```
        int width=(int)stack.getLayoutBounds().getWidth();
        int height=(int)stack.getLayoutBounds().getHeight();

        Rectangle rec=new Rectangle(x,y,width,height);
        BufferedImage screenShot = robot.createScreenCapture(rec);

FileChooser fileChooserIn=new FileChooser();
fileChooserIn.setInitialDirectory(new java.io.File( pathname: "C:/users"));
fileChooserIn.setTitle("Save image");
fileChooserIn.getExtensionFilters().add(new FileChooser.ExtensionFilter( description: "jpg, png, bmp, gif",  ...extensions: "*.jpg", "*.png",
 "*.bmp", "*.gif"));
java.io.File file=fileChooserIn.showSaveDialog(primaryStage);
int extIndex=file.getPath().lastIndexOf( str: ".");
String ext=file.getPath().substring(extIndex+1);
ImageIO.write(screenShot, ext, file);
 root.getChildren().clear();
 stack.setPadding(new Insets( top: 5, right: 5, bottom: 5, left: 5));
 stack.setLayoutX(5);
 stack.setLayoutY(5);
 root.getChildren().addAll(vboxR);
        vboxStack.getChildren().clear();
        vboxStack.setSpacing(10);
        vboxStack.setLayoutX(5);
        vboxStack.setLayoutY(5);
```

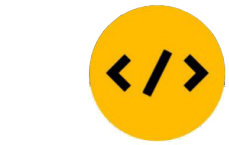
ОСНОВЫ ПРОГРАММИРОВАНИЯ С JAVA

ТИМУР МАШНИН

12+

Тимур Машнин

Основы программирования с Java



«Автор»

2022

Машнин Т.

Основы программирования с Java / Т. Машнин — «Автор», 2022

Эта книга предназначена для всех, кто хочет изучить основы программирования с использованием языка Java.Эта книга даст понимание основных элементов программирования на Java и абстракции данных с использованием объектно-ориентированного подхода. С этой книгой Вы научитесь писать программы с использованием переменных, массивов, управляющих операторов, циклов, рекурсии, абстракции данных и объектов в интегрированной среде разработки. Вы изучите основы языка программирования Java, познакомитесь с его синтаксисом, типами данных, объектами и классами и многим другим.

Содержание

Тимур Машнин
Основы программирования с Java

Введение. Что такое хорошо определенные задачи

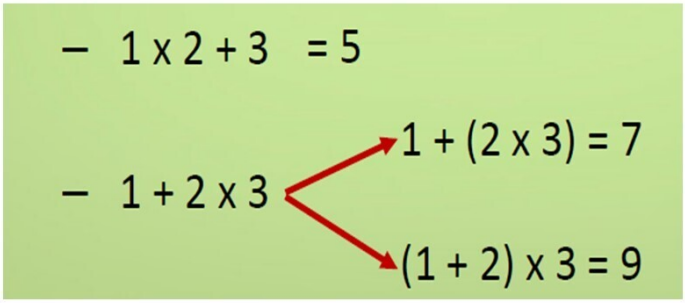
Все мы знаем, что основная цель использования компьютера, это помочь нам в решении задач. И мы используем компьютеры для решения задач, потому что они, как правило, более эффективны и надежны в решении тех или иных задач.

Тем не менее, это не всегда так, по крайней мере пока. Например, если кто-то попытается использовать компьютерную программу, чтобы понять пейзаж вокруг нас, производительность компьютера все равно будет далека от того, чтобы совпадать с ощущениями человека.

Но вы как программист должны уметь взять реальную задачу и определить соответствующую информацию, необходимую для решения этой задачи. И логически сформулировать выражения по четко определенной задаче с использованием языка программирования.

Что мы подразумеваем под четко определенной задачей.

В общем, четко определенная задача означает, что решение для задачи существует и решение может быть найдено через конечное число шагов.



Например, если вас попросили найти решение арифметического выражения "1 умножить на 2 плюс 3", вы бы знали, что мы должны сначала умножить 1 на 2, что дает промежуточный результат 2, и тогда результат умножения будет добавлен к 3, и 5 будет в качестве окончательного ответа.

Но если я попрошу вас решить задачу "1 плюс 2 умножить 3", я хочу, чтобы вы подумали о том, каким может быть ответ.

Разные люди могут давать различные ответы, потому что некоторые могут подумать, что умножение должно быть выполнено до прибавления, что во первых надо умножить 2 на 3, что дает 6 в качестве промежуточного результата, а затем добавить 6 к 1, и это дает 7 в качестве окончательного ответа, но некоторые могут просто следовать порядку операторов в арифме-

тическом выражении, в этом случае мы сначала добавим 1 к 2, и это дает 3 в качестве промежуточного результата, и затем умножим 3 на 3, и получим 9 как конечный результат.

Так какой из этих двух ответов является правильным ответом?

Это зависит от того, какие правила должны быть использованы при определении порядка операций.

И мы вернемся к этому вопросу позднее.

Из этого простого примера, можно увидеть, что некоторая дополнительная информация может быть необходима для решения этой задачи.

И вы можете реализовать решения задач с помощью Java в интегрированной среде разработки (IDE).

Есть много доступных языков программирования, и Java является лишь одним из них.

Язык Java является объектно-ориентированным языком. И вы должны научиться основам абстракции данных с помощью объектно-ориентированного программирования.

Поговорим немного о том, что такое абстракции данных.

Абстракция данных отделяет существенные свойства объектов данных от деталей того, как они реализуются

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

I, II, III, IV, V, VI, VII, VIII, IX, X

〇, 一, 二, 三, 四, 五, 六, 七, 八, 九, 十

0, 1

Абстракция данных отделяет существенные свойства объектов данных от деталей того, как они реализуются, то есть, детали реализации объектов данных скрыты от пользователей.

В большинстве приложений мы обычно заботимся только о том, как объекты данных могут быть использованы, но не как они представлены.

Например, число может быть представлено разными способами.

На человеческом языке, наиболее часто используемая система чисел – это арабские цифры 0, 1, … 9, но есть и римские цифры, обратите внимание, что только три символа, вертикальная черточка, V и крест используются для представления чисел от 1 до 10, и нет никакой специальной римской цифры для нуля.

Есть также китайские цифры и многие другие. И в компьютерах, числа представляются нулем и 1.

При этом, что независимо от того, как числа представляются, мы ожидаем, что операции, выполняемые над числами, будут подчиняться определенным правилам.

Например, при добавлении 1 к 2, как ожидается, три будет в качестве ответа, независимо оттого, что это делается китайцем, итальянцем, американцем или компьютером. И можно придумать другие жизненные примеры, использующие абстракцию.

Например, за рулем автомобиля, когда мы нажимаем на газ, автомобиль должен ускориться, и когда мы нажимаем на тормоз, предполагается, что автомобиль замедлится.

Но при этом мы не заботимся о механике, как это работает.

При использовании приложений на смартфонах, скажем iPhone, все, что вам нужно знать, это только то, что приложения должны делать, но вам действительно все равно, как они были фактически реализованы.

Вы увидите, что абстракция данных является очень важным понятием в программировании, особенно в объектно-ориентированном языке, таком как Java.

С помощью этого подхода, даже если реализация изменяется через некоторое время, поведение объекта данных или программы, как ожидается, останется прежним.

Теперь, правильный подход к решению задач требует знания информатики.

Информатика является дисциплиной, изучающей теорию, дизайн и применения вычислительных систем.

И когда мы изучаем вычислительные системы, есть три важных аспекта, а именно, аппаратные средства, программное обеспечение и аспекты применения.

Изучение аппаратных средств включает в себя проектирование и строительство компьютерных систем в виде физических устройств для выполнения программы.

Изучение программного обеспечения включает в себя рассмотрение поведения алгоритмов, компьютерных программ, чтобы определить, работают ли они правильно и эффективно.

В центре внимания данной книги – программное обеспечение компьютерных систем.

В конце концов, основное использование компьютеров, это решение реальных задач.

В этой книги вы узнаете некоторые фундаментальные концепции программирования для решения задач с использованием компьютерных программ.

Здесь мы будем использовать много примеров, которые работают с фотографиями и изображениями, чтобы проиллюстрировать некоторые важные концепции программирования.

И вы сможете применить эти понятия для решения реальных задач.

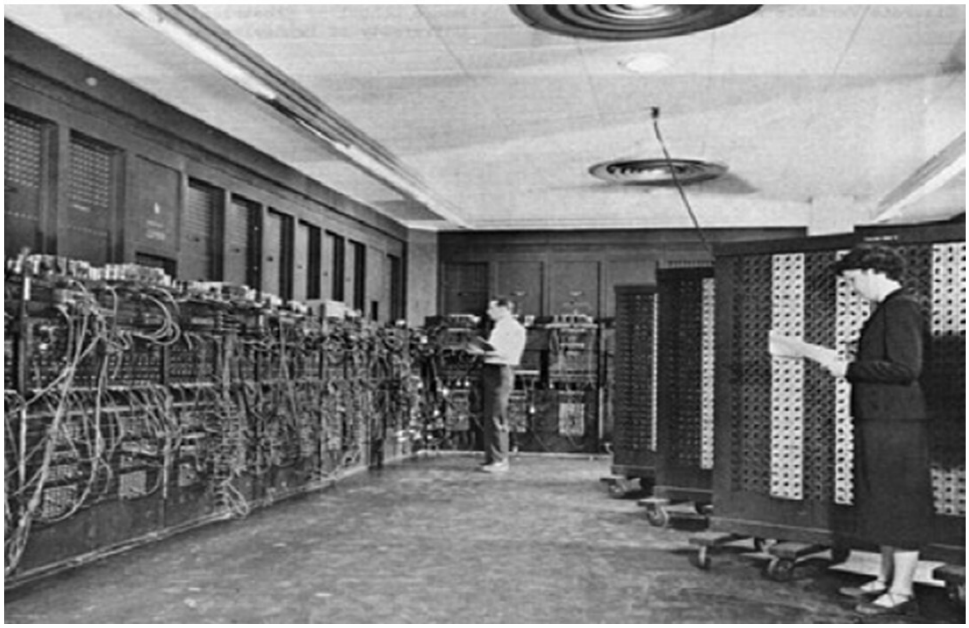
Аппаратные средства

Давайте начнем рассмотрение основных компонентов компьютерных систем. И начнем мы с аппаратных средств.

В общем и целом, есть два основных компонента в компьютере, а именно аппаратные и программные компоненты.

В 1946 году, первый, общего назначения, электронный компьютер в мире был построен Пенном.

Он назывался ENIAC. ENIAC означает электронный цифровой интегратор и компьютер.



Как вы можете видеть на этом слайде, ENIAC был большой машиной, которая весила более 30 тонн, содержала приблизительно 18000 вакуумных трубок, и размещалась в большой комнате 180 квадратных метров.

Если сравнивать с сегодняшним компьютером, вы увидите, что компьютерный чип, который используется в вашем мобильном телефоне, во много-много раз более мощный, чем ENIAC.

Существует важное наблюдение, сделанное Гордоном Муром, сооснователем Intel, он предсказал в 1965 году, что мощность компьютерного чипа удваивается примерно каждые 18 месяцев.

Это наблюдение еще работает и сегодня.

Это имеет некоторые очень важные последствия, потому что задача, которая требует одну минуту времени обработки, используя сегодняшнюю машину, потребовала бы более 40 лет времени обработки с помощью компьютера, разработанного 40 лет назад, то есть, машина все равно бы не выполнила задачу после 40 лет.

В общем и целом, компьютер представляет собой электронное устройство, которое работает под управлением команд (или программ), хранящихся в запоминающем устройстве.

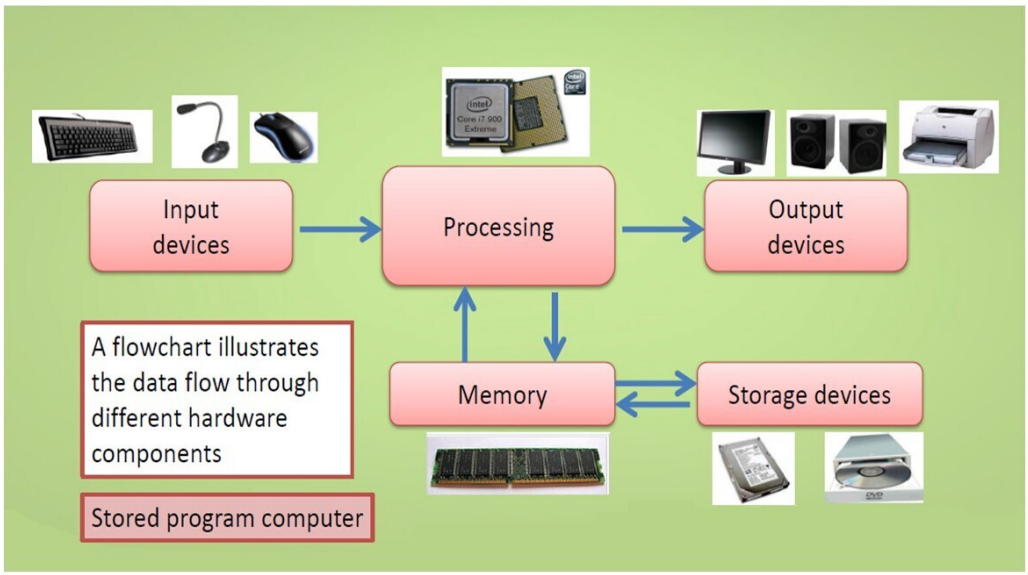


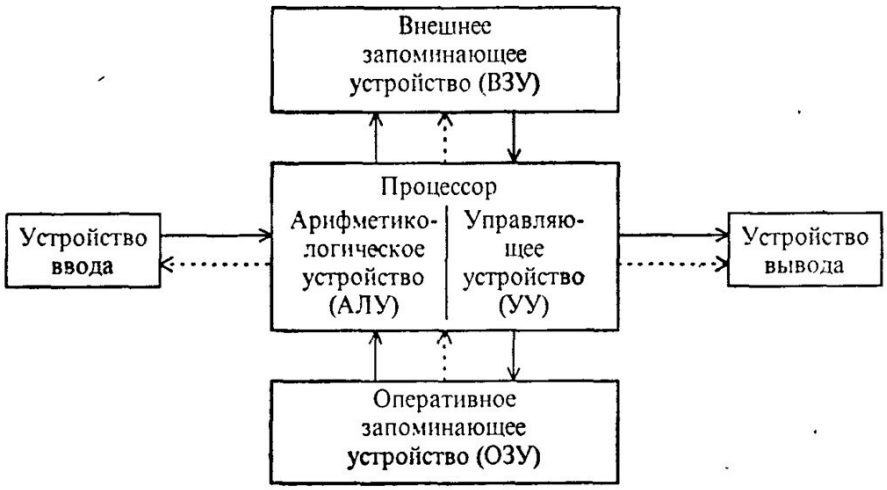
Диаграмма здесь иллюстрирует поток данных с помощью различных аппаратных компонентов.

Сначала принимаются данные с устройств ввода, далее данные обрабатываются арифметически и логически с помощью получения инструкций и данных из основной памяти, иногда необходимо получить больше данных и файлов из устройств хранения.

Затем программа будет выдавать результаты на устройства вывода, иногда она может передавать результаты обратно к устройствам хранения данных для использования в будущем.

Эту модель вычислений часто называют хранимой программой компьютера.

Основным компонентом компьютера является центральный процессор, который считается мозгом компьютера.



Процессоры можно найти во многих современных устройствах, включая ПК, ноутбуки и мобильные устройства, такие как смартфоны и планшеты.

Центральный процессор получает инструкции из памяти и выполняет вычисление данных.

И ЦП, как правило, состоит из двух частей, а именно, это арифметико-логическое устройство и блок управления.

Арифметико-логическое устройство ALU отвечает за вычисления, в том числе основных арифметических операций, таких как сложение, вычитание, умножение и деление, и логической оценки данных, в том числе логических сравнений, таких как "равно", "больше чем" или "меньше чем".

Блок управления контролирует и координирует общие операции внутри компьютера.

Основные функции блока управления включают в себя:

- Управление доступом к главной памяти хранения.
- Управление последовательностью, в которой команды выполняются.
- Регулирование времени всех операций, осуществляемых в CPU.
- Отправка и прием сигналов управления в и из периферийных устройств, таких как клавиатура и принтер.
- Управление потоком данных между ALU и основной памятью.

Арифметико-логическое устройство ALU отвечает за вычисления, в том числе основных арифметических операций, таких как сложение, вычитание, умножение и деление, и логической оценки данных, в том числе логических сравнений, таких как "равно", "больше чем" или "меньше чем".

Блок управления контролирует и координирует общие операции внутри компьютера.

Основные функции блока управления включают в себя:

Управление доступом к главной памяти хранения.

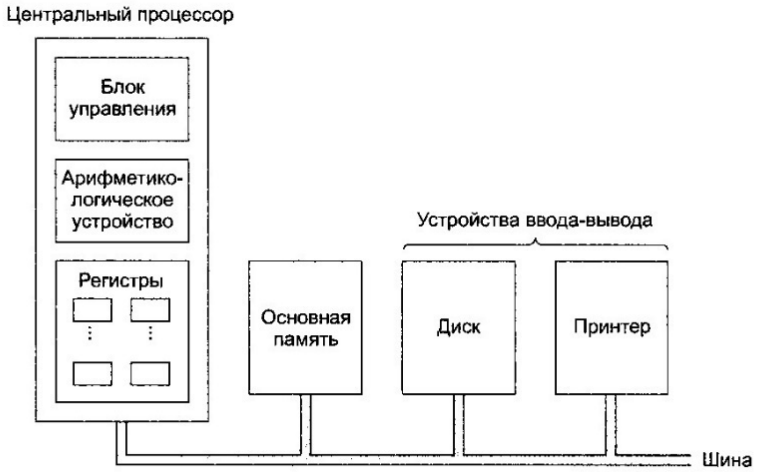
Управление последовательностью, в которой команды выполняются.

Регулирование времени всех операций, осуществляемых в CPU.

Отправка и прием сигналов управления в и из периферийных устройств, таких как клавиатура и принтер.

Управление потоком данных между АЛУ и основной памятью.

Существует три вида памяти для работы компьютера.



Это память, которая находится внутри процессора, и которая известна как «регистр».

Регистры, это быстрая память для хранения данных, которые процессор в настоящее время обрабатывает.

Все данные должны быть сохранены в регистре, прежде чем они могут быть обработаны.

Например, в сложении двух чисел, оба числа должны быть в регистрах, а результат также будет помещен в регистр.

Это основная память, которая также известна как оперативное запоминающее устройство (ОЗУ), которая содержит инструкции программы и данные для программы.

В современных компьютерах, процессоры снабжены также кэш-памятью, которая хранит часто используемые данные для того, чтобы сократить время доступа к оперативной памяти.

Устройства ввода несут ответственность за получение информации от пользователей.

Самым распространенным устройством ввода является клавиатура, которая стала стандартным устройством для большинства компьютеров.

Кроме того, существуют и другие устройства ввода, которые помогают улучшить пользовательский интерфейс, например, мышь.

Некоторые устройства ввода разработаны с конкретной целью.

Например, микрофон для записи звука, считыватель штрих-кода для считывания штрих-кодов, сканер для сканирования документов и цифровой фотоаппарат для съемки.

Я уверен, что вы можете придумать и другие примеры.

Устройства вывода отвечают за представление информации для пользователей.

Два основные типы устройств вывода, это мониторы и принтеры.

Мониторы отображают информацию на экране.

Тем не менее, информация, представленная таким образом, является энергозависимой и не-портативный.

Принтеры часто используются, чтобы вывести информацию на внешний носитель, который можно хранить отдельно и переносить.

Существуют и другие устройства вывода, например, динамики для вывода звука, плоттеры для построения графиков, и некоторые специально разработанные устройства вывода, такие как дисплей Брайля, который можно использовать в качестве устройства для тактильного зрения.

В настоящее время, так как мобильные устройства набирают популярность, сенсорные экраны используются в качестве устройств ввода и вывода.

Опять же, вы можете придумать и другие примеры.

Кроме того, существуют внешние запоминающие устройства, такие как CD, DVD и жесткие диски.

Емкость жесткого диска в настоящее время может легко держать несколько сотен гигабайт данных.

Запоминающие устройства являются энергонезависимыми носителями данных, то есть, они могут хранить данные постоянно, даже когда питание отключено.

В общем и целом, они медленнее и менее эффективны, чем основная память.

В течение последних 10 лет, такой вид носителей, становится очень популярным, как USB или карты памяти.

USB накопители в настоящее время могут легко хранить 10-ки и даже более 100 Гбайт данных.

Одним из последних направлений является развитие облачных систем хранения данных, которые позволяют хранить большое количество данных, которые хранятся в центрах обработки данных, доступных через Интернет.

Программное обеспечение

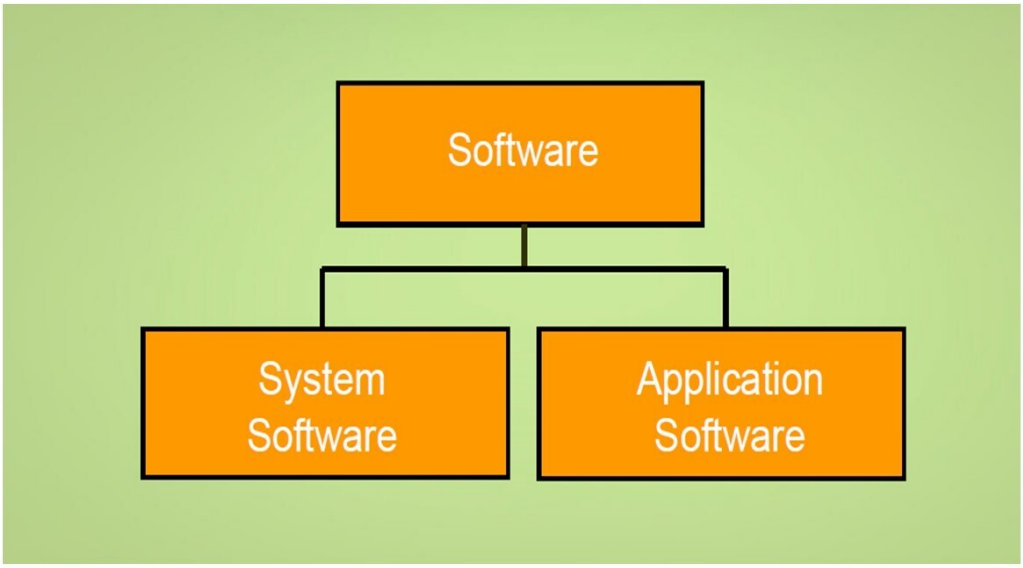
Давайте продолжим обсуждение основных компонентов компьютерных систем и рассмотрим программное обеспечение компьютера.

Компьютерная техника сама по себе не была бы очень полезна.

Это как после строительства здания для библиотеки. До того, как книги помещаются на книжные полки, здание не будет являться библиотекой.

Нужно предоставить компьютеру четкие инструкции для того, чтобы выполнить что-то полезное.

Этот вид инструкций можно назвать программным обеспечением.



Программное обеспечение представляет собой набор инструкций, которые даются компьютеру для выполнения определенных задач.

В общем и целом, программное обеспечение, используемое в компьютере, может быть классифицировано на две основные категории, а именно, системное программное обеспечение и прикладное программное обеспечение.

Прикладное программное обеспечение представляет собой программы, которые предназначены для выполнения конкретных задач и может легко использоваться пользователями.

В центре внимания этого курса – разработка прикладного программного обеспечения.

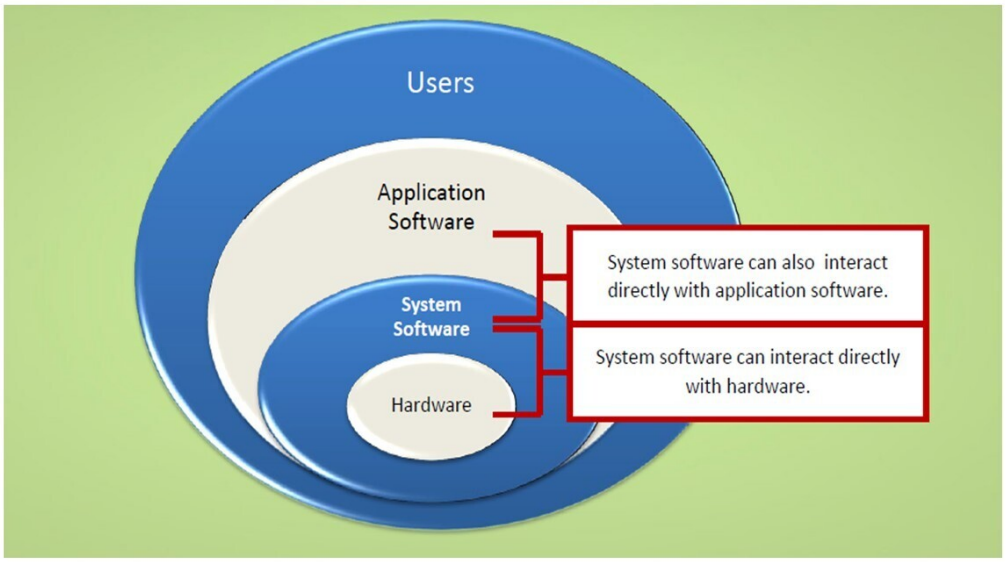
Системное программное обеспечение, это программы, которые поддерживают выполнение и разработку других программ.

Некоторые примеры системного программного обеспечения, это операционные системы и компиляторы для языков программирования, таких как Java, Python и C + +.

Другой тип системного программного обеспечения называется утилитами.

Некоторые примеры утилит включают в себя антивирусную программу и драйверы для подключения различных устройств к компьютерам.

Вот схема, которая иллюстрирует взаимодействие между железом и различными типами программного обеспечения.



На схеме видно, что системное программное обеспечение может с одной стороны взаимодействовать с оборудованием и с другой стороны взаимодействует с прикладным программным обеспечением.

Например, прикладному программному обеспечению, возможно, придется выдавать команды оборудованию, подключенному к компьютеру через системное программное обеспечение, например, сказать, когда программа хочет распечатать некоторые результаты на принтере.

Диаграмма также показывает, что существует взаимодействие между пользователями и прикладным программным обеспечением.

Эти пользователи могут быть прикладными программистами, которые разрабатывают программы, или пользователями приложений.

Существует также прямой интерфейс между пользователями и системным программным обеспечением, и таких пользователей обычно называют системными программистами.

Давайте также взглянем на то, что мы подразумеваем под пользовательским интерфейсом.

В общем и целом, пользовательский интерфейс обеспечивает взаимодействие между человеком и компьютером.

Общая цель обеспечения взаимодействия между человеком и компьютером заключается в создании пользовательского интерфейса, который прост в использовании и легко изучается, другими словами, мы хотим удобный интерфейс.

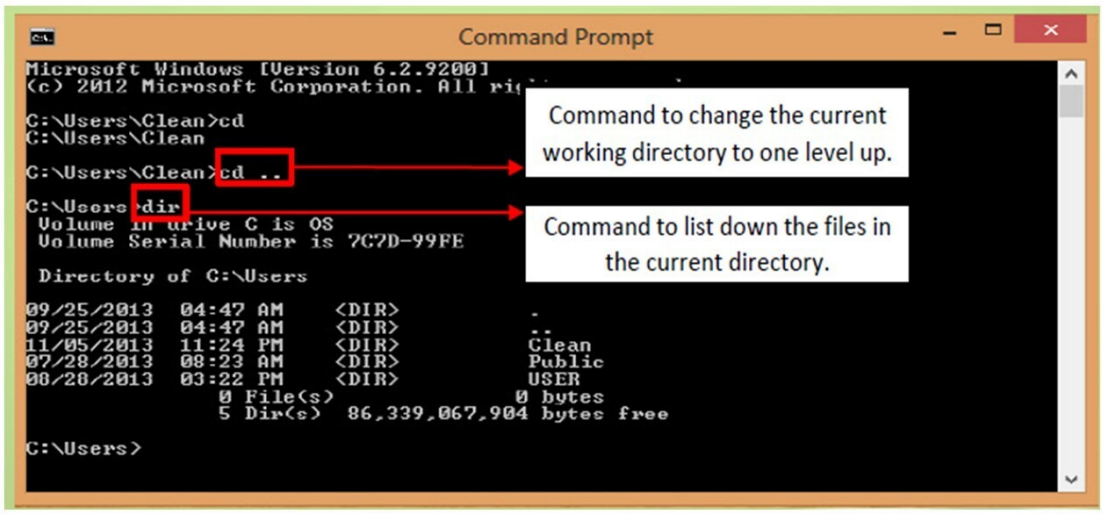
Важной частью ОС является обеспечение пользовательского интерфейса между операционной системой и пользователем.

Например, файловая система является частью операционной системы.

Хороший интерфейс для системы управления файлами позволит пользователям поддерживать и манипулировать своими файлами эффективно и результативно.

В существующих компьютерах есть в основном два вида пользовательских интерфейсов, а именно интерфейс командной строки и графический интерфейс пользователя (или GUI).

С интерфейсом командной строки можно ввести определенные ключевые слова или команды, чтобы инструктировать ОС для выполнения определенных действий.



Примером командной строки является программа cmd, которую вы найдете в системе Microsoft Windows.

Для этого вида интерфейса необходимо помнить набор команд для того, чтобы взаимодействовать с системой.

Например, команда "cd ..", чтобы изменить текущий рабочий каталог на один уровень вверх, и "dir", чтобы вывести список файлов в текущем каталоге.

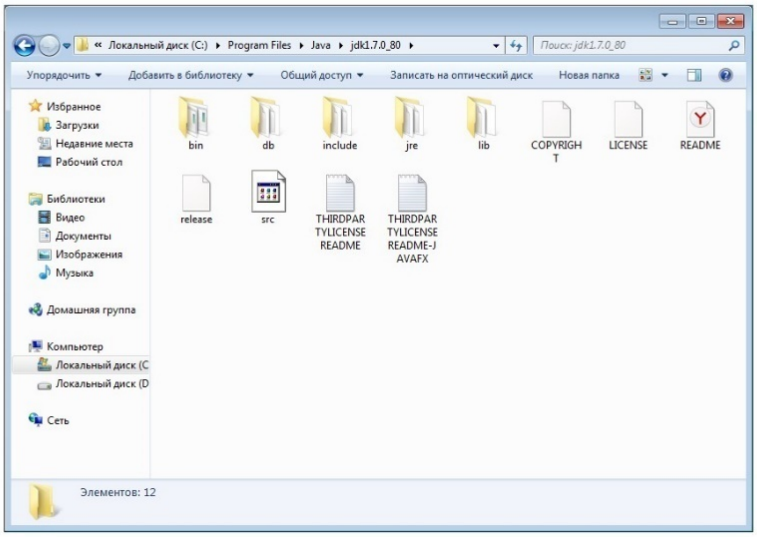
Из этого простого примера, можно понять, почему такие системы стали настолько непопулярны и редко используются, за исключением системных программистов или системных администраторов.

Я уверен, что все из вас знакомы с каким-нибудь графическим интерфейсом пользователя (GUI). В GUI вы можете найти на экране графические объекты, такие как значки, выпадающие меню и окна.

Пользователь может навести на точку и нажать на нее с помощью указывающего устройства, такого как мышь, чтобы активировать необходимые операции.

Одним из основных преимуществ, для такого рода интерфейса, является то, что вам не придется запоминать различные команды, как в интерфейсе командной строки.

Например, в окне каталога файловой системы, вы можете легко манипулировать файлами с помощью перетаскивания.



Я не думаю, что здесь необходимы дополнительные объяснения, так как вы все должны быть знакомы с такими операциями.

Мы часто используем графический пользовательский интерфейс, разработанный другими.

И в этой книге вы также узнаете, как разрабатывать свой собственный графический интерфейс пользователя с помощью Event Driven программирования.

Прикладное программное обеспечение и операционная система

Давайте теперь обсудим прикладное программное обеспечение.

Компьютеры совершили революцию в нашей жизни, потому что они могут выполнять различные задачи эффективным и надежным способом.

Задачи, которые компьютер может выполнять, включают в себя многие бизнес-операции и повседневные приложения.

И цель состоит в том, чтобы увеличить производительность труда и качество нашей жизни.

Например, компании используют информационные системы учета людских ресурсов, чтобы выплачивать заработную платой, и университеты используют информационные системы учета студентов, чтобы отслеживать прогресс обучения студентов.

В настоящее время, есть программы, с помощью которых вы контролируете приборы дома (их часто называют смарт-устройства) с помощью мобильных устройств, так что, например, еда будет готова для вас, прежде чем вы вернетесь домой.

Некоторое широко используемое программное обеспечение может быть сгруппировано в следующие категории.

Производительное программное обеспечение:
MS Word для подготовки документов, Excel Access для создания баз данных, PPT для подготовки презентаций

Коммуникационное программное обеспечение:
Internet Explorer, Safari, Chrome или Firefox для поиска информации в WWW

Мульти-медиа программное обеспечение:
PhotoShop

Мобильные приложения

Первая категория, это производительное программное обеспечение, например, MS Word для подготовки документов, Excel Access для создания баз данных и PPT для подготовки презентаций.

Легко видеть, что обработка программным обеспечением документа отличается, если сравнить усилия, необходимые для создания отчета с использованием в настоящее время компьютера и пишущей машинки 30 или 40 лет назад.

Я думаю, большинство из вас, вероятно, никогда использовали или даже не видели пишущую машинку раньше.

Другой вид прикладного программного обеспечения, с которым я уверен, многие из вас знакомы, является коммуникационным программным обеспечением, особенно то, которое вы используете для общения через Интернет.

Здесь вы можете использовать интернет-браузеры, такие как Internet Explorer, Safari, Chrome или Firefox для поиска информации в WWW.

Многие из вас общаются с друзьями по электронной почте или через социальные сети.

В прошлом, большинство книг и документов состояли в основном из текста, так как было не легко и часто дорогостояще включать в них изображения, но с достижениями в области мультимедийных средств, таких как Photoshop, можно легко создавать документы с красивой графикой и изображениями или даже видео.

На самом деле, много бумажных носителей были заменены на электронные средства массовой информации, включая газеты и журналы.

В последнее время из-за достижений в области технологий мобильной связи, в том числе мобильных сетей и мобильных устройств, таких как смартфоны и планшеты, очень быстро увеличилось использование и доступность мобильных приложений.

Есть мобильные приложения для игр и для общения с другими людьми, развлекательные приложения для прослушивания музыки или просмотра фильмов, навигационные приложения, такие как Google Maps для путешествий, и многое другое.

Хорошая новость состоит в том, что многие из этих приложений написаны на Java, так что вы можете применить свои навыки программирования на Java после этой книги.

Давайте теперь посмотрим на операционные системы.

ОС, это в основном системное программное обеспечение для контроля и управления компьютерными ресурсами, включая устройства ввода или вывода, такие как жесткие диски, монитор, клавиатура и принтер. Некоторые из них могут совместно использоваться сразу несколькими пользователями.

Некоторые популярные ОС предназначены для компьютеров, например Apple, это Mac OS, другие для мобильных устройств, например IOS для iPhone и IPad.

Компания Microsoft начала продавать MS DOS в начале 1980-х и операционную систему Windows в середине 80-х.

Последняя версия – Windows 10, которая обеспечивает пользовательский интерфейс для мобильных устройств.

Другие ОС включают UNIX и Linux, которая также является UNIX-подобной ОС.

Другая популярная ОС для мобильных устройств – Android, разработанная Google.

Android разработан на основе ядра Linux и имеет сейчас самую большую долю рынка среди всех мобильных ОС.

Вот некоторые из важных сервисов, предлагаемых операционной системой.

Некоторые из важных сервисов, предлагаемых операционной системой:

Управление файловой системой

Интерпретация команд для управления файловой системой

Ввод и вывод на различные устройства

Управление окнами

Одной из наиболее важных задач является управление файловой системой, где хранятся файлы, которые организованы в виде иерархической структуры в виде папок, это особенно важно, когда вы пытаетесь найти и извлечь определенный файл из большого количества файлов.

Как упоминалось ранее, ОС позволяет вам манипулировать файловой системы легко с помощью графического интерфейса пользователя, чтобы копировать и удалять файлы или перемещать файлы с помощью перетаскивания файлов в разные папки.

Операционная система также несет ответственность за управление устройствами ввода и вывода.

Когда программа выдает команду для получения некоторой информации из устройства ввода или написать что-то на устройство вывода, ОС берет это на себя.

Нам часто приходится открывать несколько окон одновременно, чтобы обрабатывать различные задачи, например, когда вы смотрите YouTube, вы можете также что-писать.

И многочисленные окна, которые вы открываете, управляются ОС.

Языки программирования

Давайте теперь обсудим языки программирования.

Перед этим мы говорили о прикладном программном обеспечении и системном программном обеспечении.

Это на самом деле программы, которые были написаны на определенных языках программирования.

Например, операционная система UNIX, которая является системным программным обеспечением, написана в основном на языке C.

Когда компьютеры общего назначения впервые появились в 1940 году, программы должны были быть написаны на машинном языке или языке ассемблер, который, как правило, называется языком низкоуровневого программирования, потому что программы должны были быть написаны на основе примитивных машинных инструкций для конкретной компьютерной архитектуры.

То есть, программа должна была быть переписана полностью, когда она переносилась в другую машину.

В настоящее время, большинство программ написано на языках программирования высокого уровня.

Языки программирования высокого уровня используют возможности языка ближе к человеческим языкам, например, к английскому, чем машинному языку.

Преимущество использования языков программирования высокого уровня состоит в том, что их легче читать, писать и поддерживать.

Вот некоторые примеры из языков программирования высокого уровня:

Fortran рассматривается как один из первых языков программирования высокого уровня. Он был разработан в 1950-х годах.

Cobol является вторым старейшим языком программирования высокого уровня и используется в основном для бизнес-приложений.

Basic является одним из самых простых в изучении языков.

C это язык программирования общего назначения, разработанный в AT&T Bell Labs в начале 1970-х.

Java был разработан в 1990-х годах Sun Microsystems, которая была приобретена корпорацией Oracle.

Fortran рассматривается как один из первых языков программирования высокого уровня. Он был разработан в 1950-х годах.

Fortran особенно подходит для научных приложений, так как есть обширная коллекция научных пакетов, написанных на языке Fortran за всю его долгую историю существования.

Кроме того, многие из научных приложений, написанных для суперкомпьютеров или высокопроизводительных компьютеров, написаны на языке Fortran.

Cobol является вторым старейшим языком программирования высокого уровня и используется в основном для бизнес-приложений.

Вы до сих пор можете найти, что многие системы банков и финансовых учреждений написаны на Cobol.

Basic является одним из самых простых в изучении языков.

VBA Microsoft или Visual Basic for Applications является реализацией Visual Basic вместе со своей интегрированной средой разработки.

«C» – это язык программирования общего назначения, разработанный в AT&T Bell Labs в начале 1970-х.

Многие черты C были приняты многими более поздними языками, включая C++, который является объектно-ориентированной версией C.

Другой язык программирования, который приобрел популярность в последнее время, это Python, который появился в 1990-х годах.

Он поддерживает несколько парадигм программирования, в том числе императивное, объектно-ориентированное и функциональное программирование.

Java это язык программирования, который мы собираемся использовать здесь, был разработан в 1990-х годах Sun Microsystems, которая была приобретена Oracle.

Он особенно популярен для веб-приложений и мобильных приложений.

Одним из важных преимуществ Java является то, что Java код, который был компилирован на одной платформе, не придется перекомпилировать для работы на другой платформе.

Это потому, что программы Java компилируются в форме, называемой байт-кодом, который может быть запущен на виртуальной машине Java (JVM), установленной на другом компьютере.

Вышеуказанный список приводит здесь лишь некоторые из наиболее популярных языков программирования высокого уровня,

Есть еще очень много, которые были созданы в прошлом, а новые, безусловно, будут созданы в будущем.

Следующий список дает основные мероприятия в цикле разработки программного обеспечения.

Основные виды деятельности в области разработки программного обеспечения:

Редактирование исходного кода

Компиляция (создает файл .class)

Сборка скомпилированных файлов (создает файл .exe)

Загрузка и выполнение

Тестирование программы

Программы могут быть написаны, используя какой-либо из редакторов.

Это может быть простой текстовый редактор, такой как блокнот, или более сложный редактор, предоставляемый средой разработки.

В настоящее время большинство популярных языков программирования оснащены специально разработанными редакторами для языка.

Программы, написанные на языке программирования высокого уровня, должны быть откомпилированы или переведены на машинный язык, прежде чем они могут быть выполнены.

Программа компиляции является своего рода системным программным обеспечением, потому что она взаимодействует с другими программами.

Программа компиляции берет программу в качестве входных данных, а затем переводит ее в понятный машиной язык или объектный код.

Этот процесс называют компиляцией.

После компиляции программы, она должна пройти через другой процесс, который называется связыванием или сборкой и который связывает программу с другими программами или библиотеками, которые были включены в оригинальную программу.

В случае Java, есть очень богатая коллекция существующих программ, упакованных в виде библиотек, например, библиотека для часто используемых математических функций, таких как корень квадратный и тригонометрические функции – синус, косинус и тангенс.

Это очень важно при разработке программ, потому что мы можем использовать то, что было написано раньше и не должны разрабатывать все с нуля.

Повторное использование программы является общим действием программной инженерии, которое может сэкономить время и усилия за счет сокращения лишней работы.

Процесс связывания создает компьютерный исполняемый код, который обычно хранится в .exe файле или .jar файле.

Чтобы действительно выполнить программу, исполняемый код должен быть перемещен оттуда, где он хранится, например, на жестком диске, в основную память, где может быть осуществлено выполнение программы, и этот процесс называется загрузка.

Программу необходимо протестировать для разных вводов, прежде чем она может быть опубликована.

Если обнаружена ошибка, программу нужно пересмотреть и провести через последовательность шагов разработки снова.

В настоящее время, все эти процессы могут осуществляться в интегрированной среде разработки (IDE).

Интегрированная среда разработки (IDE) является программным приложением, которое обеспечивает интерактивные инструменты для программистов, чтобы упростить цикл разработки программного обеспечения и таким образом, улучшить производительность.

Далее приведем общие компоненты IDE.

Общие компоненты среды IDE:

Редактор

Компилятор

Сборщик

Загрузчик

Отладчик

Они в основном соответствуют шагам в цикле разработки программного обеспечения, о которых мы только что говорили.

Редактор в IDE часто предоставляет инструменты, которые помогут вам отформатировать и документировать программы.

Некоторые могут даже помочь вам сделать некоторую первоначальную проверку на синтаксис.

Компилятор, сборщик и загрузчик обеспечивают компиляцию, связывание и загрузку программы для исполнения, как обсуждалось ранее.

Часто существуют ошибки в программах, даже для программ, написанных опытными программистами.

Отладчик может помочь в выявлении и локализации ошибок.

Он позволяет программисту проследить шаг за шагом выполнение программы.

В отладке программы, есть два распространенных типа ошибок, а именно синтаксические ошибки и семантические ошибки.

Синтаксис языка программирования представляет собой набор правил, которые определяют комбинацию символов, которые могут быть правильно использованы вместе в языке.

Это похоже на грамматику в естественном языке, таком как русский или английский язык.

Семантика относится к значению программы, то есть, что программа должна выполнить.

Программа может быть синтаксически правильной, но она может не давать предполагаемое значение.

В естественных языках, таких как русский или английский, может присутствовать двусмысленность, человек делает толкование и может попросить разъяснений, если значение не ясно.

Например, если вы бы дали следующее указание, подумайте о том, какое может быть значение этой инструкции.

Например, инструкция может быть "казнить нельзя помиловать".

Таким образом, это предложение являются синтаксически или грамматически правильным, но семантически неоднозначным.

Для компьютера, он всегда даст каждой программе ровно одну интерпретацию.

Мы будем использовать в основном IntelliJ IDEA как нашу интегрированную среду разработки или IDE здесь, и вы будете иметь лучшее представление о каждом из этих компонентов.

Как решать задачи?

Прежде чем мы рассмотрим, как компьютеры могут быть использованы для решения задач, давайте вначале рассмотрим, как мы обычно решаем задачи в реальной жизни.

Процесс решения задачи, которому мы обычно следуем, не ограничивается использованием только компьютера.

Определите и проанализируйте задачу.

Разработайте решение.

Запишите шаги решения подробно.

Проверьте и оцените решение, при необходимости измените его.

Документируйте и поддерживайте решение.

1-й шаг должен определить и проанализировать задачу, которую вы пытаетесь решить, чтобы мы могли получить хорошее понимание задачи.

На этом этапе, в основном, вы пытаетесь придумать спецификацию задачи. Это особенно важно, когда вы решаете задачу с помощью компьютера.

Компьютер не может читать ваши мысли, вы должны дать компьютеру точные инструкции о шагах выполнения.

Так что этот шаг очень важен, потому что вы должны сначала дать себе четкое понимание задачи, прежде чем вы можете сказать компьютеру, что вы от него хотите.

После того как вы получили спецификацию задачи, следующим шагом будет разработать решение.

Во многих задачах может быть несколько решений.

Итак, вы хотите разработать решение, которое наилучшим образом соответствует текущей ситуации или ограничениям.

Например, при попытке решить задачу добраться из одного места в другое, ограничения могут быть в том, что вы должны попасть в определенное время и с определенным бюджетом.

После того как вы определились с решением, вы должны разработать детали реализации, в том числе шаг за шагом реализацию решения.

После того как вы закончили разработку реализации решения, вы должны выполнить некоторые тесты и оценки, чтобы убедиться, что ваша реализация правильно решает проблему.

Это очень важно, потому что ваше решение может оказаться не в состоянии обработать все проблемные случаи.

Этот шаг часто является повторяющимся процессом, и возможно, придется пересмотреть решение или улучшить его, чтобы удовлетворить всем ограничениям и условиям.

И последнее, но не менее важное, вы должны задокументировать решение так, что, если вы или другие люди захотят вновь обратиться к проблеме в более позднее время, вы все равно сможете легко понять это решение.

Документация также помогает поддерживать решение в случае, если оно нуждается в пересмотре или обновлении для возможных будущих изменений.

Используем пример поиска способа путешествовать из Москвы в Лондон.

Предположим, что вы турагент, и определение задачи может быть что-то вроде "Найти лучший способ для вашего клиента, чтобы проехать из центра Москвы в Лондон в Великобритании".

Анализируя эту задачу, вам придется узнать у клиента, что он или она имеет в виду под самым лучшим способом, это кратчайшее расстояние или лучшее время или дешевая стоимость.

Эскизный проект может начаться с рассмотрения всех возможных маршрутов и видов транспорта, возможно, с помощью Google Maps или других сайтов туристических услуг.

И чтобы уточнить решение, нужно оценить различные маршруты и виды транспорта, а затем выбрать маршрут, который наилучшим образом соответствует вашей цели, например, самый короткий маршрут или самый дешевый по стоимости.

Как только вы создали решение, тестирование может быть трудным для этой конкретной задачи, если вы не можете фактически проделать это путешествие.

Если вы не можете выполнить эту поездку, тогда вы можете оценить решение, проверив записи определенных рейсов, чтобы убедиться, что есть достаточно времени для соединения рейсов, если это необходимо.

Или опросить кого-нибудь с предыдущим опытом создания подобной поездки.

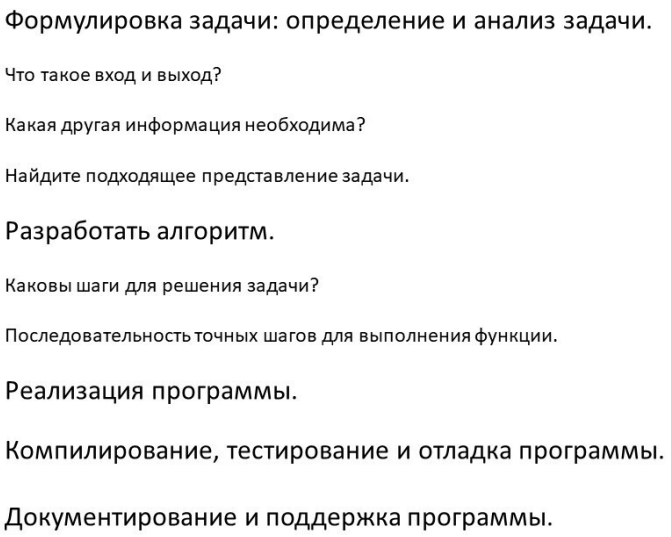
Затем вам нужно будет задокументировать свое решение, предоставляя простые инструкции вашему клиенту, чтобы он не упустил соединение рейсов или чтобы было достаточно местной валюты для использования общественного транспорта.

После того как ваш клиент завершил поездку, вы сможете получить обратную связь от него, чтобы можно было вести учет того, был ли положительным опыт или пересмотреть решение, если ваш клиент не был доволен.

И это эквивалентные шаги при попытке запрограммировать компьютер как процесс решения задач.

При определении задачи для компьютера, мы должны придумать очень точные спецификации задачи.

И одним из распространенных подходов, является придумать спецификацию для начального состояния или входную спецификацию и итоговую спецификацию для конечного состояния или выходную спецификацию для задачи.



Это полезно, потому что путь пользователя для взаимодействия с компьютером часто лежит через устройства ввода/вывода.

Вам также необходимо определить, какая дополнительная информация нужна для решения задачи.

Используя предыдущую задачу в качестве примера, вам, возможно, придется узнать, во сколько вы должны прибыть в пункт назначения и сколько денег вы готовы потратить.

Когда мы разрабатываем решения для реальных жизненных задач, мы часто записываем шаги на определенном языке, таком как русский или английский.

При решении задач с помощью компьютера, мы также хотим перечислить шаги на языке, который могут легко понять все те, кто участвует в решении проблемы, это особенно важно, если вы работаете в команде. Такую последовательность шагов часто называют алгоритмом.

В общем, алгоритм представляет собой последовательность точных шагов на английском или другом человеческом языке для выполнения определенных функций.

Очень часто, алгоритм разрабатывается на разных уровнях детализации с помощью интерактивного процесса. Такой подход часто называют пошаговым уточнением.

Выше были приведены подготовительные шаги перед тем, как вы на самом деле реализуете программу или начнете программировать решение задачи.

Можно подумать, что реализация программы является наиболее трудным шагом, но, если вы проделали хорошую работу в процессе подготовки, шаг кодирования будет сделан просто, и перевод с очень хорошо продуманного алгоритма может быть самой легкой частью среди всех этих шагов.

Поэтому, когда вы решаете компактную проблему, мы советуем вам удержаться от соблазна начать кодирование, прежде чем вы получите отчетливое понимание решения задачи.

Еще одно решение, которое нужно сделать в процессе реализации, это нужно определить, какой язык программирования использовать и какое представление будет использоваться для различных аспектов задачи.

После окончания кодирования программы, она должна быть скомпилирована в исполняемый код компьютера, и она должна быть протестирована, используя различные сценарии, чтобы проверить все ли в порядке при различных условиях.

Если возникают ошибки, или если программа не работает, как задумано, нужно отследить ошибки. Этот шаг часто называют шагом отладки.

Для исторической справки. Термин отладки (debugging) используется, потому что первая известная компьютерная ошибка была найдена в 1947 году, когда мотылек (насекомое – bug) был пойман в ловушку в реле компьютера.

Насекомое было записано на пленку в журнале учета Грейс Хоппер и в настоящее время хранится в Смитсоновском музее.

Для сложных проблем, важно протестировать программу в рамках различных сценариев, так как программа, которая работала в одном случае, не означает, что она будет работать для всех случаев.

И полезно придумать план тестирования, чтобы охватить набор общих случаев, которые можно было бы ожидать при работе программы.

Тесты могут привести к двум распространенным типам ошибок, а именно синтаксической ошибки или семантической ошибки.

Также важно документировать и поддерживать программу, особенно для программ, жизнь которых, как ожидается, продлится в течение длительного времени, и для повторного использования, и программ, которые могут быть пересмотрены или изменены другими пользователями.

Игра

Прежде чем погрузиться в программирование, давайте более внимательно посмотрим на важность постановки задачи и представление процесса путем изучения обычно используемого подхода к решению задачи под названием представление пространства состояний.

В представлении пространства состояний, задача представляется в виде множества состояний.

И я буду использовать примеры для иллюстрации того, что я имею в виду под состояниями.

Пространство состояний, это множество всех возможных состояний, в которых задача может находиться.

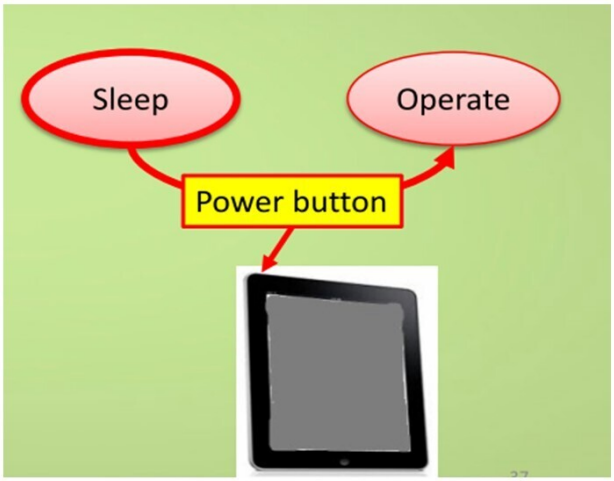
Пространство состояний, это множество всех возможных состояний, в которых задача может находиться

В частности, есть множество начальных состояний, то есть там, где начинается задача, и набор конечных состояний, в том числе всех возможных решений задачи.

Два состояния связаны, если существует действительная операция, которая может превратить одно состояние в другое.

Давайте рассмотрим несколько примеров, чтобы получить более полное понимание представления пространства состояний.

Это простой пример, и задача заключается в том, чтобы включить смартфон.



Здесь есть два состояния "сон" и "работа" и они соединены операцией нажатия на кнопку питания.

Первоначально, смартфон находится в «спящем» состоянии.

Для перехода к состоянию "работать", пользователь нажимает на кнопку питания.

И смартфон переключается из состояния "сна" в состояние "работать".

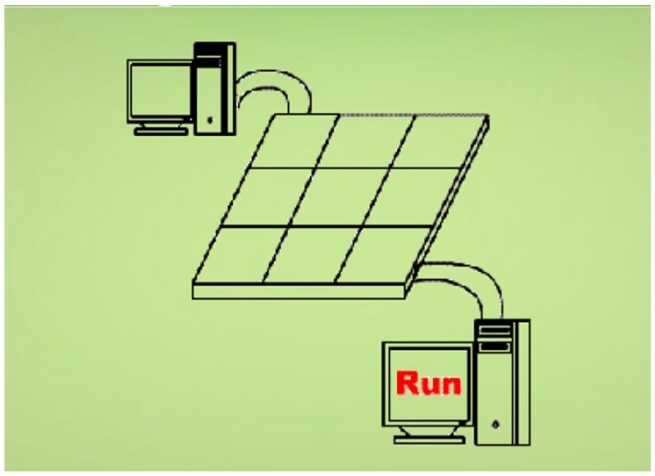
Пользователь может переключиться на «спящее» состояние смартфона снова, нажав на кнопку питания.

В этом примере, вероятно, нельзя сказать много о полезности представления пространства состояний.

Давайте теперь рассмотрим более интересный пример – игру крестики-нолики.

В этой игре, два игрока ставят "крестик" или "кружок" в таблице 3x3.

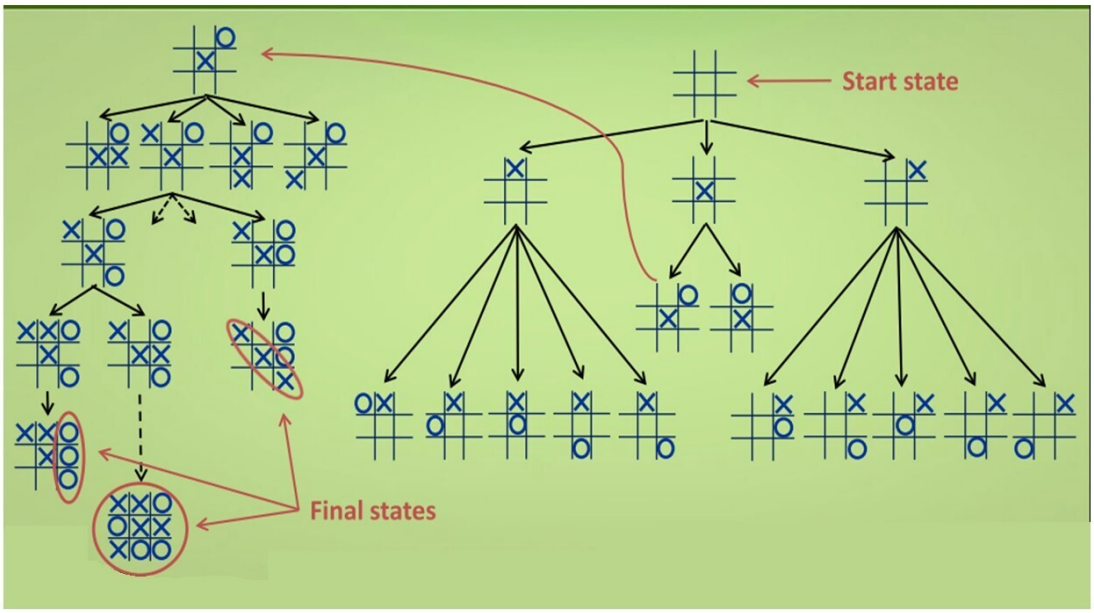
И тот, кто получает 3 крестика или нолика в ряд первым либо горизонтально, либо вертикально, либо по диагонали будет победителем.



Здесь показывается, как два компьютера играют в крестики-нолики друг с другом.

Поскольку компьютеры не делают ошибок, мы можем написать программу, чтобы гарантировать, что они не проиграют в этой игре.

Это своего рода задача, изучаемая в области искусственного интеллекта.



На этой диаграмме видно, что игра начинается с пустой сетки 3x3, мы назовем это начальным состоянием.

Пусть игрок, который ставит крестик, начнет первым.

Есть 9 возможных мест, где 1-й игрок может разместить крестик, но из-за симметрии, график показывает только три варианта, в том числе средний из которых является уникальным, а два других представляет 4 случая.

2-й шаг мог бы быть сделан игроком, который ставит кружок.

Здесь все возможные ходы для 2-го игрока после удаления симметричных случаев.

Если мы будем продолжать, чтобы пройти все возможные ходы, в результате график даст нам пространство состояний.

Угадайте, сколько возможных состояний есть? Если вы достаточно терпеливы, чтобы проследить все возможности, вы увидите, что есть более чем 250000 возможных последовательностей. Вместо того чтобы идти через все случаи, давайте дальше расширять только один конкретный случай.

Здесь все возможные размещения 2-го крестика после этого конкретного выбранного состояния, и все возможные места размещения 2-го кружка.

3-я шаг со стороны игрока с крестиком приведет к первому возможному состоянию выигрыша.

Кроме того, 3-й шаг для кружка приведет ко второму возможному состоянию выигрыша.

Также есть состояние ничьей.

Эти результаты состояния победы вместе с состоянием ничьей образуют множество конечных состояний.

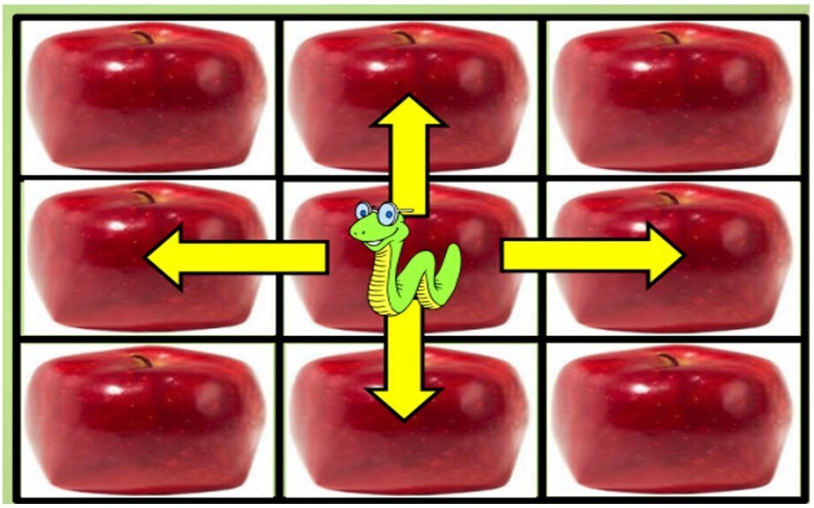
Есть много больше конечных состояний, которые не показаны здесь.

И после прочтения этой книги, вы должны уметь писать программы для такого рода игр или даже более сложных задач.

Пример задачи

Давайте рассмотрим другой пример, чтобы проиллюстрировать, как выбор соответствующего представления может упростить поиск решений более сложной задачи.

Как и в игре крестики-нолики, задача здесь начинается с матрицы 3x3, но в этой задаче, клетки занимают квадратные яблоки.



Давайте назовем это задачей квадратных яблок.

Люди на самом деле могут вырастить квадратные яблоки или даже квадратные арбузы.

Предположим, что в исходном состоянии этой задачи существует червь в средней ячейке.

Вопрос в том, может ли червь съесть все яблоки, выполнив следующие два правила.

Во-первых, после того, как червь закончил целое яблоко в текущей ячейке, он может двигаться только в другую ячейку, которую разделяет общая сторона, так что эти стрелки показывают 4 возможных хода и 2-е правило состоит в том, что вы не можете переместиться в ту ячейку, которую посещали прежде.

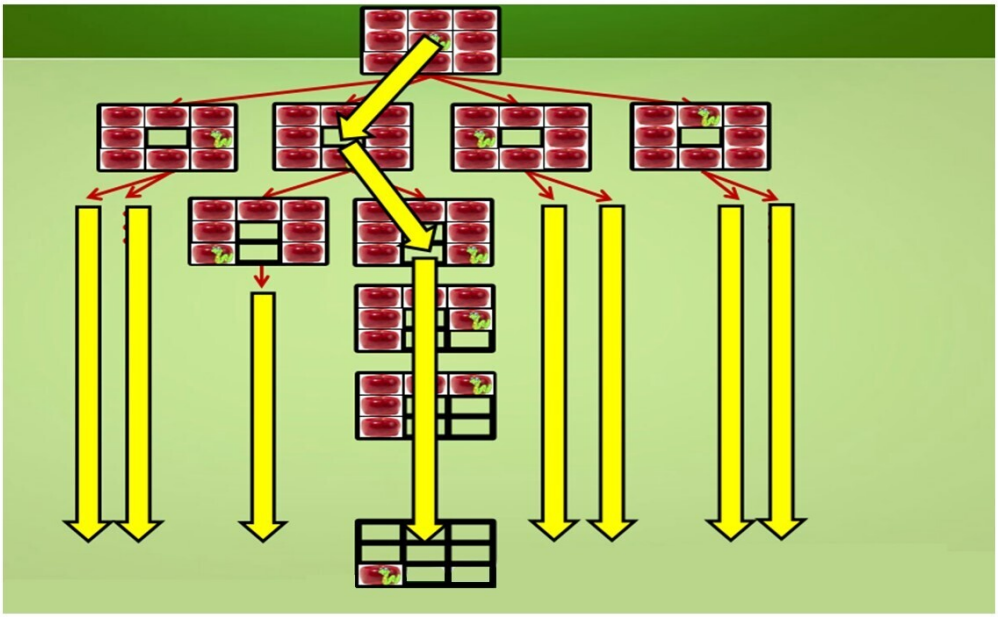
То есть, червь может двигаться только в ячейку, где есть еще несъеденное яблоко внутри.

Рисунок показывает, что червь начинает с середины. После того, как заканчивается среднее яблоко, он может двигаться в одну из четырех клеток на стороне, но не в углу. Таким образом, червь может двигаться вправо, двигаться вниз, двигаться влево и двигаться вверх.

Я хочу, чтобы вы подумали, есть ли решение этой задачи.

То есть, может ли червь съесть все 9 яблок.

Это должно быть совершенно очевидно.



Здесь часть пространства состояний графа и это возможный путь, который может привести к решению.

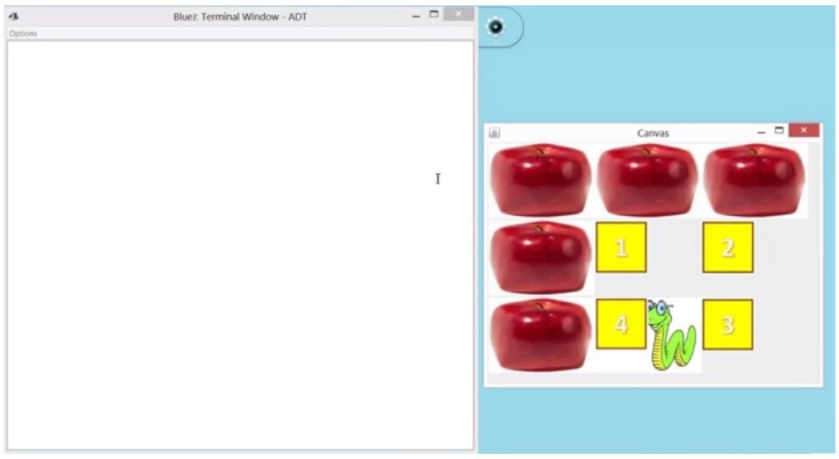
На самом деле, вы обнаружите, что есть 7 других пути для решения, пробуя другие ходы.

Можно написать программу Java, которая была бы создана для решения этой задачи.

И после прочтения этой книги, вы должны быть в состоянии написать собственную программу для решения таких задач, как эта.

Это демо-программа, которая была написана для вас, чтобы проиллюстрировать решение задачи квадратного яблока.

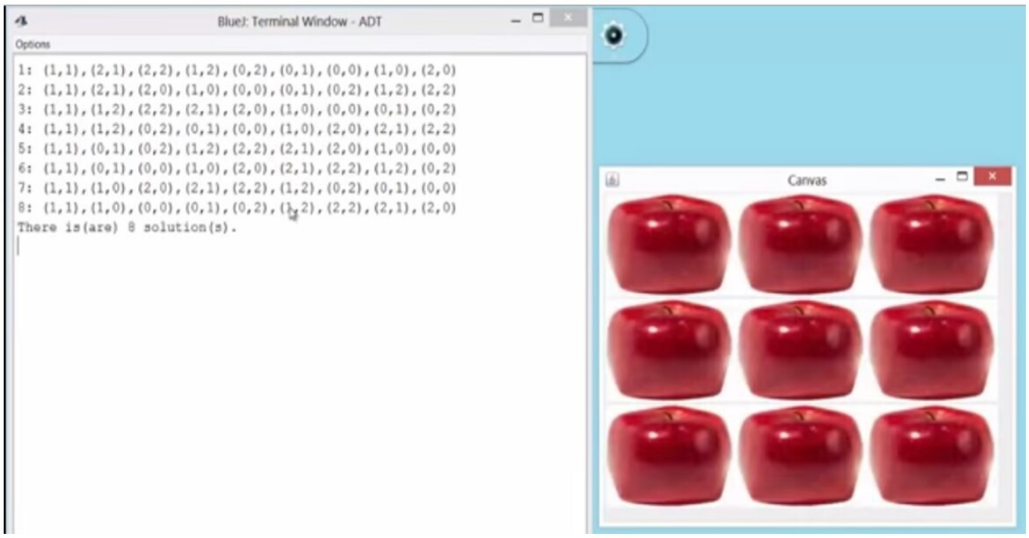
Исходный код к данной книги вы можете скачать по адресу https://github.com/novts/java-base.



Задача будет начинаться с червем в середине. Червь настигает яблоко, перемещаясь в другую ячейку.

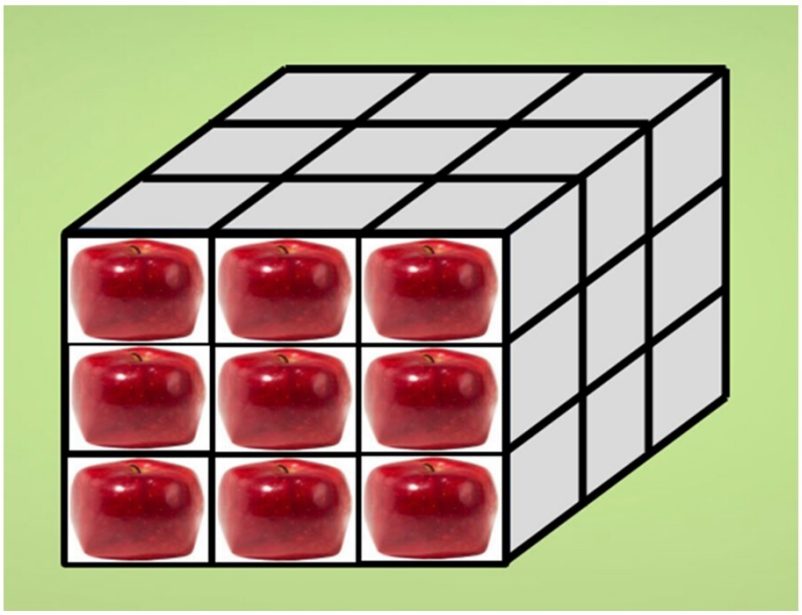
Отображая условие – не посещать клетки, которые были захвачены ранее, цифры показывают последовательность шагов.

Программа также отображает последовательность шагов в другом окне.

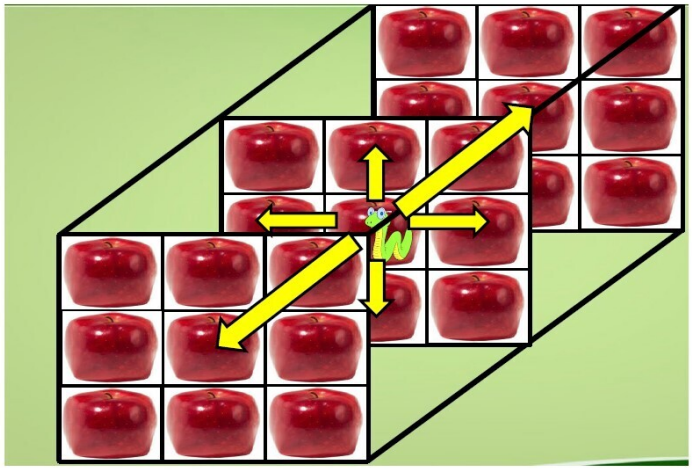


Таким образом, вы можете видеть, что есть в общей сложности 8 решений, как обсуждалось ранее.

Давайте теперь посмотрим на 3D-версию задачи. Задача в основном такая же, как и раньше, за исключением того, что у вас есть 3x3x3 куб как кубик Рубика и с квадратным яблоком в каждой ячейке.



То есть, есть в общей сложности, есть 27 яблок. Чтобы помочь вам визуализировать проблему, диаграмма здесь отображает куб в три слоя.



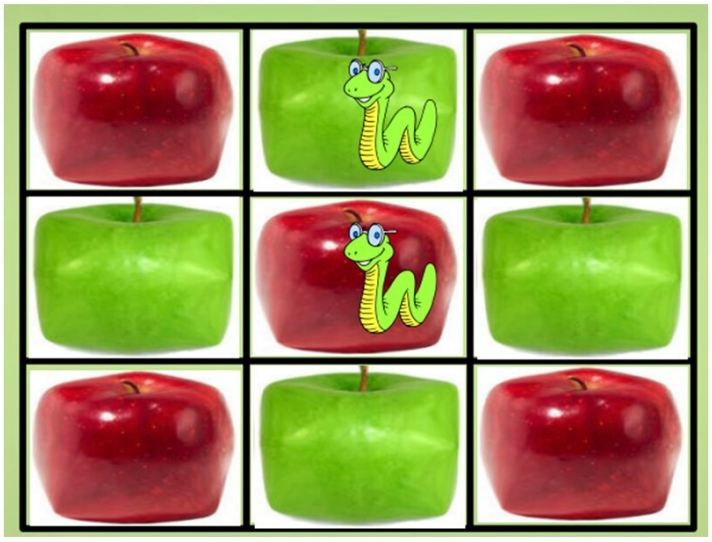
Подобно тому, что у нас было раньше, червь прячется в середине, и мы должны следовать тем же правилам.

То есть, в начале, есть 6 возможных ходов, вместо 4, как в случае 2D, потому что в дополнение к 4 ячейкам на сторонах в среднем слое, червь также может перейти к средней ячейке на переднем плане и средней ячейке на заднем плане.

И теперь вопрос в том, может ли червь по-прежнему съесть все 27 яблок.

Помните, что червь не может вернуться к любым тем ячейкам, которые были захвачены ранее.

Давайте теперь вернемся и посмотрим на 2D-задачу немного по-другому. Некоторые из красных яблок заменены на зеленые яблоки, и они расположены по такой схеме.

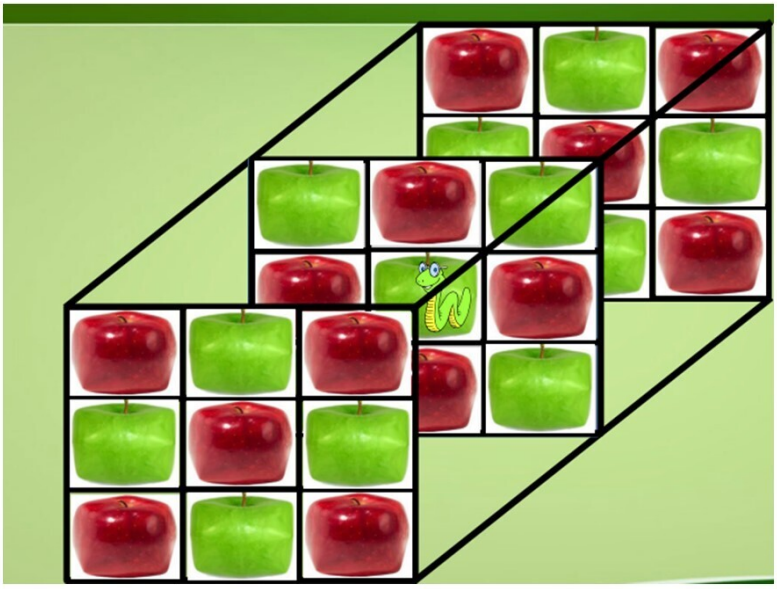


Если червь опять должен начать с середины, следуя тем же правилам, это та же задача, как и раньше, независимо от цвета яблок, и у нас есть те же 8 решений, как и в предыдущем случае 2D.

Теперь рассмотрим, что, если вместо того, чтобы начать с середины, червь начинает с одной из ячеек на стороне, а все другие правила те же самые.

Подсказка в том, что надо посмотреть, как меняется цвет, когда червь переходит из одного яблока в другое.

Давайте теперь вернемся к задаче квадратных яблок 3D и будем чередовать цвета яблок, как мы сделали это в 2D случае.



Как и в предыдущем 3D случае червь прячется в середине.

Так что это все та же 3D задача, как и раньше, хотя цвета некоторых из яблок были изменены.

Теперь используйте то, что вы наблюдали в случае 2D, и попытайтесь придумать быстрое решение этой задачи.

Вопросы

Задача

Может ли червь съесть все 27 яблок, если он начинает от центра куба?

1. Да
2. Нет

Ответ: 2.

Задача

Может ли червь съесть все 27 яблок, если он начинается с одного из углов куба?

1. Да
2. Нет

Ответ: 1.

Из задачи 2D квадратных яблок, мы можем наблюдать, что червь сможет съесть все яблоки, если он начинает с ячейки с красным яблоком. Применяя то же правило в задаче квадратных яблок 3D, мы можем быстро сказать, что червь не сможет съесть все яблоки, если он начинает из центра куба, в котором содержится зеленое яблоко. Аналогичным образом можно быстро сказать, что червь может съесть все яблоки, если он начинает с угла куба, который содержит красное яблоко.

Важность представления задачи

В задаче 2D квадратных яблок, используя красное и зеленое представления яблок, вы увидите, что есть 5 красных яблок и 4 зеленых яблока.

Каждое движение будет чередовать яблоки разных цветов.

Если начинать с зеленого яблока, тогда не будет больше зеленого яблока после употребления четвертого красного яблока.

В 3D случае, существует 14 красных яблок и 13 зеленых яблок.

Используя те же рассуждения, что и в случае 2D, если начинать с зеленого яблока, тогда не будет больше зеленого яблока после окончания 13-го красного яблока.

Так что решения этой задачи не будет, если червь начинает со средней ячейки, которая содержит зеленое яблоко.

И нам удается сразу найти решение, без необходимости искать все возможные пути, представляя задачу.

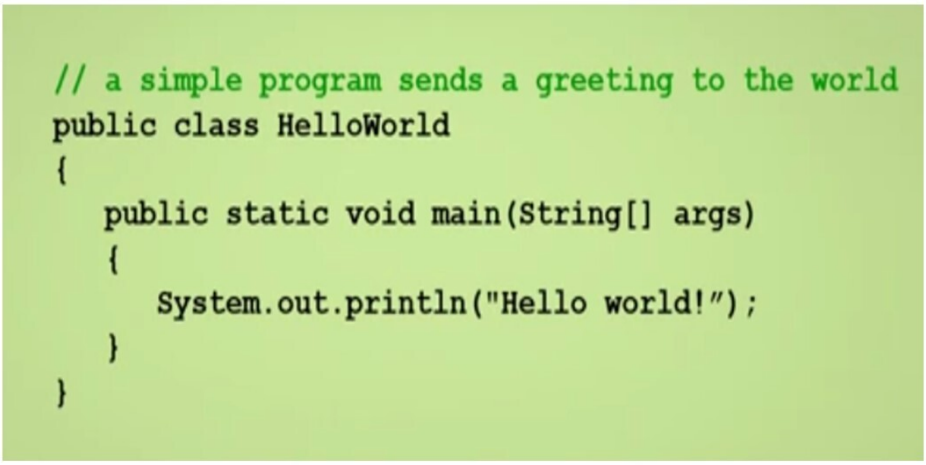
На самом деле, тот же аргумент может быть использован для более широкой задачи, скажем 5x5x5, 7x7x7, или даже больше.

Этот пример показывает важность нахождения правильного представления до решения задачи. Это может сэкономить много времени и усилий.

Первая Java программа

Теперь мы можем начать писать нашу первую Java программу.

По традиции изучения нового языка программирования, ваша первая программа на Java будет печатать приветствие "Hello, World!".



```
// a simple program sends a greeting to the world
public class HelloWorld
{
   public static void main(String[] args)
   {
      System.out.println("Hello world!");
   }
}
```

Первая строка здесь содержит комментарии о том, что, как предполагается, программа должна делать.

И мы обсудим документирование программы позже.

Классы являются основными единицами в программах Java. Все программы Java это классы.

Здесь первая строка кода объявляет имя этой программы, как HelloWorld.

Объявление здесь выглядит немного сложно, но вы не должны беспокоиться о том, что здесь сейчас означают различные слова.

Вы можете смотреть на это как на какой-то формат или шаблон, которому вы должны следовать.

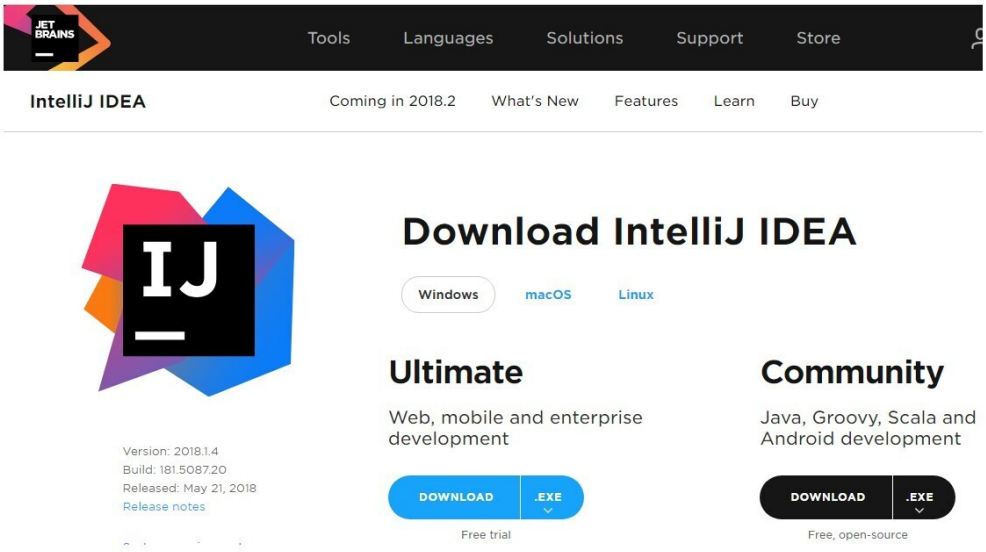
Аналогия, когда вы пишете письмо, у вас есть определенный формат, которому вы должны следовать, такой как адреса отправителя и получателя, а также письмо, как правило, начинается со слова "Уважаемый", а затем следует имя из адреса.

Вы просто должны обратить внимание на слово " main ", которое указывает на главную точку входа в программу.

Строка кода внутри фигурных скобок main является важной частью программы, которая явно указывает компьютеру, чтобы распечатать сообщение-приветствие "привет мир".

Перейдем теперь к среде разработке, чтобы посмотреть на программу более детально.

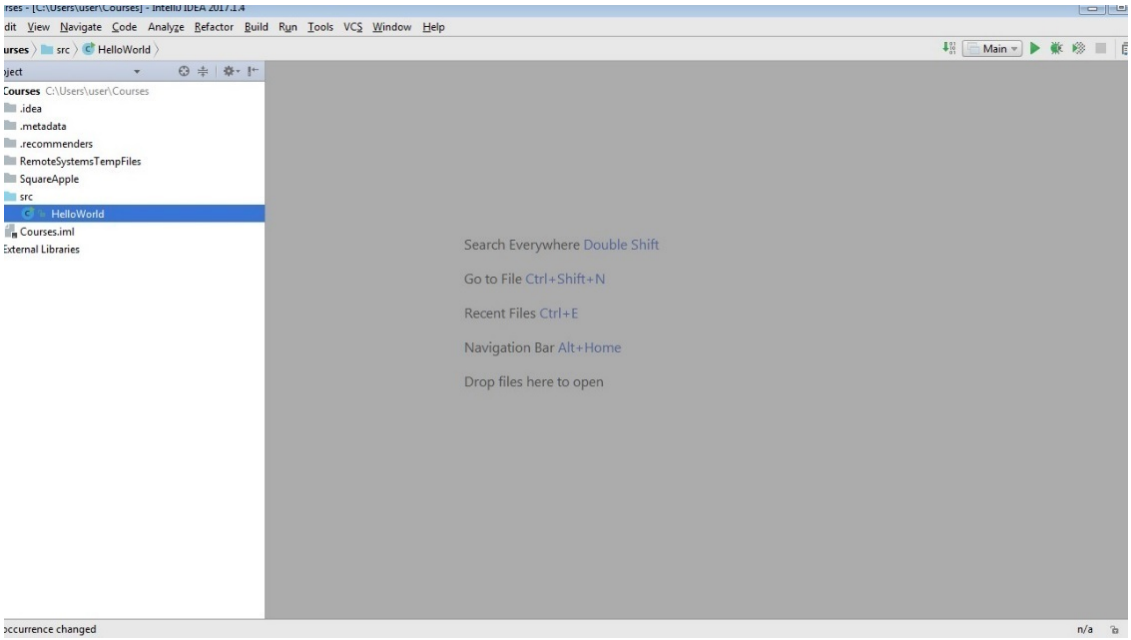
https://www.jetbrains.com/idea/download/#section=windows



Скачайте и установите среду разработки IntelliJ IDEA версии Community.

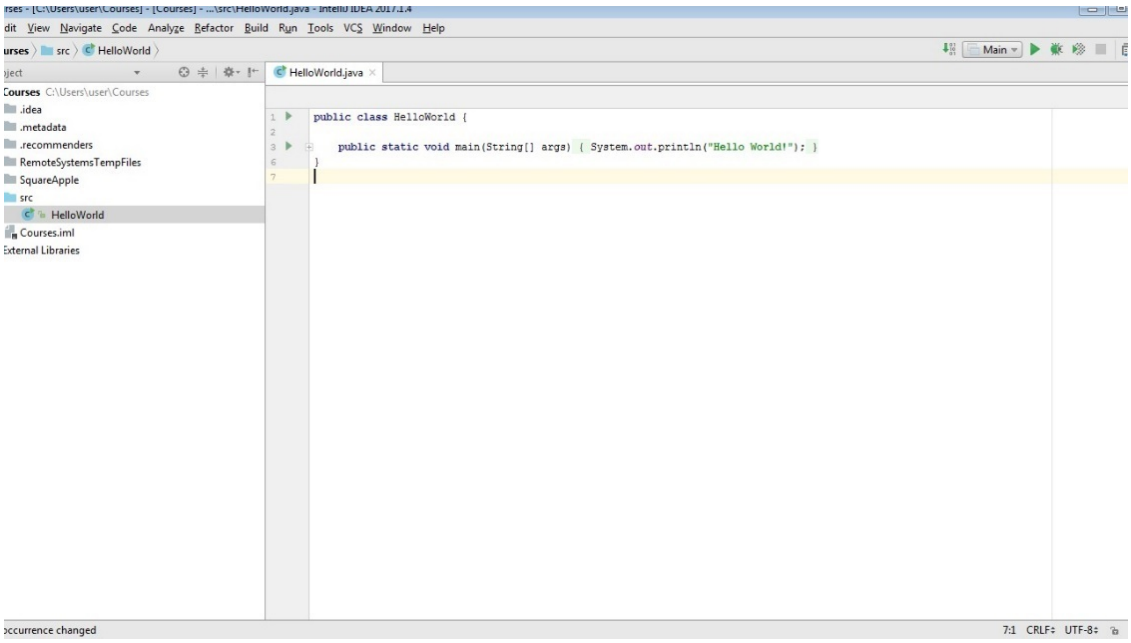
И скачайте проект приложения, который доступен в ресурсах к этой книги (https://github.com/novts/java-base).

Распакуйте этот проект и откройте его в среде разработки.

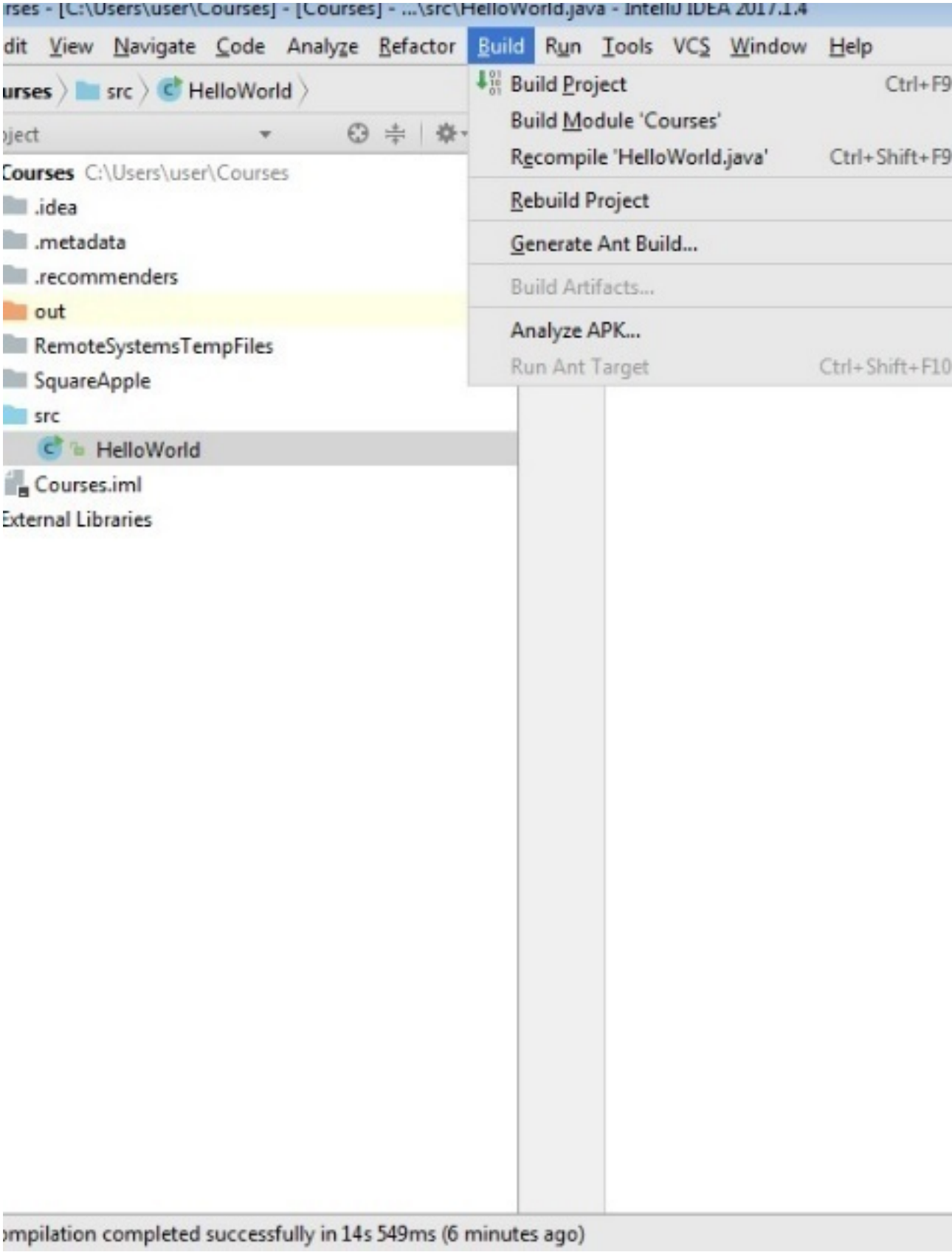


Вот этот файл, соответствующий классу приложения.

Вы можете дважды щелкните по нему, чтобы открыть программу.

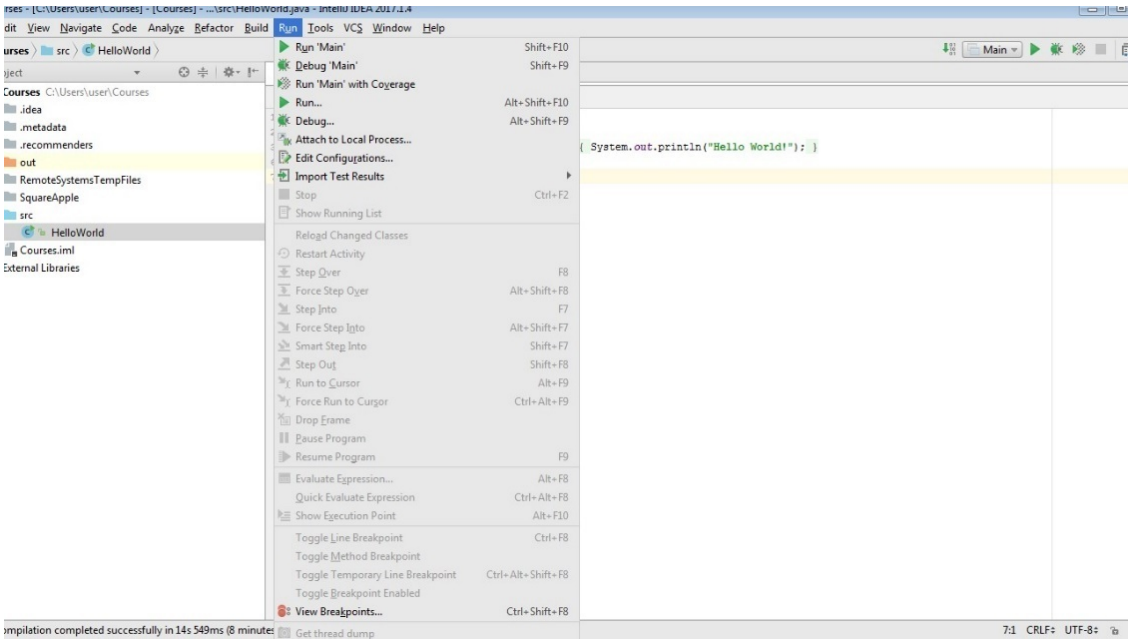


И вы можете скомпилировать программу, нажав на кнопку "Build Project".

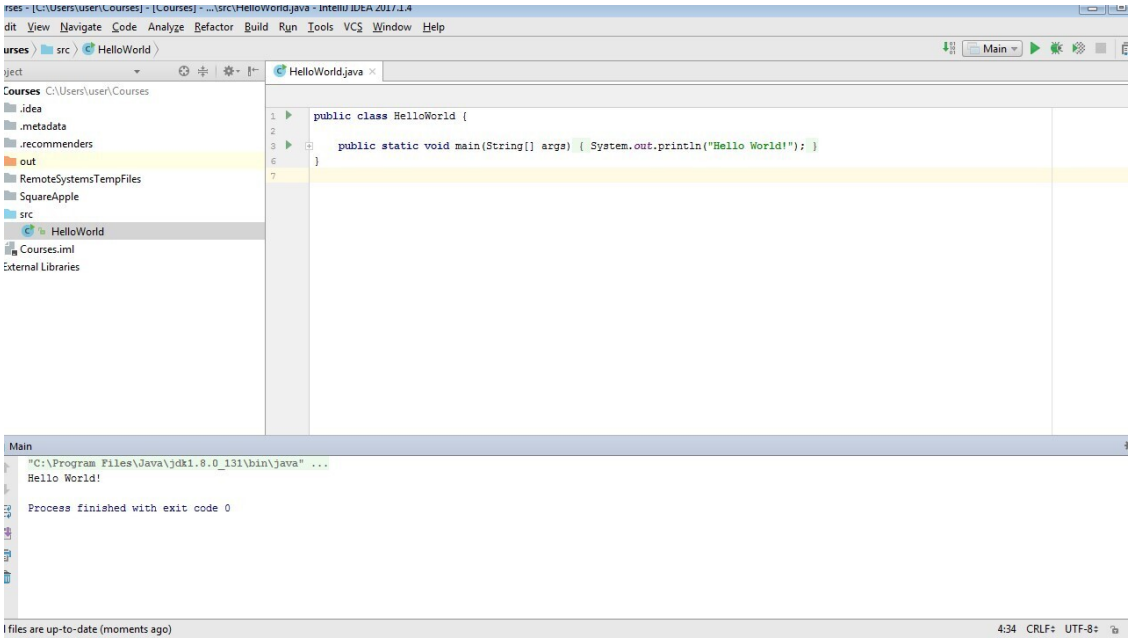


И вы можете увидеть, что программа составлена правильно, так как в сообщении будет указано, что здесь нет ошибок синтаксиса.

Теперь ваша программа готова к работе. Вы можете запустить программу, выбрав Run.



И в другом экране появится сообщение: «Привет, мир!".

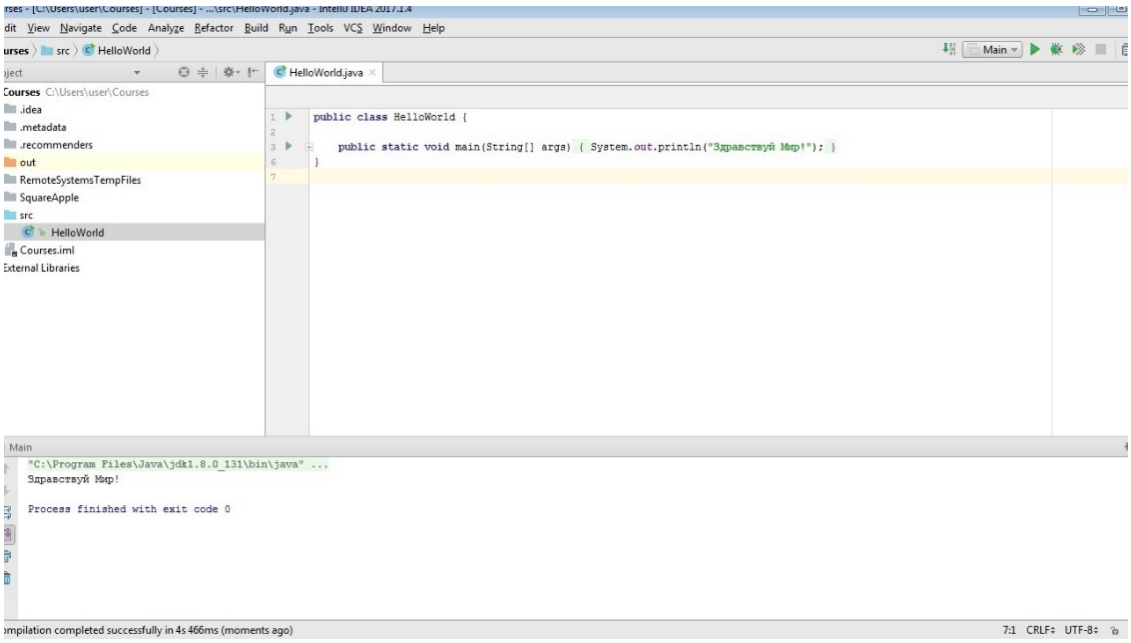


Поздравляем! Вы успешно написали свою первую программу Java.

Язык Java был разработан с самого начала с учетом интернационализации, вместо поддержки только английского языка, и как и в большинстве других языков программирования, Java поддерживает 16-битный стандарт Unicode, который включает в себя много других языков, кроме английского.

Попробуем заменить сообщение в двойных кавычках на аналогичное сообщение на другом языке.

Последовательность символов, заключенная в двойные кавычки, называется строкой символов в Java.



Мы поговорим об этом позже. Теперь мы можем скомпилировать программу и запустить ее снова.

Основы программирования. Введение

Теперь давайте перейдем к рассмотрению собственно основ программирования.

В этом разделе мы собираемся рассмотреть некоторые элементарные понятия программирования, в том числе примитивные типы данных, идентификаторы и переменные, операторы присваивания и арифметические выражения.

Элементарное программирование:

Примитивные типы данных

Идентификаторы и переменные

Выражения присваивания

Арифметические выражения

Java программы, которые мы видели до сих пор, главным образом работали со строками символов, то есть, текстовыми символами, представленными в виде строк символов.

Компьютерные системы в настоящее время широко используются для обработки текстовой информации, например, системы обработки текстов, такие как Microsoft Word, который берет текст в качестве входных данных и выводит наглядное представление текстовых документов,

Это поисковые системы, такие как Google или Yandex, представляющие собой программные системы, которые ищут информацию из веб-документов всемирной паутины, содержащие текстовую информацию.

Однако компьютеры, созданные в самом начале, были предназначены для работы только с числами.

С древних времен, человек признал свою слабость в борьбе с числами и создавал устройства, чтобы помочь себе в выполнении вычислений, например, китайские счеты, которые часто называют первым вычислительным устройством, были изобретены более 4000 лет назад.

И самые мощные компьютеры в настоящее время по-прежнему предназначены в основном для работы с числами, например, в таких приложениях, как моделирование погоды, биоинформатика и финансовое моделирование.

Теперь давайте посмотрим на некоторые примеры, которые работают с числами для решения задач. И будем следовать шагам решения задачи, которые мы обсуждали ранее.

Рассмотрим простую задачу, которая может возникнуть в ходе изучения курса, такого как этот.

Ваше присутствие на курсе может оцениваться с помощью различных видов активности.

Например, итоговая оценка на курсе может зависеть от работы на экзаменах, лабораторных работах и домашних заданиях, но они не имеют одинакового веса.

Как правило, экзамены будут иметь более значимый вес.

Задача, которую нужно решить, это вычислить итоговую оценку как взвешенную комбинацию работы на экзаменах, лабораторных работах и домашних заданиях.

Первый шаг заключается в анализе, какая информация необходима для решения задачи.

Очевидно, что вам понадобятся оценки для экзаменов, лабораторных и домашних заданий.

Подумайте о том, что еще будет необходимо при расчете итоговой оценки?

Подумайте об этом, и я вернусь к этому позже.

Задача может быть сформулирована с помощью определения набора входных данных и набора выходных данных.

В этом случае, баллы за экзамены, лабораторные и домашние задания необходимо будет предоставить в качестве входных данных.

И курс может потребовать несколько экзаменов, лабораторных и домашних работ, но для простоты давайте предположим, что у нас есть агрегированный балл по каждому из этих компонентов.

Результат решения этой задачи совершенно очевиден, мы хотим получить итоговую оценку, которая является взвешенной суммой баллов в качестве выходных данных.

Какая необходима дополнительная информация?

Как уже упоминалось, мы используем различные вес для различных типов оценок, так что этот набор весов должен быть указан.

Один момент, который следует отметить, в то время как оценки могут отличаться от ученика к ученику, веса должны быть одинаковыми для всех учащихся на курсе, и они, как правило, заранее определены и фиксируются в начале курса.

После того как мы получили понимание задачи, мы можем приступить к решению задачи, определив алгоритм.

```
Определить веса для каждого оцениваемого компонента.

Получить оценки экзаменов, лабораторных и домашних заданий.

Вычислить окончательный результат:
    Вычислить взвешенный экзамен.
    Вычислить взвешенную лабораторную оценку
    Вычислить взвешенный домашний результат
    Вычислить сумму приведенных выше взвешенных оценок

Вывести окончательный результат.
```

Как уже говорилось в прошлый раз, алгоритм представляет собой последовательность инструкций, которые могут привести к решению проблемы.

Как было замечено на стадии анализа задачи, мы должны указать набор заранее определенных весов.

Так что примем это в качестве первого шага.

Далее мы должны получить баллы за экзамены, лабораторные и домашние задания – это второй шаг.

После того как мы получили все входные данные, мы можем начать вычисления окончательной оценки – третий шаг.

Расширим этот шаг дальше, так как может быть не очевидно, что понимается под взвешенной суммой баллов.

Поэтому мы можем определить, как могут быть вычислены отдельные взвешенные оценки, а затем взвешенная сумма сложит все взвешенные оценки.

После вычисления окончательной оценки, она выводится пользователю для проверки – четвертый шаг.

Пример

Как только у нас есть алгоритм, следующим шагом является реализация программы.

Как упоминалось ранее, хорошо продуманный алгоритм значительно облегчит процесс кодирования.

Давайте посмотрим на начальный проект программы, которая уже была написана для вас (CourseGrade, https://github.com/novts/java-base).

```
import comp102x.IO;
public class CourseGrade
{
   public static void main(String[] args)
   {
      int examWeight = 70;   // Percentage weight given to examination
      int labWeight = 20;    // Percentage weight given to lab work
      int hwWeight = 10;     // Percentage weight given to homework assignment
      double examScore;      // Examination score obtained by student
      double labScore;       // Lab score obtained by student
      double hwScore;        // Homework score obtained by student
      double finalGrade;     // Final grade obtained by student
      // Ask student to input scores for exam, lab and homework
      IO.output("Enter your exam grade: ");
      examScore = IO.inputDouble( );
      IO.output("Enter your lab grade: ");
      labScore = IO.inputDouble( );
      IO.output("Enter your homework grade: ");
      hwScore = IO.inputDouble( );
      // Computer final grade as the weighted sum of exam, lab and homework scores
      examScore = examScore * (examWeight / 100.0);
      labScore = labScore * (labWeight / 100.0);
      hwScore = hwScore * (hwWeight / 100.0);
      finalGrade = examScore + labScore + hwScore;
      // Output the final grade
      IO.outputln("Your final grade is " + finalGrade);

   }
}
```

Программа начинается, следуя такому же формату, который мы видели в нашей первой программе Java.

В этом случае, название программы (в Java это называется класс) является CourseGrade.

Далее идет выражение, которое определяет главную точку входа для программы.

Это выражение точно такое же, как то, которое мы использовали для HelloWorld.

Это похоже на написание формального письма, которое начинается с фирменного бланка.

Первая часть программы здесь является определением или объявлением переменных.

И я вернусь к этой теме позже.

То, что вы здесь найдете, достаточно хорошо соответствует спецификациям входных и выходных данных и другой информации, которую мы придумали на этапе анализа задачи.

Порядок определения здесь не имеет значения.

Так как мы решили, что веса должны быть предопределены, мы также определяем эти имена в первую очередь.

Имена examScore, labScore и hwScore соответствуют входным данным, в то время как finalgrade представляет желаемый результат.

И обратите внимание, что эта часть программы предусматривает некоторые пояснения, что каждое из этих имен означает.

Я должен отметить, что существует также краткое описание цели программы в самом начале.

Это комментарии, которые следуют определенному формату.

Я вернусь к комментариям программ позже.

Для основной части программы, вы можете увидеть, что каждый основной раздел программы, который описывается комментарием, соответствует шагу алгоритма, как это было предусмотрено в алгоритме.

Вы должны также заметить, что различные участки кода идут с отступом. Это поможет улучшить читаемость программы.

И обратите внимание, что блок операторов в шаге «Ask student to input scores for exam, lab and homework» будет предлагать пользователю ввести оценки экзамена, лабораторной и домашних заданий с помощью объявления IO – IO.output и IO.inputDouble.

И существует еще одно объявление IO.outputln на шаге «Output the final grade».

Я вернусь к этим объявлениям IO, когда мы будем обсуждать простой ввод-вывод IO позже.

Следующим шагом после реализации решения, это придумать план тестирования для этой реализации.

Подумайте о том, что будет считаться хорошим набором входных чисел для оценок экзаменов, лабораторных и домашних работ для тестирования программы.

Вопросы

Задача

Как уже говорилось, важно придумать план тестирования, чтобы проверить, работает ли программа как ожидалось. В примере CourseGrade, вы можете протестировать программу на разных входных значениях для examScore, labScore и hwScore.

Учитывая только examScore, попробуйте придумать план тестирования из 5 осмысленно различных тестов, при условии, что диапазон фактических баллов составляет от 0 до 100.

Ответ:

1. Минимальное значение диапазона 0.
2. Максимальное значение диапазона 100.
3. За минимальной границей -1.
4. За максимальной границей 100.
5. В диапазоне 50.

Такой метод тестирования называется тестированием границ.

Идентификаторы

Как вы видели в предыдущих примерах, такие имена, как HelloWorld и CourseGrade были использованы в качестве имен для классов (или программ), а имена examWeight, examScore, labScroe и т.д. были определены в программе CourseGrade.

Эти имена называются идентификаторами.

Ранее мы говорили о абстракции, и в Java, как и в большинстве языков программирования высокого уровня, можно связать значения или атрибуты определенного типа с идентификатором.

Это очень важное понятие, потому что для человека, намного легче помнить имена, а не ряд чисел, таких как идентификационный номер страховки.

Хотя каждый из нас был связан с различными идентификационными номерами, например, номер паспорта, многие из вас, возможно, не помнят все эти цифры, но я уверен, что вы не забудете свое имя или ваших друзей и членов семьи.

Одним из больших преимуществ в языках программирования высокого уровня является то, что можно использовать значимые символы для представления объектов, в отличие от машинного языка, где все представляется в виде 0 и 1.

Именование идентификаторов должно следовать определенным правилам в Java.

Идентификатор представляет собой последовательность символов, состоящую из:

Букв (a-z, A-Z)

Подчеркивания (_)

Знака доллара ($)

Цифр (0-9, первый символ не может быть цифрой)

Идентификаторы чувствительны к регистру

Идентификатор может быть любой длины, но не может быть одним из зарезервированных слов

В Java, правильный идентификатор, это последовательность символов, состоящая из букв от А до Я в нижнем регистре и верхнем регистре, символа подчеркивания и знака доллара, также могут быть использованы и цифры от нуля до девяти, за исключением того, что цифра не может быть использована в качестве первого символа идентификатора.

Идентификаторы чувствительны к регистру. Например, «Привет» с заглавной буквы отличается от «привет» со всеми строчными буквами.

Давайте рассмотрим несколько примеров.

```
int examWeight = 70;   // Percentage weight given to examination
int labWeight = 20;    // Percentage weight given to lab work
int hwWeight = 10;     // Percentage weight given to homework assignment
double examScore;      // Examination score obtained by student
double labScore;       // Lab score obtained by student
double hwScore;        // Homework score obtained by student
double finalGrade;     // Final grade obtained by student
```

```
exam_score и perfect_score_10

perfect score 10
2017y
int
```

Раньше вы видели, что examWeight и examScore были использованы в предыдущей программе, и они являются допустимыми идентификаторами. Вы также видели слова int и double.

Это типы этого идентификатора.

Давайте проигнорируем их пока.

Вы также можете добавить подчеркивания между словами, например, exam_score и perfect_score_10, но пробел не может быть включен в качестве части идентификатора, так что " perfect score 10" с пространством между словами не является допустимым идентификатором.

Если вы хотите отделить два слова в качестве идентификатора, следует использовать подчеркивания.

И не путайте подчеркивания "_" с дефисом "-", дефис "-" не может быть использован в качестве идентификатора, так как это можно было бы интерпретировать как минус.

2017y является недействительным идентификатором, потому что он начинается с цифры.

И один последний пример недопустимого идентификатора является int, так как int является зарезервированным словом в Java и используется для объявления определенного идентификатора в виде целого числа.

Таким образом, еще одно правило в том, что зарезервированные слова не могут быть использованы в качестве идентификатора.

Давайте посмотрим на то, что является зарезервированным словом. Не трудно найти зарезервированные слова в нашей повседневной жизни.

Для названий доменов в Интернете, .gov зарезервирован для государственных организаций, .edu зарезервирован для учебных заведений. Точно так же, и Java использует некоторые зарезервированные слова.

В таблице здесь показаны некоторые из зарезервированных слов Java.

Elementary programming	Branching and loops	Object and classes	Miscellaneous
boolean	break	class	import
byte	case	instanceof	package
char	continue	new	return
double	default	private	
final	do	protected	
float	else	public	
int	for	static	
long	if	this	
short	switch		
void	while		

Этот список неполон, но охватывает большую часть зарезервированных слов, которые будут использоваться в этой книге.

Различные программисты могут следовать разным стилям программирования, но есть определенные часто используемые стили, которые могли бы улучшить читаемость вашей программы. То есть, сделать проще для других понимание вашей программы.

Вот некоторые правила для названий идентификаторов Java.

Важно, чтобы использовались значимые имена. Использование бессмысленных названий, таких как x, y, r следует заменить более значимыми именами, такими как radius, area, score если это возможно. Для длинных имен, полезно использовать смешанный регистр, то есть, смесь нижнего регистра и заглавных букв, разделяя слова, используя заглавные буквы.

Эти заглавные буквы обычно называются CamelCase, как горбы верблюдов. И есть два типа CamelCase, нижний верблюд начинается со строчной буквы, например, examScore и areaOfCircle. Нижний CamelCase обычно используется в Java для переменных и методов.

Существует также верхний CamelCase, который начинается с буквы верхнего регистра, и который обычно используется для именования классов.

В предыдущих примерах, HelloWorld и CourseGrade являются именами классов, которые начинаются с заглавных букв.

Вот еще один пример объявления для класса BankAccount.

Мы обсудим подробнее позже, что означают методы и классы.

Вопросы

Задача
Просьба указать допустимый идентификатор Java из списка ниже.
1. Last_Name
2. 1dentifier
3. You&Me
4. COMP-102
Ответ: 1

Переменные

Один вид идентификатора, который очень часто используется в программе является переменной.

Во многих приложениях необходимо зарезервировать память компьютера для хранения значений, которые будут использоваться в программе.

Переменная представляет собой кусок памяти компьютера, который может хранить значение.

Как правило, такая память выделяется для переменной по запросу программой.

Значение переменной может быть изменено, так что программа должна быть гибкой для обработки различных входных данных.

Например, в программе расчета оценки, переменные examScore, labScore и hwScore могут принимать различные значения в зависимости от входных данных.

Идентификатор finalgrade также является переменной.

Он хранит результат взвешенной суммы, которая зависит от значений входных баллов.

Имя переменной это метка участка памяти.

С точки зрения компьютера, адреса кучи памяти будет достаточно, но для человека, нам нужен хороший способ, чтобы различать и ссылаться на участок памяти.

Аналогией является использование почтовых ящиков, где каждый почтовый ящик помечен его владельцем (или имеет идентификатор) и различные виды почты (или значений) могут быть оставлены в этом почтовом ящике.

Переменная создается через процесс объявления.

Цель объявления, это сделать понятным для компьютера, что имя конкретного идентификатора означает в программе.

Объявление переменной в программе состоит из трех основных частей:

Оно начинается типом данных, далее следует идентификатор и заканчивается точкой с запятой.

```
int examWeight;
```

```
int examWeight = 70;
```

Так что с помощью этого синтаксиса объявляется переменная.

Например, в программе CourseGrade, было сделано объявление переменной int examWeight; где int является целочисленным типом данных, examWeight является идентификатором и объявление заканчивается точкой с запятой.

Если вы обратитесь к программе расчета оценки, объявление int examWeight не просто заканчивается сразу после examWeight, но за ним следует "= 70".

В объявлении int examweight = 70, examweight объявляется и инициализируется в одном определении.

Объявление и инициализация также могут быть сделаны отдельно.

В любом случае это приведет к тому же эффекту.

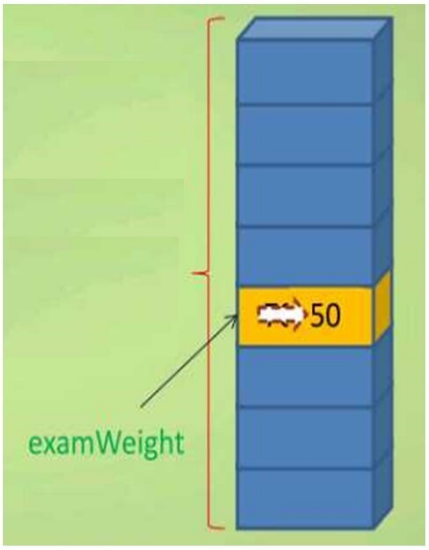
Знак равенства, который вы видите здесь это оператор присваивания, в данном случае, examweight = 70 является утверждением присваивания.

Также вы можете увидеть, что, если программа ссылается на examWeight до инициализации, это приведет к ошибке компиляции.

Хотя Java инициализирует определенные переменные, я поговорю об этом подробнее, когда будем обсуждать классы и объекты, тем не менее это всегда хорошая практика программирования – инициализировать переменную перед ее использованием.

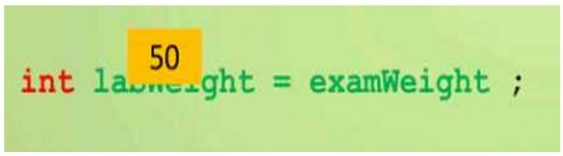
После объявления переменной и инициализации, ее значение может быть изменено с помощью оператора присваивания.

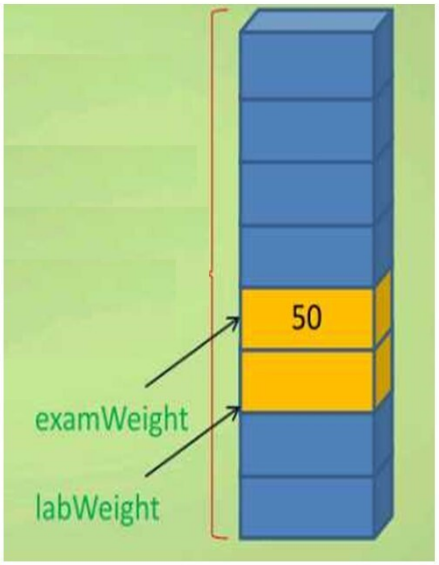
```
examWeight = 50;
```



Старое значение, в данном случае, 70 будет удалено и заменено новым значением 50.

Также можно получить значение переменной, ссылаясь на ее имя.





В этом примере со ссылкой на examweight, в правой стороне от знака равенства, значение examweight извлекается, и значение затем присваивается labweight и результат будет храниться в ячейке памяти, выделенной для labWeight.

Иногда мы хотим сохранить значение переменной неизменным на протяжении всей программы, например, идентификаторы для математических констант, таких как пи (PI присваивается значение 3,14159), это постоянная, которую используют для вычисления периметра и площади круга.

В программе CourseGrade, переменные examWeight, labWeight, hwWeight можно рассматривать как константы.

Если вы хотите сделать фиксированными веса для каждого студенты, вы можете предотвратить случайные изменения в значении переменной с помощью "финализации" его значения. Это делается с помощью ключевого слова "final" перед объявлением переменной. Такие переменные часто называют константами.

И вы можете назначить значение для финальной переменной только один раз.

В примере, который мы видели, объявление int examWeight = 70 объявляет и инициализирует examWeight.

Если мы добавим ключевое слово final перед декларацией, эффект проявится в виде блокировки памяти, и значение не может быть изменено снова.

Попытка повторно присвоить значение финальной переменной вызовет ошибку компиляции.

```
final int examWeight = 70;
```

Например, если examWeight была объявлена как финальная, компилятор будет жаловаться при попытке изменить значение examWeight на 50.

Типы данных

Тип данных является очень важным понятием в языке программирования высокого уровня.

Java является строго типизированным языком программирования.

Это означает, что должны быть определены все типы имен, упомянутых в программе Java, прежде чем они могут быть использованы.

Мы только что видели, что способом объявить переменную является указание типа.

Мы использовали тип int для целого числа в предыдущих примерах.

Другие типы данных также поддерживаются Java.

Каждый тип данных имеет свои свойства и требования к пространству памяти.

Это позволяет компьютеру сделать выделение памяти, когда программа выполняется, так как различные типы данных имеют разные требования к размеру памяти.

Свойства типа данных включают в себя набор значений, которые он может взять на себя и набор операторов, которые могут применяться к этим значениям.

Например, значение целого числа может отрицательным, положительным или нулевым, и операции, которые могут быть выполнены для целого числа включают + сложение, – вычитание, * умножение и / деление.

Набор значений известен как домен для этого типа.

Java поддерживает восемь простых типов данных в рамках 4-х основных категорий, а именно целые числа, числа с плавающей точкой, символы и логические значения.

byte, short, int и long являются различными целочисленными типами, которые занимают разное количество памяти.

float и double представляют числа с плавающей точкой, то есть, числа с дробной частью.

Тип char для символов, представленных 16-битным стандартом Unicode.

boolean это тип данных, которые могут взять на себя только два возможных значения, истина и ложь.

byte, short, int и long являются различными целочисленными типами, которые занимают разное количество памяти.

float и double представляют числа с плавающей точкой, то есть, числа с дробной частью.

Значения, сохраненные в float и double типах, являются только приблизительными.

Существует также тип char для символов, представленных 16-битным стандартом Unicode.

boolean это тип данных, которые могут взять на себя только два возможных значения, истина и ложь.

Сначала мы сконцентрируемся на целых числах и числах с плавающей точкой и вернемся к char и логическим типам позже.

Среди 4 целых типов:

byte это 8-разрядные целые числа, со значениями в диапазоне от -128 до 127,

short составляет 16 бит,

int 32 бита и

long 64 бита, соответствующие диапазоны значений приведены в этой таблице.

Name	Range
byte	-2^7 to 2^7-1 (-128 to 127)
short	-2^{15} to $2^{15}-1$ (-32768 to 32767)
int	-2^{31} to $2^{31}-1$
long	-2^{63} to $2^{63}-1$

Основное преимущество целочисленного представления в том, что оно представляет собой точное значение без приближения, но оно не может представлять значения с плавающей запятой и его область значений ограничена.

Хотя long может составлять до 2 в 63-й степени, его диапазон по-прежнему намного меньше, чем float или double.

Name	Range
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38
double	Negative range: -1.7976931348523157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348523157E+308

В таблице здесь показан диапазон значений, которые могут быть представлены float, который использует 32 бита и double, использующий 64 бита.

Как вы можете видеть, это астрономические цифры.

Выражения

При написании программ, выражения часто используются как строительные блоки для определения действий, которые программа предполагает выполнить.

Существует два типа выражений – арифметические выражения и логические выражения.

Примером арифметического выражения может служить вычисление окончательной оценки как взвешенной суммы оценок, как мы видели в программе CourseGrade.

Примером логического выражения может служить действие проверки студента на получение оценки.

Сначала остановимся на арифметических выражениях, а затем рассмотрим логические выражения, когда будем говорить о разветвленных выражениях.

Арифметическое выражение это последовательность операндов, включающих переменные и константы или литералы, соединенные арифметическими операторами, такими как +, -, *, /

Арифметическое выражение это последовательность операндов, включающих переменные и константы или литералы, соединенные арифметическими операторами, такими как +, -, * & / .

Мы уже обсуждали переменные и константы.

И первым примером здесь будет использование литералов.

```
2+3

c*9/5+32
```

Арифметическое выражение 2+3 соединяет два литерала 2 и 3, используя оператор сложения.

Второй пример, это арифметическое выражение для преобразования температуры из Цельсия в Форенгейт:

Цельсий*9/5+32

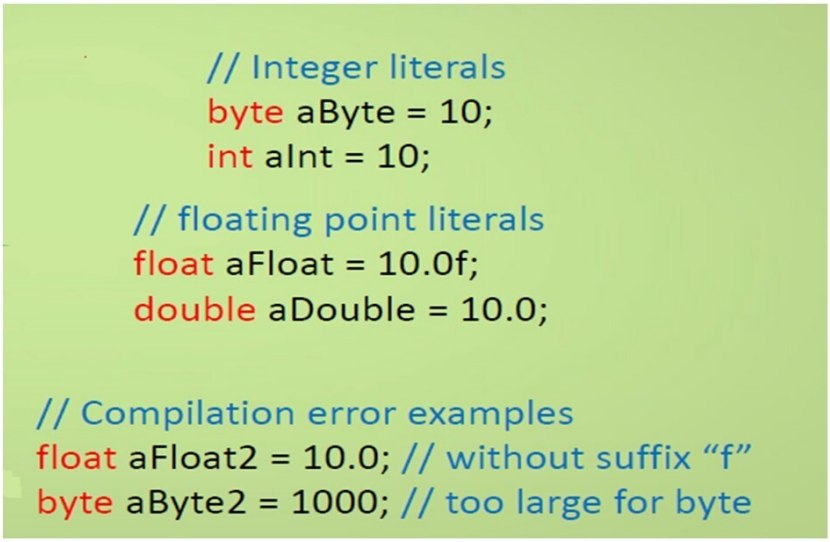
Цельсий, – это переменная, 9, 5 и 32 – литералы, и литералы соединены тремя операторами *, / и +.

Позже, вы обнаружите, что, если это будет использовано как Java выражение, возникнет ошибка вычисления.

Вы узнаете, что я имею в виду, когда мы будем говорить о делении целых чисел.

Так что вы видите, что литерал, – это константное значение, которое появляется непосредственно в Java программе.

И существует два типа численных литералов в Java – это целочисленные литералы и литералы с плавающей запятой.



Здесь показаны некоторые примеры литералов.

Заметьте, что для переменной aFloat значение установлено 10.0f с суффиксом f после 10.0 – это потому, что литерал с плавающей запятой 10.0 представлен как double – с размером 64 bit, в то время как float имеет размер 32 bit.

Присвоение double в float ведет потенциально к потере информации, и Java компилятор будет жаловаться, если суффикс f отсутствует.

Я поговорю о преобразовании типов позже.

Также присвоение 1000 в byte, который может содержать только целое значение до 128, приведет к ошибке компиляции, потому что литерал слишком большой, чтобы поместиться в переменную.

Operator	Examples	Result
(+) Addition	2 + 3	5
(-) Subtraction	2 – 3	-1
(*) Multiplication	2 * 3	6
(/) Division	2 / 3	0 (Integer division)
(%) Modulus Would work on floating point numbers too!	2 % 3	2 (The remainder of 2 / 3)

Здесь показаны общераспространенные арифметические операторы.

Операторы +, – и * довольно простые, однако оператор деления целых чисел слегка хитрый.

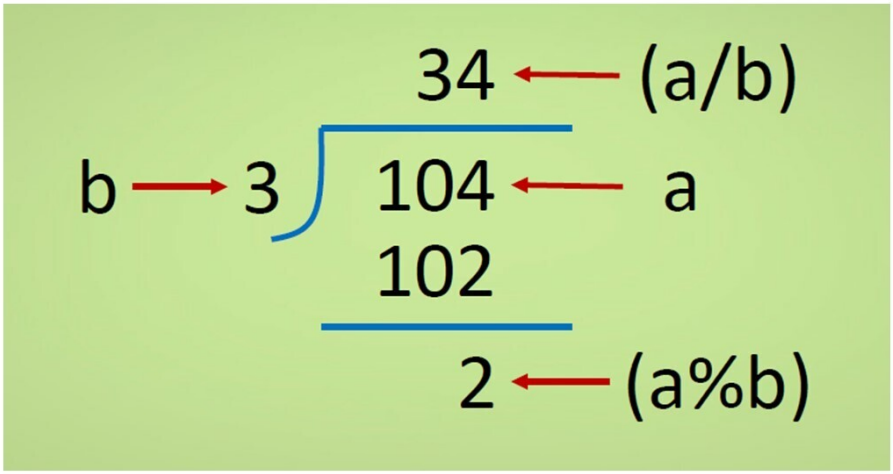
Например, если 2 разделить на 3, результат будет 0.

Оператор модуля, который представлен знаком процентов, дает остаток от деления первого операнда на второй операнд.

Например, 2 модуль 3 возвращает 2 как результат.

Оператор модуля в Java работает также для чисел с плавающей запятой.

Рассмотрим другой пример использования оператора модуля.



Когда переменная со значением 104 делится на другую переменную b со значением 3, результат будет 34 с остатком 2.

Числовые значения хранятся как целые числа или числа с плавающей запятой.

Для целочисленного деления результат, – это целая часть деления, десятичная часть результата игнорируется.

Другими словами, целочисленное деление всегда возвращает целое число путем усечения без округления.

При этом может быть потеряна информация, когда десятичная часть отсекается.

Например, при делении 2 на 3 результат должен быть 0.66, но мы получаем 0 из-за отсечения.

Или 3 делим на 2 и получаем 1 вместо 1.5.

Если вы делите double на double, результат будет double, как и ожидалось.

Теперь вопрос, как Java оперирует со смешанными делениями, включающими целые числа и числа с плавающей запятой?

В общем, деление double дает double.

Когда целое делится на double или double делится на целое, результатом будет double.

Это позволяет программе максимально сохранить информацию.

Например, если 2 делится на 3.0, результатом будет 0.6666 вместо 0.

Также, деление 3.0 на 2 даст результат 1.5.

Деление двух double 10.0 и 2.0 даст результат double 5.0.

Когда выражение вычисляется, мы должны определить порядок для выполнения операций, если в выражении больше одного оператора.

Когда мы изучали алгебру, мы узнали, что операции * умножения и / деления выполняются перед операциями + сложения и – вычитания, и такое же правило действует и в Java.

Например, в выражении m*x + b, m умножается на x перед прибавлением b к результату умножения.

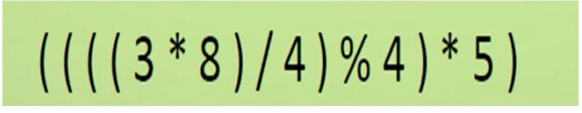
Приоритет операторов задает порядок, в котором различные операторы выражения вычисляются.

( )

* / %

\+ -

Здесь показан стандартный порядок, которому следует Java:
( )
* / %
– +
Выражение, заключенное в круглые скобки, вычисляется первым.



Для вложенных скобок внутреннее выражение вычисляется первым.

Операторы * умножения, / деления и % остатка вычисляются вторыми, и, если их несколько, вычисление идет слева направо.

Операторы сложения и вычитания вычисляются после остальных операторов, и, если их несколько, вычисление идет слева направо.

Другая важная вещь в вычислении выражений, это концепция ассоциативности.

Ассоциативность используется для определения порядка, в котором операторы с одинаковым приоритетом вычисляются в выражении.

Правило ассоциации в этом примере, – это вычисление слева направо, и называется левой ассоциативностью.

При этом круглые скобки могут быть вставлены для усиления порядка вычисления.

Вы можете подумать, что все операции должны следовать левой ассоциации.

Однако это не всегда случается в Java, и мы уже видели оператор, который следует правой ассоциации, – это оператор присваивания =.

Вопросы

Задача
Что является результатом каждого из следующих выражений?
Expression X: 3 % 4 – 10 * 5
Expression Y: 5 + 11 / 2 * 2.0
Expression Z: 100 / 0
Варианты:
1.
X: -47
Y: 10.0
Z: 0
2.
X: 1
Y: 10.0
Z: 0
3.
X: -47
Y: 15.0
Z: ERROR
4.
X: -47
Y: 10.0
Z: ERROR
Ответ: 3.

Присваивание

Мы видели много выражений со знаком равенства в предыдущих примерах.

Все они использовали оператор присваивания.

Синтаксис оператора присваивания представляет собой размещение переменной на левой стороне знака равенства, выражения на его правой стороне и точки с запятой в конце.


```
int one = 1;   // variable one is initialized to 1
one = one + 1; // Now, the variable one has a value of 2
```

Смысл или семантика оператора присваивания – это присвоить значение, вычисленное выражением на правой стороне, переменной на левой стороне, и исходное значение, хранимое в переменной, будет заменено.

Это обозначение может быть немного запутанным, поскольку в большинстве утверждений присваивания, левая сторона может быть не равна правой стороне в математическом смысле.

Например, вы можете иметь что-то вроде, a = a + 1;

Это не корректно в качестве математического выражения, но это верное утверждение присваивания.

Переменная здесь имеет начальное значение 1, и ее значение будет изменено на 2 после присвоения.

Для определенного типа существует набор действительных операторов, которые могут быть применены к этому типу.

Если кто-то хочет применить некоторый оператор, который действителен только для другого типа, необходимо преобразование типов, чтобы преобразовать тип данных из одного типа в другой.

Преобразование типа может быть сделано двумя способами: это явные и неявные преобразования.

Неявное преобразование изменяет значение одного типа в другой без специальной инструкции от программиста.

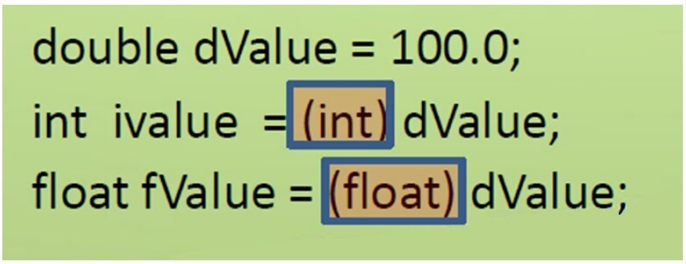
Например, целый тип int разрешено присваивать типу float, хотя при этом может быть потеряна точность.

Основное правило заключается в том, что неявное преобразование разрешено, если диапазон значений первого типа является подмножеством второго.

Его часто называют расширяющим преобразованием.

Обратное это сужающее преобразование, которое, как правило, не допускается без явного преобразования.

Явное преобразование выполняется приведением типов.



Приведение типов делается путем размещения имени целевого типа в скобках перед типом данных, которые будут преобразованы.

Например, если dValue является double переменной, компилятор не допустит присвоения int или float без преобразования явного типа, указав int или float в скобках.

Далее, мы вернемся к примеру CourseGrade и используем то, что мы только что узнали о переменных, объявлениях, типах данных и арифметических выражениях, чтобы получить более глубокое понимание того, как эта программа выполняется.

Выделение памяти

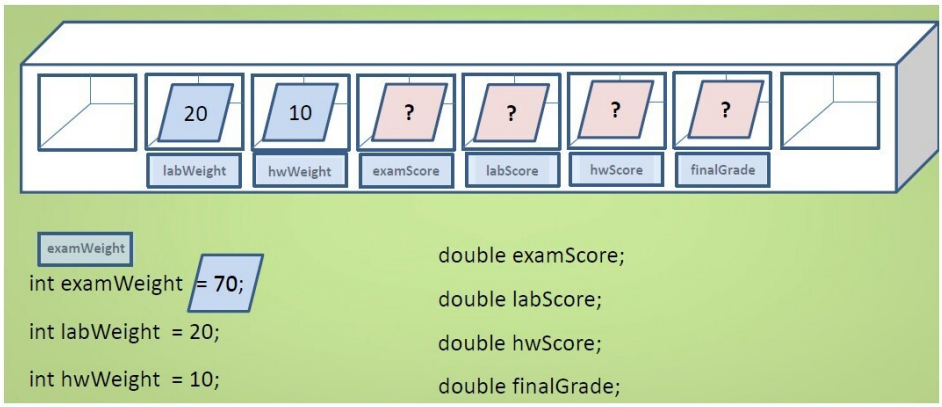
Давайте теперь используем пример CourseGrade для того, чтобы проиллюстрировать эффект объявления и выполнения программы.

```
/**
 * CourseGrade determines the final grade which is computed as
 * the weighted sum of the grades obtained in exam, lab and homework
 */
public class CourseGrade
{
    public static void main(String[] args)
    {
        int examWeight = 70;    // Percentage weight given to examination
        int labWeight = 20;     // Percentage weight given to lab work
        int hwWeight = 10;      // Percentage weight given to homework assignment
        double examScore;       // Examination score obtained by student
        double labScore;        // Lab score obtained by student
        double hwScore;         // Homework score obtained by student
        double finalGrade;      // Final grade obtained by student
```

Как уже говорилось, объявление идентификатора, в дополнение к определению его типа данных, также позволяет компьютеру делать выделение памяти, когда программа выполняется.

В этой программе, объявляются 7 идентификаторов, а именно examweight, labWeight, hwWeight, examScore, labScore, hwScore и finalGrade.

Я буду использовать схему, которая показывает набор из ячеек, чтобы проиллюстрировать выделение памяти, когда переменная объявлена.



Это только для иллюстрации и объем памяти, конечно отличается от ячейки.

Если объявление int examWeight сделано, пространство памяти выделяется в соответствии с размером типа данных, в данном случае int.

Присвоение значения 70 для examWeight приведет к инициализации значения, хранимого в памяти, до 70. В компьютере, значение 70 на самом деле представлено как строка битов 0 и 1.

Объявление и инициализация для labWeight и hwWeight проходит через аналогичный процесс.

Объявление для examScore выделяет достаточно памяти для хранения числа с плавающей точкой типа double. Большинство реализаций для double требует 8 байт, поэтому размер будет отличаться от int, который требует 4 байта.

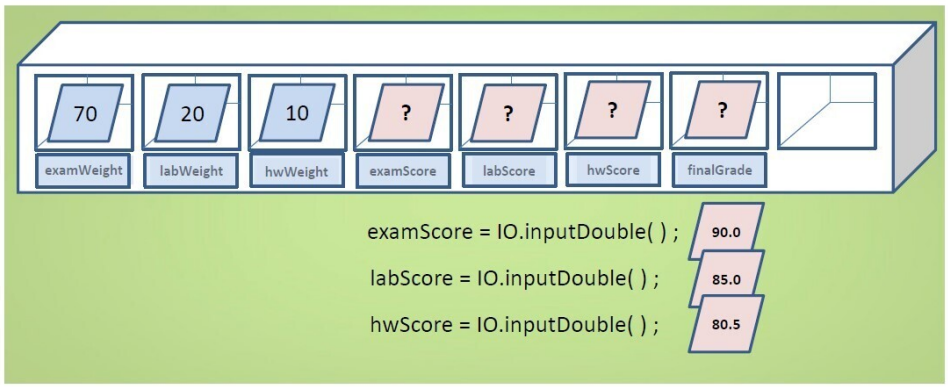
Подобное выделение памяти будет сделано для labScore, hwScore и finalGrade.

Я буду использовать здесь другой цвет, чтобы проиллюстрировать, что int и double имеют разные требования к памяти.

Поскольку значения не были присвоены для этих переменных, их значения не известны.

Фактически, сначала им должны быть присвоены значения до того, как может быть сделана на них ссылка.

После того, как объявления выполнены, программа предложит пользователю ввести значения для examScore, labScore и hwScore.



Предположим, что пользователь ввел 90,0 для examscore.

Обратите внимание, что даже если пользователь ввел 90, без десятичной точки, значение будет преобразовано в число с плавающей точкой.

Опять же, я использую здесь другой цвет, чтобы отличить double тип от int типа, который находится в синих ячейках.

Аналогично, значение 85,0 вводится для labScore, и 80,5 вводится для hwScore.

Чтобы продолжить выполнение кода, будет выполнен другой оператор присваивания, который изменяет значение examScore.

```
// Ask student to input scores for exam, lab and nomework
IO.output("Enter your exam grade: ");
examScore = IO.inputDouble( );
IO.output("Enter your lab grade: ");
labScore = IO.inputDouble( );
IO.output("Enter your homework grade: ");
hwScore = IO.inputDouble( );

// Computer final grade as the weighted sum of exam, lab and homework scores
examScore = examScore * (examWeight / 100.0);
labScore = labScore * (labWeight / 100.0);
hwScore = hwScore * (hwWeight / 100.0);
finalGrade = examScore + labScore + hwScore;

// Output the final grade
IO.outputln("Your final grade is " + finalGrade);
```

Вычисление выражения справа от оператора присваивания сначала вычисляет выражение внутри круглых скобок.

Значение examWeight извлекается из памяти, а затем делится на 100,0.

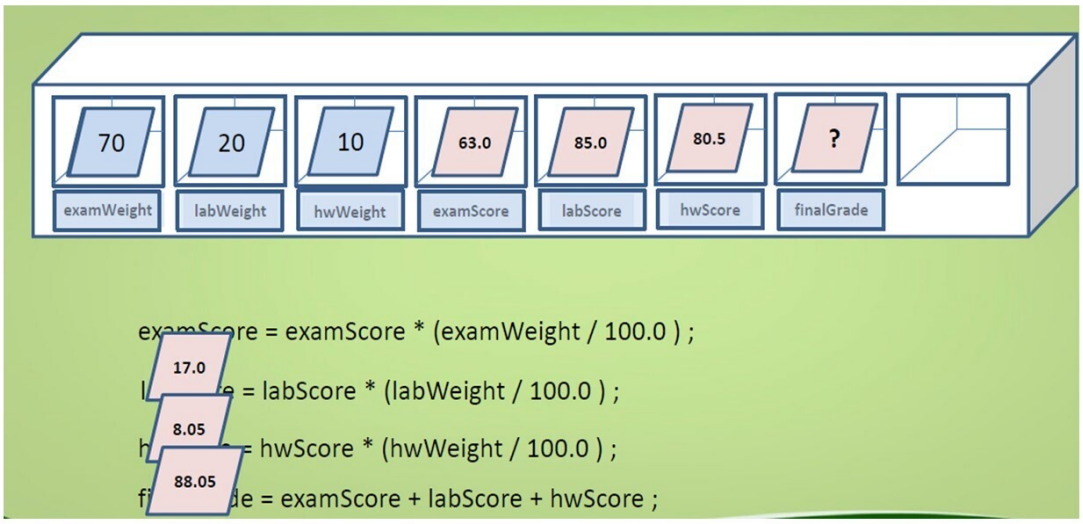
Следует отметить, что examWeight представляет собой целое число, и если оно делилось бы на другое целое число 100, то результатом был бы ноль.

Но так как мы используем 100.0, которое является числом с плавающей точкой, результатом деления будет число с плавающей точкой 0.7.

Значение examScore затем будет извлечено из памяти и умножится на 0,7.

Полученное значение 63,0 затем будет присвоено переменной на левой стороне оператора присваивания.

Результат выражения заменит исходное значение в памяти для examscore новым значением 63.0.



Аналогично, значения для labScore и hwScore обновятся и, наконец, значение finalGrade будет рассчитано путем добавления обновленных значений для examScore, labScore и hwScore.

Полученное значение 88,05 затем будет присвоено участку памяти для finalGrade.

Во время стадии анализа задачи при проектировании исходной задачи, было определено, что веса экзаменов, лабораторных и домашних заданий должны быть предварительно определены, и их значения должны быть одинаковыми для всех студентов в том же курсе.

Если мы хотим предотвратить случайное изменение весов, мы объявим эти идентификаторы как константы, поставив final в качестве ключевого слова в начале объявления.

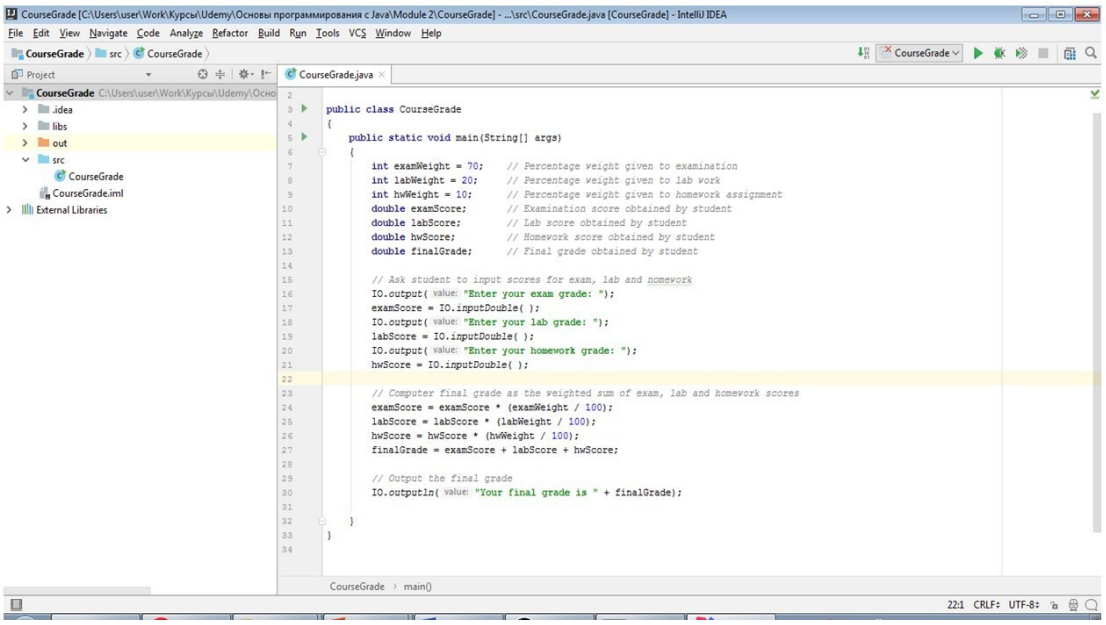
```
/**
 * CourseGrade determines the final grade which is computed as
 * the weighted sum of the grades obtained in exam, lab and homework
 */
public class CourseGrade
{
    public static void main(String[] args)
    {
        final int examWeight = 70;  // Percentage weight given to examination
        final int labWeight = 20;   // Percentage weight given to lab work
        final int hwWeight = 10;    // Percentage weight given to homework assignment
        double examScore;       // Examination score obtained by student
        double labScore;        // Lab score obtained by student
        double hwScore;         // Homework score obtained by student
        double finalGrade;      // Final grade obtained by student
```

В некотором смысле, вы можете думать об этом как запереть ячейку памяти и запретить любую попытку изменить ее значение в другой части программы.

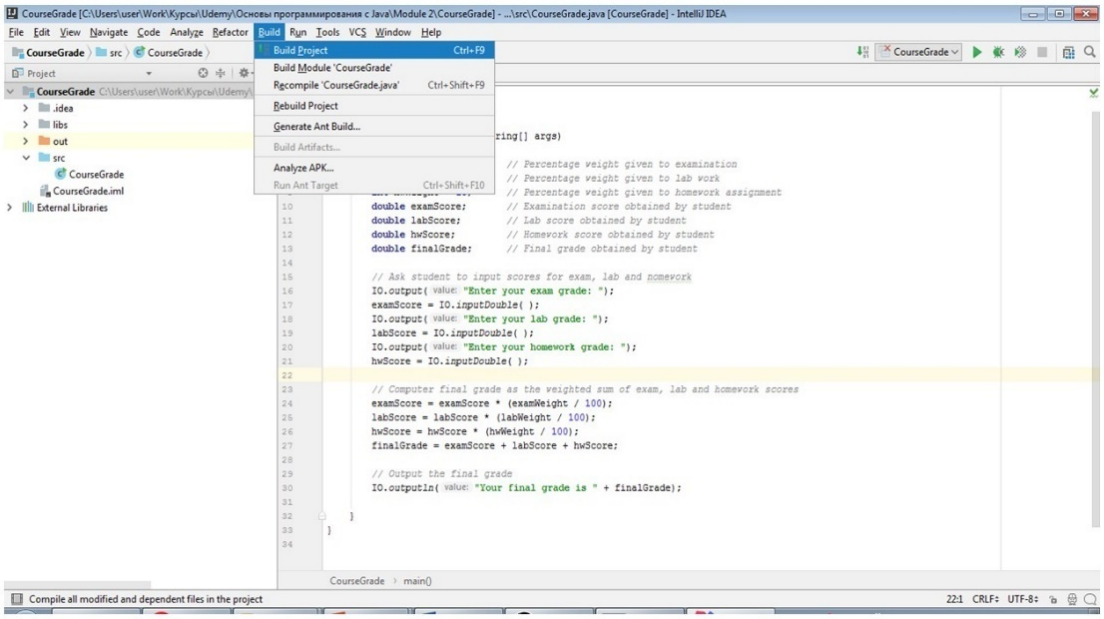
Можно сказать, что, если студент сдал плохо экзамены, но сделал хорошо лабораторные работы, попытка уменьшить вес для экзамена и увеличить вес для лабораторных будет блокирована.

Демонстрация примера

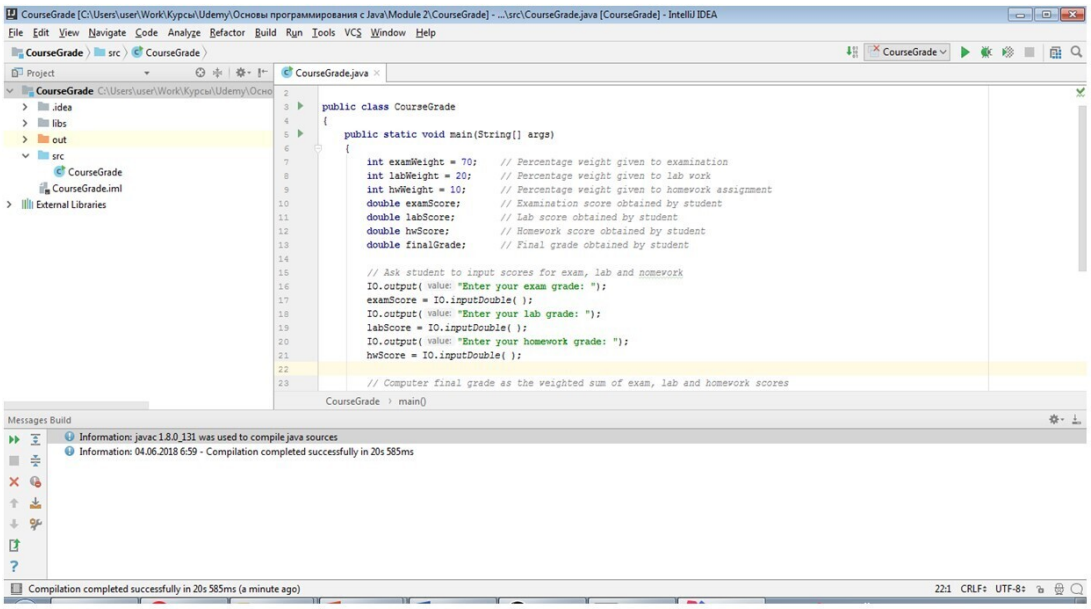
Давайте теперь посмотрим на программу в среде IntelliJ IDEA. Мы откроем проект под названием CourseGrade, который является программой, которую мы только что обсуждали.



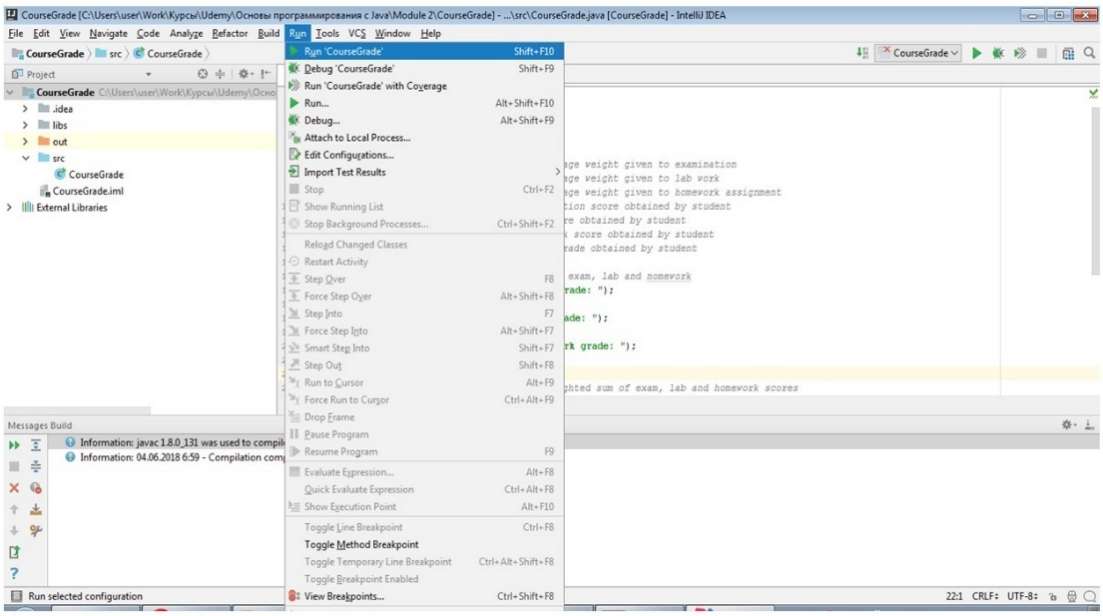
Откроем файл CourseGrade в редакторе исходного кода.
И вы можете видеть, что эта программа та, которую мы только что обсуждали.
И эта программа еще не скомпилирована.
Попробуем скомпилировать программу с помощью меню Build Project.



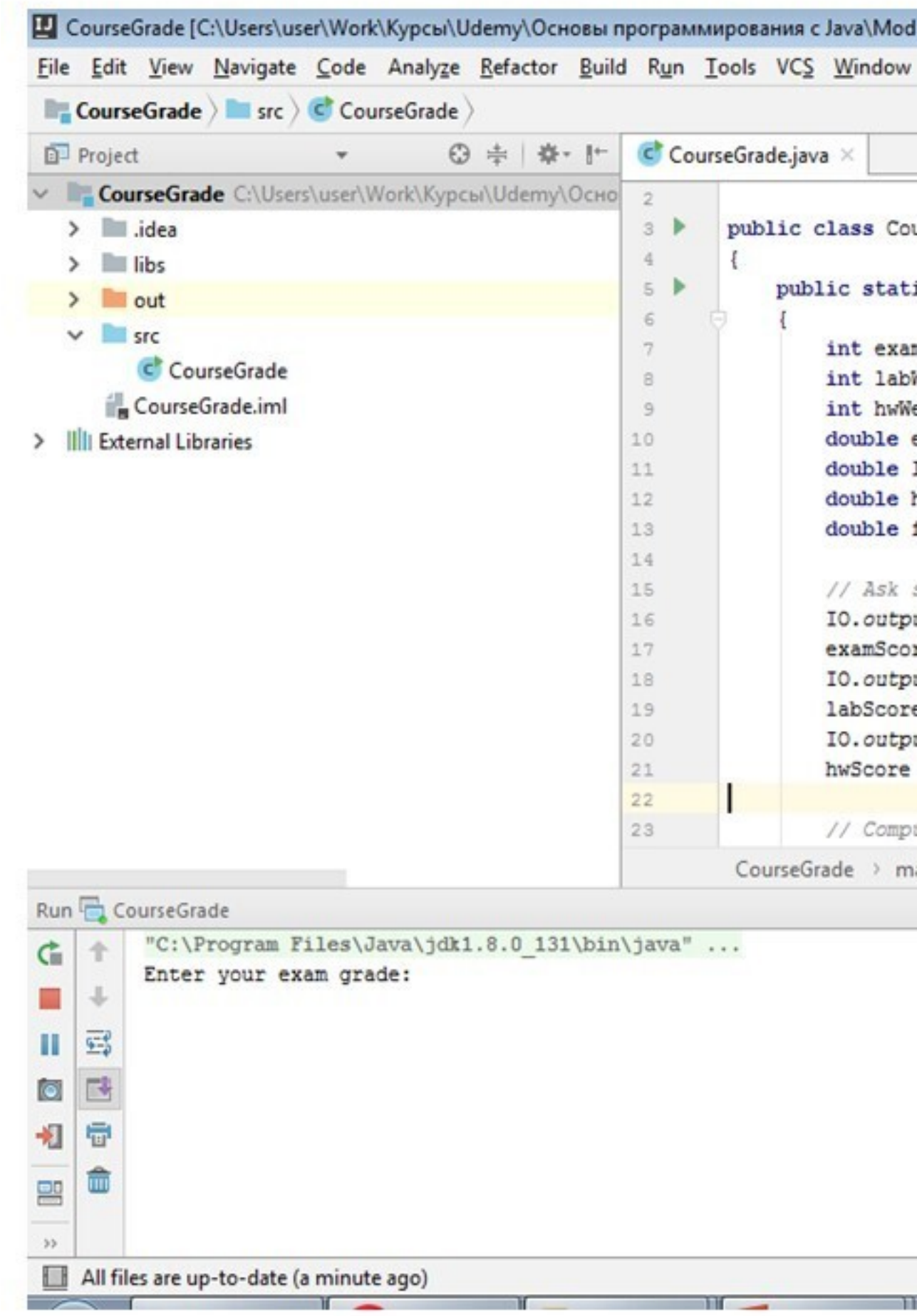
Вы можете видеть, что программа успешно компилируется без ошибок синтаксиса.



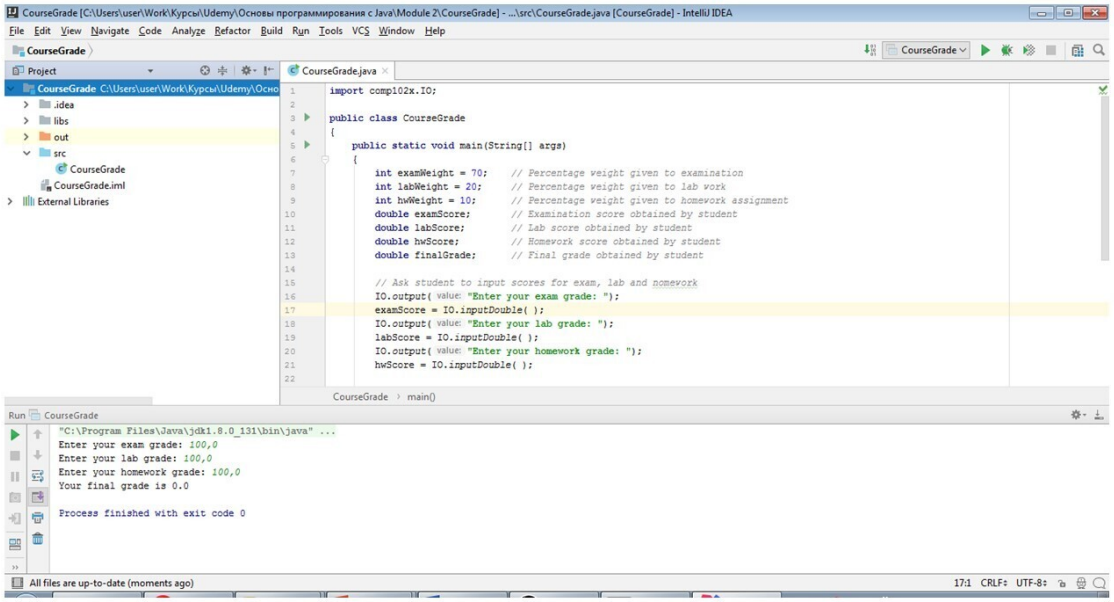
Давайте попробуем запустить программу с помощью кнопки Run.



Теперь вы можете видеть, что приглашение ввести ваши оценки экзаменов отображается в окне терминала.


CourseGrade [C:\Users\user\Work\Курсы\Udemy\Основы программирования с Java\Mod
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window
CourseGrade src CourseGrade
Project
CourseGrade C:\Users\user\Work\Курсы\Udemy\Осно
.idea
libs
out
src
CourseGrade
CourseGrade.iml
External Libraries
CourseGrade.java
public class Cou
{
public stati
{
int exa
int labW
int hwWe
double e
double l
double h
double f
// Ask
IO.outpu
examScor
IO.outpu
labScore
IO.outpu
hwScore
// Compu
CourseGrade
Run CourseGrade
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...
Enter your exam grade:
All files are up-to-date (a minute ago)

Скажем, что это очень хороший ученик и получил отличные оценки на экзаменах, лабораторных, а также домашних заданиях.



Обратите внимание, что мы вводим с десятичной запятой, 100,0, для каждой из оценок.

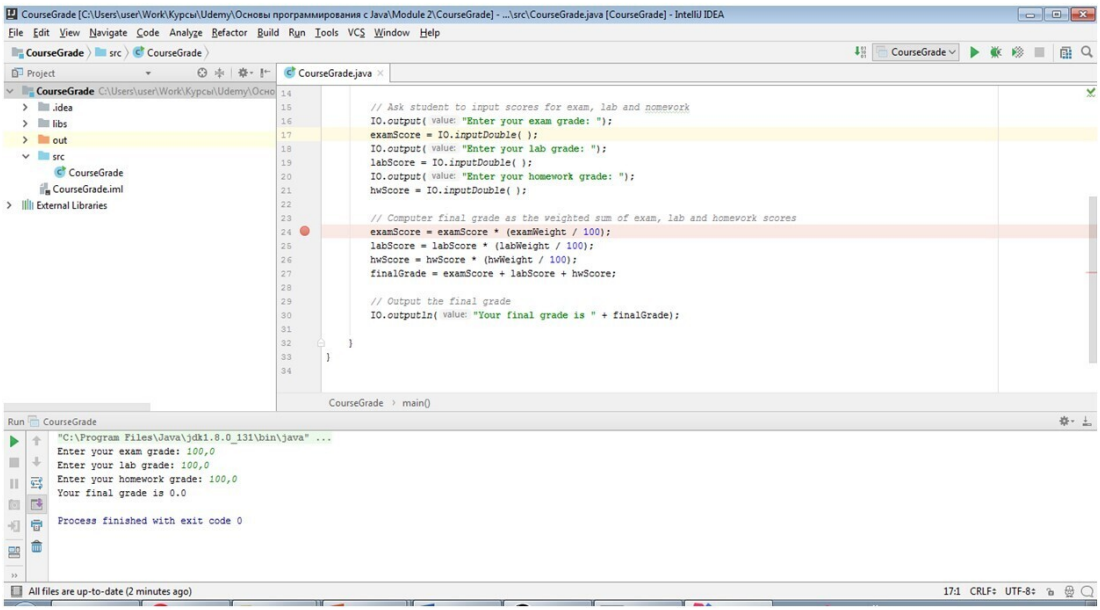
И в результате, по какой-то причине, итоговая оценка вернулась программой как 0.0.

Я уверен, что студент будет очень недоволен. Давайте попробуем выяснить, что вызывает эту проблему.

В IntelliJ IDEA, есть очень полезный инструмент отладки, который позволит нам проследить выполнение программы.

Инструмент очень полезен для отслеживания ошибок.

Давайте попробуем выяснить, есть ли какие-либо проблемы после того, как оценки были введены в программу, установив точку останова после объявлений IO.

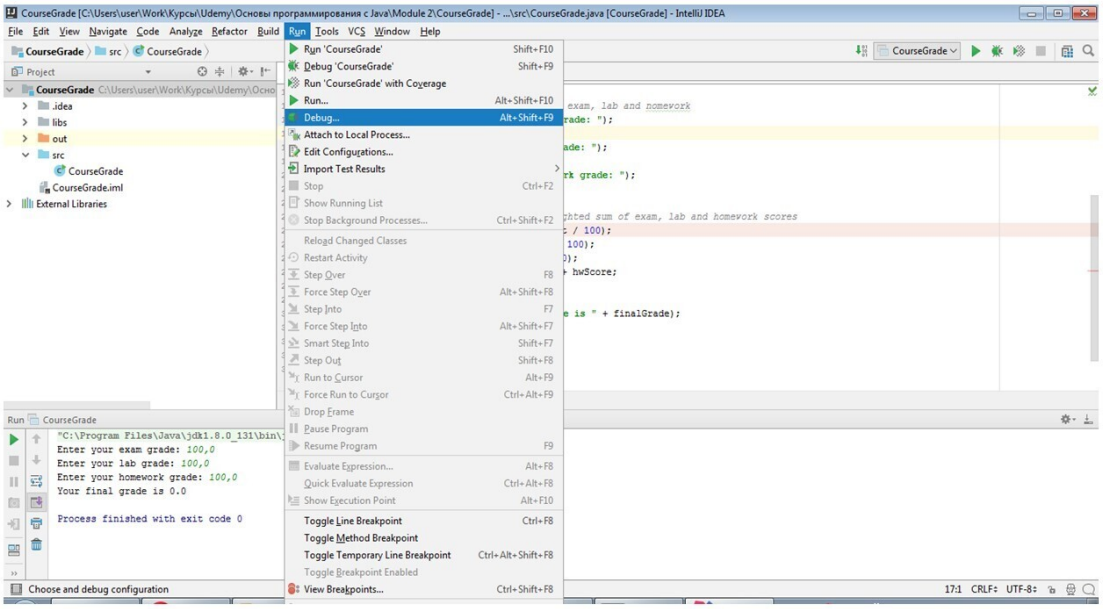


Для этого просто нажмем на соответствующее место вдоль правого столбца в окне редактора.

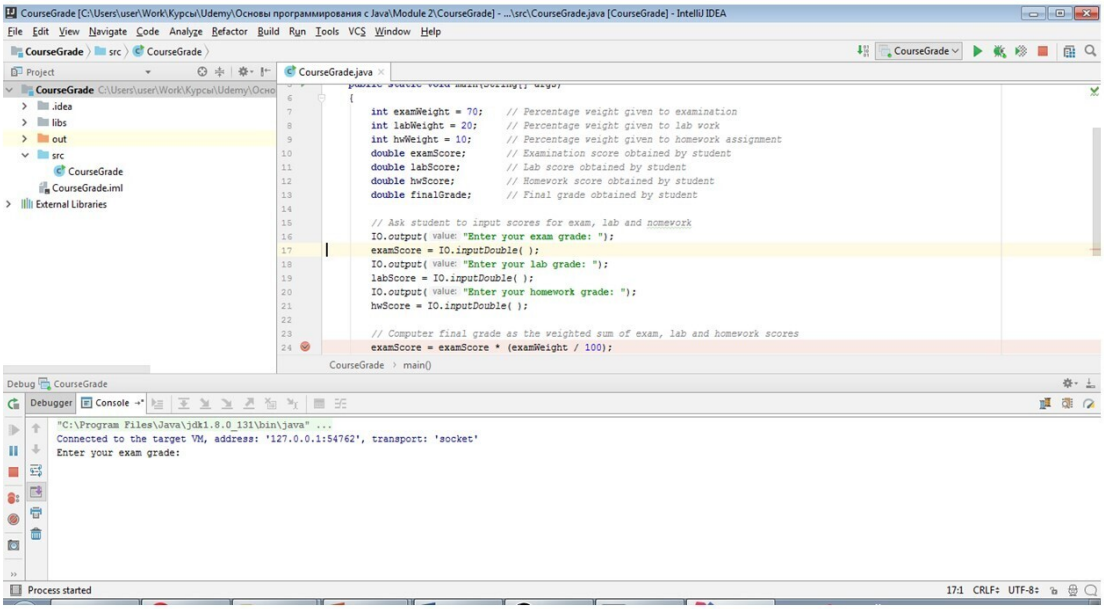
Вы можете нажимать в этих местах, чтобы установить или очистить точки останова.

После того, как точка останова устанавливается, IntelliJ IDEA выполняет программу до инструкции, где точка останова была установлена.

Давайте начнем выполнение снова, нажав кнопку Debug.



И введем ту же оценку, 100,0, для экзаменационной оценки, 100,0 для лабораторной.



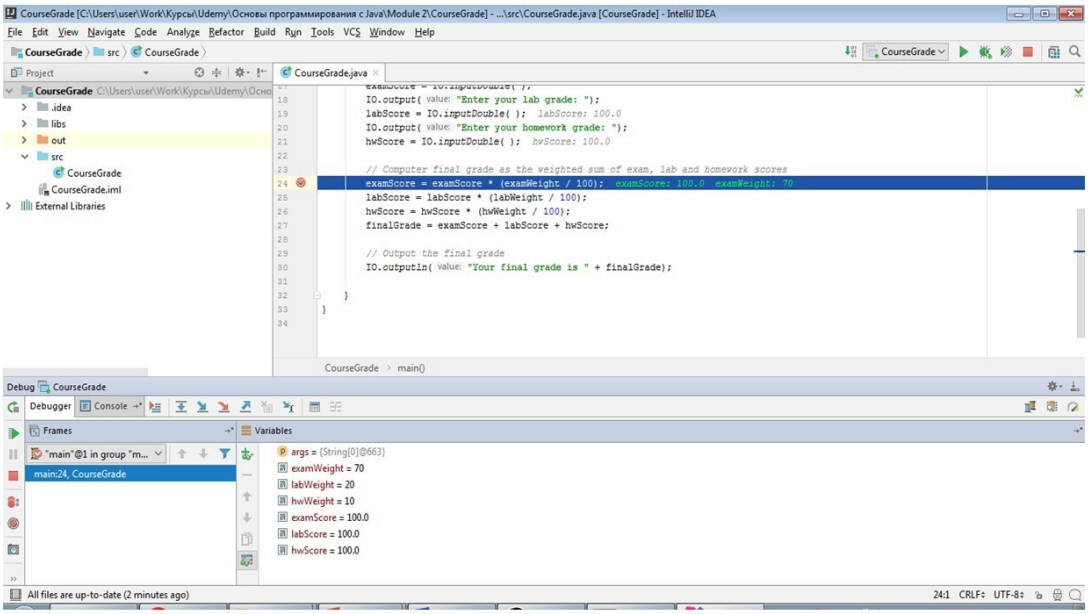
Давайте попробуем для домашних работ введем просто 100 без 0,0, чтобы сделать преобразование неявного типа, так как переменная hwScore имеет double тип и диапазон значений для int представляет собой подмножество для double.

Когда вы нажмете return, вы увидете, что окно редактора появляется с выражением, выделенным в точке останова.

Что еще более важно, при этом выскочило окно отладчика.

Это позволит вам просматривать текущее состояние программы, когда выполнение останавливается в точке останова.

Вы можете увидеть окно локальных переменных и изучить их значения.



Вы можете видеть, что значения весов определены при инициализации, и examScore, labScore и hwScore, все получили значение 100,0.

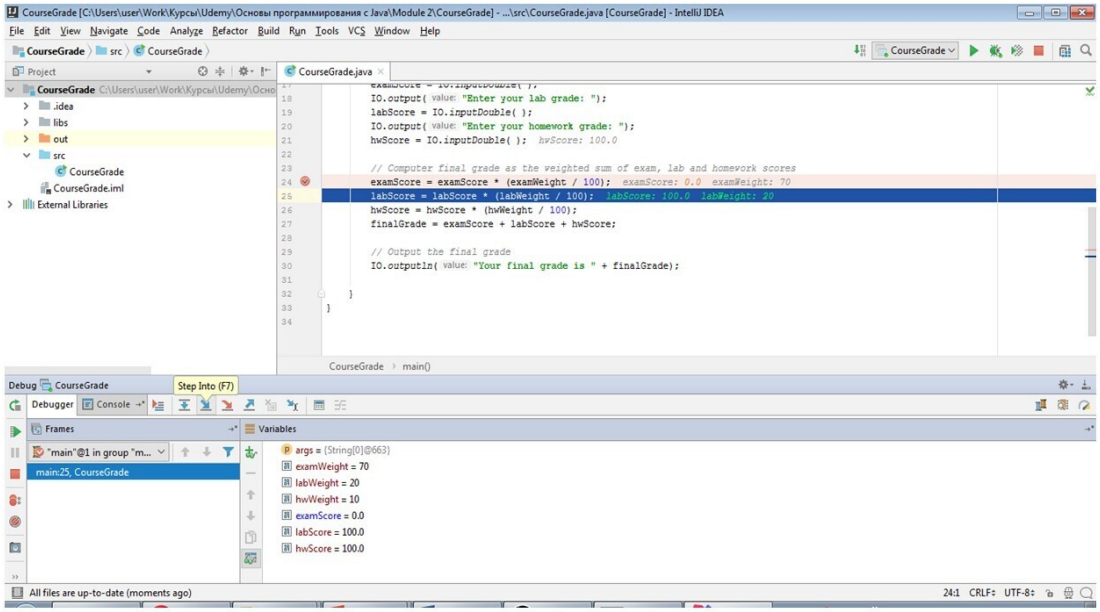
Вы можете продолжить выполнение программы, нажав на кнопку 'step', которая будет выполнять следующую инструкцию в программе.

Есть и другие кнопки здесь и лучший способ узнать, что они делают, это попробовать их самостоятельно.

Давайте просто использовать кнопку step на данный момент.

Вы можете видеть, что следующее утверждение подсвечено в окне редактора.

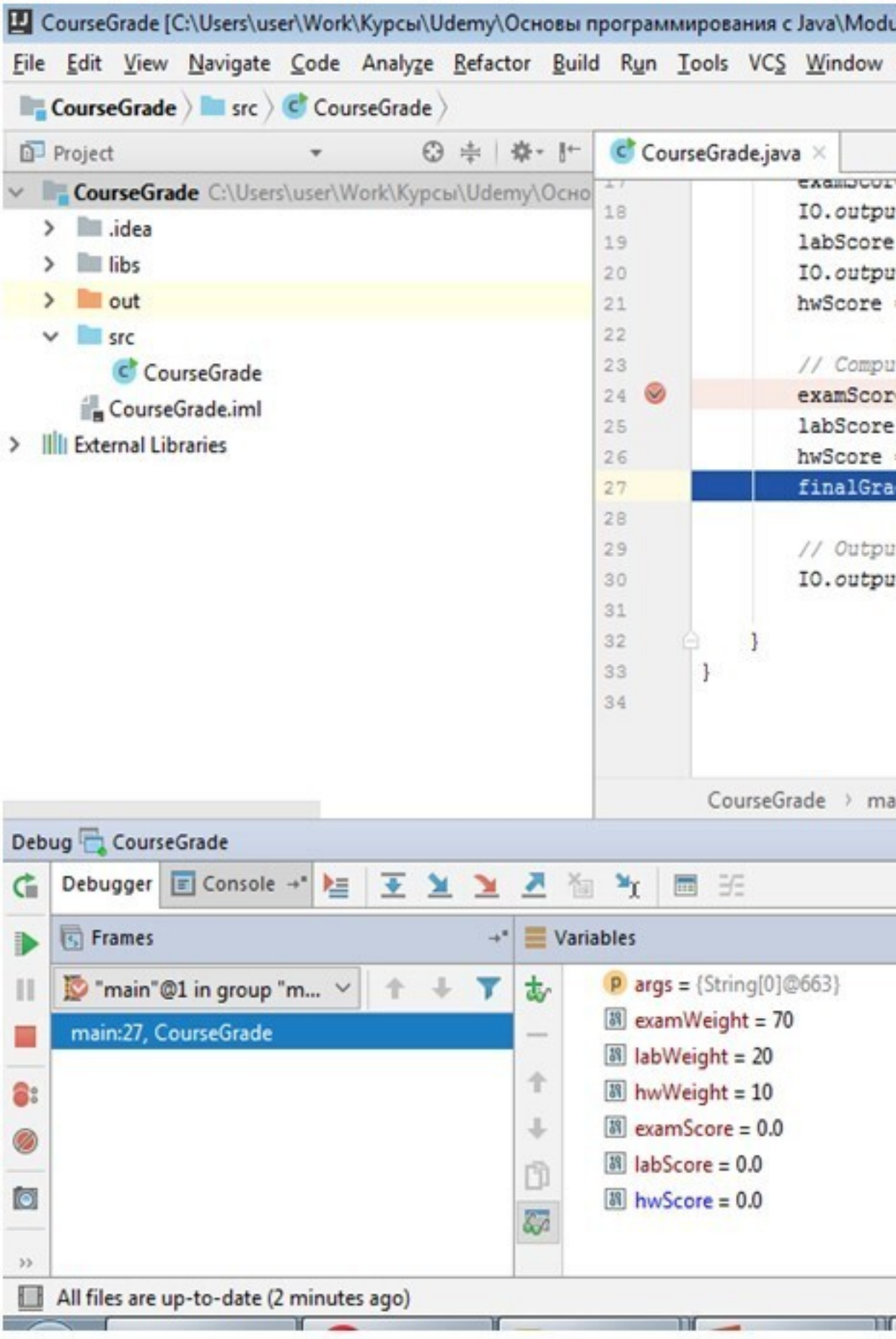
Это означает, что программа только что закончила выполнение инструкции для вычисления examScore.



Если изучить значения переменных снова, вы найдете, что все остальные значения остаются такими же, за исключением того, что для examScore теперь дается значение 0.0.

Поэтому здесь должна быть некоторая проблема с выполнением этого выражения.

Давайте продолжим выполнение чтобы увидеть, если у нас аналогичная проблема с вычислением и других показателей.


CourseGrade [C:\Users\user\Work\Курсы\Udemy\Основы программирования с Java\Modu
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window
CourseGrade src CourseGrade
Project
CourseGrade C:\Users\user\Work\Курсы\Udemy\Осно
.idea
libs
out
src
CourseGrade
CourseGrade.iml
External Libraries
CourseGrade.java
18 IO.outpu
19 labScore
20 IO.outpu
21 hwScore
22
23 // Compu
24 examScor
25 labScore
26 hwScore
27 finalGra
28
29 // Outpu
30 IO.outpu
31
32 }
33 }
34
CourseGrade › ma
Debug CourseGrade
Debugger Console
Frames
Variables
"main"@1 in group "m...
main:27, CourseGrade
args = {String[0]@663}
examWeight = 70
labWeight = 20
hwWeight = 10
examScore = 0.0
labScore = 0.0
hwScore = 0.0
All files are up-to-date (2 minutes ago)

Опять же, мы получаем 0 для labScore а затем 0 для hwScore, так что должны быть некоторые общие проблемы с расчетом и обновлением оценок, использующих эти выражения присваивания.

Можете ли вы найти эту проблему? Намек, что это связано с делением целых, которое я обсуждал ранее.

Обратите внимание, что вес экзаменов со значением 70 имеет тип int. Когда он делится на 100, что также является целым числом, результат деления 0,7 будет урезан и возвращает 0 в результате.

Далее 0 умножается на examScore, результирующее значение равно 0, который затем присваивается examScore.

Та же проблема возникает в labScore и hwScore.

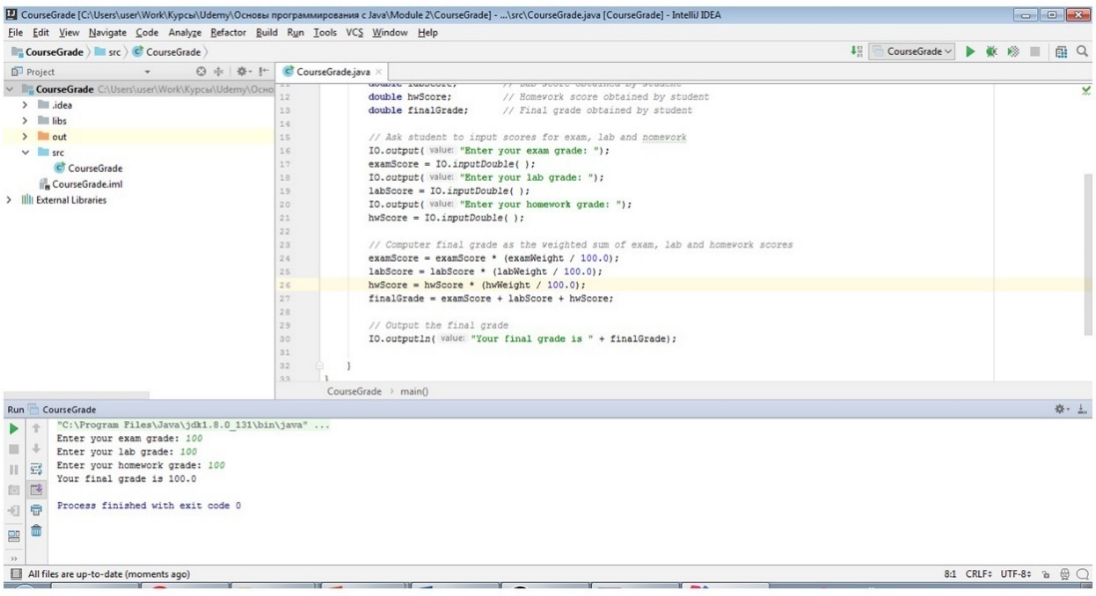
И эту проблему легко исправить. Для этого нужно заменить 100 на 100,0, и, если вы помните правило преобразования типов, которое мы только что обсудили, когда целое делится на число с плавающей точкой, результат будет преобразован в число с плавающей точкой.

Давайте теперь прекратим предыдущее выполнение программы и скомпилируем программу снова.

Теперь также нет ошибки синтаксиса.

В предыдущем выполнении программы было показано, что, хотя программа не имеет синтаксических ошибок, это не означает, что она будет работать так как надо, или семантически правильно.

Давайте попробуем выполнить программу, нажав кнопку Run.



Если ввести те же баллы 100, 100 и 100, вычисляется окончательная оценка 100 и все сейчас должны быть счастливы.

Вопросы

Задача

Каким вы думаете будет значение переменной result после выполнения следующего сегмента Java кода?

```
int i = 1234567890;
float f = i;
int result = i – (int)f;
```

1. 0
2. nonzero
3. an error

Ответ: 2.

Оба int и float 32-битные типы.

Все 32-битные int используются для представления целой части числового значения.

Тем не менее, для float, некоторые биты используются для представления целой части и некоторые для десятичной части.

При присвоении большого целого значения переменной с плавающей точкой, уменьшенное количество битов для его целой части, возможно, не в состоянии точно представить большое значение.

Таким образом, вычисление разности между int и float переменных одного и того же большого численного значения, не всегда может дать результат 0.

Вы можете проверить код, указанный выше, в программе Java.

Обсуждение отладки

Достаточно часто случается, что мы не можем написать без ошибок программу с первой попытки.

Нам часто нужно отлаживать нашу программу несколько раз, чтобы сделать безупречную версию.

Ниже пример программы содержит ряд ошибок. Попробуйте определить все эти ошибки.

```
1
2  public Class InputDemo
3  {
4      public static void main(String[] args) {
5
6          IO-output('Enter degree in Celsius: '),
7
8          double celsius = IO.inputDouble();
9
10         double fahrenheit = Celsius * (9/5) + 32;
11
12         IO.outputln(celsius + " celsius is " + fahranheit + " degrees in fahrenheit")
13     }
14 }
15
```

Ответ:

```
1  import compl02x.IO;
2
3  public class InputDemo
4  {
5      public static void main(St
6
7          IO.output("Enter degre
8
9          double celsius = IO.in
10
11         double fahrenheit = ce
12
13         IO.outputln(celsius +
14     }
15 }
16
17 // line  1: missing import com
18 // line  3: small letter "c" f
19 // line  7: should be IO.outpu
20 // line  7: should be double q
21 // line  7: should be a semico
22 // line 11: small letter "c" f
23 // line 11: should not use int
   rrecting it is (9.0 / 5)
24 // line 13: "fahrenheit" is mi
25 // line 13: missing semicolon
26
```

Простой IO

Когда мы обсуждали аппаратную часть компьютера, мы говорили, что компьютеры взаимодействуют с пользователями через устройства ввода и вывода.

Большинство языков программирования высокого уровня обеспечивают стандартные методы ввода и вывода программ для получения информации от пользователей и вывода результатов пользователям для просмотра.

```
System.out.println("Hello, world!");
```

```
import comp102x.IO;
```

```
IO.output("Enter your exam grade: ");

IO.outputln("Your final grade is " + finalGrade);
```

```
double examScore = IO.inputDouble( );

int intValue = IO.inputInteger( );
```

В Java, стандартные методы для вывода текста в консоль являются System.out.println и System.out.print.

System представляет собой класс Java.

Вы видели println в программе HelloWorld.

Разница между print и println в том, что println будет переводить на новую строку после печати, в то время как print останется на той же строке.

Технически, println и print являются методами класса PrintStream.

Обратите внимание, что они разделены точкой или оператором точка.

Я вернусь к этому, когда будем обсуждать классы, объекты и методы позже.

Есть аналогичные стандартные методы для ввода на основе класса Scanner, но он немного неуклюжий, поэтому здесь создан свой пользовательский класс ввода-вывода I/O с именем IO в качестве оболочки, который будет сочетать как ввод, так и вывод и, надеюсь, предоставляет более простой в использовании интерфейс ввода / вывода для вас.

Методы обертки очень часто используются в Java.

Они в основном предоставляют альтернативный интерфейс на основе существующих методов, чтобы удовлетворить определенной цели.

В этом случае, новый класс I/O, можно надеяться, будет легче в изучении для начинающих программистов.

Я вернусь к классу Scanner, когда мы будем обсуждать файловый ввод/вывод.

Методами вывода являются IO.outputln и IO.output.

Вы видели их обоих в программе CourseGrade.

Разница между ними состоит в том, что IO.outputln будет перемещаться на новую строку, в то время как IO.output останется на той же строке.

Два метода output и outputln берут строку символов в качестве параметра и выводят ее в консоль.

В случае println, finalGrade, который имеет double тип, будет преобразован в символьную строку, и знак плюс, который вы видите здесь является оператором конкатенации, который соединяет две строки в единую строку символов.

Методы ввода – это IO.inputInteger для целых чисел и IO.inputDouble для double.

Вы видели IO.inputDouble, когда программа просила пользователя ввести examScore, labScore и hwScore.

Аналогичным образом, IO.inputInteger может быть использован для ввода целых.

Для того чтобы использовать класс IO, есть еще одна вещь, которую вы должны будете сделать.

Вам нужно импортировать пакет comp102x, который предоставлен в виде библиотеки.

В частности, необходимо будет включить выражение «import comp102x.IO;» в начало программы.

Где import является зарезервированным словом Java.

Вы увидите много примеров по использованию import и этих методов ввода-вывода далее.

Я хочу сказать немного больше о пользовательском интерфейсе для ввода, особенно для чисел и текста, что часто бывает необходимо во многих приложениях.

Например, когда вы идете в супермаркет, представьте, как долго вы будете рассчитываться, если кассир должен вводить каждый элемент, который вы приобрели, с помощью клавиатуры.

Обратите внимание, что не только цены должны быть введены, магазину необходимо также вести учет всех деталей для каждой транзакции, то есть, должно быть введено название каждого элемента и т.д.

Так что, компьютеры не очень хороши в чтении рукописных или даже печатных слов или в понимании человеческой речи, по крайней мере пока, хотя технология улучшается, но мы все еще довольно далеко от систем, которые могли бы обеспечить скорость и точность для многих приложений.

Альтернативные интерфейсы ввода необходимы.

Клавиатура является наиболее часто используемым устройством ввода. Однако существуют альтернативные технологии, такие как штрих-коды.

Я уверен, что каждый из вас часто сталкивался со штрих-кодами в повседневной жизни.

Например, в супермаркетах, библиотеках и на почте.

В последнее время появляются и новые технологии для распознавания голоса.

Например, Siri, который iPhone использует для голосового ввода, Google поиск, и у многих из вас, вероятно, есть опыт разговоров с компьютером, когда вы делали телефонные звонки.

В последнее время радиочастотная технология идентификации RFID позволяет отслеживать продукты через беспроводные бесконтактные средства.

Некоторые супермаркеты экспериментируют с радиометками RFID, которые позволяют продавцу осуществлять проверку, не вынимая каждый товар из вашей корзины.

Я покажу вам пример манипулирования со штрих-кодом в программе.

Штрих-код является машиночитаемым представлением данных в виде изображения.

Я буду говорить об одномерных штрихкодах.

2D штрих-коды, такие как QR-коды, также набирают популярность.

В основном, линейный штрих-код представляет данные путем изменения ширины и промежутка между набором параллельных линий.

Здесь показаны некоторые примеры штрих-кодов.

Штрих-технологии были разработаны в 1960-х годах и получили коммерческий успех, так как они широко используются в автоматизированных кассовых системах, таких как те, что используются в супермаркетах.

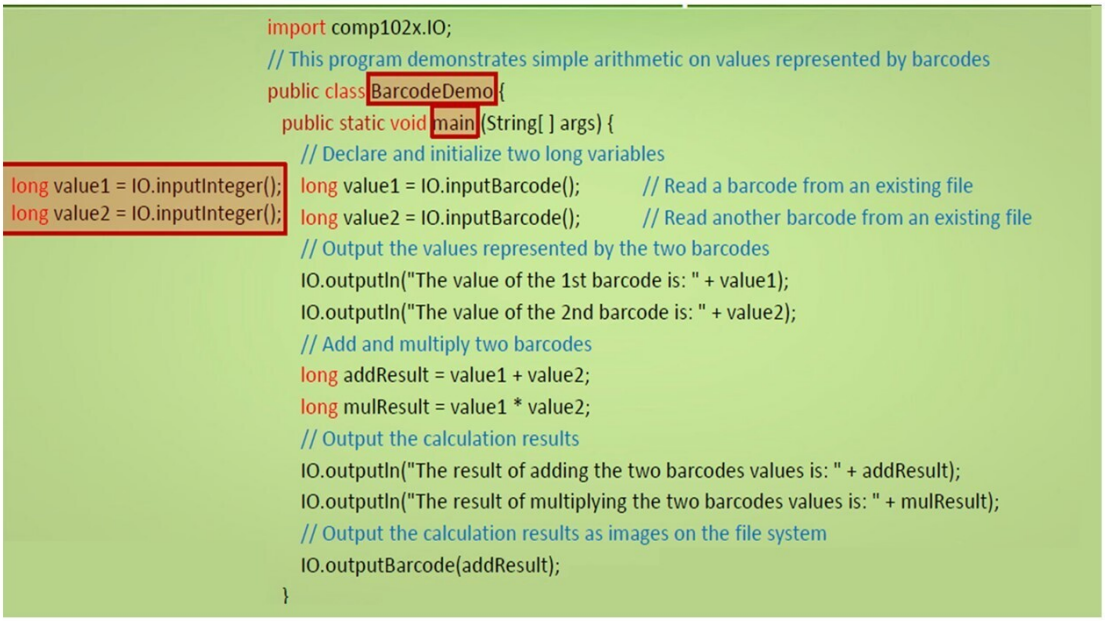
Я хочу отметить, что в следующем примере, мы используем абстракцию данных для работы со штрих-кодом.

То есть, нам нужно только знать, что штрих-код представляет собой число в виде серии цифр и считыватель штрих-кода будет способен декодировать штрих-код и ввести число в компьютер.

Как это на самом деле сделано или реализовано – не важно для пользователя.

Можно рассматривать его таким же образом, как номера на клавиатуре.

Эта программа здесь показывает, что простая арифметика может быть применена к числовым значениям, представленным с использованием штрих-кодов.




```
import comp102x.IO;
// This program demonstrates simple arithmetic on values represented by barcodes
public class BarcodeDemo{
  public static void main (String[ ] args) {
    // Declare and initialize two long variables
    long value1 = IO.inputBarcode();          // Read a barcode from an existing file
    long value2 = IO.inputBarcode();          // Read another barcode from an existing file
    // Output the values represented by the two barcodes
    IO.outputln("The value of the 1st barcode is: " + value1);
    IO.outputln("The value of the 2nd barcode is: " + value2);
    // Add and multiply two barcodes
    long addResult = value1 + value2;
    long mulResult = value1 * value2;
    // Output the calculation results
    IO.outputln("The result of adding the two barcodes values is: " + addResult);
    IO.outputln("The result of multiplying the two barcodes values is: " + mulResult);
    // Output the calculation results as images on the file system
    IO.outputBarcode(addResult);
  }
```

Программа начинается с импорта класса IO из пакета comp102x.

В дополнение к выполнению операций ввода/вывода от стандартных устройств ввода и вывода, класс IO может также принимать входные и выходные данные как штрих-код.

Этот пример, иллюстрирующий, что, когда используется компьютер, тогда не важно, получены ли входные данные от пользователя через клавиатуру или с помощью штрих-кода.

Подробное представление не важно до тех пор, пока для программы обеспечены методы декодирования информации.

Как и прежде, программа получает имя класса, в данном случае BarcodeDemo, и метод main как главную точку входа в программу.

Первая часть тела программы принимает два штрих-кода в качестве входных данных с помощью метода inputBarcode от класса IO.

Чуть позже вы увидите в демо программе, что вместо ввода числа с помощью клавиатуры, пользователю будет предложено выбрать изображение штрих-кода, которое представляет некоторое число из существующего файла.

Числа затем будут расшифрованы и присвоены переменным value1 и value2.

Это похоже на использование inputInteger или inputDouble, если входные данные должны были быть введены с консоли.

Цифры, введенные с клавиатуры, по-прежнему должны быть декодированы перед присвоением в соответствующие переменные.

Обратите внимание, что здесь мы используем тип long, потому что штрих-коды могут представлять очень большие числа, которые могут быть вне диапазона типа int.

После того, как штриховые коды декодируются в виде чисел, они могут быть использованы так же, как если они были созданы с помощью других средств.

Два IO.outputln объявления здесь выводят значения, представленные двумя штрих-кодами.

Значения можно обрабатывать так же, как числа, представленные в других примитивных типах в Java, и арифметические операции, такие как сложение и умножение, могут быть применены к этим числам.

В этом случае, результаты операций будут присвоены переменным addResult и multResult с примитивным целочисленным типом данных long снова, потому что штрих-коды могут представлять очень большие числа.

И результаты арифметических операций распечатываются для просмотра.

Кроме того, результаты могут также быть выведены в виде штрих-кода, используя метод outputBarcode для класса IO.

В этом случае штрих-код будет создан для значения addResult и сохранится в выходном файле.

Демонстрация примера

Прежде чем продемонстрировать программу штрих-кодов, давайте сначала посмотрим на некоторые образцы штрих-кодов.

Эти штрих-коды хранятся в виде файлов изображений.

Первый из них сфотографировали с обложки книги с помощью камеры сотового телефона.

Вы можете увидеть, что качество не очень хорошее.

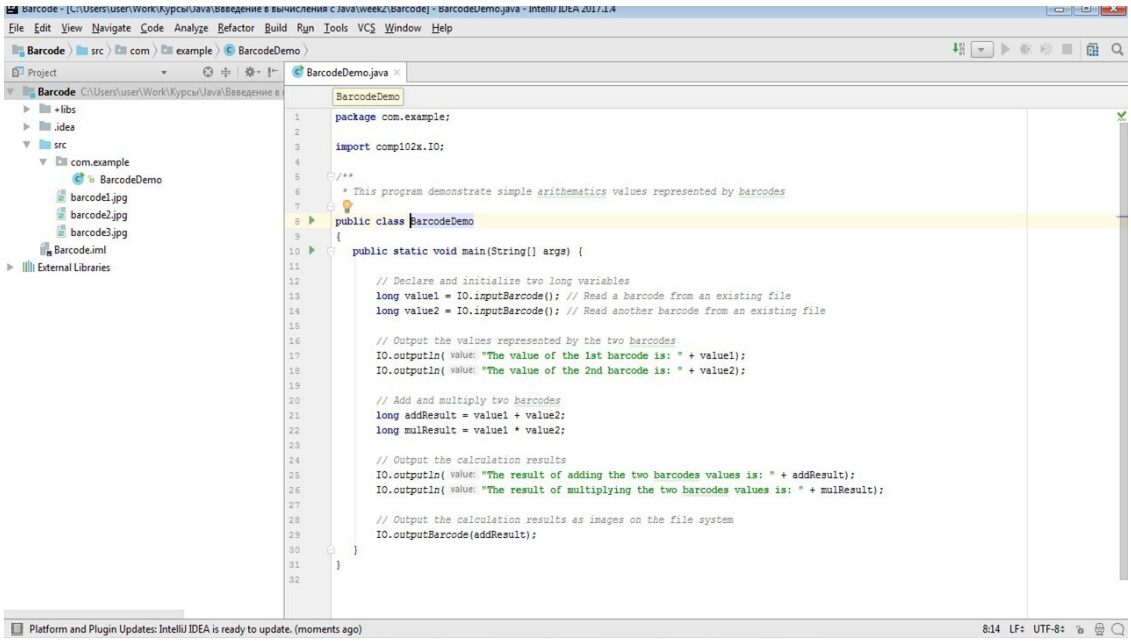
Второй сгенерирован компьютером, и мы не можем сказать, какие числа этот штрих-код представляет без помощи считывателя штрих-кодов.

Третий штрих-код представляет собой число с цифрами от 0 до 9.

Обратите внимание, что эти штрих-коды разных размеров и качества.

Теперь мы можем открыть проект BarcodeDemo.

Вы можете видеть, что это та же программа, которую мы только что обсуждали.



Программа составлена без ошибок.

Мы можем запустить программу, нажав кнопку Run.

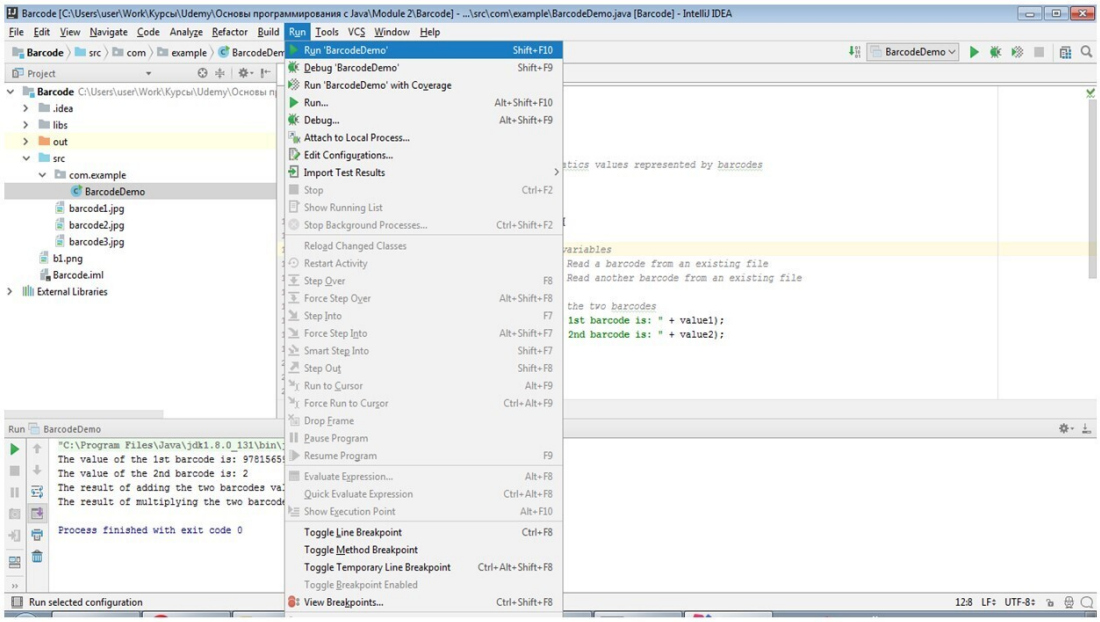
Теперь вы можете увидеть, что появилось окно диалога, которое запрашивает местоположение первого изображения штрих-кода.

Это происходит при выполнении метода inputBarcode().

Давайте выберем barcode1 в качестве первого штрих-кода.

Затем программа запрашивает второй штрих-код при втором вызове inputBarcode().

Давайте выберем barcode2.



И вы можете видеть, что значения штрих-кодов отображаются в окне консоли.

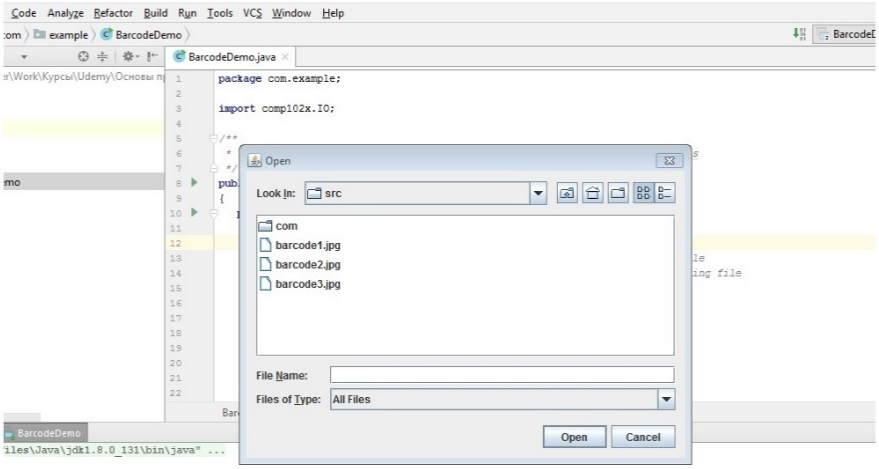
Значение первого штрих-кода огромное число из 12 цифр.

Второй штрих-код является небольшим числом со значением 2.

Легко проверить, что сумма двух штрих-кодов отображается здесь правильно.

Вы также можете проверить, что результат умножения также должен быть правильным.

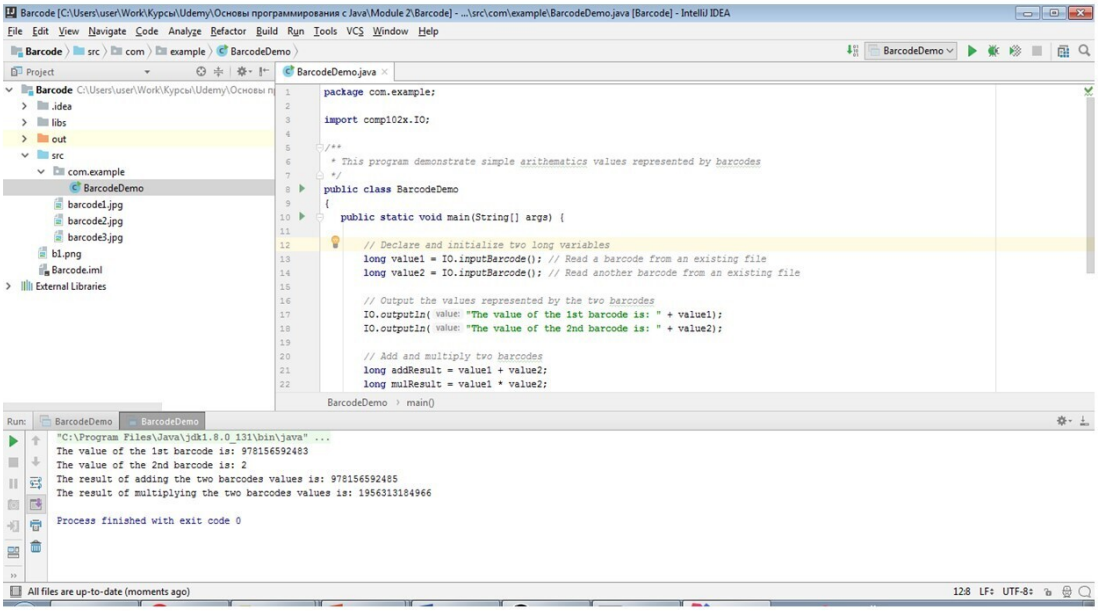
Обратите внимание, что мы еще не сделали – есть еще всплывающее окно, которое запрашивает имя файла.



Это потому, что последнее выражение в программе, outputBarcode() выводит штрих-код со значением addResult в изображение штрих-кода.

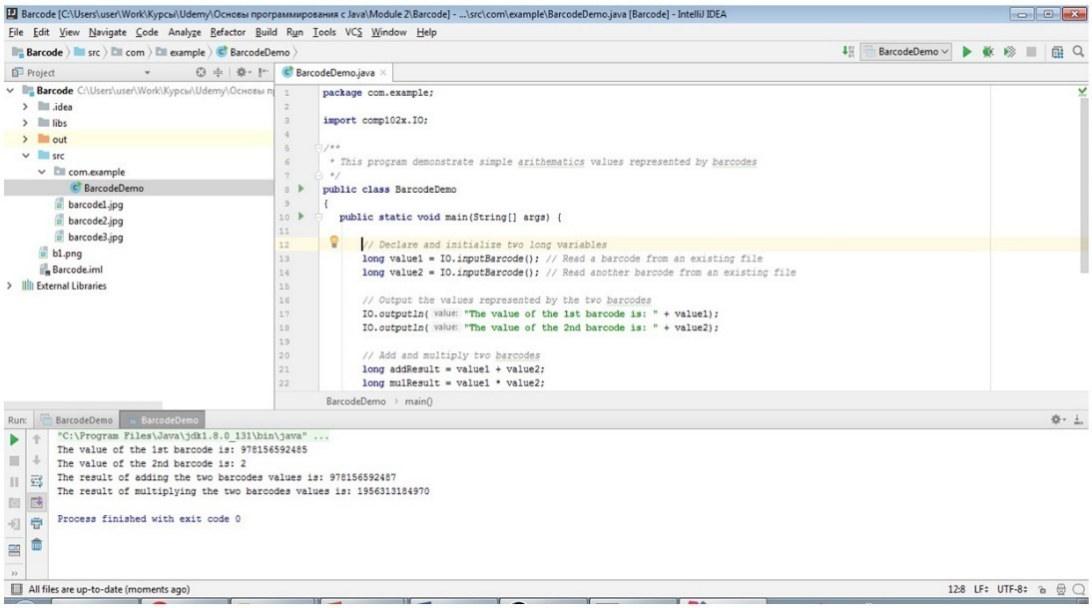
Давайте использовать название b1 в качестве выходного файла.

Теперь программа завершается.



Если вы проверите папку проекта, вы увидите, что есть файл с именем b1, и вы сможете увидеть, что это штрих-код, но мы не можем сказать его значение, просто взглянув на изображение.

Нет проблем, давайте попробуем еще раз запустить программу.



На этот раз, давайте использовать b1 качестве первого изображения штрих-кода и использовать barcode2 как второй штрих-код, как и раньше.

Вы можете видеть, что значение первого штрихкода теперь то же самое, как значение addResult в предыдущем выполнении программы.

Таким образом, мы убедились, что штрих-код с addResult как значением, действительно был сформирован.

Вы также можете увидеть результат сложения и умножения.

Давайте введите b2 в качестве выходного файла, чтобы завершить программу.

Я уверен, что вы сможете найти много штрих-кодов, чтобы поэкспериментировать с программой.

Вы можете использовать ваш смартфон или цифровую камеру, чтобы сфотографировать штрих-кода, а затем ввести их в программу.

Объектно-ориентированное программирование. Введение

Мы уже рассмотрели некоторые элементарные понятия программирования. И мы говорили о правилах именования идентификаторов.

И один важный тип идентификатора – это переменная.

Переменная имеет имя и тип, который определяет, какой тип значений может быть сохранен в памяти, выделенной для переменной.

Мы говорили о примитивных типах данных, типах, которые встроены в Java.

Мы также говорили о выражениях, особенно арифметических выражениях, которые обеспечивают выполнение математических вычислений для определенных задач.

Комбинируя выражения и операторы присваивания, можно изменять значения переменных.

Затем мы обсудили преобразования типов, которые позволяют операции, выполняемые со смешанными типами данных.

Наконец, мы ввели некоторые простые методы ввода и вывода с помощью класса IO, который предоставлен пользовательской библиотекой.

Примитивные типы данных могут использоваться для задач, которые имеют дело с числами и текстом, но многие задачи в реальном мире должны иметь дело с более сложными объектами, чем просто цифры и текст, например, когда вы хотите работать с изображениями и видео.

Объектно-ориентированный подход позволяет вам создавать вычислительные объекты для решения комплексных задач, используя абстракцию данных, которую мы обсуждали ранее.

Если вы посмотрите на окружающую среду вокруг вас, вы найдете много физических объектов.

Вы увидите, что у этих физических объектов есть две общие характеристики:

Все они имеют свои отдельные состояния и поведение.

Например, большинство источников света могут иметь только два состояния – включен и выключен и два поведения – включить свет или выключить свет.

Но есть также источники света, которые могут иметь более двух состояний и ведут себя по-разному, например, есть диммер, который может регулировать свет постепенно до различной яркости.

Или вы можете даже задать для него мигающее состояние, как у новогодней гирлянды.

Если брать смартфон, он также может быть во включенном или выключенном состоянии.

И если он включен, его яркость может быть скорректирована до определенного уровня.

Вы можете изменять поведение смартфона, звоня своему другу, проигрывая музыку, занимаясь веб-серфингом, чтобы узнать последние новости.

Вы увидите, что все эти объекты реального мира могут быть смоделированы как вычислительные объекты, используя объектно-ориентированное программирование.

Например, умным выключателем света со сложным поведением можно управлять с помощью объектно-ориентированной программы в соответствии с условиями освещения в комнате, так что можно обеспечить энергосбережение.

И многие приложения, написанные для смартфонов, являются объектно-ориентированными программами.

Здесь я представлю основные понятия объектно-ориентированного программирования с использованием Java.

Я буду обсуждать такие важные понятия как классы, объекты и методы.

И переменные можно найти в классах, объектах и методах.

Мы также кратко поговорим о правилах области применения для значений переменных в различных условиях.

Правила области применения будут обсуждаться более подробно позже.

Java является объектно-ориентированным языком программирования.

Целью этого раздела является познакомить вас с классами, объектами и методами, которые являются фундаментальными понятиями в объектно-ориентированном программировании.

Так почему же объектно-ориентированный подход хорош для решения проблем?

В нашей повседневной жизни мы часто используем инструменты или артефакты, чтобы помочь нам в решении задач.

Мы используем плиты, печи, тостеры и микроволновые печи, чтобы сделать продукты, и при этом мы не должны понимать, как они работают.

Важно лишь иметь хороший рецепт, и рецепт таким образом, подобен алгоритму.

Когда нам нужно попасть из одного места в другое, мы используем различные транспортные средства, мы можем путешествовать с помощью машины, поезда и велосипеда по суше; или путешествовать по воздуху с помощью самолетов и вертолетов, или путешествовать по морю с помощью кораблей и катеров.

Это конкретные или материальные объекты.

Есть также нематериальные объекты.

Если вы хотите получить доступ к Интернету с помощью мобильного устройства, вы можете подключиться к сети, используя сеть сотовой связи телефонной компании, вы также можете использовать WiFi или подключиться к другим устройствам.

Когда вы используете смартфон для общения с другими людьми, вы можете открыть панель набора номера телефона или e-mail приложение, каждое из них может рассматриваться как вычислительный объект или программный объект.

С помощью этих примеров мы можем наблюдать, что люди любят группировать объекты с аналогичными свойствами вместе и давать им коллективное имя, например, печь, автомобиль, самолет и телефон.

В то время как эти объекты имеют сходное поведение с точки зрения их использования, каждый из них также имеет некоторые свои определенные свойства.

Например, в то время как печи могут быть использованы для приготовления пищи, печь может быть электрической или газовой, автомобили используются для перевозки, но каждый автомобиль может иметь различный цвет и пробег, для смартфонов, я уверен, что настройки и данные в моем телефоне отличаются от вашего.

Объектно-ориентированный подход в состоянии описать все эти сходства и различия.

Давайте использовать машину в качестве примера для дальнейшего рассмотрения концепции объектно-ориентированного подхода.

Вот коллекция автомобилей.

Они все, кажется, очень хорошие автомобили, которые мы все хотели бы иметь.

Мы знаем, что автомобиль является своего рода транспортным средством, который имеет определенные свойства, такие как колеса, у него есть двигатель, и он нуждается в топливе, чтобы двигатель работал, и каждый автомобиль имеет владельца.

Автомобиль выполняет определенные функции, например, он может ускориться и замедлиться, он может ехать назад и выполнять повороты, и надеюсь, не будет врезаться в другие автомобили.

И существует собственник для каждого отдельного экземпляра автомобиля, это может быть моя машина, или эти автомобили находятся в собственности других людей, и все эти экземпляры имеют некоторые различия, такие как цвет, количество пассажиров, которые могут поместиться, год выпуска, размер двигателя и т.д.

В объектно-ориентированной терминологии, это можно рассматривать как класс автомобилей.

При этом экземпляры индивидуальных автомобилей являются объектами и характеристики этих объектов, это их атрибуты или свойства, такие как цвет, количество пассажиров, год выпуска, размер двигателя и так далее.

Эти свойства часто называются полями объектов.

Объект также может демонстрировать определенное поведение или действия, которые он может выполнять, например, движение вперед, перемещение назад и поворот.

Эти действия называются методами в объектно-ориентированном программировании.

Обратите внимание, что здесь также может быть иерархия автомобилей, это легковые автомобили, вы могли бы также иметь фургоны, грузовики и внедорожники.

Позже, я представлю идею подкласса и суперкласса, где подкласс может наследовать свойства суперкласса.

Таким образом, вы можете видеть, что объекты являются фундаментальными строительными блоками объектно-ориентированных программ.

В объектно-ориентированных программах, программные объекты используются для моделирования объектов реального мира, которые имеют определенные состояния или атрибуты и поведения или действия.

Давайте теперь посмотрим на классы, объекты и методы в Java.

Мы видели, что класс описывает группу объектов с общими свойствами и поведением.

Например, мы можем определить класс автомобиля, который основывается на общей концепции транспортного средства, которое двигается на колесах и может перемещаться из одного места в другое.

Или мы можем определить класс "смартфонов", это мобильные электронные устройства, которые могут быть использованы для совершения телефонных звонков, веб-серфинга, воспроизведения музыки, отправки SMS и т.д.

Мы можем использовать ключевое слово class, чтобы определить класс в Java.

На самом деле, мы уже использовали ключевое слово class в нашей программе CourseGrade, и я уже упоминал ранее, что все программы Java, это классы.

```
public class Car

public class SmartPhone
```

Два выражения здесь объявляют два класса, один для автомобиля, а другой для смартфона, заметьте, что имена Car и SmartPhone являются Java идентификаторами.

Здесь используется верхний CamelCase по соглашению об именах для классов.

Цель определения класса заключается в разработке шаблона для создания объектов.

Т.е. класс – это шаблон для создания объектов.

После того, как класс определен, мы можем создавать экземпляры или объекты в этом классе.

Понятия объект и экземпляр являются взаимозаменяемыми.

В предыдущем примере, в классе автомобилей, могут быть различные экземпляры (или объекты) автомобилей, которые могут принадлежать мне и другим людям.

В классе "смартфон", различные экземпляры могут быть созданы для каждого студента на этом курсе.

Аналогичным образом, если класс студентов определен, экземпляры студентов могут быть созданы для представления каждого студента в классе.

В Java, экземпляры или объекты создаются с помощью конструкторов и ключевого слова new.

Я вернусь к этому позже в лекции.

Как я уже говорил, класс выступает в качестве шаблона или плана для объекта.

Определение класса должно охватить две основные характеристики.

Первая характеристика – это состояния или свойства объекта в классе, которые часто называют полями.

Для объекта автомобиля, поля могут включать имя его владельца и его цвет или местоположение.

Для смартфона, это его марка и модель, такие как iPhone 7 или Samsung Galaxy 5, могут быть сохранены в полях объекта.

2-я характеристика – это поведение объекта.

И объекты демонстрируют свое поведение с помощью методов.

Методы являются операциями, которые могут быть выполнены, чтобы изменить состояние объекта.

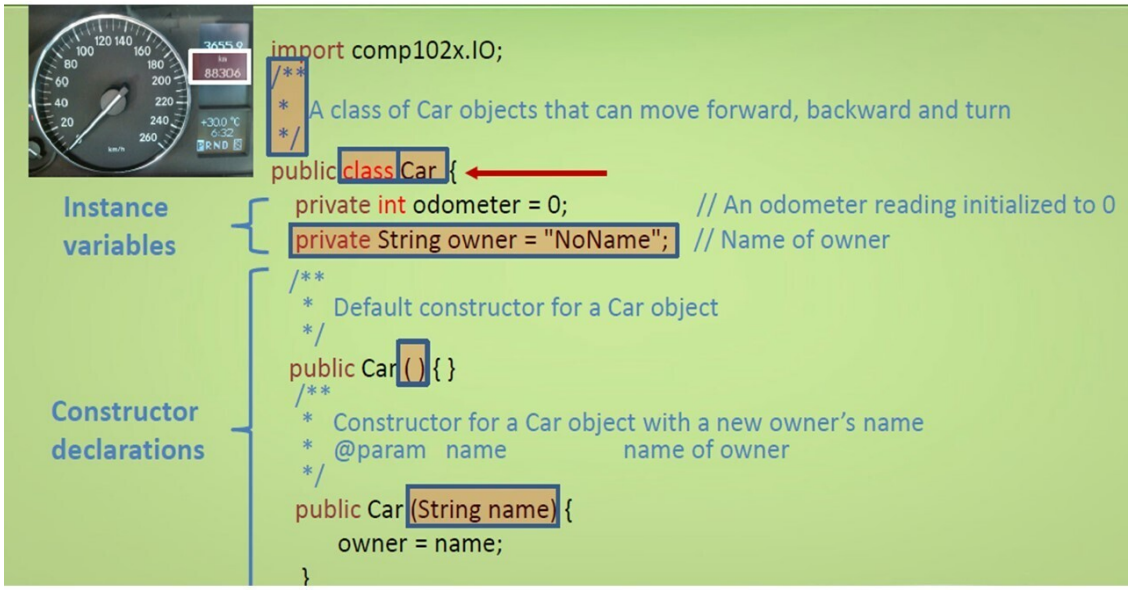
Например, чтобы изменить положение автомобиля нужно двигаться вперед или назад, или перевести смартфон в спящий режим, или увеличить громкость.

Прежде чем обсуждать в деталях, как определить поля и методы в Java, давайте сначала посмотрим на простой пример.

Пример

Перед погружением в детальную структуру объектно-ориентированного программирования, я сначала покажу вам пример, чтобы дать вам некоторое представление о том, как объектно-ориентированное программирование может быть использовано для моделирования внутренних состояний и поведения объектов.

Некоторые термины, которые я собираюсь использовать здесь, могут быть для вас в новинку, но не беспокойтесь, мы разберем все из них подробно.



В первой строке программы импортируется класс IO, который мы уже обсуждали.

Далее следует блок комментариев, который начинается с / ** и заканчивается * /.

Это является форматом Javadoc.

Это дает краткое описание того, что программа должна делать.

И эта программа определяет класс объектов автомобиль, который может двигаться вперед, назад и поворачивать.

Есть также комментарии в других различных разделах программы.

Я вернусь к документированию программы позже.

Первая строка после комментария является фактическим объявлением класса с именем Car (автомобиль) с помощью ключевого слова class.

Вы видели ключевое слово class раньше, потому что все программы – это Java классы.

Основная часть определения класса начинается с открытой фигурной скобки, и есть также закрытая фигурная скобка в конце программы.

Определение класса начинается с объявления переменных.

Они называются переменными экземпляра.

И эти переменные экземпляра могут быть использованы для моделирования внутренних состояний или атрибутов объектов в этом классе.

Я вернусь к различным типам переменных позже.

Первой объявляется переменная одометр целого типа, и она устанавливается в нуль.

Как вы знаете, одометр является инструментом, который записывает общее расстояние, пройденное транспортным средством.

Второе объявление определяет переменную с именем владельцем типа String.

String на самом деле класс, определенный в Java.

Он представляет строку символов или последовательность символов.

Мы будем использовать много строк в программах Java, и я буду говорить детально о строках позже.

Это объявление также инициализирует владельца как символьную строку "NoName" с NoName, заключенным в паре двойных кавычек. Обратите внимание, что оба объявления начинаются с ключевого слова private.

private является модификатором доступа.

Я вернусь к этому позже.

После объявление переменных экземпляра следует определение конструкторов.

Конструкторы используются для создания новых объектов в классе.

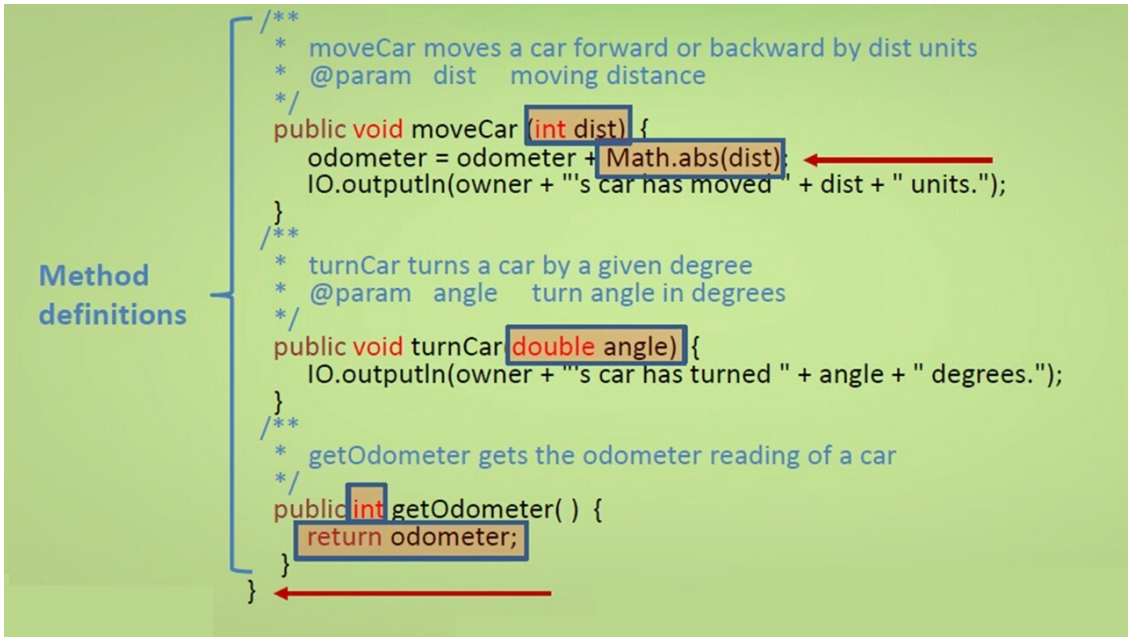
Здесь определены два конструктора.

Первый без параметров внутри пары скобок.

В то время как 2-й конструктор имеет один параметр с типом String.

В принципе, все, что 2-й конструктор делает, это присваивает переменной владельца экземпляра новое имя, данное параметром, в то время как 1-й конструктор будет оставлять имя по умолчанию "NoName" без изменений.

Остальная часть программы является определением методов.



Методы моделируют определенное поведение автомобиля.

Здесь определяются три метода.

Первый метод moveCar перемещает автомобиль на определенное расстояние, и расстояние указано здесь параметром dist целого типа.

Обратите внимание, комментарий говорит, что метод moveCar может двигать автомобиль вперед или назад.

Значение dist будет положительным, если автомобиль двигается вперед, и отрицательным, если автомобиль двигается назад.

Так как автомобиль переехал на некоторое расстояние, показания одометра должны быть обновлены путем добавления пройденного расстояния.

Но тут нельзя просто добавить значение dist.

Подумайте о том, что произойдет, если автомобиль движется назад или расстояние отрицательно?

Добавление отрицательного значения в одометр уменьшит показания одометра.

Но вы знаете, отмотка одометра транспортного средства, является незаконной.

Таким образом, вместо того чтобы просто добавить расстояние, расстояние должно быть заменено абсолютной величиной dist, так что общее пройденное расстояние будет увеличено независимо от того, перемещается автомобиль вперед или назад.

Можно вычислить абсолютное значение с помощью метода abs библиотеки Math.

Метод затем просто печатает сообщение о действии, которое метод, как предполагается, выполняет.

Следующий метод turnCar будет осуществлять действие поворота автомобиля, и угол поворота задается в виде параметра angle, но в этом случае типа double.

Опять же, этот метод просто печатает сообщение, без фактической реализации поворота.

Последний метод getOdometer используется для получения показаний одометра.

Заметьте, что метод getOdometer возвращает тип int и выражение начинается с ключевого слова return внутри метода.

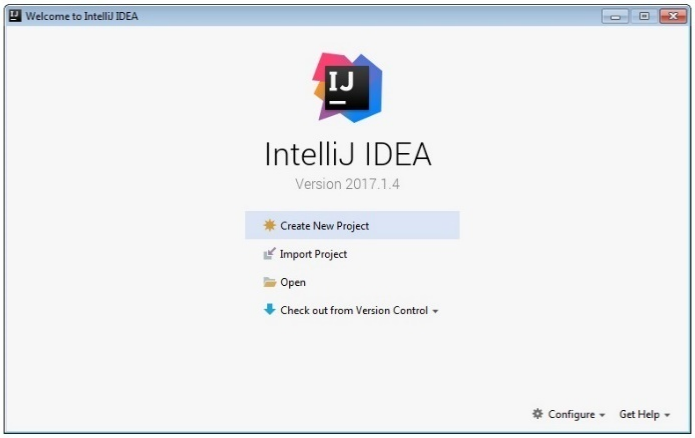
Я буду обсуждать метод, который возвращает значение, детально позже.

Программа заканчивается закрывающей фигурной скобкой.

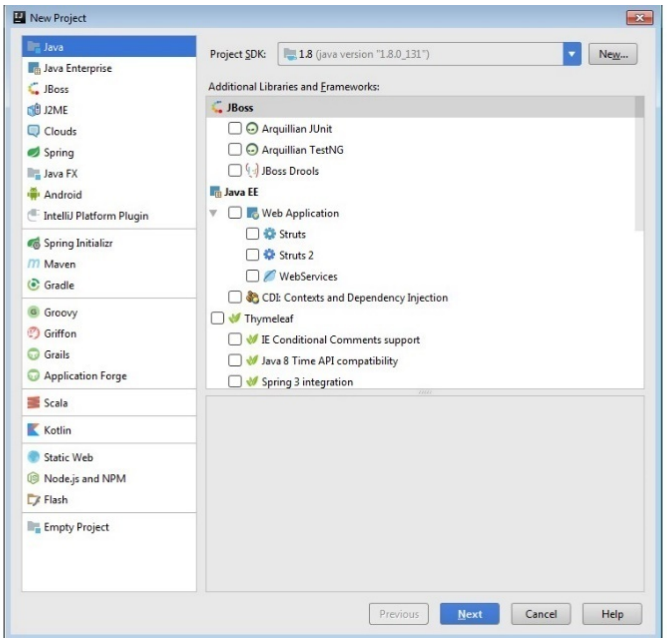
Теперь давайте вернемся к среде IntelliJ IDEA, чтобы увидеть, как этот класс автомобилей работает.

Демонстрация примера

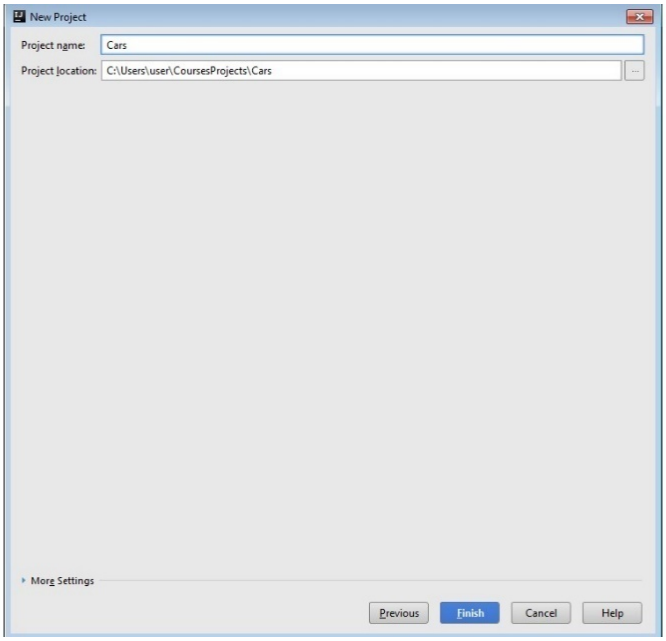
Давайте откроем IntelliJ IDEA.



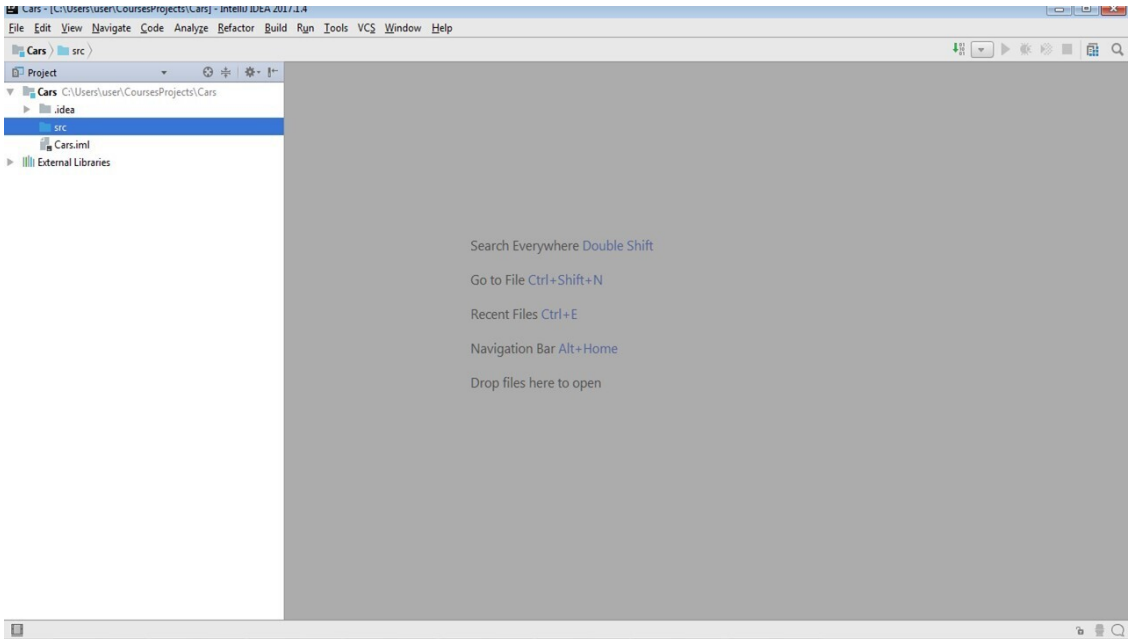
Вы можете создать новый проект, выбрав "Create New Project".



Выберем создание Java проекта.

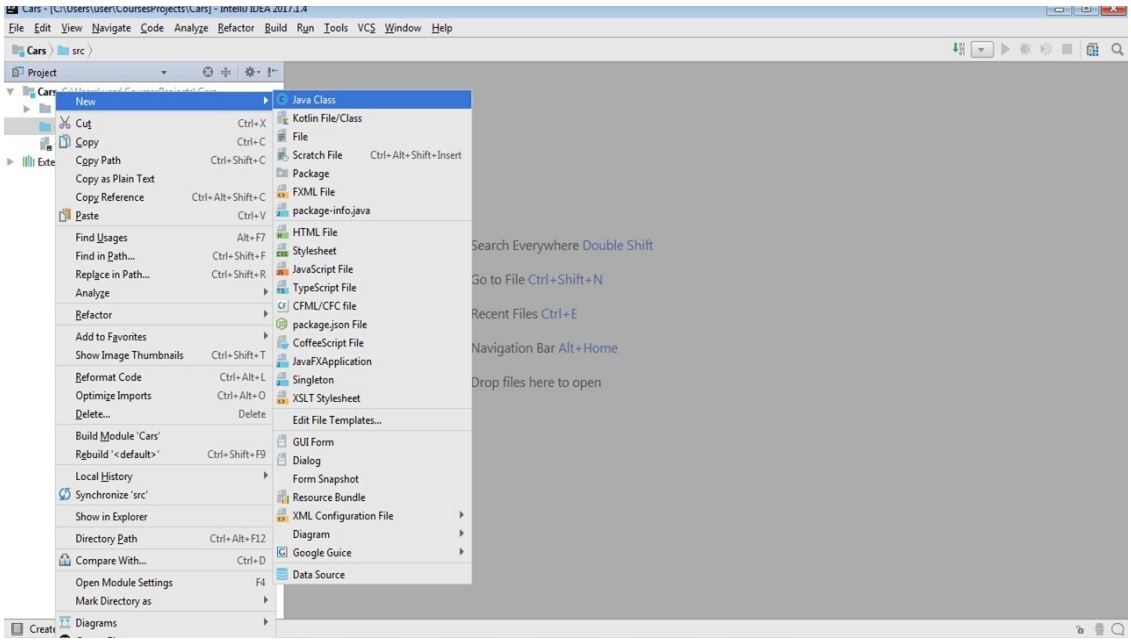


Вы можете выбрать место, где вы хотите сохранить ваш проект, а затем дать ему имя.
Давайте просто назовем проект «Cars».
И нажмем кнопку "Finish".

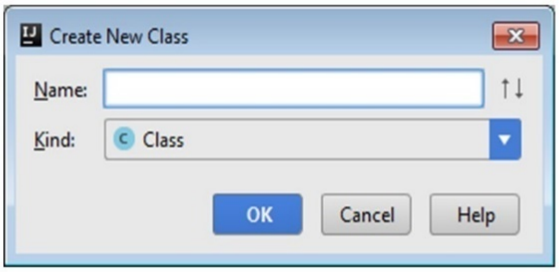


Появится окно нового проекта, который не содержит пока никакого кода.

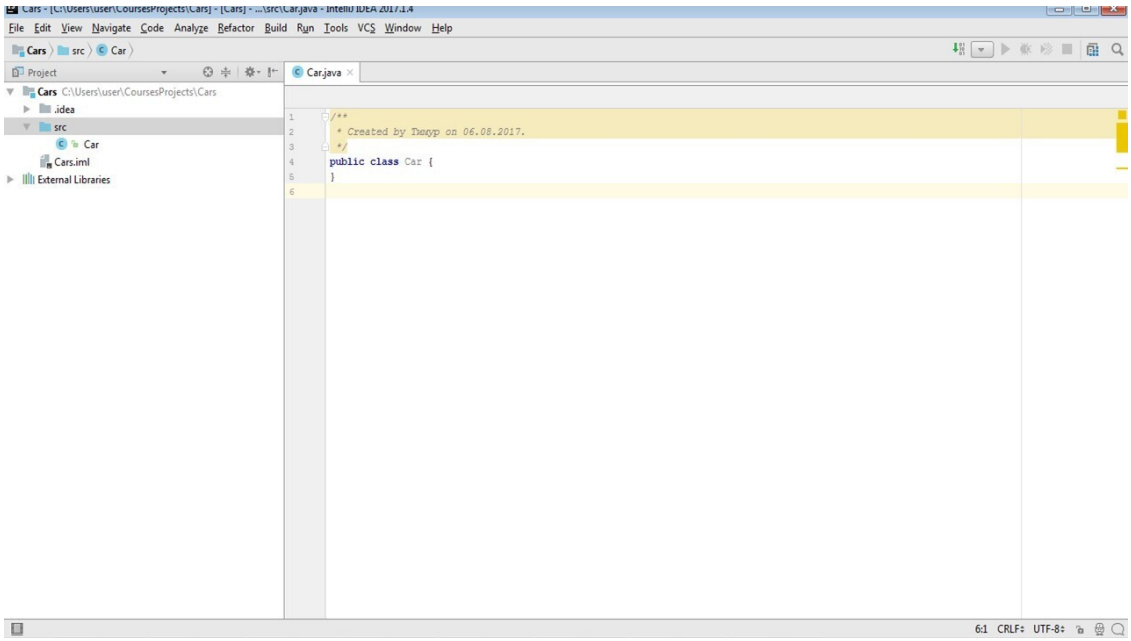
Мы можем начать писать нашу программу, создав новый класс, так как все программы Java это классы.



После нажатия "New Java Class", появится диалоговое окно.



И вы должны ввести имя класса, который вы создаете.
В нашем случае, Car является именем класса, с которым мы хотим работать.



Вы увидите значок Car, созданный в окне проекта.

Дважды щелкните по нему, и класс откроется в окне редактора.

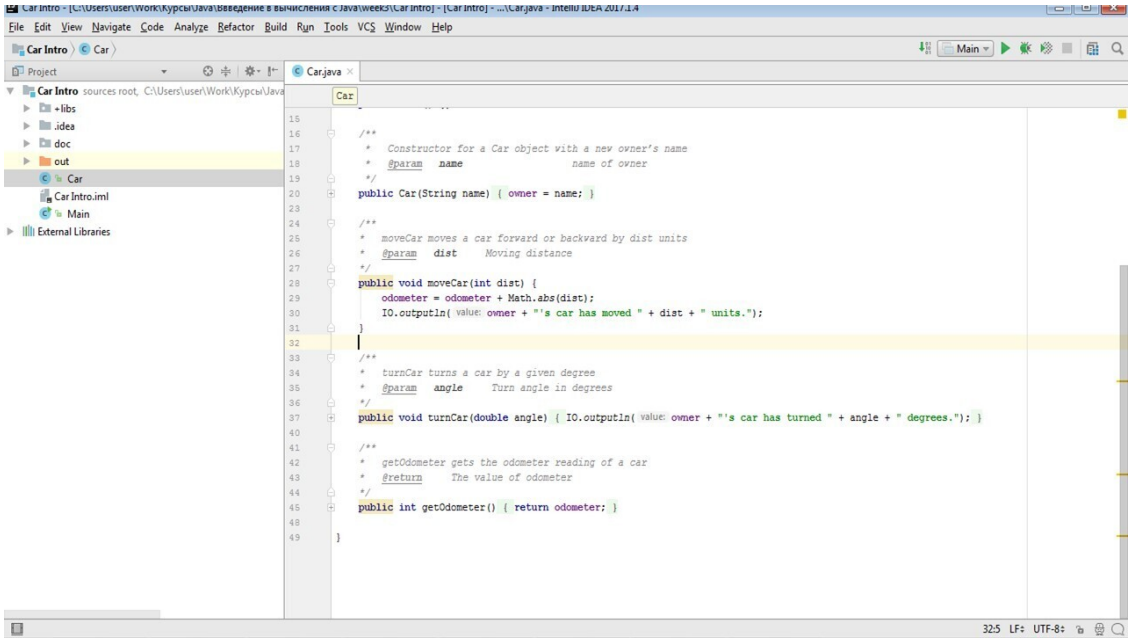
Вы можете увидеть, что здесь для вас уже создан шаблон, который начинается с документации, а затем идет заголовок класса, вместе с телом определения класса, заключенным в фигурные скобки.

Это на самом деле уже готовый класс, который может быть скомпилирован без ошибок, хотя он не может делать ничего полезного.

Я вернусь к этому шаблону, чтобы обсудить общую структуру программ Java.

На данный момент, давайте просто заменим тело определения класса примером автомобиля, что мы только что обсуждали.

Кроме того, вы можете просто открыть проект, который подготовлен для вас, выбрав "Open", а затем открыть готовый проект.

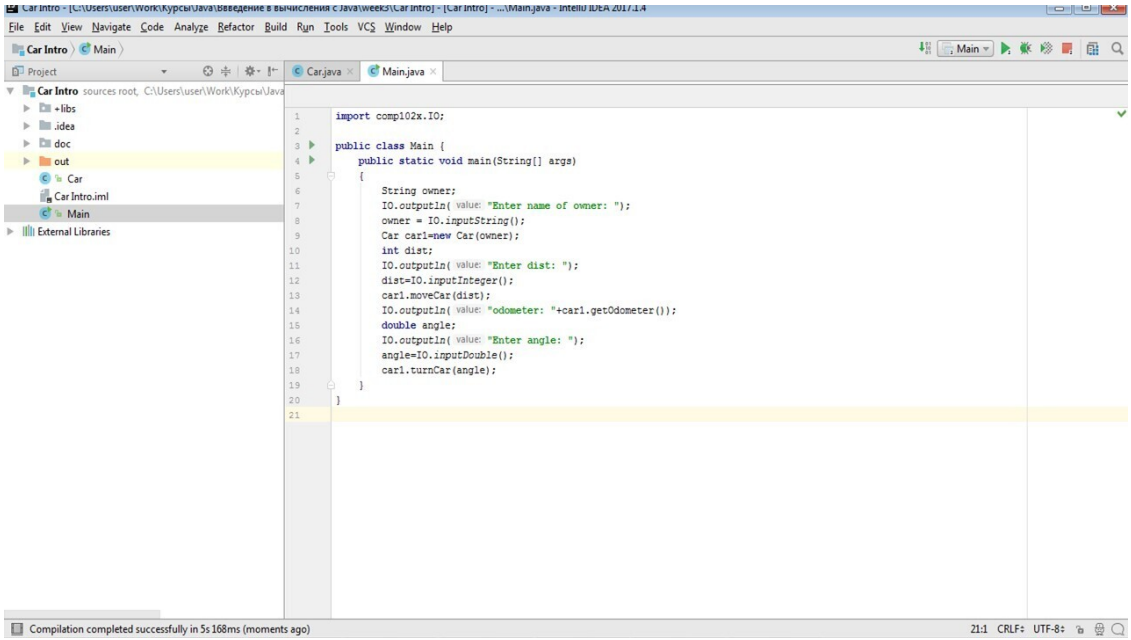


Вы можете дважды нажать на значок Car, чтобы увидеть исходный код.

Опять же, это та же программа, которую мы обсудили, но как мы можем запустить эту программу?

Помните, что метод main является точкой входа в программу Java, но мы еще не написали метод main.

Поэтому создадим еще один класс Main с методом main, в который добавим код ввода значений полей класса и вызова его методов.



Здесь мы сначала запрашиваем ввод имени владельца автомобиля, и затем создаем экземпляр класса.

Запрашиваем ввод дистанции, на которую нужно переместить автомобиль, и вызываем метод класса, перемещающий автомобиль.

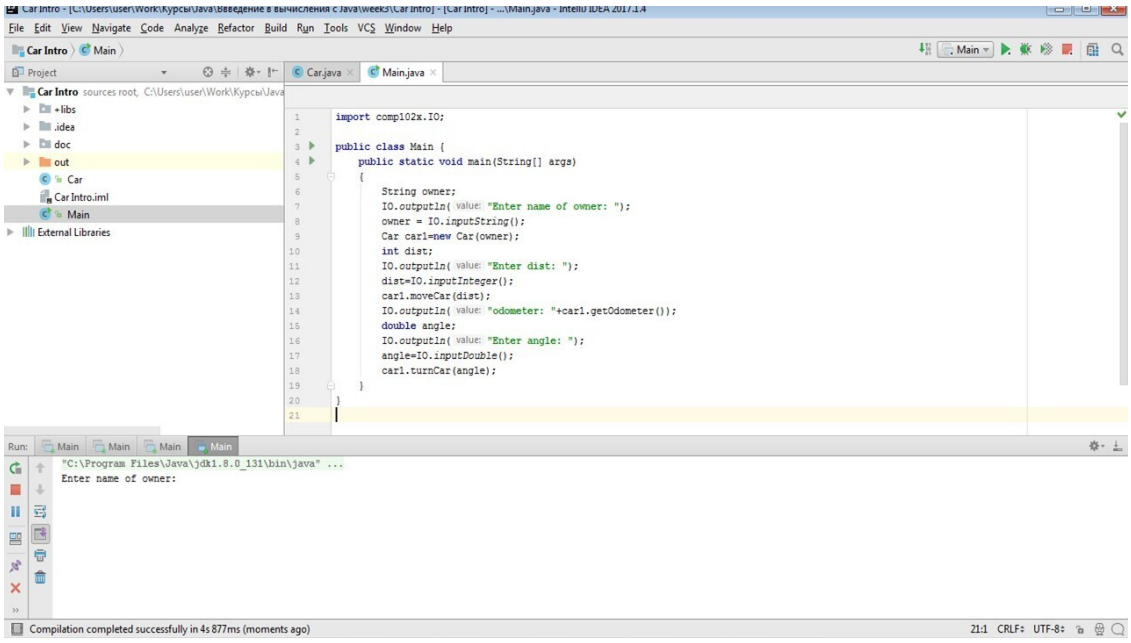
Выводим показание одометра, и запрашиваем ввод угла, на который нужно переместить автомобиль.

Вызываем метод перемещения автомобиля.

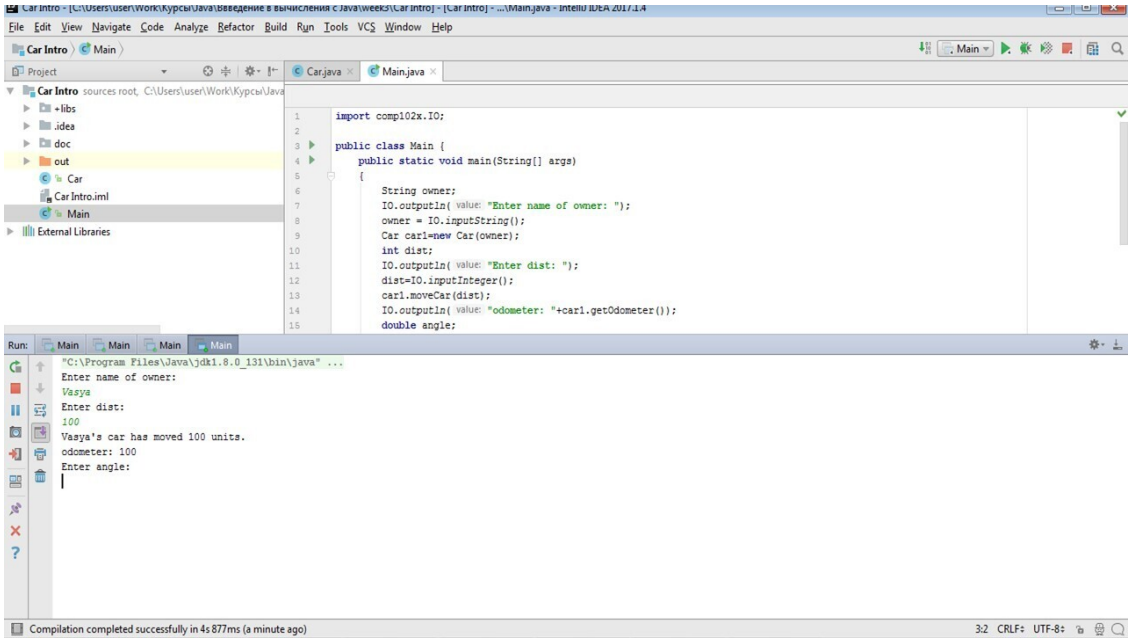
Давайте сначала попытаемся скомпилировать программу.

Хорошо, что здесь нет синтаксической ошибки.

И нажмем кнопку Run.

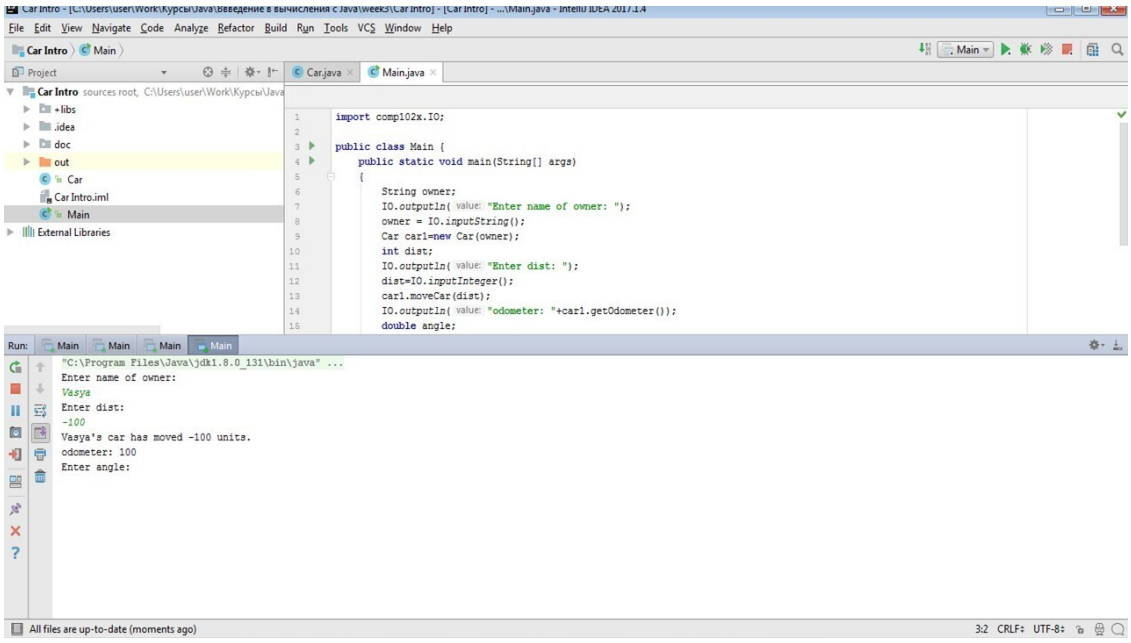


Появится запрос на ввод имени владельца.
Введем имя, затем введем дистанцию.

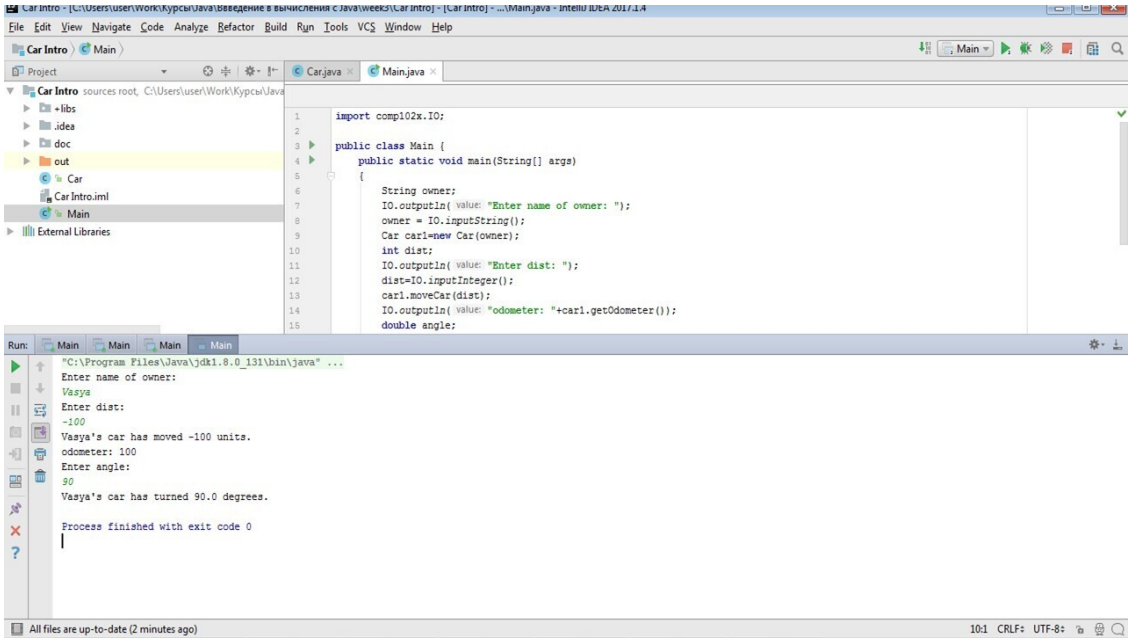


В результате появится вывод, что этот автомобиль проехал введенную дистанцию и мы увидим показание одометра.

Запустим программу заново и введем теперь расстояние «– 100».



Показание одометра все равно останутся 100.
Введем угол поворота и повернем автомобиль.



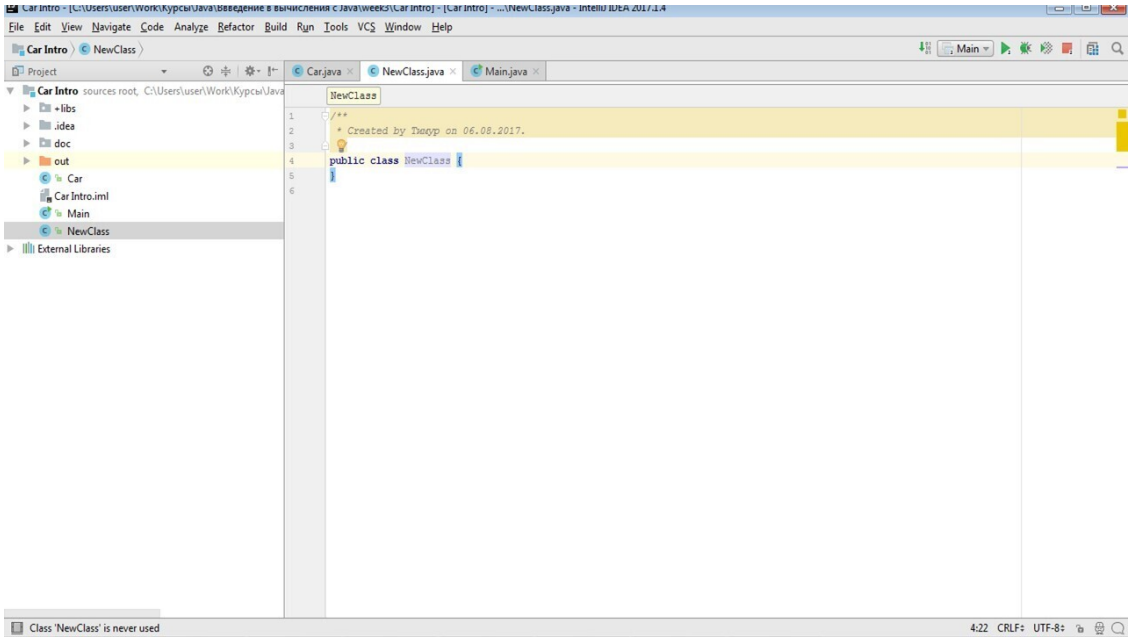
Конструктор

Теперь, когда вы увидели пример по определению класса для автомобилей, давайте вернемся и посмотрим на общую структуру объявления класса.

При попытке создать новый класс в IntelliJ IDEA, вы увидите всплывающее окно, которое запрашивает имя класса.

Скажем, мы хотим назвать класс, который будет создан как NewClass.

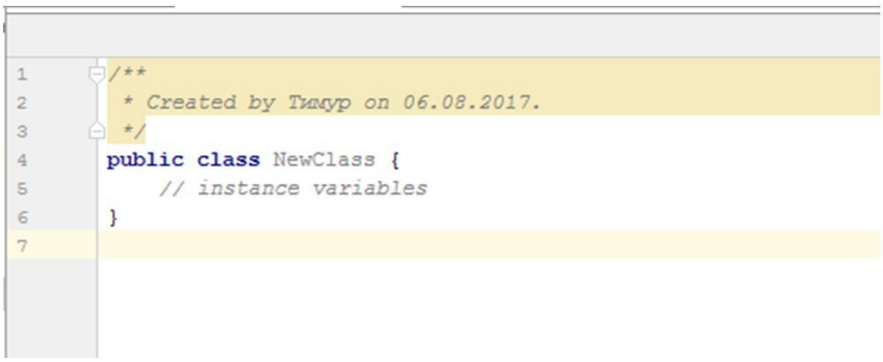
После того как вы обеспечили имя класса, вы увидите шаблон для создания нового класса.



Давайте рассмотрим каждый компонент объявления класса более тщательно.

Программа начинается с некоторых комментариев, которые начинаются с косой черты, а затем двумя звездочками и заканчиваются звездочкой и косой чертой.

Все между этими разделителями будет рассматриваться как комментарии и будет игнорироваться компилятором.



Это специальный формат размещения комментария для создания особого рода документа под названием JavaDoc.

Вы также можете использовать еще один вид комментариев, который начинается с двойных косых черт.

Это еще один способ размещения комментария.

Я вернусь к теме комментариев позже, когда будем обсуждать документирование программы более подробно.

После комментариев, следующая строка начинается с ключевого слова public, а затем после ключевого слова class следует имя класса, который вы создаете.

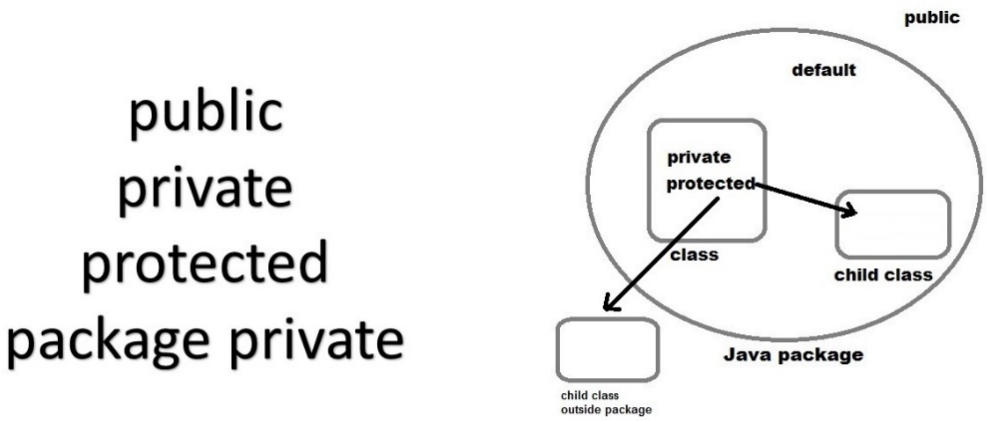
В качестве именования классов используется верхний CamelCase.

Ключевое слово public является идентификатором доступа.

В примере Car, у нас был public class Car, чтобы начать определение класса.

И Java определяет четыре уровня идентификаторов доступа.

Их также называют модификаторы доступа.



Это public и private, а также protected и package private.

Для классов, методов, данных или полей, объявленных как public, они могут быть видны для любого класса в любом пакете.

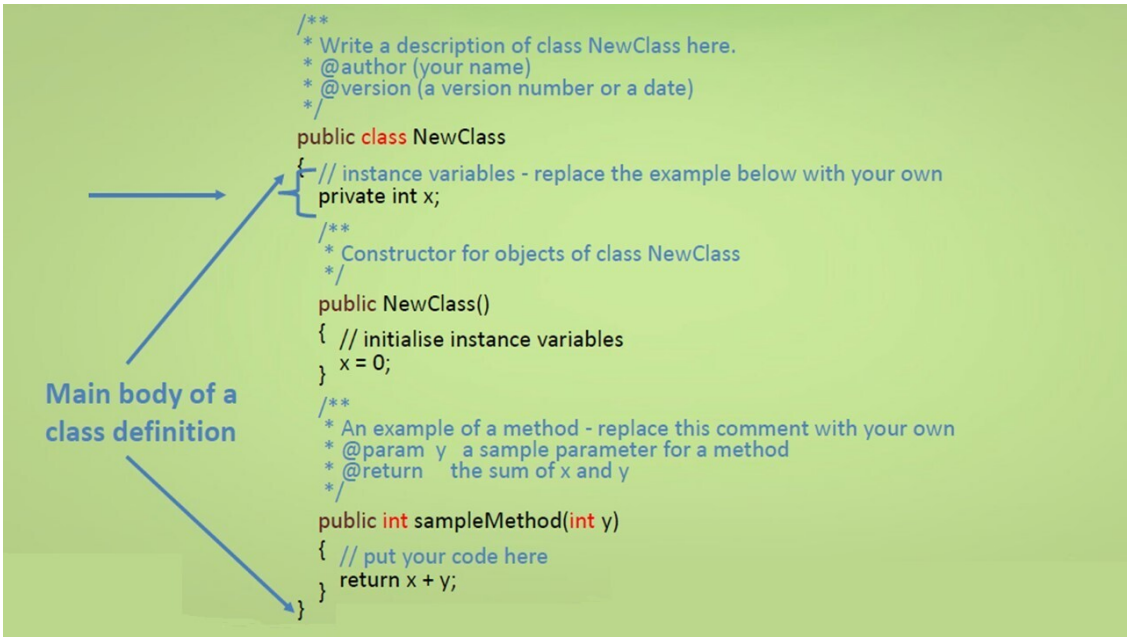
Пакеты представляют собой группы связанных классов.

Мы вернемся к этому позже.

Идентификатор private указывает, что методы и данные или поля, объявленные в классе, могут быть доступны только в пределах одного класса.

Классы не могут быть объявлены как private, за исключением случая вложенных классов.

Основная часть определения класса заключена в фигурные скобки.



В основной части первый блок предназначен для определения переменных экземпляра, то есть, эти переменные будут связаны с конкретным экземпляром объекта в NewClass.

В этом примере, переменная x с типом int будет создана при создании нового объекта в NewClass и переменная будет доступна только в NewClass, потому она объявлена как private.

Чуть позже вы увидите, что там также могут быть переменные класса или статические переменные, которые также могут быть здесь объявлены.

Далее блок программы предназначен для определения конструкторов.

Конструкторы обеспечивают экземпляры объекта, которые будет созданы из шаблона класса.

Конструкторы объявляются аналогично тому, как объявляются методы, но имя класса используется в качестве названия конструктора, в этом примере, имя класса NewClass используется в качестве имени конструктора.

Конструкторы используются для инициализации свойства или поля объекта.

Конструктор вызывается один раз и только один раз во время создания объекта с помощью оператора new и не имеет возвращаемого типа.

Я буду говорить больше о возвращаемом типа позже.

Так что вот общий синтаксис для объявления конструктора.

Он начинается с public, а затем идет имя класса и далее следует список параметров в круглых скобках или пустой список, если нет параметров.

Так, конструктор может не иметь параметров, может иметь один или несколько параметров.

В Java могут быть конструкторы с различным числом параметров на основе числа параметров и их типов.

Вы не можете написать два конструктора, которые имеют одинаковое количество и типы параметров для одного и того же класса, потому что Java не сможет отличить их друг от друга.

Рассмотрим конструктор для NewClass.

Тело конструктора окружено фигурными скобками.

Этот конструктор не имеет параметров и просто инициализирует переменную экземпляра x=0.

В программе Car были объявлены два конструктора для одного и того же класса.

Первый без параметров и ничего не делающий конструктор.

Второй принимал один параметр типа String, и изменял значение переменной владельца экземпляра, как было указано в параметре.

Вопросы

Задача

```
import comp102x.IO;

public class Week3Quiz
{
   private double x;

   public Week3Quiz(int i, double d) {
       x = i + d;
   }
   public Week3Quiz(double d, int i) {
       x = i - d;
   }
   public static void main(String[] args) {

       Week3Quiz q1 = new Week3Quiz(10.0, 10);
       Week3Quiz q2 = new Week3Quiz(10, 10.0);

       IO.outputln(q1.x); // statement 1
       IO.outputln(q2.x); // statement 2
   }
}
```

Укажите вывод выражения 1 и 2.

Ответ: 0.0 и 20.0

Когда у нас есть несколько конструкторов, Java будет пытаться решить, какой конструктор вызвать путем проверки списка параметров конструктора в вызывающем выражении.

Первое выражение используется для создания q1.

Первый параметр типа double, а второй параметр имеет тип int.

Таким образом, вызывается третий конструктор и x инициализируется как i – d, что эквивалентно 0,0.

Второе выражение используется для создания q2.

Первый параметр имеет тип int, а второй параметр имеет тип double.

Таким образом, вызывается второй конструктор и x инициализируется как i + d, что эквивалентно 20,0.

Задача

Вы только что узнали, что при наличии нескольких конструкторов, Java будет пытаться решить, какой конструктор вызывать путем проверки списка параметров конструктора.

Рассмотрим определение класса:

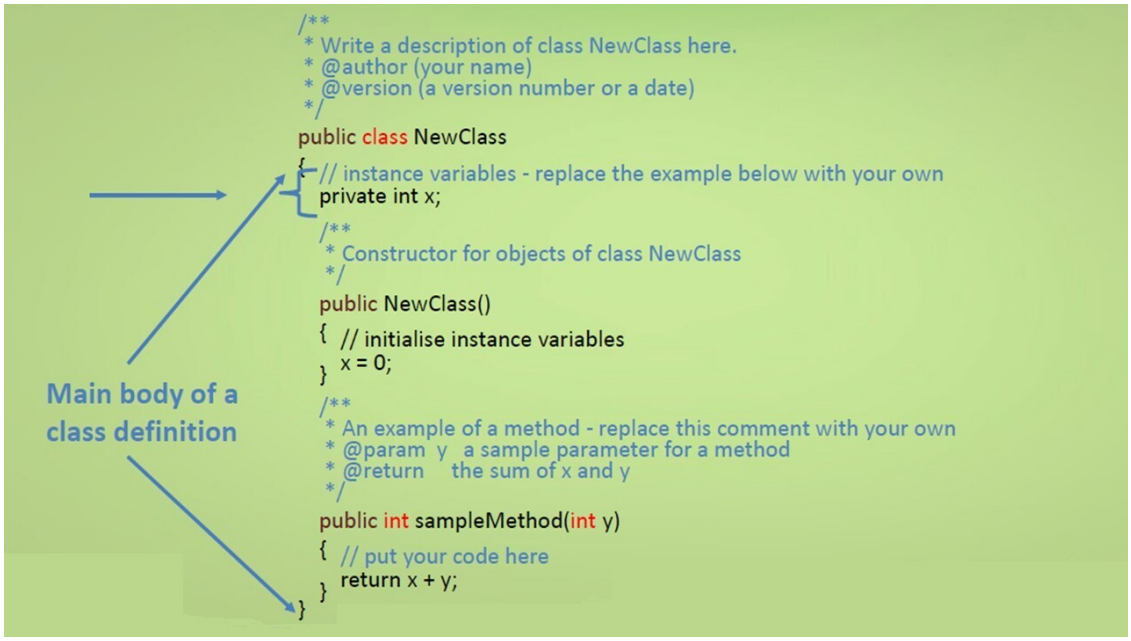
```
import comp102x.IO;

public class Week3Quiz
{
   private double x;

   public Week3Quiz(int i, double d) {
       x = i + d;
   }
   public Week3Quiz(double d, int i) {
       x = i - d;
   }
   public static void main(String[] args) {

       Week3Quiz q3 = new Week3Quiz(10.0, 10.0);
       Week3Quiz q4 = new Week3Quiz(10, 10);

       IO.outputln(q3.x);
       IO.outputln(q4.x);
   }
}
```

Что вы думаете произойдет с вышеприведенным кодом, если параметры вызываемого конструктора не будут соответствовать объявлениям конструкторов.

Ответ: компилятор выдаст ошибку.

Методы

Последняя часть программы состоит в определении методов.



Методы реализуют определенное поведение и действия, которые объект может выполнять.

Каждое объявление метода имеет 5 основных компонентов.

Первым идет идентификатор доступа.

public метод может быть доступен для всех других классов, а доступ к private методу можно получить только в пределах одного класса.

Далее идет возвращаемый тип метода, который является типом данных значения, возвращаемого методом.

Например, метод может возвращать целочисленное значение, тогда возвращаемый тип может быть одним из четырех примитивных типов целых чисел, в том числе, byte, short, int и long.

Если метод возвращает число с плавающей точкой, возвращаемый тип может быть float или double.

Кроме того, возможно, что никакое значение не возвращается методом, в этом случае используется тип void.

Далее идет имя метода.

Любой допустимый идентификатор Java может быть использован в качестве имени метода.

По соглашению в качестве имени метода используется нижний CamelCase.

Далее идет список параметров.

Список параметров является списком пар тип-параметр, разделенных запятыми. И список параметров окружен скобками, даже если список параметров пуст.

И наконец, идет тело метода.

Тело метода окружено парой фигурных скобок. И тело метода часто делится на две основные части.

Первая часть, это декларация локальных переменных и вторая часть состоит из Java выражений, которые делают реальную работу.

В примере NewClass, определяется метод с именем sampleMethod.

Доступность метода является public. А тип значения, возвращаемого методом, является int.

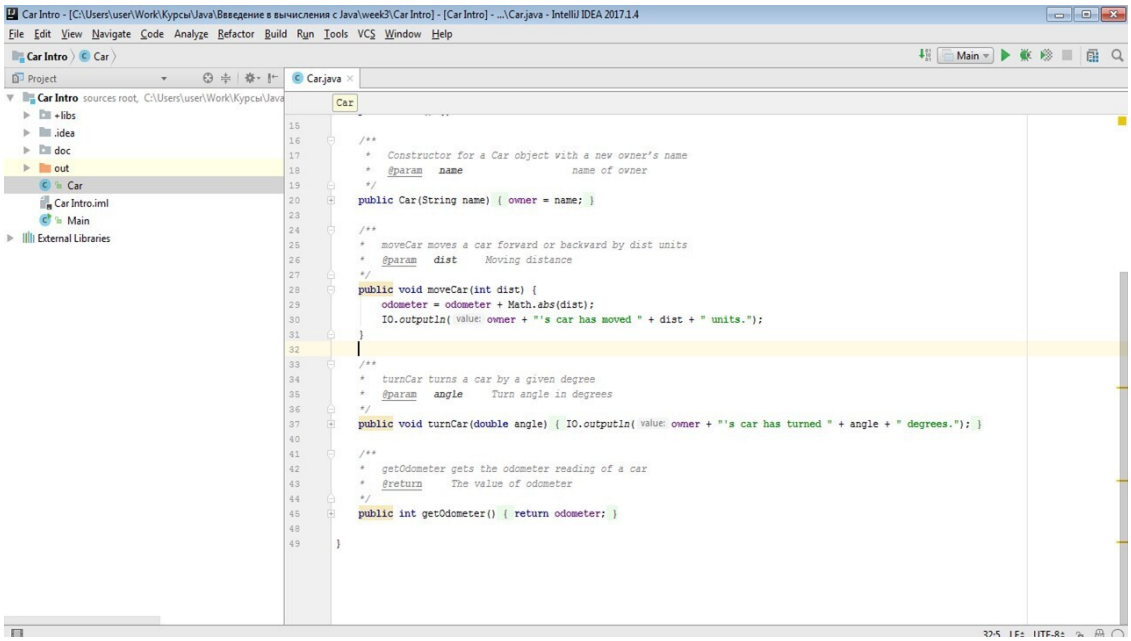
В списке параметров существует только один параметр с типом int.

Тело метода имеет только один оператор, return x + y, где x является переменной экземпляра, объявленной в начале, и y является параметром, передаваемым в метод.

Обратите внимание, что слово return здесь это ключевое слово Java.

Выражение «x + y» будет складывать две переменные и результат, который имеет тип int, будет возвращаемым значением метода sampleMethod.

Если мы вернемся к примеру с автомобилем, вы обнаружите, что объявление класса Car следует той же структуре.



Оно начинается с заголовка, который определяет название класса, в этом случае, Car.

Тело объявления Car начинается с объявления переменной экземпляра.

После объявления переменных следует декларация конструктора.

Конструкторы используются для создания новых экземпляров объектов и в этом случае есть два конструктора. Один без параметров, другой с одним параметром строкового типа.

После объявления конструкторов следуют объявления методов.

Здесь определены три метода.

Два из методов имеют тип void.

Это означает, что методом не возвращается значение.

Оба метода печатают сообщение в консоли о действиях, которые они призваны выполнять.

Метод moveCar также изменяет значение одной из переменных экземпляра, счетчик километража.

Третий метод getsOdometer возвращает значение одометра, который имеет int тип.

Метод, который возвращает значение закрытого поля, часто называют getter методом.

Вопросы

Обсуждение отладки.
Давайте попробуем определить все ошибки в следующем классе IronMan.

```
12
13  public class IronMan
14  {
15      private String id = "?";
16
17      public void IronMan(String newID) {
18          id = newID;
19      }
20
21      private String dance() {
22          IO.outputln("You must be joking +_+");
23      }
24
25      public cleanSlateProtocol() {
26          IO.outputln("3 ... 2 ... 1 ... BOOM!");
27      }
28
29      public setID(int givenId) {
30          id = givenId;
31      }
```

Ответ:

```
32
33      public void getName() {
34
35          // ("Mark " + id) conc
36          return "Mark " + id;
37      }
38
39      public static void main(St
40
41          // construct an IronMan
42          IronMan defaultIronMan
43          String name = defaultIr
44          IO.outputln("This is: "
45
46          // construct an IronMan
47          IronMan mark42 = new Ir
48          IO.outputln("This is: "
49          mark42.dance();
50          mark42.cleanSlateProtoc
51
52      }
53  }
```

Комментарии

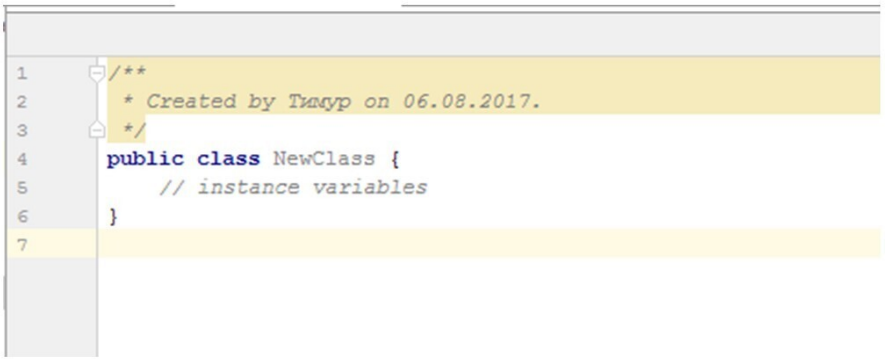
Прежде чем мы двинемся дальше, давайте поговорим о программной документации.

Как уже говорилось, документирование программы важно, потому что также как и компьютер, другие люди могут также читать вашу программу, которая представлена для широкой публики.

Кто бы ни оценивал вашу программу, он должен понимать, как работает ваша программа, даже если она не работает, чтобы быть в состоянии помочь вам в определении проблемы.

Документация поможет сопровождению программы и в случае, если программа может быть изменена в будущем.

Программная документация предоставляется в виде комментариев.



```
1  /**
2   * Created by Тимур on 06.08.2017.
3   */
4  public class NewClass {
5      // instance variables
6  }
7
```

Как правило, комментарии используются для описания следующих аспектов программы:

– Что программа делает и что требует, например, описание спецификации входных и выходных данных;

– Что представляют собой переменные, и использование значимых имен здесь также поможет;

– Что следует учитывать при использовании переменных или методов, например, описание диапазона допустимых входных данных и типов параметров, ожидаемых методом;

– Краткое описание логического потока программы;

– Некоторые сведения об авторе и самом коде, так что люди смогли с вами связаться, если обнаружат проблемы с кодом.

И есть три основных способа, чтобы расположить комментарии в коде, так, чтобы компьютер не рассматривал их как часть программного кода.

Комментарии, которые помещаются на одной строке, предваряются двумя косыми чертами (/ /).

```
String owner = "NoName";    // Name of owner

// Method to move the car forward
public void moveForward( )
{ IO.outputln(owner + "'s car is moving forward."); }
```

Все, что написано в строке за двойными косыми чертами будет рассматриваться как комментарии.

Строка комментариев обычно помещается на верху или на правой стороне программного выражения.

Блок комментариев может быть заключен между / * и * / в одну или несколько строк, и все между этими разделителями будет рассматриваться как комментарии.

```
/*
 * Compute final grade as the weighted some of exam scores, lab scores
 * and homework scores. Note that all scores should be within the range
 * of 0 and 100
*/
        examScore = examScore * (examWeight / 100.0);
        labScore = labScore * (labWeight / 100.0);
        hwScore = hwScore * (hwWeight / 100.0);
        finalGrade = examScore + labScore + hwScore;
```

Это обычно используется, когда подробное описание требуется для объяснения набора выражений программирования, содержащихся в блоке кода.

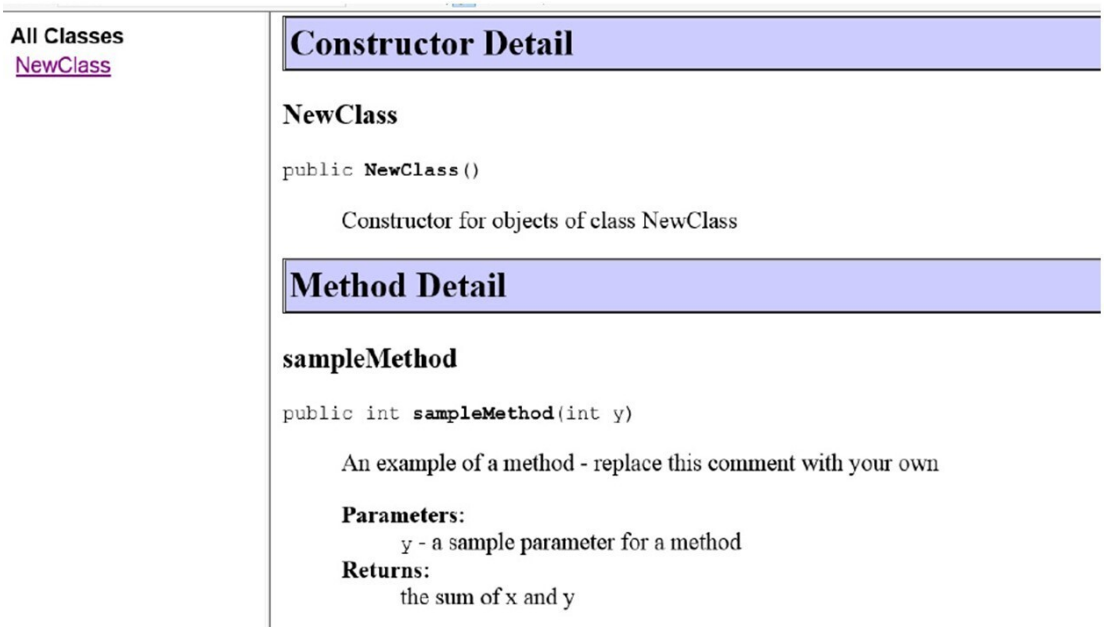
Javadoc является полезным инструментом для генерации документации автоматически из программ Java.

```
/**
 * Write a description of class NewClass here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NewClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class NewClass
     */
    public NewClass()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     *
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

Комментарии Javadoc написаны между двумя разделителями / **, заметьте, что здесь есть две звездочки, и * /.

Комментарии могут занимать несколько строк и все между двумя такими разделителями будет рассматриваться как комментарии.

Javadoc также предоставляет некоторые полезные описательные теги, два самые полезные из них включают @param для описания параметров, используемых в методе или конструкторе, и @return используется для описания возвращаемого методом значения.



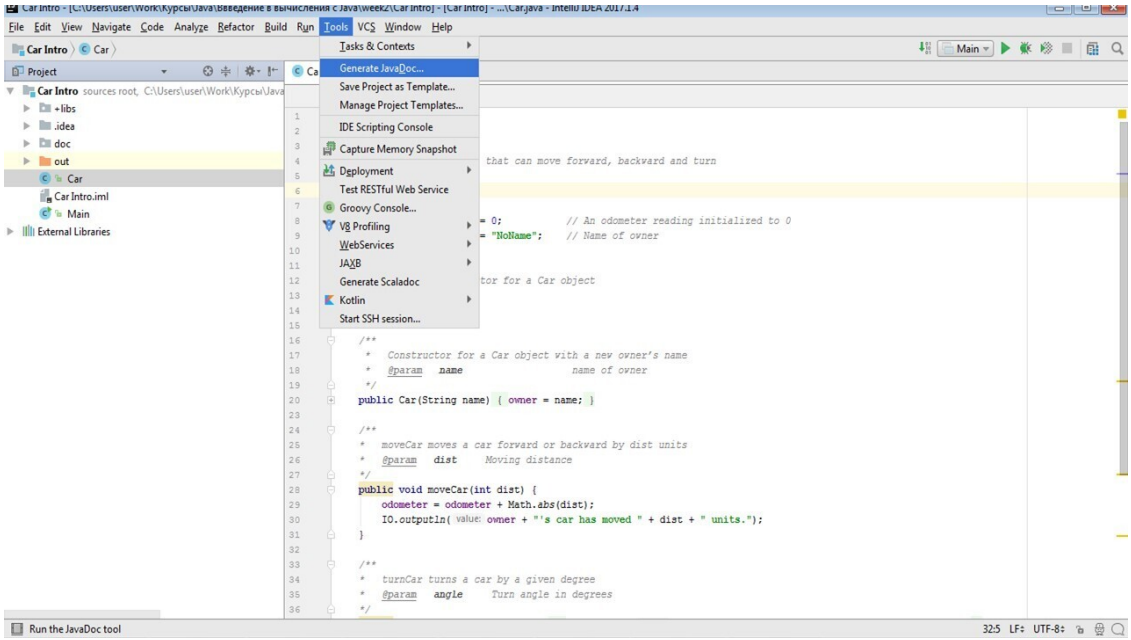
В результате работы инструмента Javadoc генерируется документ в формате HTML, который может быть отображен как веб-страница.

```
/**
 * Write a description of class NewClass here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NewClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class NewClass
     */
    public NewClass()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     *
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

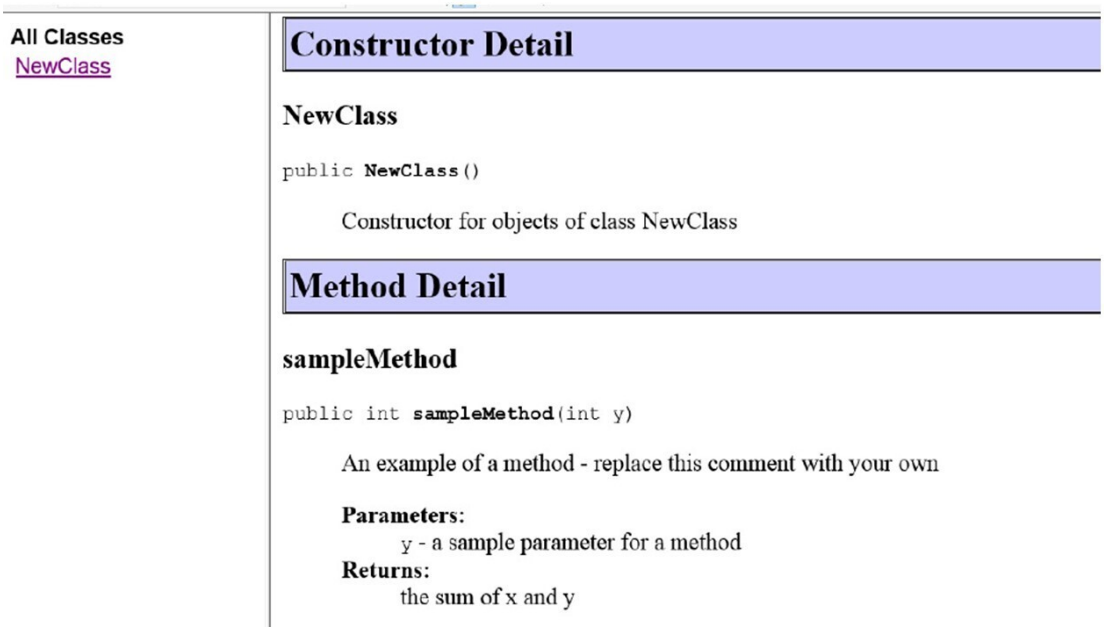
В NewClass, например, разделы комментариев синего цвета могут быть извлечены в Javadoc.

Это описательные теги.

Для метода, определенного здесь, дано описание для параметра y и значения, возвращаемого методом.



В IntelliJ IDEA сгенерировать Javadoc документацию для проекта можно соответствующей командой меню.



В общем, Javadoc начинается с общего описания и резюме конструкторов и методов, которые сопровождаются подробным описанием.

Например, вы можете посмотреть здесь подробное описание для метода sampleMethod, включая описание для параметров и значения, возвращенного методом.

И формат Javadoc считается отраслевым стандартом для документирования классов Java.

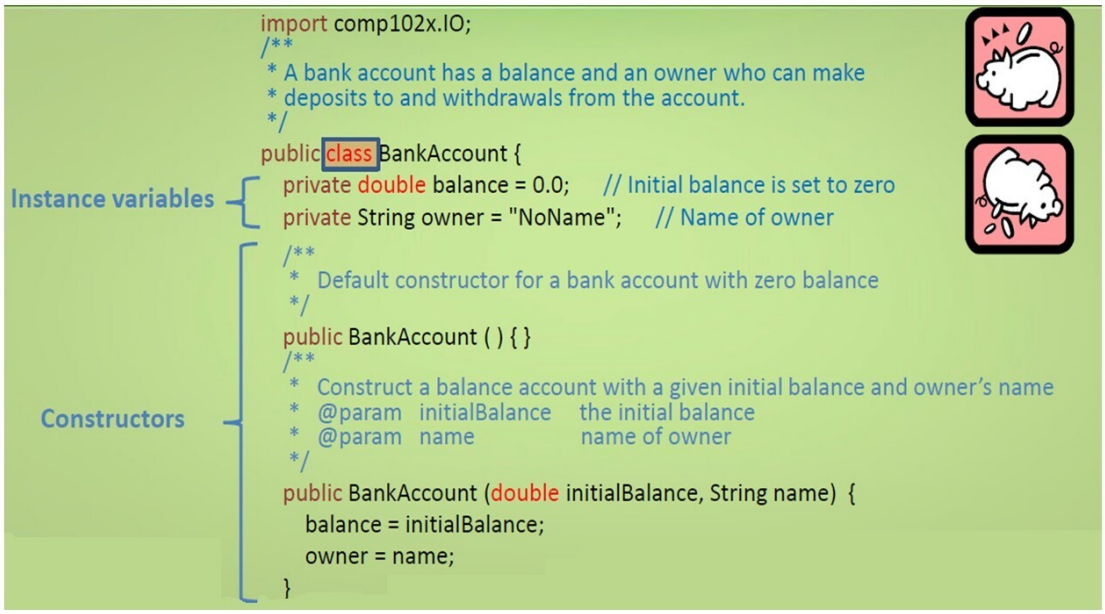
Вы увидите, что все популярные Java библиотеки, очень хорошо задокументированы. И вы должны усвоить привычку писать хорошо документированные программы.

Пример

Давайте посмотрим на другой пример, чтобы проиллюстрировать концепции объектно-ориентированного программирования, которые мы только что обсудили.

Эта программа определяет класс банковских счетов, которые имеют свой баланс и владельца, который может делать вклады и снимать средства со счета.

Даже если у вас нет счета в банке, идеи, описанные здесь, могут быть использованы для других финансовых инструментов, которые могут быть использованы для пополнения и снятия денег. Даже для копилки, хотя вынимать деньги из копилки может быть немного сложнее.



В первой строке программы импортируем класс IO, как мы делали в предыдущих программах, так что мы сможем сделать ввод и вывод в консоли.

Далее следует блок комментариев, используя формат Javadoc.

Вы увидите позже, что такие комментарии в различных разделах программы обеспечат генерацию Javadoc для класса BankAccount.

Первая строка после комментария, это объявление класса с именем BankAccount с помощью ключевого слова class.

Название BankAccount является идентификатором с верхним CamelCase. Фактическое определение BankAccount заключено в паре скобок и начинается с некоторых объявлений переменных экземпляра.

Эти переменные описывают некоторые важные свойства различных экземпляров банковских счетов.

В этом случае, здесь объявляются две переменные экземпляра.

Баланс является переменной doubld типа, для хранения текущего баланса банковского счета, который инициализируется с нулевым значением.

Вторая переменная экземпляра определяет владельца, имеет тип String и при инициализации получает значение NoName.

Обратите внимание, что NoName заключено в двойные кавычки, чтобы обозначить, что это строка символов. И вы далее увидите много символьных строк.

Отметим также, что оба объявления начинаются с модификатора доступа private.

Переменные экземпляра, как правило, объявляются как private, если не появится хороший повод, чтобы сделать их достоянием общественности.

Легко видеть, что в случае банковского счета, эти переменные баланса и информации о собственности не должны быть преданы гласности и стать доступными для всех.

После объявления переменной экземпляра идет определение конструкторов.

Здесь определены два конструктора.

Помните, что имя класса, в данном случае, BankAccount, используется в качестве имени конструкторов.

Первый конструктор по умолчанию предназначен для создания нового объекта банковского счета с нулевым балансом и NoName качестве владельца.

Обратите внимание, что этот конструктор не несет параметра внутри пары скобок.

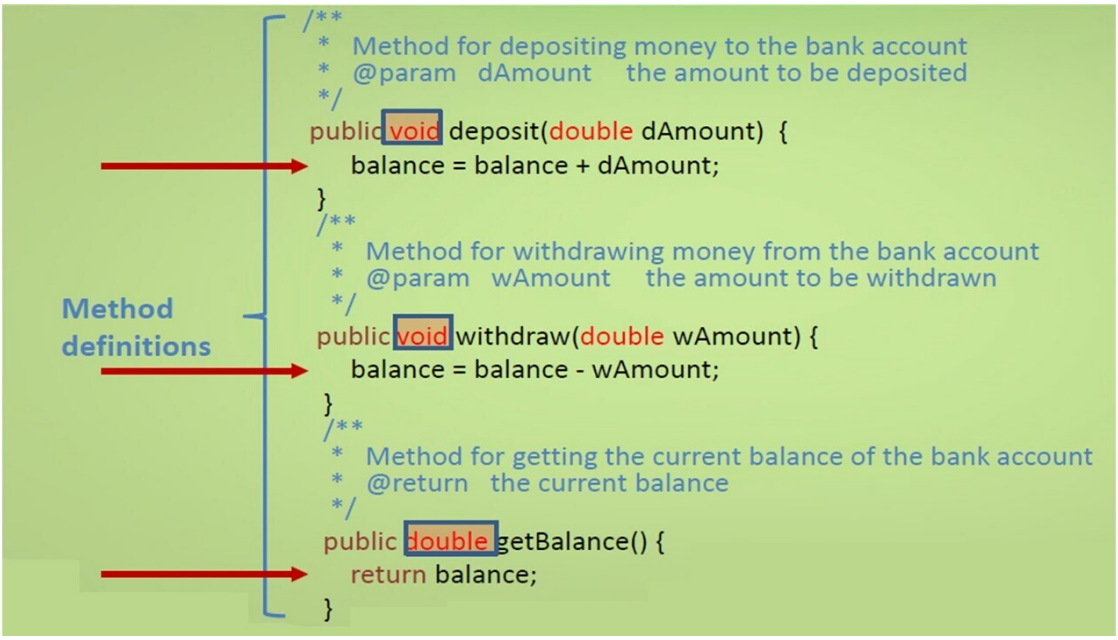
Второй конструктор принимает два параметра, первый initialbalance имеет double тип и второй name имеет тип String.

Этот второй конструктор изменяет баланс банковского счета на значение, определенное в intialbalance.

Это как когда новый счет открывается, начальная сумма зачисляется на счет.

Счету также дается имя владельца, как указано в втором параметре.

Остальная часть программы посвящена определению методов.



Здесь определяются три метода.

Первый это deposit, который используется для внесения денег на банковский счет.

Он принимает один параметр dAmount double типа, который представляет собой сумму для внесения на счет.

В выражении здесь dAmount добавляется к переменной экземпляра balance и результат снова присваивается balance.

Второй метод называется withdraw, который можно использовать для снятия денег с банковского счета.

Он принимает один параметр wAmount double типа, который представляет собой сумму для вывода со счета.

В выражении здесь wAmount вычитается из переменной экземпляра balance и результат снова присваивается balance.

Третий метод называется getBalance, с помощью которого получается значение текущего баланса банковского счета.

Этот метод не несет никакого параметра.

Обратите внимание, что он отличается от предыдущих двух методов.

Метод getBalance возвращает значение типа double, в то время как предыдущие два метода определены void типа, потому что они не возвращают никакого значения.

Значение баланса возвращается методом с помощью выражения с ключевым словом return.

Метод, как этот, который возвращает значение private поля, часто называют getter методом.

Предыдущие два метода, которые изменяют значение private поля, часто называют мутаторы или setter методы.

Использование getter и setter методов может помочь в обеспечении инкапсуляции данных и защитить переменные экземпляра от случайного изменения.

Теперь, когда мы закончили определение класса BankAccount, это как закончить писать рецепт. И чтобы на самом деле произвести что-то съедобное, мы все еще должны смешать ингредиенты и сделать реальную кулинарию.

Небольшие отклонения в смеси ингредиентов могут привести к разным результатам.

Например, вы можете использовать различное количество сахара или использовать маргарин вместо масла в выпечки печенья для производства различных печений или различных экземпляров объектов.

Давайте на самом деле попытаемся создать некоторые банковские счета с помощью класса BankAccount, который мы только что определили.

Мы можем создать различные банковские счета, используя различные конструкторы или предоставляя конструкторы с различными параметрами.

Это как использовать различные сочетания ингредиентов для вашего рецепта.

Мы напишем основной метод main для тестирования различных банковских счетов.

```
/**
 *  Main method for testing the bank account
 */
public static void main (String[ ] args) {
   BankAccount testAccount = new BankAccount( );
   testAccount.deposit(100);
   testAccount.withdraw(50);
   IO.outputln (testAccount.owner + "'s account has a balance of $"
         + testAccount.balance);

   BankAccount myAccount = new BankAccount(100, "TC");
   myAccount.deposit(100);
   myAccount.withdraw(50);
   IO.outputln (myAccount.owner + "'s account has a balance of $"
         + myAccount.balance);
  }
}
```

Метод main является точкой входа в программу Java и это выражение такое же, как те, которые мы уже видели.

Чтобы создать новый счет в банке, вы должны сначала дать банковскому счету имя идентификатора.

В этом случае, мы назовем это счет testAccount.

Это похоже на объявление переменной примитивного типа, как int или double.

В этом случае, testAccount объявлен как переменная типа BankAccount, но не инициализирована каким-либо значением.

Чтобы действительно создать новый банковский счет, используется ключевое слово new перед конструктором.

Здесь мы используем конструктор по умолчанию без параметров.

Этот конструктор создаст банковский счет с нулевым балансом и NoName в качестве имени владельца.

На самом деле, вы можете сделать как объявление, так и инициализацию в одном выражении. И результат будет тот же самый.

После создания счета, можно сделать депозит на счет, вызвав метод deposit.

Так как может быть несколько счетов, то для того, чтобы указать, что депозит должен быть произведен для данного конкретного счета testAccount, оператор точка используется для указания, что метод deposit должен быть использован для объекта testAccount.

В терминологии объектно-ориентированного программирования это называется передачей сообщения.

Мы также должны указать размер вносимой суммы, в данном случае, 100 долларов.

Мы можем выполнить снятие со счета с помощью метода withdraw, снова с помощью оператора точки, чтобы указать, что это для объекта testAccount, и указать количество выводимых денег, например, 50 долларов.

Здесь используется выражение IO.outputln, чтобы распечатать итоговый баланс после двух операций.

Существуют три части сообщения, которые должны быть распечатаны.

Первая из них, это имя владельца для testAccount, снова с помощью оператора точка между testAccount и переменной owner экземпляра.

Вторая часть, это строка символов, заключенная в двойные кавычки.

И третья часть, это баланс для testAccount.

Обратите внимание, переменная экземпляра balance имеет тип double, так что она будет преобразована в строку символов, прежде чем распечататься.

Два знака + плюс между тремя компонентами будут объединять три символьные строки вместе.

И я хочу создать другой счет для себя. Давайте назовем его myAccount.

Опять же, ключевое слово new перед конструктором используется для создания нового экземпляра BankAccount.

Обратите внимание, что, хотя имя конструктора такое же, как и раньше, здесь есть два параметра в вызове этого конструктора.

Компилятор сможет понять, что мы используем второй конструктор, исследуя количество и типы параметров.

В этом случае первоначальный баланс установлен как 100 и имя владельца установлено как "TC", то есть мое имя.

Обратите внимание, что TC ставится внутри пары двойных кавычек, потому что второй параметр ожидает строку символов.

Подобно тому как и раньше, вносится депозит в размере 100 долларов, но на этот раз на myAccount вместо testAccount.

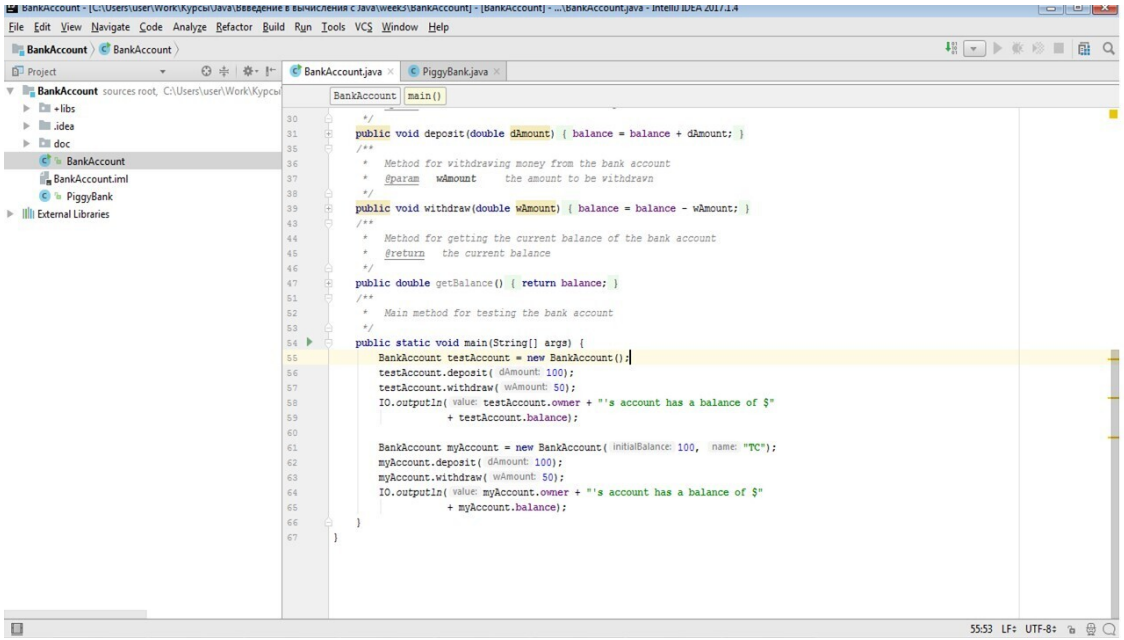
Вывод 50 долларов также делается с myAccount.

В результате баланс для myAccount затем выводится с помощью IO.outputln. Так что это конец всей программы.

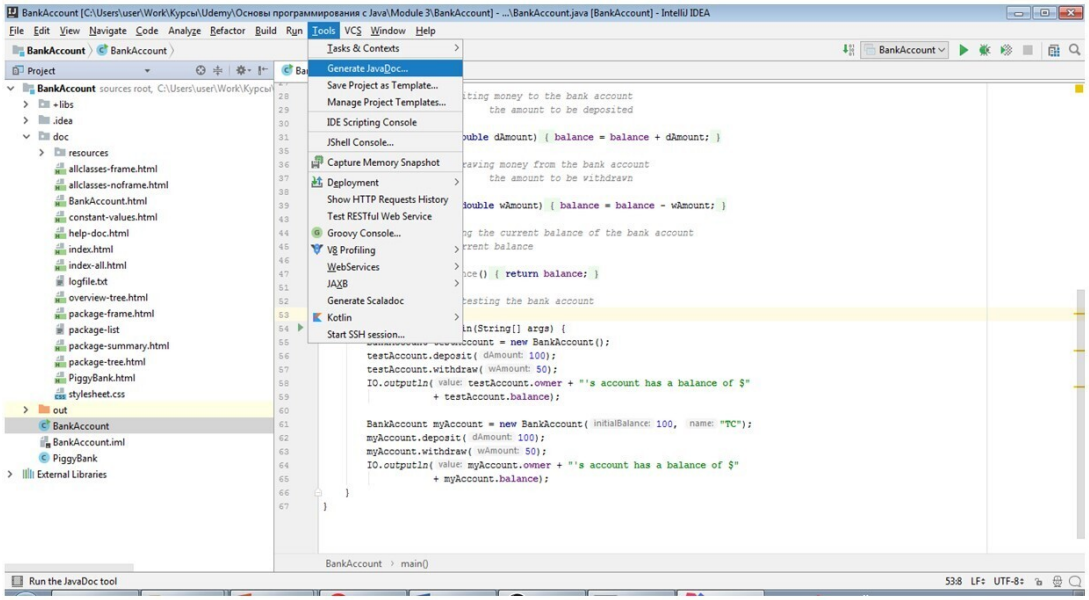
Давайте посмотрим на фактическое выполнение программы.

Демонстрация примера

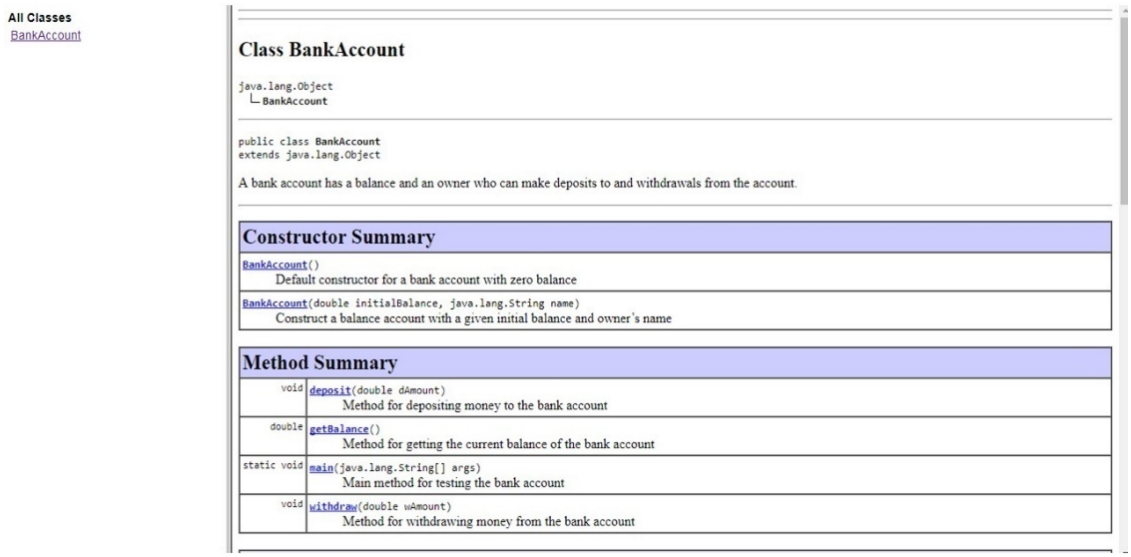
Проект банковский счет уже открыт в среде IntelliJ IDEA.



Сгенерируем Javadoc из класса BankAccount.



Это Javadoc-документ, который генерируется из комментариев, которые мы включили в программу.



All Classes

BankAccount

Class BankAccount

java.lang.Object
└ BankAccount

public class BankAccount
extends java.lang.Object

A bank account has a balance and an owner who can make deposits to and withdrawals from the account.

Constructor Summary

BankAccount()	Default constructor for a bank account with zero balance
BankAccount(double initialBalance, java.lang.String name)	Construct a balance account with a given initial balance and owner's name

Method Summary

void	deposit(double dAmount) Method for depositing money to the bank account
double	getBalance() Method for getting the current balance of the bank account
static void	main(java.lang.String[] args) Main method for testing the bank account
void	withdraw(double wAmount) Method for withdrawing money from the bank account

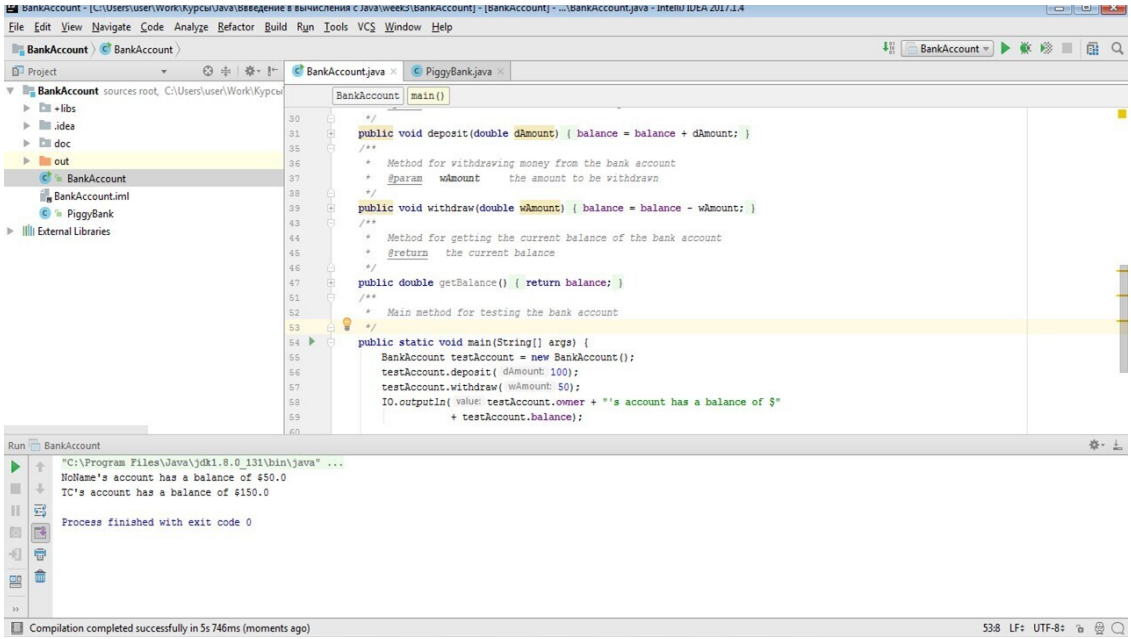
И там есть резюме конструктора и методов.

Подробные описания конструкторов и методов также предоставляются в Javadoc.

И я хочу еще раз подчеркнуть, что важно обеспечить хорошую документацию для ваших программ.

Давайте вернемся к исходному коду и скомпилируем его. Хорошо, что нет никаких ошибок.

Давайте запустим программу, нажав кнопку Run.



Результаты выводятся в консоль.

Обратите внимание, что два счета для NoName и TC имеют разный итоговый баланс, один с 50 долларами, а другой со 150 долларами, хотя один и тот же депозит и сделка снятия были произведены для обоих счетов, как вы можете видеть здесь из программы.

Это потому, что различные копии переменной экземпляра, balance, были созданы для каждого из экземпляров банковского счета.

Баланс TC равен 150 долларам вместо 50, как в счете NoName, потому что он был создан с начальным балансом 100.

Пример

Теперь вы должны иметь общее представление о структуре объявления класса.

Это важно, поскольку все программы – это Java классы. И класс является шаблоном для объектов.

Программные **объекты** обладают **свойствами**, могут использовать **методы** и реагируют на **события**

Классы объектов являются «шаблонами», определяющими наборы свойств, методов и событий, по которым создаются объекты

Объекты становятся экземплярами класса с использованием конструкторов.

Переменные экземпляра или поля, а также методы определены в классе, чтобы описать свойства и поведение объектов.

Вы можете думать о рецепте в качестве аналогии для класса.

Когда вы готовите, скажем выпечку печенья, необходимы ингредиенты (как переменные экземпляра для объектов) и инструкции шаг за шагом, как указано в рецепте (как методы в классе).

Когда печенье пекут (или создается объект), оно будет выглядеть по-разному, и вкус немного отличается, как и различные экземпляры.

Конечно, одна большая разница в том, что вы не можете съесть объекты программного обеспечения, созданные в Java программе, но в один прекрасный день вы можете быть в состоянии написать Java программу, которая поручит компьютеру испечь вкусное печенье для вас.

Давайте теперь посмотрим на другие примеры классов и объектов.

И это будет класс ColorImage.

Class ColorImage

java.lang.Object
  comp102x.CanvasObject
    comp102x.ColorImage

```
public class ColorImage
extends CanvasObject
```

The ColorImage class represents an image read from the file system. It also provides various method for manipulating the image.

Author:

leofan

Constructor Summary

Constructor and Description
`ColorImage()` Constructs a ColorImage object by loading an image from the file system and re-scales it to fit the default canvas size.
`ColorImage(boolean rescale)` Constructs a ColorImage object by loading an image from the file system and re-scales it to fit the default canvas size if indicated.
`ColorImage(ColorImage copy)` Constructs a copy of the given ColorImage object.
`ColorImage(int width, int height)` Constructs a blank white ColorImage object with a specify width and height.
`ColorImage(java.lang.String filename)` Constructs a ColorImage object by using a filename and re-scales it to fit the default canvas size.
`ColorImage(java.lang.String filename, boolean rescale)` Constructs a ColorImage object by using a filename and re-scales it to fit the default canvas size if indicated.

ColorImage используется для создания объектов изображения, которые могут отображаться на холсте.

Холст представляет собой графическое окно, которое может использоваться для отображения графических объектов, в том числе изображений, в то время как окно консоли можно использовать только для ввода и вывода текста.

Объекты ColorImage это объекты холста.

Здесь вы можете увидеть документацию для ColorImage.

Тут есть несколько конструкторов для ColorImage.

Один конструктор без параметров, и он будет создавать объект цветного изображения при загрузке существующего изображения из папки на вашем компьютере.

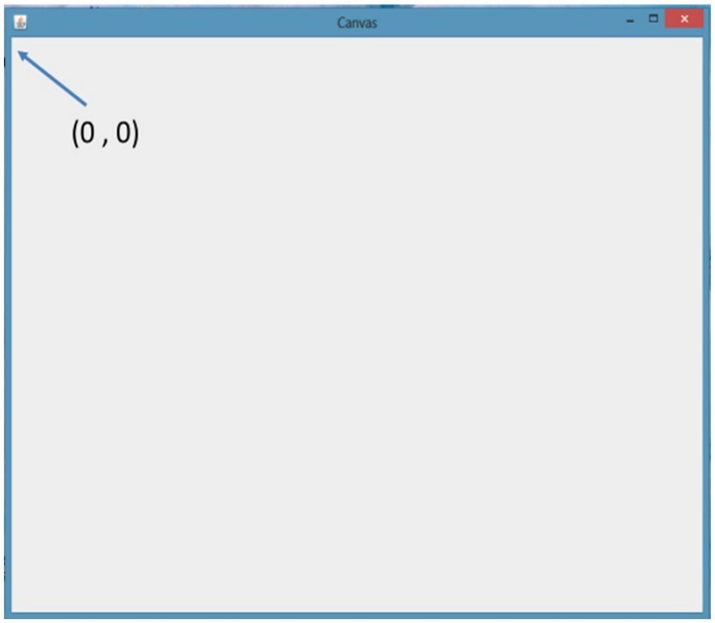
И другой конструктор принимает имя файла в виде строки в качестве параметра.

Он будет строить объект цветного изображения для изображения, которое хранится в файле, как указано в параметре.

Modifier and Type	Method and Description
`ColorImage`	`add(ColorImage operand)` Adds another ColorImage object to this ColorImage object.
`static ColorImage`	`add(ColorImage image1, ColorImage image2)` Adds two ColorImage objects together.
`void`	`add(int value)` Adds a value to all the RGB channels of this ColorImage object.
`double`	`averageDifference(ColorImage operand)` Gets the average difference between this ColorImage object and the other ColorImage object.
`void`	`convolve(float[][] kernel)` Convolves the ColorImage object with a specified kernel.
`ColorImage`	`createCyanImage()` Creates a copy of this ColorImage object with the red channel removed.
`ColorImage`	`createRedImage()` Creates a copy of this ColorImage object with the green and blue channel removed.
`void`	`decreaseBlue(int value)` Decreases the value of blue channel for all pixels.
`void`	`decreaseGreen(int value)` Decreases the value of green channel for all pixels.
`void`	`decreaseRed(int value)` Decreases the value of red channel for all pixels.
`ColorImage`	`divide(ColorImage operand)` Divides another ColorImage object to this ColorImage object.
`static ColorImage`	`divide(ColorImage image1, ColorImage image2)` Divides a ColorImage object object by another ColorImage object.
`void`	`divide(int value)` Divides all the RGB channels of this ColorImage object by a value.
`void`	`drawImage(ColorImage overImage, int x, int y)` Overlays another ColorImage object on top of the current one.
`void`	`drawOval(int x, int y, int width, int height)` Draws an oval on this ColorImage object
`void`	`drawOval(int x, int y, int width, int height, int red, int green, int blue)` Draws an oval on this ColorImage object with a specific color.

Здесь показаны некоторые из методов, которые могут быть применены к объектам в ColorImage.

Холст имеет целочисленные координаты x и y.



Это скриншот холста с координатами (0,0) в верхнем левом углу. Размер по умолчанию 800x600.

В следующих примерах мы будем использовать методы getRotation, getX и getY, которые позволяют получить позицию и ориентацию объекта изображения на холсте, и методы setRotation, setScale, setX и setY, которые изменяют положение, размер и ориентацию объекта изображения.

Давайте рассмотрим простую демо-программу ColorImageDemo, чтобы посмотреть на некоторое поведение ColorImage.

Instance variable

Constructors

```
/**
 * A simple demo on ColorImage.
 */
public class ColorImageDemo
{
   private Canvas canvas = new Canvas();      // Create a canvas for ColorImage
   private ColorImage image1 = new ColorImage("Car1.png");  // Default image

   // Define two constructors for ColorImageDemo
   public ColorImageDemo() {
      canvas.add(image1, 0, 0);            // Display ColorImage at (0,0) position
   }

   public ColorImageDemo(int xPos, int yPos) {
      image1 = new ColorImage( );         // Create a new ColorImage from user file
      canvas.add(image1, xPos, yPos);    // Display ColorImage at (xPos,yPos) position
   }
```

Здесь есть объявление переменных экземпляра – создается холст для отображения объекта ColorImage и создается новый объект ColorImage с именем переменной image1, который создается с использованием по умолчанию изображения с именем Car1.png в формате под названием PNG или Portable Network Graphics.

Другие популярные графические форматы, такие как JPG и GIF также могут быть использованы.

Дальше идет объявление конструкторов.

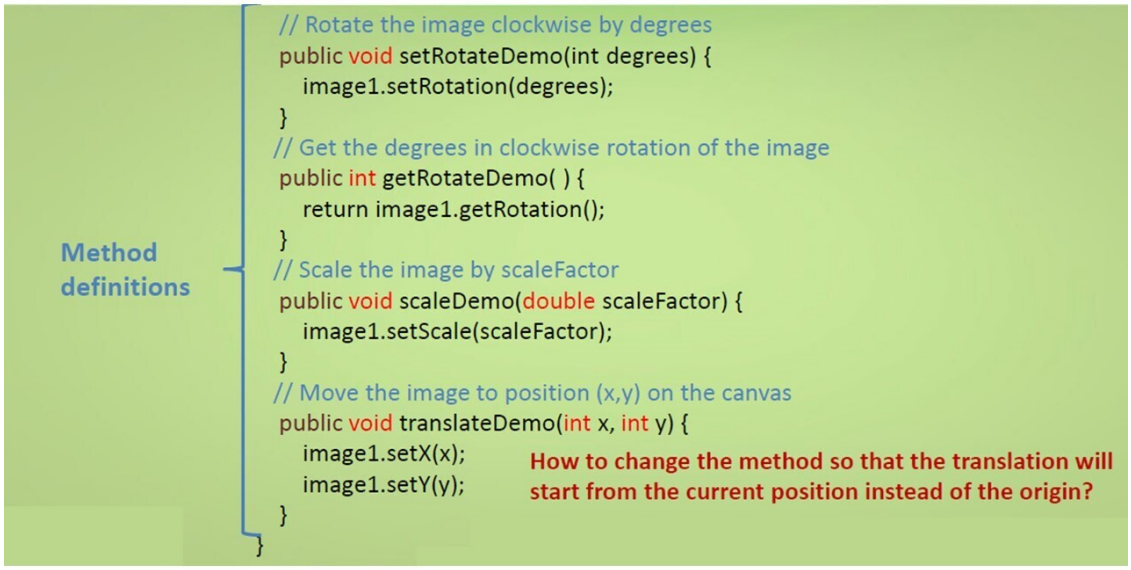
Здесь определяются два конструктора.

Один без параметров, который просто отображает по умолчанию ColorImage в позиции (0,0).

Второй конструктор с двумя параметрами, оба целого типа.

Этот конструктор заменяет image1 новым изображением ColorImage, которое может быть выбрано из существующих файлов изображений пользователем.

В результате изображение будет отображаться в позиции xPos, yPos на холсте.



И здесь определены четыре метода.

Первый вращает по часовой стрелке изображение на угол, указанный в качестве параметра.

Второй метод возвращает целое число, указывающее текущую ориентацию изображения при вращении по часовой стрелке.

Обратите внимание, что возвращаемый тип указывается как int, который отличается от возвращаемого типа void в предыдущем методе.

Третий метод масштабирует изображение на коэффициент масштабирования, указанный в качестве параметра типа double.

Этот метод уменьшает или увеличивает изображение объекта.

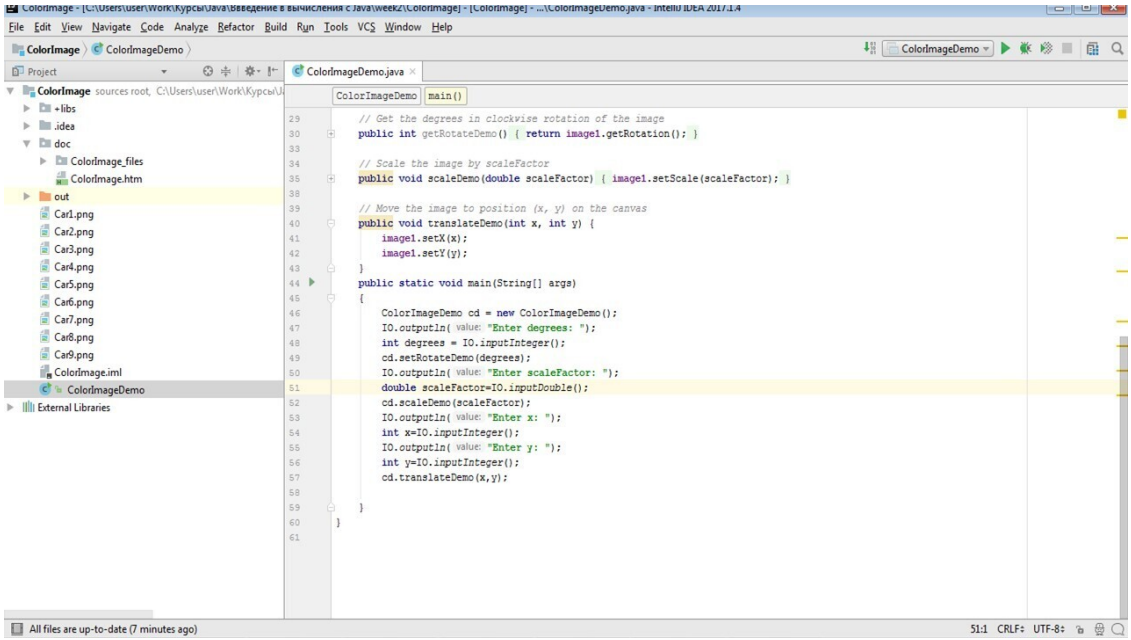
И последний метод перемещает изображение в положение (X, Y) на холсте.

Обратите внимание, что движение начинается из начального положения, а не из текущего местоположения.

Я хочу, чтобы вы подумали о том, что должно быть сделано для движения из текущей позиции вместо начала координат.

Демонстрация примера

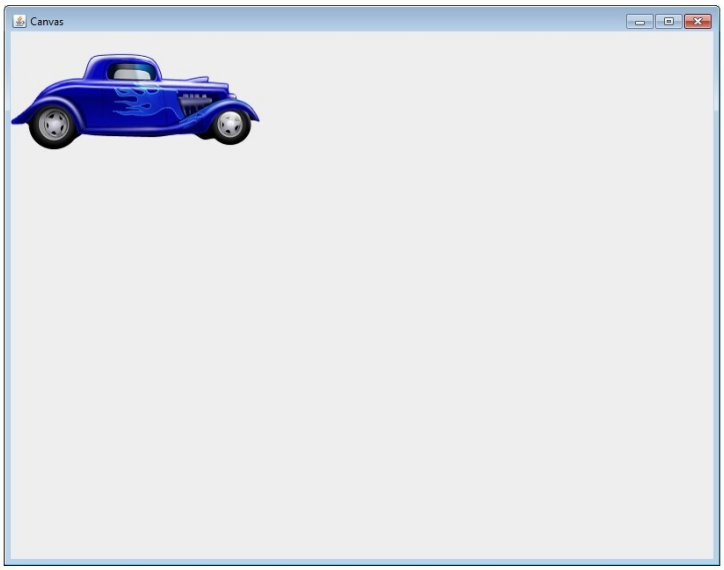
Давайте теперь посмотрим на ColorImageDemo в среде разработки IntelliJ IDEA.



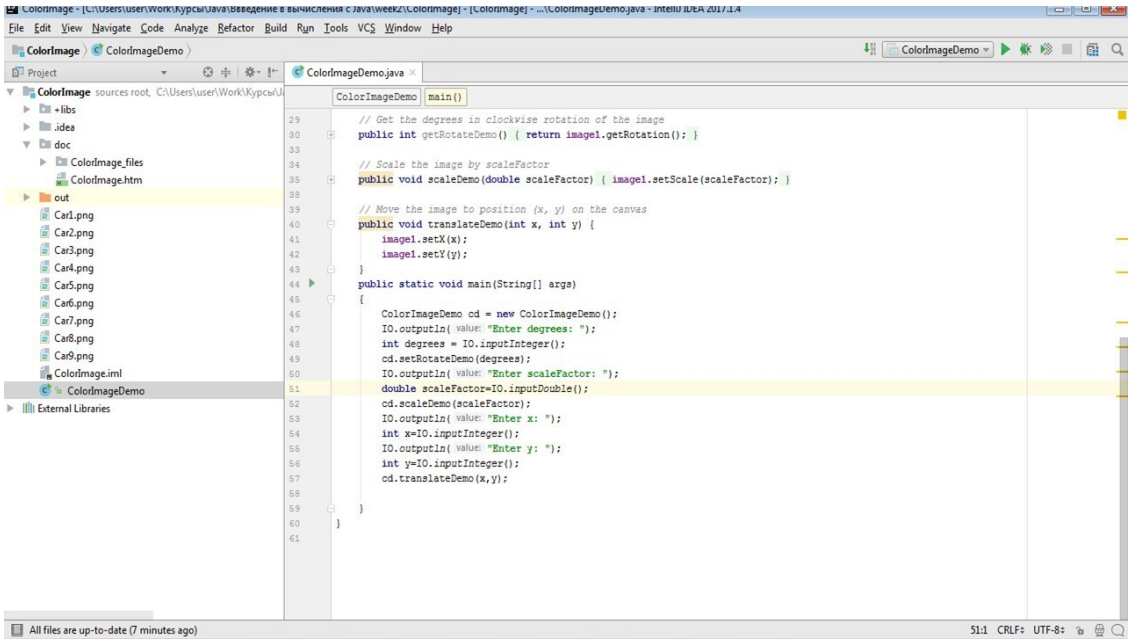
Здесь есть определение класса ColorImageDemo, как мы только что видели.

И есть метод main для запуска программы.

В этом методе мы создаем экземпляр класса ColorImageDemo, с помощью конструктора по умолчанию, используя настройки по умолчанию.



Изображение car1, то есть изображение по умолчанию, отображается на холсте в верхнем левом угле.



Затем мы с помощью ввода в консоли поворачиваем изображение, масштабируем его и перемещаем по холсту.

Вопросы

Задача

```
import comp102x.IO;
import comp102x.Canvas;
import comp102x.ColorImage;
public class ColorImageDemo
{
     public Canvas canvas = new Canvas();
     public ColorImage image1 = new ColorImage();

     public void setAndOutput() {
          canvas.add(image1, 0, 0);
          image1.setX(50);
          image1.setY(100);
          image1.setRotation(45);
          image1.setY(image1.getX() + 100);
          image1.setRotation(image1.getRotation() - 90);

          IO.output(" X: " + image1.getX());
          IO.output(" Y: " + image1.getY());
          IO.output(" A: " + image1.getRotation());
     }}
```

1. **X: 50 Y: 100 A: 45**
2. **X: 150 Y: 200 A: -45**
3. **X: 50 Y: 150 A: -45**
4. **X: 50 Y: 150 A: 45**
5. **Compilation Error**

Каким будет вывод на экране, если методы set и output вызываются в следующей программе?

Варианты ответа:

1. X: 50 Y: 100 A: 45
2. X: 150 Y: 200 A: -45
3. X: 50 Y: 150 A: -45
4. X: 50 Y: 150 A: 45
5. Compilation Error

Ответ: 3.

x и y координаты image1 установлены на 50 и 100 соответственно. Угол поворота изображения затем устанавливается на 45 (градусов). После этого, y-координата изображения обновляется на текущую x-координату изображения плюс 100 (т.е. 100 + 50 = 150).

Наконец, угол поворота изображения обновляется на текущий угол поворота минус 90 (т.е. 45 – 90 = -45 градусов).

Демонстрация примера

Я хочу немного больше сказать о классе ColorImage.

Class ColorImage

java.lang.Object
comp102x.CanvasObject
comp102x.ColorImage

```
public class ColorImage
extends CanvasObject
```

The ColorImage class represents an image read from the file system. It also provides various method for manipulating the image.

Author:

leofan

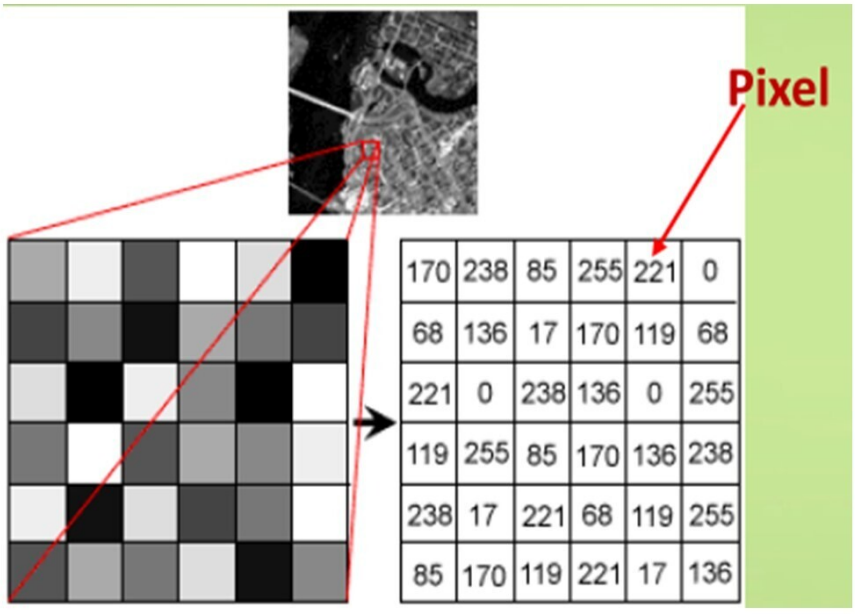
Constructor Summary

Constructor and Description
`ColorImage()` Constructs a ColorImage object by loading an image from the file system and re-scales it to fit the default canvas size.
`ColorImage(boolean rescale)` Constructs a ColorImage object by loading an image from the file system and re-scales it to fit the default canvas size if indicated.
`ColorImage(ColorImage copy)` Constructs a copy of the given ColorImage object.
`ColorImage(int width, int height)` Constructs a blank white ColorImage object with a specify width and height.
`ColorImage(java.lang.String filename)` Constructs a ColorImage object by using a filename and re-scales it to fit the default canvas size.
`ColorImage(java.lang.String filename, boolean rescale)` Constructs a ColorImage object by using a filename and re-scales it to fit the default canvas size if indicated.

Как я уже говорил, ColorImage является своего рода объектом Canvas, который может быть отображен на холсте как двумерное изображение.

Было бы полезно иметь некоторое представление о том, как представляется 2D изображение, хотя вам не нужно знать все детали.

Давайте сначала посмотрим на черно-белое изображение, как показано здесь.



Оно может быть представлено в виде 2D матрицы чисел.

Эти цифры часто называют уровни серого, которые представлены в виде 8-разрядного байта.

Байт может принимать числа от 0 до 255.

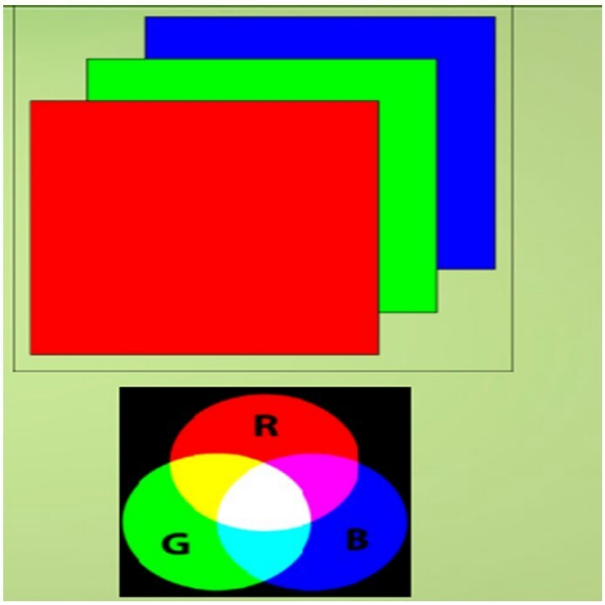
В представлении 2D-изображений как уровней серого, 0 используется для представления полностью черного и 255 для белого цвета, а значения между ними для различной интенсивности серого.

Рисунок здесь показывает небольшую область изображения, и соответствующие ей уровни серого.

Каждый элемент 2D матрицы называется пикселем или элементом изображения.

Если вы изучали физику, тогда знаете, что полный цветовой спектр может быть представлен сочетанием трех основных цветов, красного, зеленого и синего или RGB.

Так, цветное изображение можно представить в виде трех слоев 2D изображения по одному для красного, зеленого и синего цветов.



Цветные изображения в классе ColorImage имеют еще один канал, называемый альфа-каналом, который представляет собой степень прозрачности.

Таким образом, 4 канала, R, G, B и Alpha, каждый из них занимает 8 бит и все они составляют в общей сложности 32 бит.

Это все, что я хотел сказать о представлении цветного изображения.

В дополнение к методам, которые изменяют положение и ориентацию ColorImage, существуют также методы для манипулирования значениями пикселей в ColorImage. Вот некоторые из этих методов.

Methods

Modifier and Type	Method and Description
ColorImage	**add**(ColorImage operand) Add another ColorImage object to this ColorImage object
static ColorImage	**add**(ColorImage image1, ColorImage image2) Add two ColorImage objects together
void	**add**(int value) Add a value to all the RGB channels of this ColorImage object
double	**averageDifference**(ColorImage operand) Get the average difference between this ColorImage object and the other ColorImage object
void	**convolve**(float[][] kernel) Convolve the ColorImage object with a specified kernel
ColorImage	**createCyanImage**() Create a copy of this ColorImage object with the red channel removed
ColorImage	**createRedImage**() Create a copy of this ColorImage object with the green and blue channel removed
void	**decreaseBlue**(int value) Decrease the value of blue channel for all pixels
void	**decreaseGreen**(int value) Decrease the value of green channel for all pixels
void	**decreaseRed**(int value) Decrease the value of red channel for all pixels

Документация Javadoc дает полный список и подробные описания методов.

Вам на самом деле нужно будет использовать только некоторые из них для вашей лабораторной работы, например, чтобы добавить два изображения.

Я хочу показать вам интересную работу на цветных изображениях, которая позволила бы вам создавать 3-мерные изображения с помощью трех методов add, createCyanImage и createRedImage.

Метод createRedImage удалит зеленый и синий каналы из изображения и оставит только красный канал.

Метод createCyanImage удаляет красный канал и сохраняет зеленый и синий, и сочетание из оставшихся зеленого и синего каналов дает голубой цвет.

Итак, как мы можем создать 3D изображение.

Мы видим 3D, потому что у нас есть два глаза и два глаза немного по-разному смотрят на объекты вокруг нас.

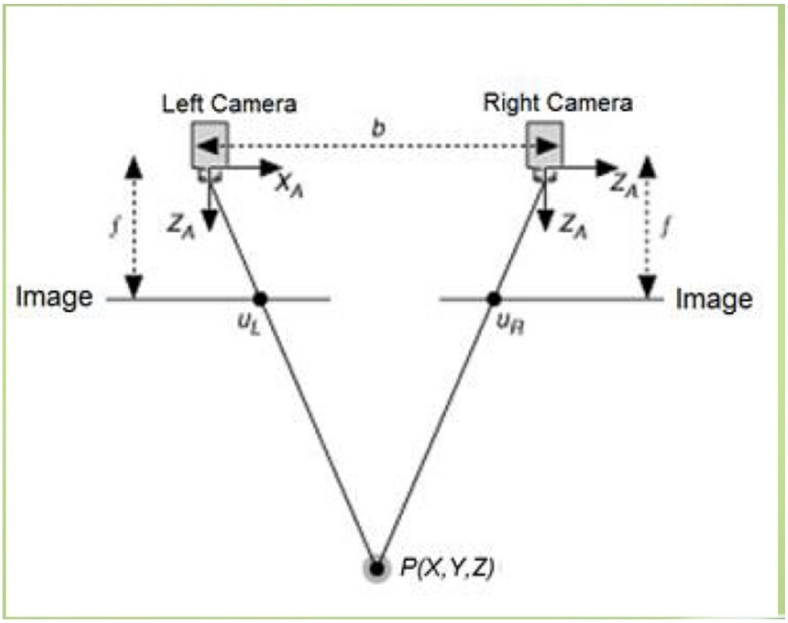
Вы можете проверить это легко, закрыв один глаз, а затем другой. Вы увидите, что изображения немного отличаются, и наша зрительная система достаточно умна, чтобы интерпретировать такие различия для получения 3D информации.

Вы можете имитировать этот процесс путем размещения двух одинаковых камер бок о бок, чтобы сфотографировать окружающую среду.

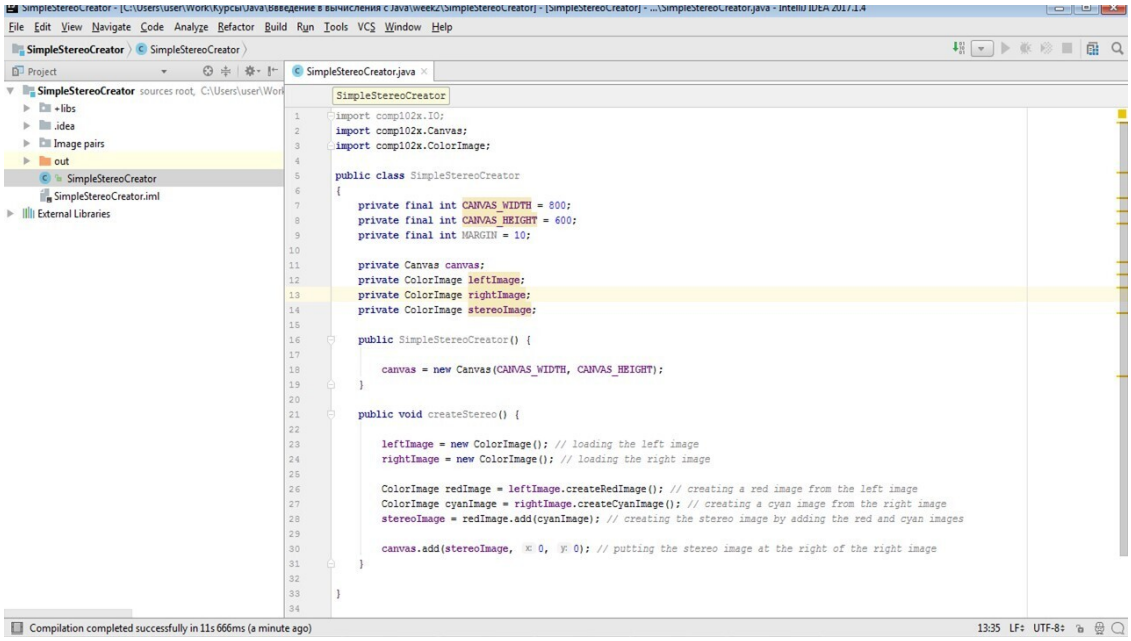
Даже если у вас нет двух камер, можно использовать только одну, например, камеру вашего мобильного телефона.

Возьмите одну картину, а затем переместите ее по горизонтали на небольшое расстояние.

Расстояние не может быть слишком большим, иначе ваша зрительная система не сможет слить два изображения.



Теперь давайте откроем проект SimpleStereoCreator.



Вы видите, что программа начинается с импорта трех классов, класса IO, класса Canvas, класса ColorImage.

И имя нашего класса SimpleStereoCreator.

Тело определения класса начинается с объявления переменных экземпляра.

Здесь определяется размер создаваемого холста, а затем объявляются переменные экземпляра для холста, левого изображения leftImage, правого изображения rightImage и полученного стереоизображения.

Конструктор класса просто создает новый объект холста с заданной шириной и высотой.

И основной метод, который делает всю работу, это метод createStereo.

Вы видите, что он начинается с запроса у пользователя существующих левого и правого изображений.

Затем метод создает красное изображение, извлекая красный канал из левого изображения, а затем метод создает голубое изображение, извлекая синий и зеленый каналы из правого изображения.

При добавлении красного и голубого изображений, получается стереоизображение, а затем стереоизображение отображается на холсте.

Вы можете скомпилировать программу и здесь нет синтаксических ошибок.

Мы можем запустить программу путем создания экземпляра объекта, а затем вы можете вызвать метод createStereo.

```
public static void main(String[] args)
  {
     SimpleStereoCreator sc = new SimpleStereoCreator();
     sc.createStereo();

  }
```

Появится диалоговое окно, которое попросит у вас сначала левое изображение, а затем правое изображение.

И вы видите, что здесь создается стереоизображение.

Но как вы можете увидеть этот стереоэффект?

Вы можете просмотреть красное и синее стереоизображение с помощью красных и синих стереоочков.

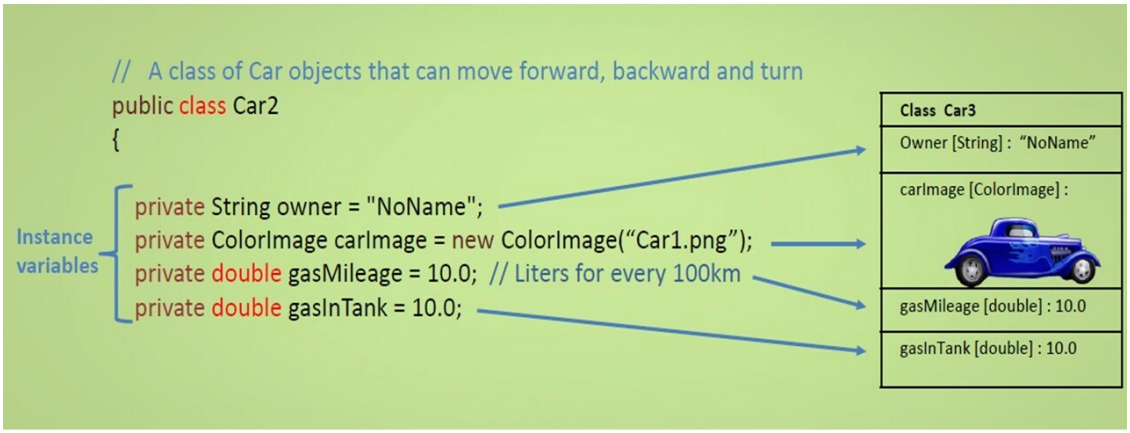
Эффект может быть не так хорош, как когда вы смотрите фильмы в формате 3D в кинотеатре, но это даст вам простой метод для просмотра красно-синих стереоизображений.

Пример

В предыдущем примере автомобиля, методы moveCar и turnCar просто распечатывали сообщение на экране о том, что методы должны делать, вместо выполнения реальных действий.

Теперь, когда вы узнали об объектах ColorImage, давайте попробуем изменить программу таким образом, что она сможет выполнять более интересные действия.

И мы дадим этому классу новое имя Car2.



Здесь у нас есть экземпляр переменной для хранения информации о владельце, как и раньше.

То, что мы хотим сделать, это использовать некоторые фотографии автомобилей для представления различных экземпляров объекта Car.

Так вот, вместо того чтобы использовать только одну переменную экземпляра владельца, у нас также теперь есть еще одна переменная экземпляра, которая имеет тип ColorImage (своего рода объект холста, о котором мы только что говорили), и давайте, теперь, инициализируем изображение автомобиля по умолчанию как Car1.png.

Кроме того, мы хотим описать больше полезных свойств автомобилей в этом классе.

В наше время предпочитают покупать экологически чистые автомобили.

Поэтому полезной информацией является расход топлива.

Для этой цели объявляется переменная экземпляра gasMileage типа double, и предполагается, что значение переменной представляет собой количество топлива в литрах, используемых на каждые 100 км.

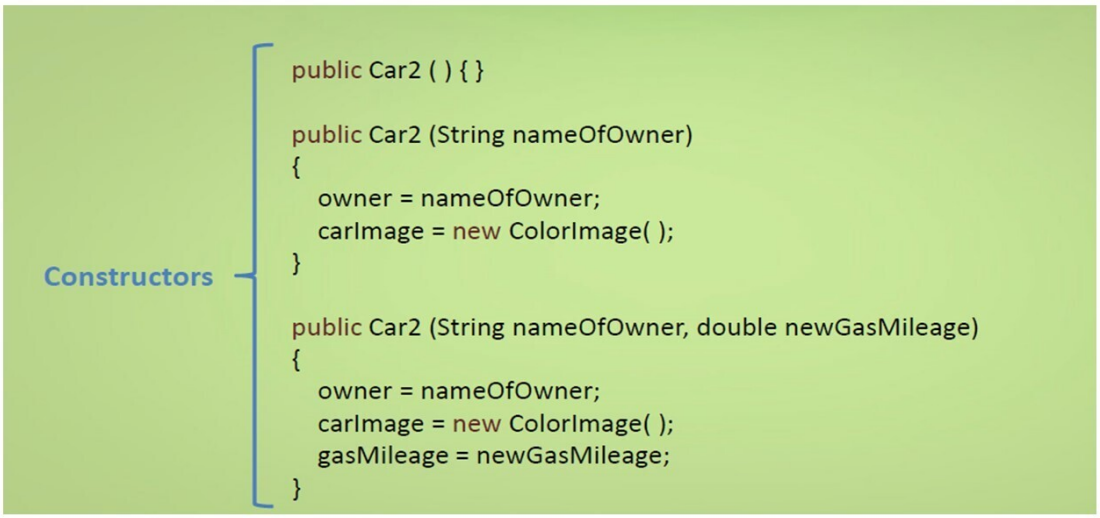
Наконец, мы хотим знать, сколько бензина в баке.

В то время как предыдущие поля были относительно стабильными, это значение часто изменяется время от времени и отличается для различных экземпляров автомобилей.

Эта коллекция переменных экземпляра дает абстрактное описание объекта автомобиля – его владельца, его внешний вид, его эффективность расхода топлива и количество топлива в баке.

И каждый раз, когда создается новый объект автомобиля, необходимо выделять объем памяти для хранения всей этой информации.

Давайте посмотрим на конструкторы класса для Car2 объектов.



Обратите внимание, что первый конструктор такой же, как и раньше, за исключением того, что именем конструктора является Car2, так как имя класса в настоящее время Car2.

Конструктор, который не имеет параметров, просто использует все настройки по умолчанию.

Я хочу отметить, что, если конструктор не определен внутри класса, Java будет предоставлять конструктор по умолчанию для класса от его суперкласса.

Но если конструктор определен, Java не позволит использование конструктора по умолчанию без явного его определения.

Как и второй конструктор в предыдущем примере, второй конструктор здесь имеет один параметр типа String.

Первое выражение в теле конструктора является таким же, как прежде, которое изменяет значение владельца на значение, заданное параметром nameOfOwner.

Мы добавили второе выражение здесь, которое изменяет carImage на новое изображение.

Ключевое слово new здесь показывает, что создается новый объект типа ColorImage.

Поскольку ни один параметр не предоставляется ColorImage, пользователю будет предложено выбрать существующее изображение из папки.

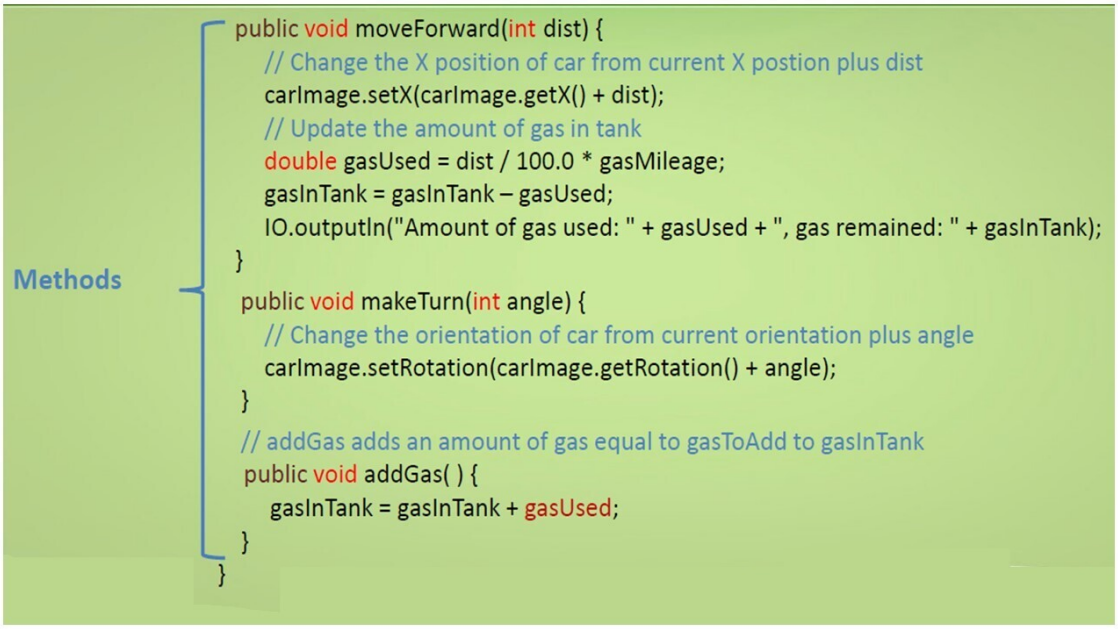
И здесь есть третий конструктор. Поскольку gasMileage может измениться, так как автомобиль стареет.

И чтобы не вводить в заблуждение клиента, который покупает подержанную машину, конструктор позволит gasMileage измениться.

Этот конструктор принимает два параметра, один типа String, другой типа double.

В дополнение к тому, что делается во втором конструкторе, значение переменной gasMileage экземпляра изменяется на newGasMileage при создании нового объекта Car2, который создается с помощью этого конструктора.

Давайте теперь посмотрим на методы Car2.



У нас были методы перемещения и поворота автомобиля, и они просто распечатывали сообщение на экране о том, что эти методы должны делать.

Вместо того чтобы просто распечатать сообщение, модифицированный метод на самом деле двигает автомобиль на холсте из текущего положения вперед.

Текущее положение получается методом getX для объекта carImage с помощью оператора точки.

А расстояние перемещения указывается в качестве параметра.

При переезде автомобиль использует некоторое количество топлива, и количество топлива в баке должно быть обновлено.

Мы сначала вычислим количество топлива, потребляемого на это расстояние.

А затем количество топлива, оставшегося в баке, теперь может быть обновлено путем вычитания gasUsed из значения gasInTank.

Результат вычислений выводится на консоль. Обратите внимание, что параметр расстояния может принимать как положительные, так и отрицательные значения, таким образом, метод позаботился о том, как двигаться вперед, а также назад, когда расстояние отрицательно.

Вместо того чтобы просто распечатать сообщение на консоли, метод makeTurn будет изменять ориентацию объекта автомобиля на угол параметра при его отображении на холсте.

Текущая ориентация получается методом getRotation для объекта carImage с помощью оператора точки.

Так как мы использовали некоторое количество топлива, здесь добавляется метод, чтобы мы могли пополнить бензобак.

Давайте назовем этот метод addGas.

Первоначальный проект метода может выглядеть примерно так, как если мы хотим пополнить только то количество топлива, которое только что было использовано.

Помните, что мы вычислили gasUsed в методе перемещения.

В выражении внутри метода addGas добавим значение gasUsed к gasInTank, а затем присвоим результат обратно к переменной gasInTank.

Как вы думаете, это будет работать?

То, что вы обнаружите, это компилятор будет выдавать ошибку: "Невозможно найти символ – переменную gasUsed".

Почему это так?

Это потому, что есть очень важное понятие – область видимости переменных, которое мы еще не обсуждали.

В принципе, областью видимости переменной является блок кода, где переменная может быть использована.

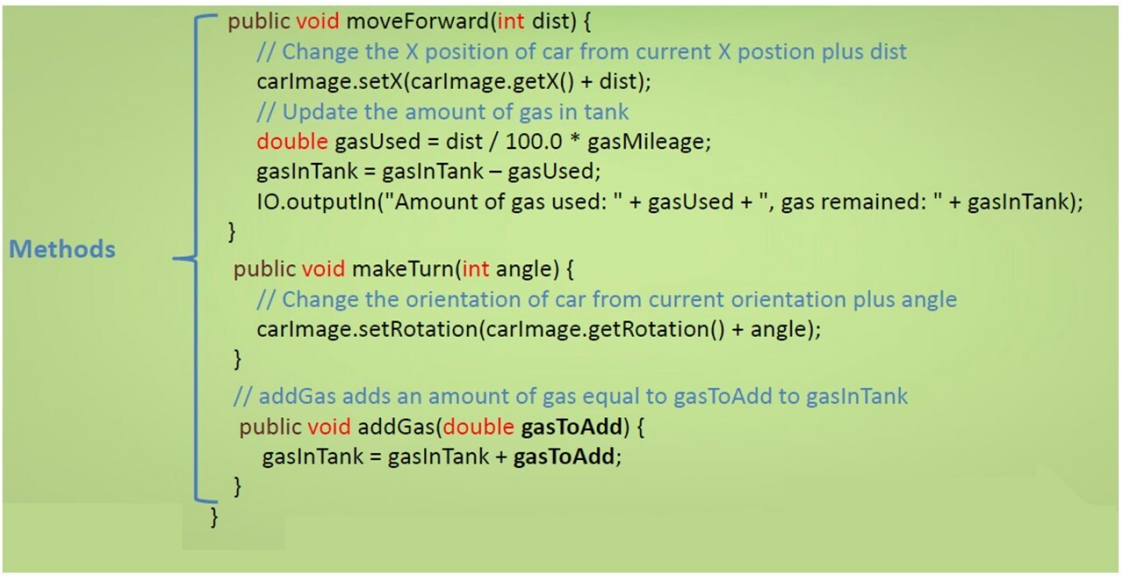
В случае переменной gasUsed, она называется локальной переменной, и она имеет смысл только в рамках метода moveForward.

Так что любая ссылка на нее вне метода moveForward вызовет ошибку.

Метод moveForward можно вызывать много раз, и переменная gasUsed используется только как временное хранилище для хранения конкретной поездки, и она принимает разные значения в других поездках.

В некотором смысле, это как кратковременная память.

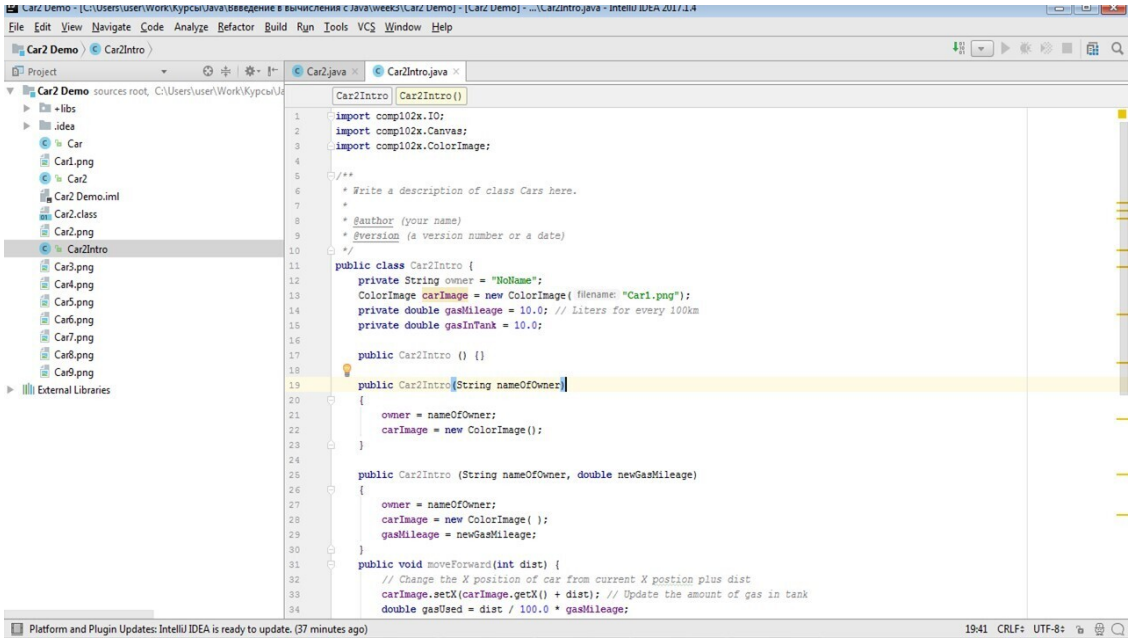
Мы вернемся к правилам области видимости в деталях позже.



Давайте теперь просто изменим метод addGas, чтобы он был методом с одним параметром gasToAdd, который дает количество топлива для добавления и выражение внутри метода addGas обновит количество топлива в баке соответственно.

Демонстрация примера

Откроем среду IntelliJ IDEA и загрузим проект Car2.



Мы увидим определение Car2.

Здесь у нас есть объявления переменных экземпляра, конструкторы, а также методы.

Прежде, чем я скомпилирую программу, я хочу отметить, что я добавил метод car2Demo, так что будет создан новый холст, и изображение автомобиля будет отображаться в положении 200, 200 на холсте.

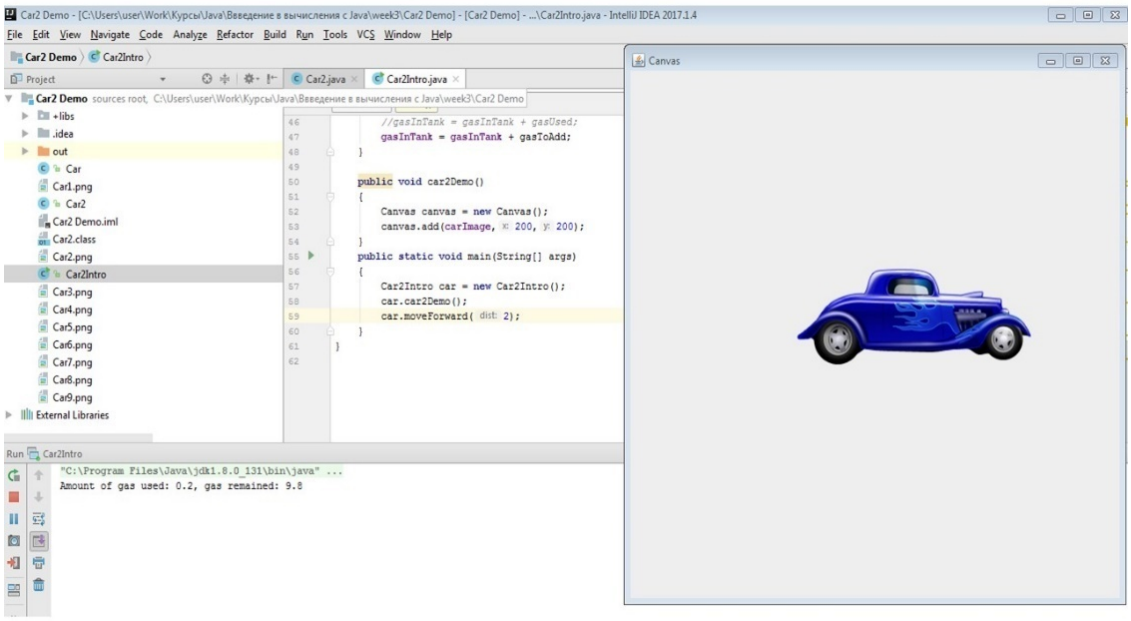
Давайте теперь скомпилируем программу.

И создадим экземпляр объекта Car2 с первым конструктором, используя настройки по умолчанию.

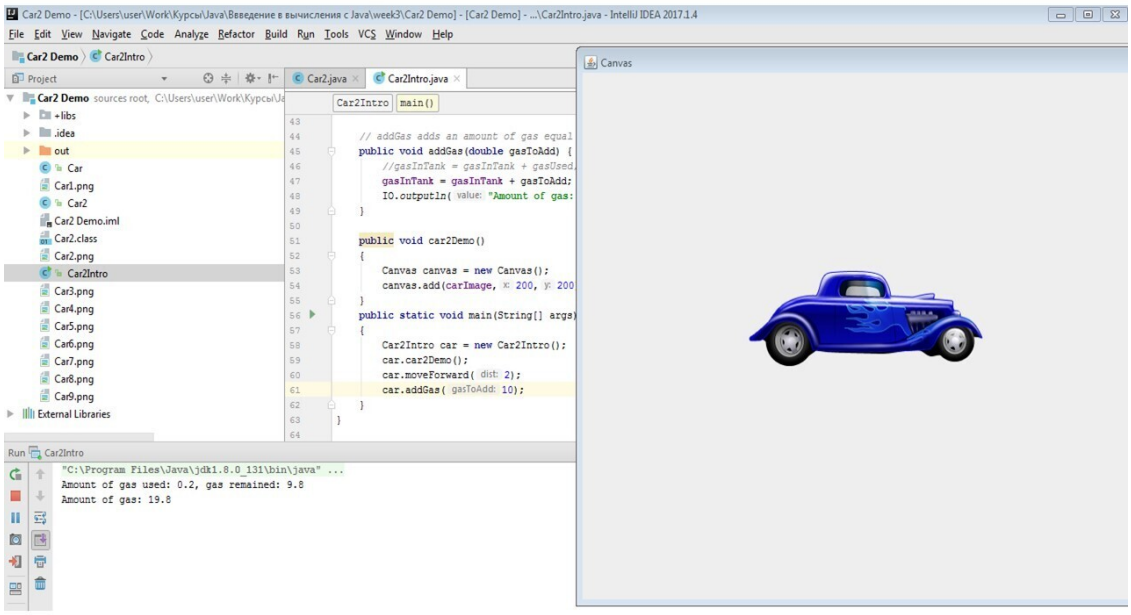
Мы сначала вызовем метод car2Demo так, чтобы изображение автомобиля отобразилось на холсте.

Затем мы можем выбрать метод, чтобы двигаться вперед. Давайте двигаться вперед на 2 единицы.

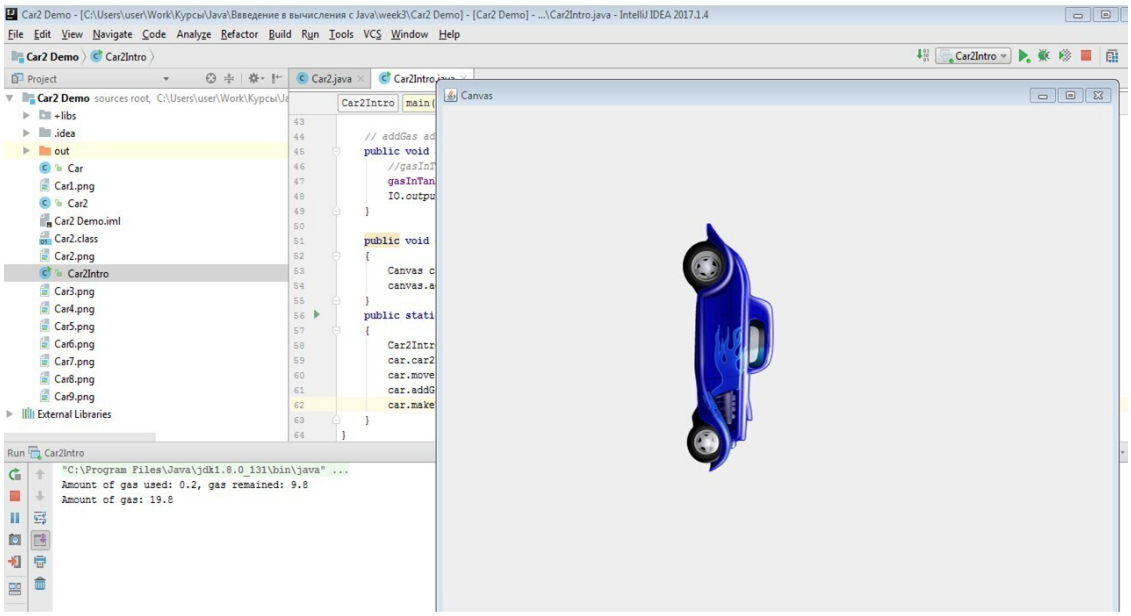
Обратите внимание, что автомобиль перемещается, и консоль также отображает, что 0,2 л бензина было использовано и количество топлива, оставшегося в баке теперь 9.8 вместо 10.



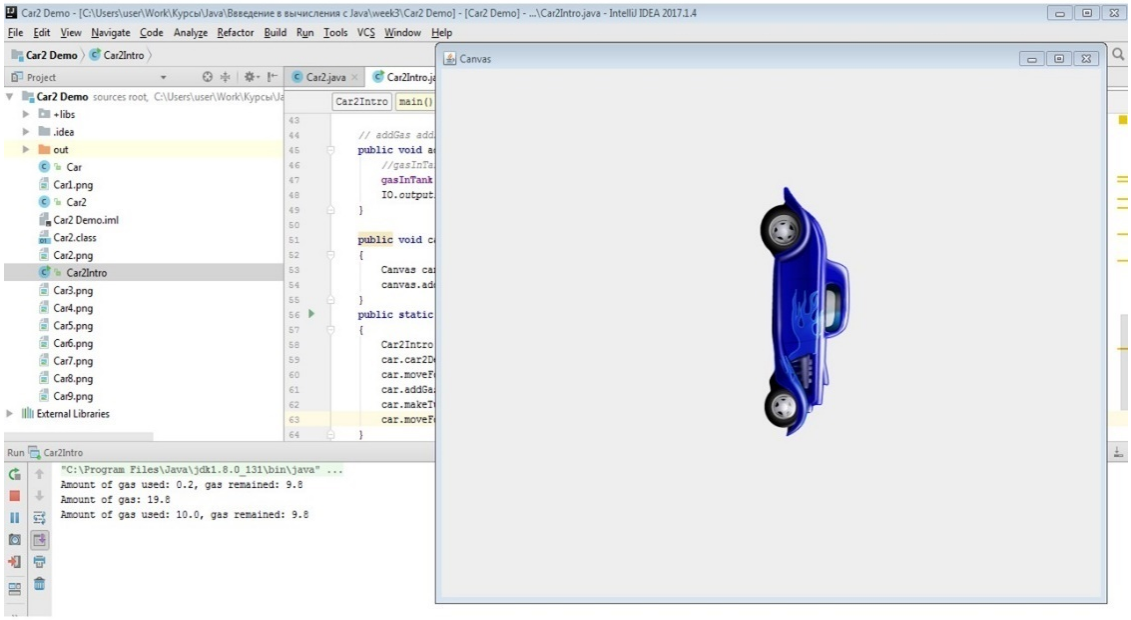
Теперь мы можем вызвать метод addGas, чтобы увидеть, может ли он на самом деле изменить переменную gasInTank экземпляра, скажем при добавлении 10 литров.



Здесь мы увидим, что gasInTank теперь 19,8, а не просто 9.8, как и прежде.
Давайте посмотрим на метод makeTurn и повернем машину, скажем на 90 градусов.



Автомобиль вращается, как и ожидалось.
Все до сих пор кажется прекрасным.
Давайте вернемся к методу moveForward и переместим автомобиль на 100 единиц.



Он движется, но, вероятно, не так, как автомобиль должен был бы продвинуться вперед, исходя из его ориентации

Я хотел бы, чтобы вы думали о том, как решить эту проблему так, чтобы автомобиль двигался вперед в том же направлении, что и ориентация автомобиля.

Если вернуться к автомобилю, вы можете попробовать другие конструкторы, вводя имена файлов и так далее.

Вопросы

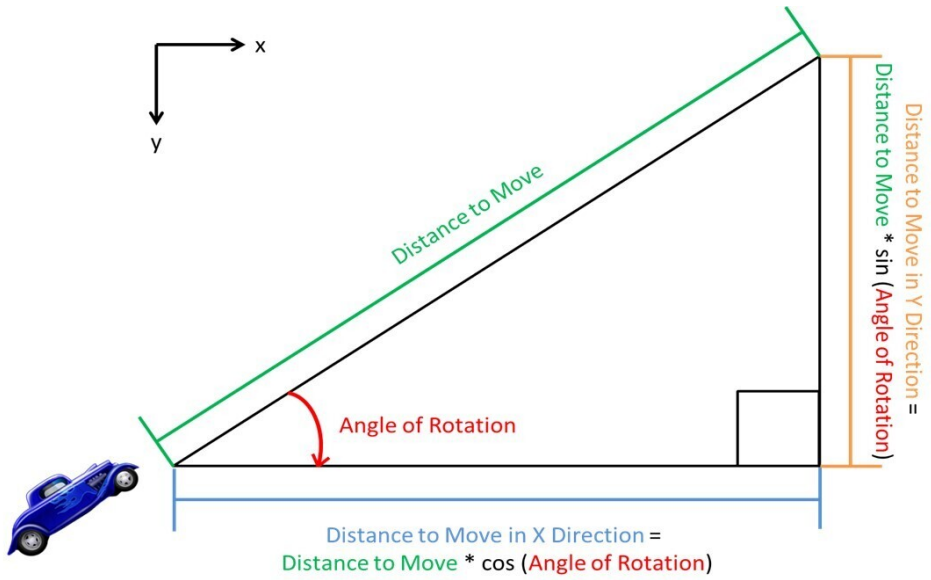
Задача

Описание

Из демо Car2, Вы заметили, что мы можем двигать автомобиль слева направо (или наоборот), просто вызывая setX, чтобы изменить его x положение. Обратите внимание, что ориентация автомобиля не рассматривалась в этой демонстрации.

В этой задаче вы должны улучшить движение автомобиля. Перемещайте машину вперед или назад с учетом ориентации автомобиля. Вы можете сделать это, изменив как x положение (с помощью setX), так и положение y (с помощью setY).

Вы можете обратиться к диаграмме, чтобы понять взаимосвязь между расстоянием перемещения и углом поворота.



http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

Как указано на схеме, должны быть вычислены косинус (COS) и синус (SIN) угла поворота.

Это может быть сделано с помощью методов cos и sin, определенных в Java классе Math (http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html).

Для того чтобы использовать эти методы, вы должны будете в первую очередь преобразовать текущий угол поворота из градусов в радианы, используя формулу:

$$rotation\ angle\ in\ radians = \frac{rotation\ angle\ in\ degrees * \pi}{180}$$

Ответ:

double rotationInRadians = Math.toRadians(rotationInDegrees);

double rotationInRadians = 22.0 / 7.0 * rotationInDegrees / 180;

После получения угла поворота в радианах, можно вычислить расстояние для перемещения автомобиля в направлениях X и Y с помощью простых математических уравнений.

Учитывая расстояние, на которое мы хотим двигать автомобиль, и угол поворота в радианах, расстояние движения в направлениях X и Y можно рассчитать следующим образом:

```
Distance to Move in x Direction = Distance to Move *
cos(Angle of Rotation)

Distance to Move in y Direction = Distance to Move *
sin(Angle of Rotation)
```

Ответ:

double distX = dist * Math.cos(rotationInRadians);

double distY = dist * Math.sin(rotationInRadians);

Правила области видимости

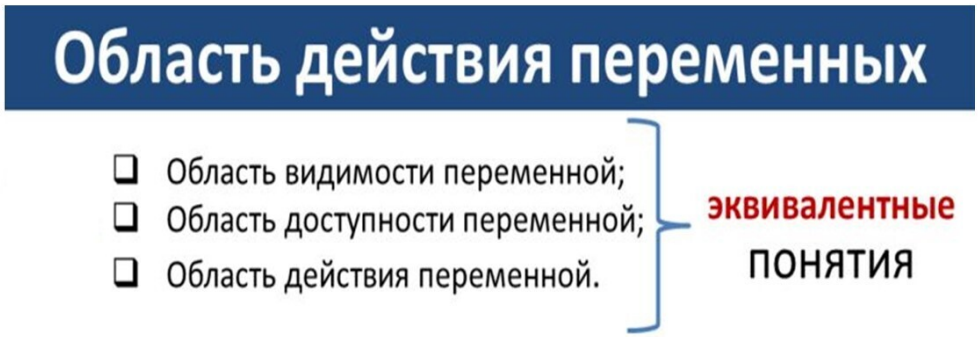
В этом разделе мы продолжим обсуждение объектно-ориентированного программирования.

В частности, мы рассмотрим различные типы переменных, в том числе переменные класса, экземпляра класса и локальные переменные, и видимость переменных.

Затем мы начнем знакомиться с операторами ветвления, в том числе с выражениями if-else и switch.

Давайте сначала рассмотрим некоторые примеры, чтобы проиллюстрировать использование различных типов переменных, их видимость и срок их действия, исходя из объектно-ориентированного подхода.

Давайте сначала посмотрим, что означает видимость переменной.



Видимость переменной, это раздел программы, в котором переменная может быть использована.

Мы часто сталкиваемся с вопросами, связанными с использованием объектов в повседневной жизни.

Когда мы путешествуем с места на место, мы можем использовать определенную валюту только в пределах конкретного региона.

Например, евро может использоваться в Европе, доллары в США и юани в Китае.

Один и тот же термин в другом контексте может нести разный смысл.

Например, то, что вы можете получить за 100 долларов в Азии, очень отличается от того, что можно было бы получить за ту же сумму в долларах в США или Европе.

В программах, мы можем объявить несколько переменных с одинаковыми именами.

В реальной жизни, мы также часто сталкиваемся с многочисленными людьми или объектами с одинаковыми именами.

На самом деле, в мире есть несколько городов с названием Лондон, и мы сможем узнать, какой город имеется в виду, только исходя из контекста.

Скажем, если вы находитесь в Англии, вы, вероятно, имеете в виду Лондон в Великобритании.

Если вы находитесь в Канаде, вы, вероятно, говорите о Лондоне в провинции Онтарио.

Другим примером является использование имен пользователей в разных доменах.

Хотя ваше имя пользователя на одном сайте является уникальным, на другом сайте оно может быть уже занятым для вашей регистрации.

Давайте рассмотрим правила видимости для различных типов переменных в Java.

Прежде всего, Java является строго типизированным языком программирования, то есть, все имена должны быть сначала определены как типы, прежде чем они могут быть использованы.

Мы уже рассматривали переменные экземпляра класса, но теперь мы поговорим о статических переменных или переменных класса.

Эти переменные объявляются вне методов и доступны из любой точки в классе, где они были объявлены.

Доступность за пределами класса будет зависеть от идентификатора доступа, используемого в объявлении.

Переменные же, объявленные внутри метода, в том числе переменные параметров, доступны только в рамках метода.

Эти переменные, как правило, называются локальными переменными, и идентификаторы доступа не используются для локальных переменных.

Кроме того, локальные переменные имеют короткое время жизни, они создаются при вызове метода и исчезают, когда выполнение метода завершается.

Переменные, объявленные в блоке программы, который заключен в фигурные скобки {}, являются локальными для блока программы, то есть, они не доступны за пределами блока программы.

А что, если у нас есть две переменные, объявленные с одним и тем же именем в программе.

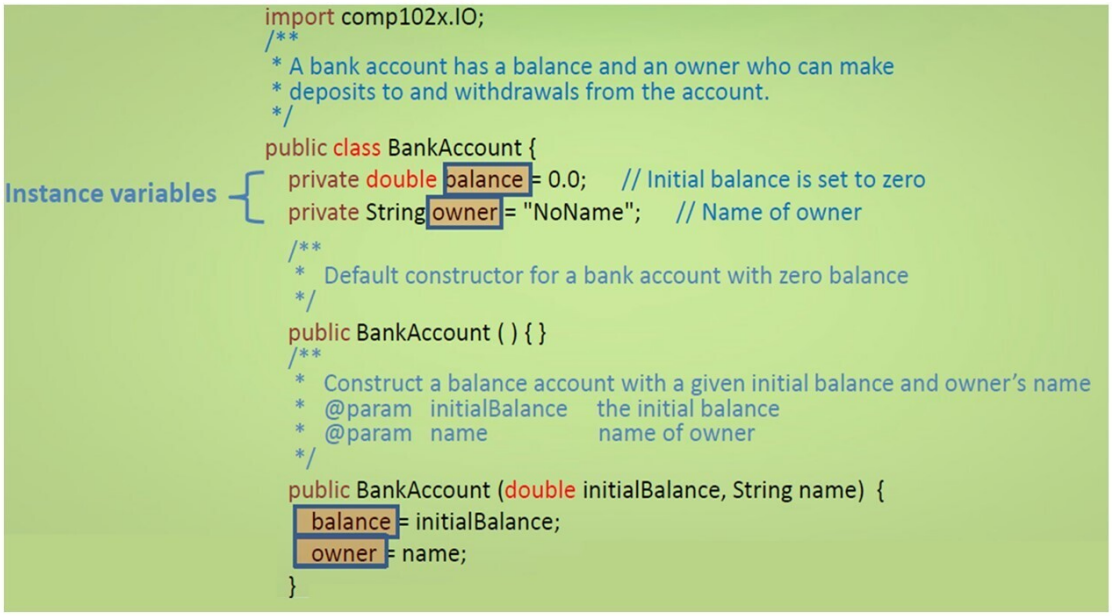
Способ, с помощью которого Java имеет дело с переменными с одним и тем же именем, это с первой попытки сначала найти соответствующее объявление переменной в текущем блоке, где появляется переменная.

Если такое объявление не найдено, делается попытка найти соответствующее объявление переменной в блоке его родителя.

Этот процесс будет повторяться до тех пор, пока область переменной не будет разрешена.

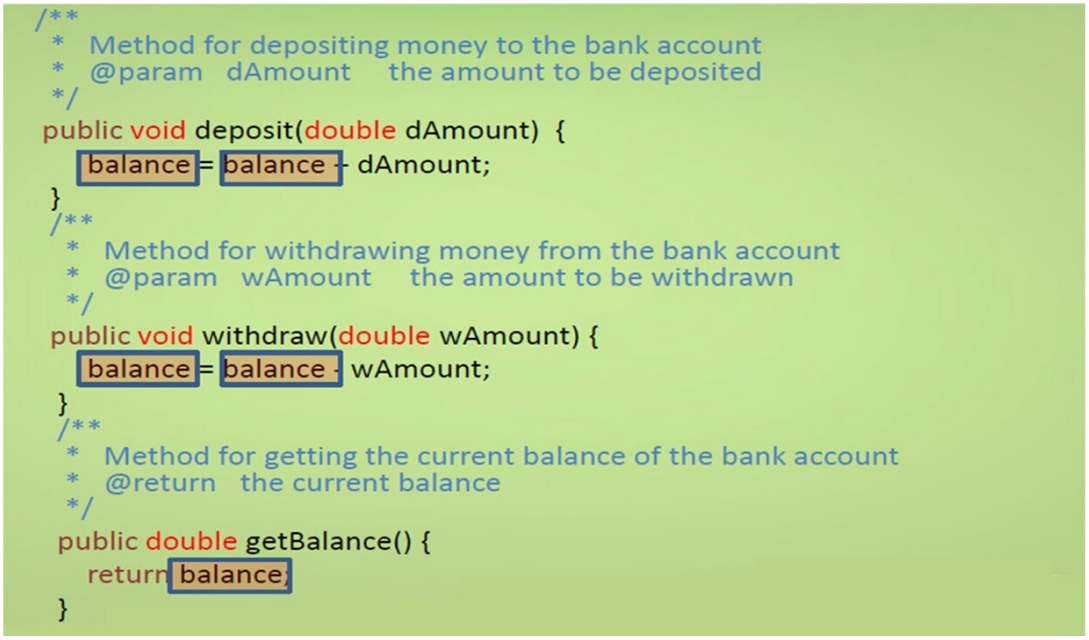
При этом будет ошибка компиляции, если имя переменной не может быть разрешено даже в самом внешнем блоке кода.

Давайте сначала посмотрим на простой пример, а именно класс BankAccount, который мы рассматривали ранее.



Здесь есть две переменные экземпляра, баланс и владелец, в объявлении банковского счета.

В соответствии с правилами видимости для переменных экземпляра, они доступны в любом месте класса, и вы можете видеть, как баланс и владелец используются во втором конструкторе и в методах класса.



Обратите внимание, что методы не только могут получить доступ к этим переменным экземпляра, они также могут вносить изменения в их значения.

Например, депозит может увеличить величину баланса путем добавления суммы, вносимой на счет.

Здесь, методы deposit и withdraw имеют один параметр.

Один с именем dAmount и другой с именем wAmount.

Как описано в правилах видимости для переменных параметров, переменные параметров являются локальными переменными.

Они могут быть использованы только в соответствующих методах, так что методу deposit не будет разрешен доступ к wAmount и методу withdraw не разрешено обращаться к переменной dAmount.

Теперь, что будет, если мы изменим как dAmount, так и wAmount на одно и то же имя Amount.

```
/**
 *  Method for depositing money to the bank account
 *  @param  dAmount    the amount to be deposited
 */
public void deposit(double amount)  {
    balance = balance + amount;
}
/**
 *  Method for withdrawing money from the bank account
 *  @param  wAmount    the amount to be withdrawn
 */
public void withdraw(double amount) {
    balance = balance - amount;
}
/**
 *  Method for getting the current balance of the bank account
 *  @return  the current balance
 */
public double getBalance() {
    return balance;
}
```

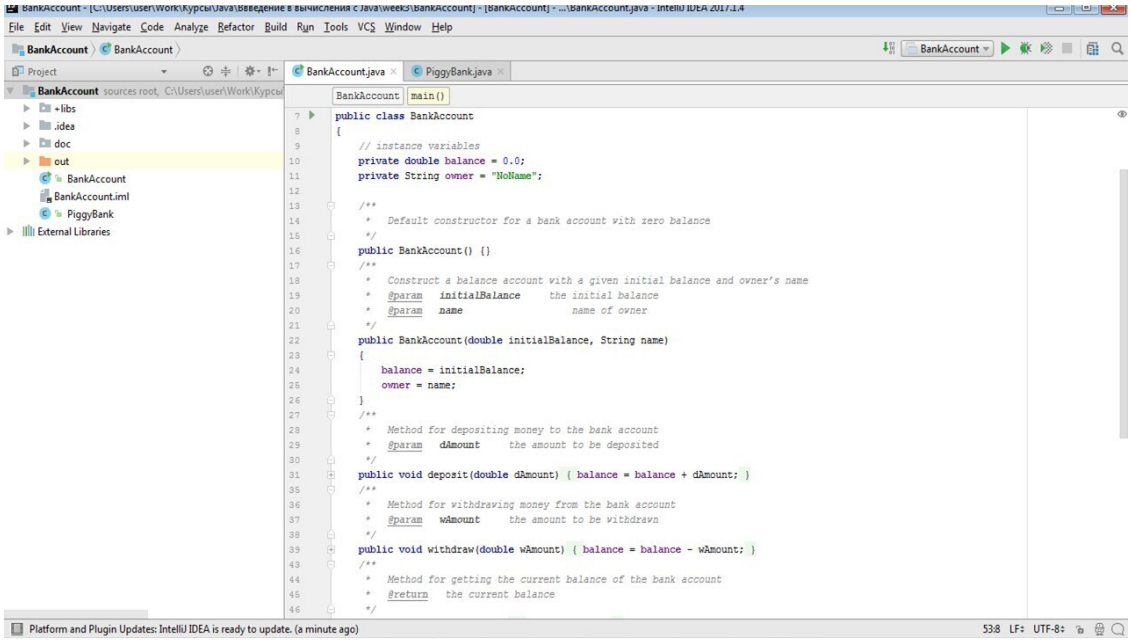
Это ничего не изменит, потому что сумма в deposit является локальной для метода deposit, и параметр в методе withdraw проживает в другом пространстве памяти, чем параметр в методе deposit.

То есть, программа не будет ошибочно вычитать сумму из баланса, которая должна быть зачислена на счет.

Давайте откроем IntelliJ IDEA и посмотрим на выполнение программы.

Демонстрация примера

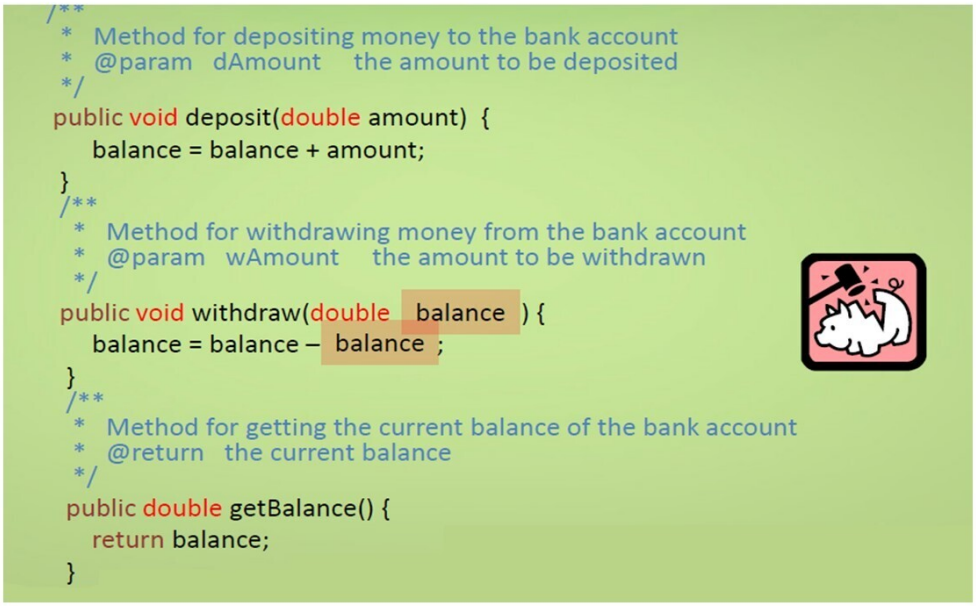
Давайте откроем проект BankAccount в среде IntelliJ IDEA.



Вы можете видеть, что это тот же самый класс BankAccount, который мы обсуждали ранее.

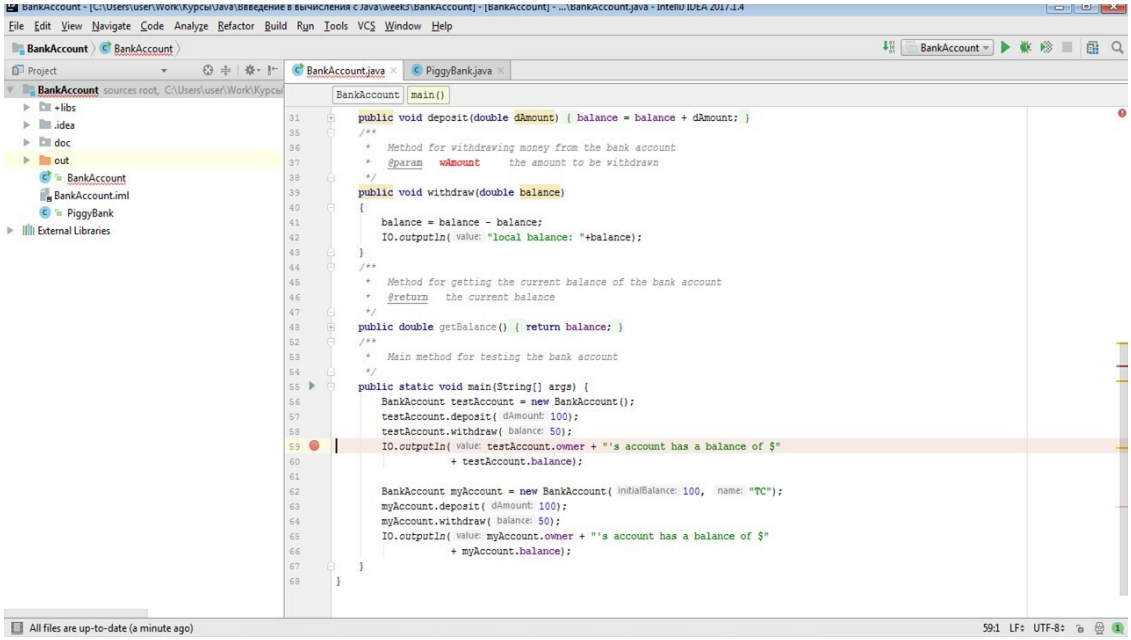
У нас есть три метода: deposit, withdraw и getBalance.

Если мы изменим параметр wAmount на balance, вы можете думать об этом как если бы вы разбили копилку и извлекли всю сумму из копилки.



При компиляции метода здесь не будет никакой ошибки.

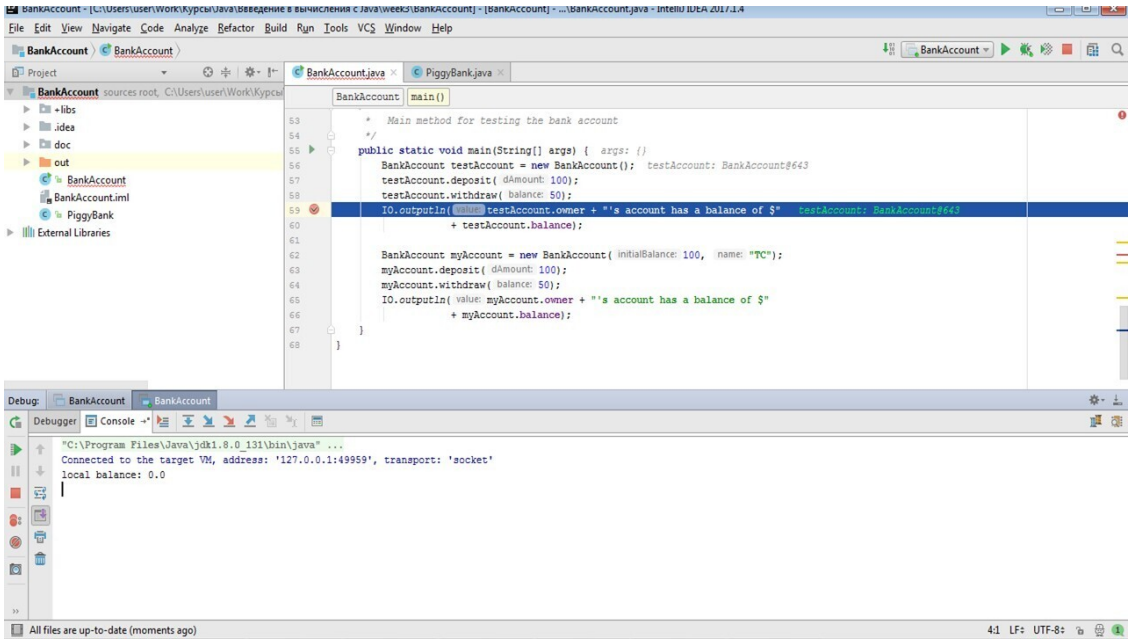
Давайте установим точку останова здесь, чтобы посмотреть, как значение баланса будет изменено внутри метода withdraw.



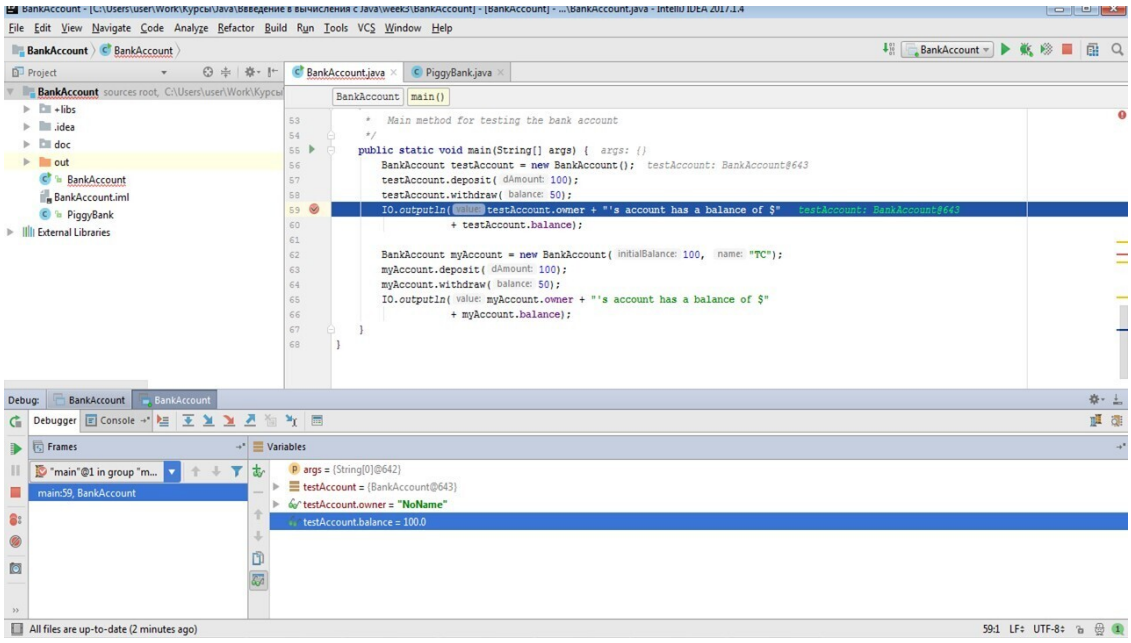
Мы можем создать экземпляр BankAccount, просто использовав конструктор по умолчанию для BankAccount.

Теперь сделаем депозит, скажем, $100.

Давайте попробуем сделать вывод той же суммы с этого счета. Скажем 100.



Здесь вы сможете увидеть, что локальная переменная balance свелась к 0.



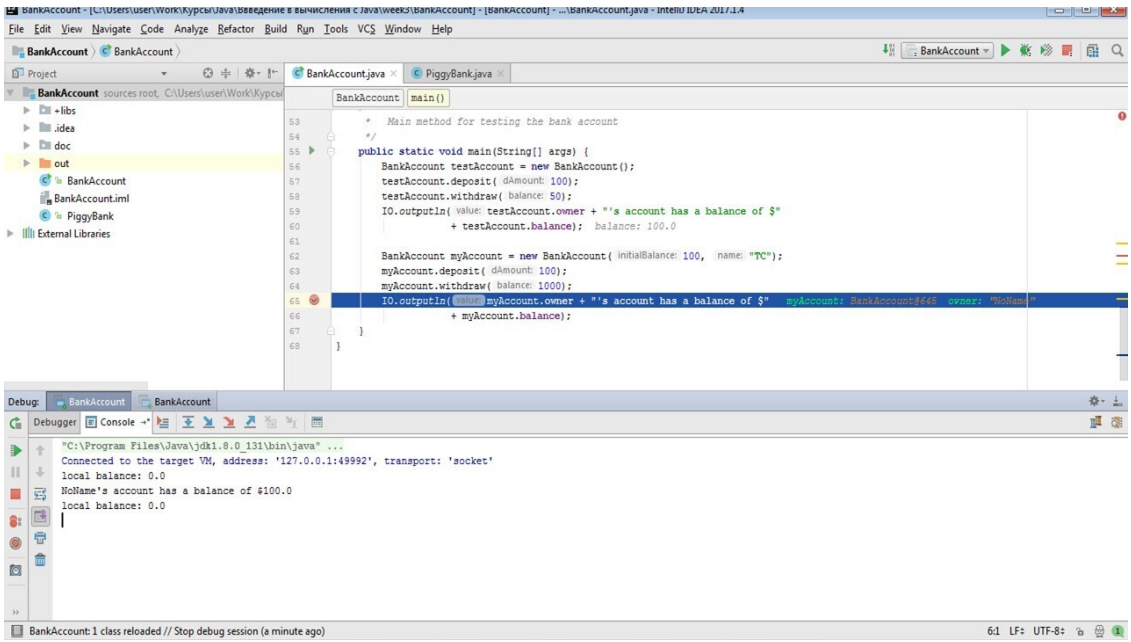
Но нет никакого эффекта для переменного экземпляра balance, и она по-прежнему имеет значение 100.

Если вы продолжите выполнение, программа заканчивается, но вы найдете в окне отладки, что переменная экземпляра balance будет по-прежнему иметь значение 100, что было бы очень хорошо, даже если вы сняли деньги со счета.

Это никак не влияет на баланс счета.

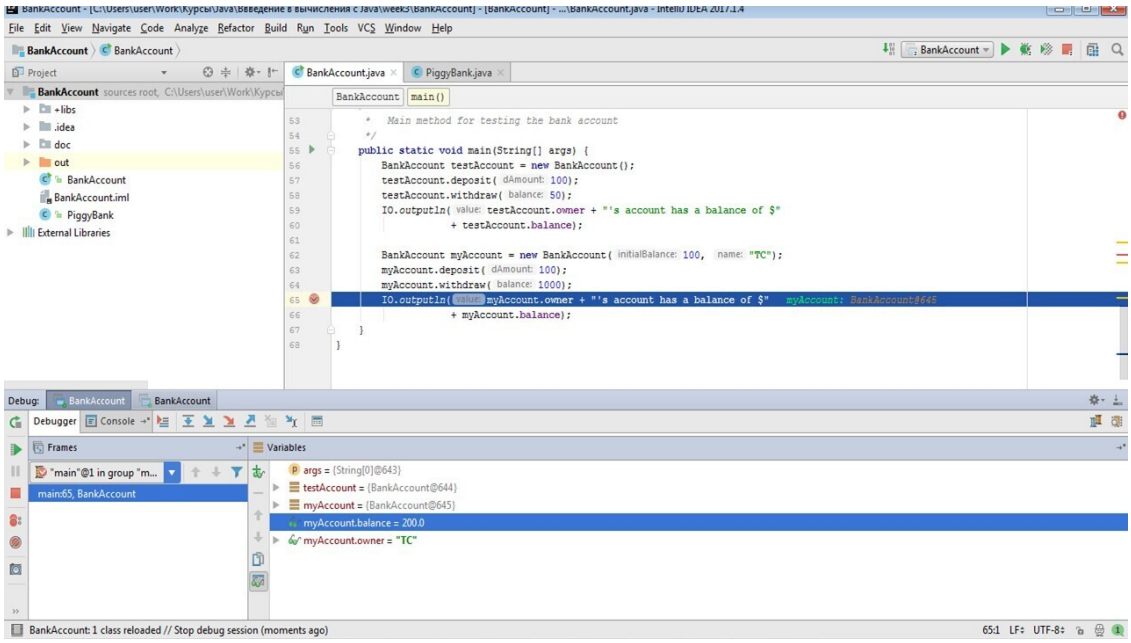
Давайте посмотрим, что, если нам просто повезло в прошлый раз, и теперь мы сделаем еще один вывод денег.

Я буду более жадным сейчас и хочу снять $1000 со счета.



Внутри метода withdraw, вы можете видеть, что локальная переменная balance теперь имеет значение 0.

Но опять же нет никакого эффекта для переменного экземпляра balance.



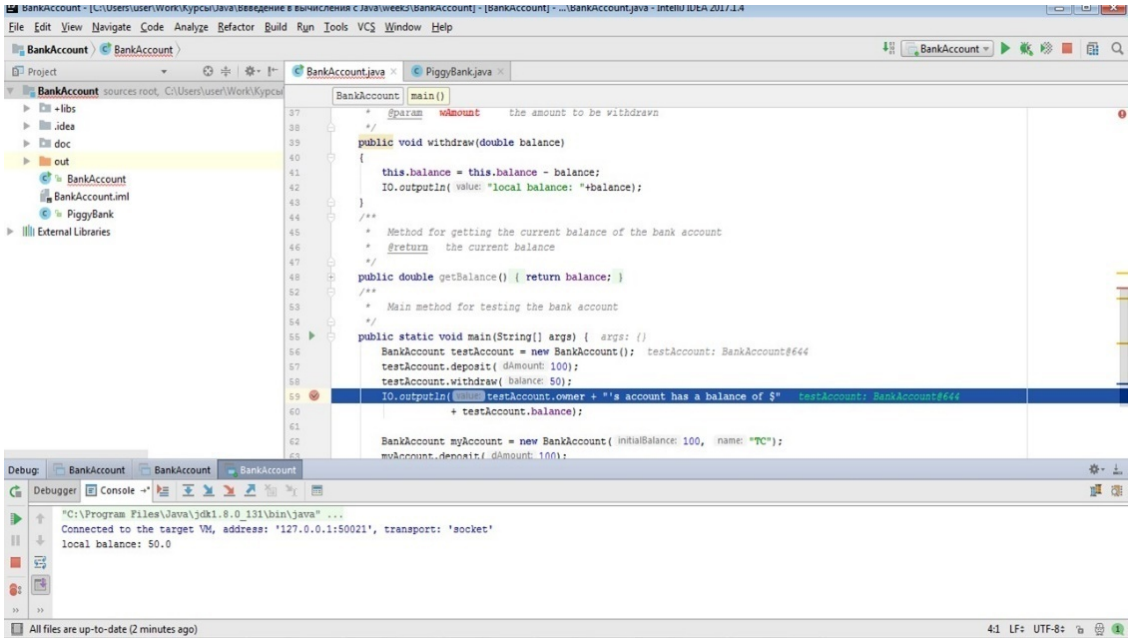
Вы закончите выполнение, и вы увидите, что переменная экземпляра по-прежнему имеет значение 200, так что должно быть что-то не так с методом.

Причина в том, что параметр balance здесь является локальным для метода withdraw, то есть, любые изменения, внесенные в такие переменные, не будут иметь никакого эффекта вне метода. Так как мы можем решить эту проблему?

Позже, я буду говорить о ключевом слове this.

Используя ключевое слово this, вы будете обращаться к переменной экземпляра для текущего объекта, с которым вы работаете.

Так что, если вы хотите сделать вывод денег, и вы хотели бы вычесть сумму из переменной экземпляра balance, используйте это ключевое слово.



С этой модификацией, давайте выполним компиляцию снова.

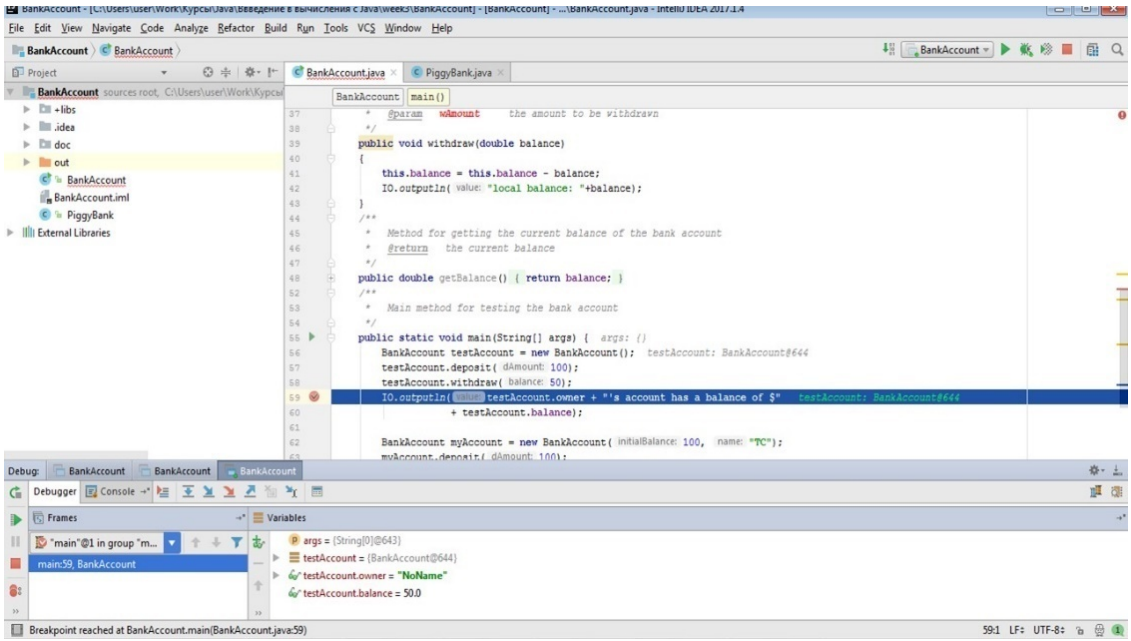
Теперь тоже нет никакой ошибки.

И снова, мы установим точку останова здесь.

Давайте опять создадим экземпляр, используя конструктор по умолчанию, и мы внесем депозит $100 на счет.

Теперь, если вы вызываете метод withdraw, давайте выведем $50.

И здесь параметр balance будет ссылкой на локальную переменную balance, в то время как this.balance будет ссылкой на переменную экземпляра balance, и результат будет присвоен переменной экземпляра balance.



Так что, теперь вы увидите, что значение локальной переменной balance не изменяется, но теперь переменная экземпляра balance сводится к 50.

Теперь программа работает правильно, но клиент банка, вероятно, не так счастлив, как прежде.

Вопросы

Задача

Что будет на выходе после компиляции и выполнения основного метода в следующей программе?

```
import comp102x.IO;

public class Quiz1 {

    private int x;

    public Quiz1(int x) {

        x = x;
    }

    public static void main(String[] args) {

        Quiz1 q1 = new Quiz1(10);
        IO.outputln(q1.x);
    }
}
```

Варианты ответа:

1. 0
2. 10
3. Compilation Error

Ответ: 1.

Объяснение

В конструкторе Quiz1, x в левой и правой части выражения x = x относится к той же локальной переменной x, объявленной в списке параметров конструктора.

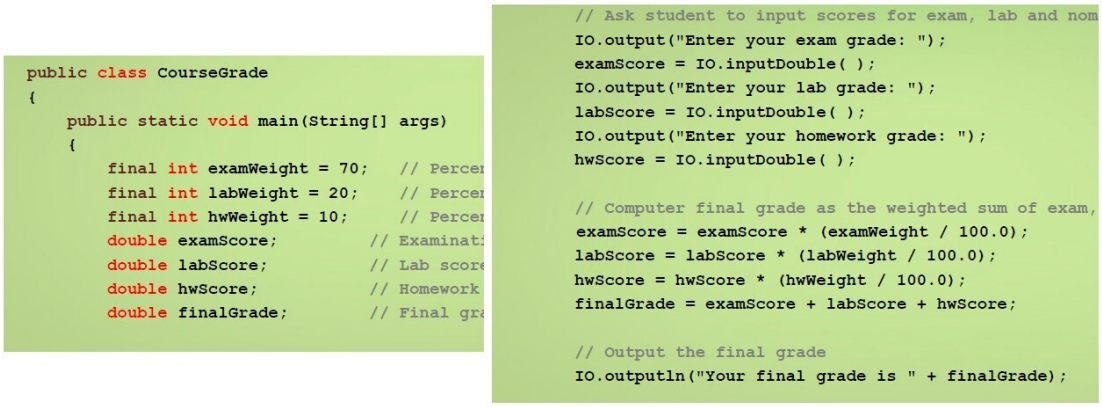
Это локальная переменная исчезает после того, как Java выполняет конструктор.

Переменная экземпляра x не изменяется вообще, и имеет начальное значение, равное нулю.

Поэтому, когда мы пытаемся вывести переменную экземпляра x объекта q1 класса Quiz1, мы получим 0, даже если мы используем конструктор со значением 10 в качестве параметра для построения объекта q1.

Пример

Теперь давайте рассмотрим пример CourseGrade, который мы обсуждали ранее.



```
public class CourseGrade
{
    public static void main(String[] args)
    {
        final int examWeight = 70;    // Percer
        final int labWeight = 20;     // Percer
        final int hwWeight = 10;      // Percer
        double examScore;          // Examinat
        double labScore;           // Lab score
        double hwScore;            // Homework
        double finalGrade;         // Final gra
```

```
        // Ask student to input scores for exam, lab and nom
        IO.output("Enter your exam grade: ");
        examScore = IO.inputDouble( );
        IO.output("Enter your lab grade: ");
        labScore = IO.inputDouble( );
        IO.output("Enter your homework grade: ");
        hwScore = IO.inputDouble( );

        // Computer final grade as the weighted sum of exam,
        examScore = examScore * (examWeight / 100.0);
        labScore = labScore * (labWeight / 100.0);
        hwScore = hwScore * (hwWeight / 100.0);
        finalGrade = examScore + labScore + hwScore;

        // Output the final grade
        IO.outputln("Your final grade is " + finalGrade);
```

Этот пример, CourseGrade, вычисляет итоговую оценку как взвешенную сумму оценок за экзамены, лабораторные работы, и домашние задания.

В этой программе, хотя мы и используем объектно-ориентированный язык в реализации программы, мы на самом деле не используем объектно-ориентированный подход.

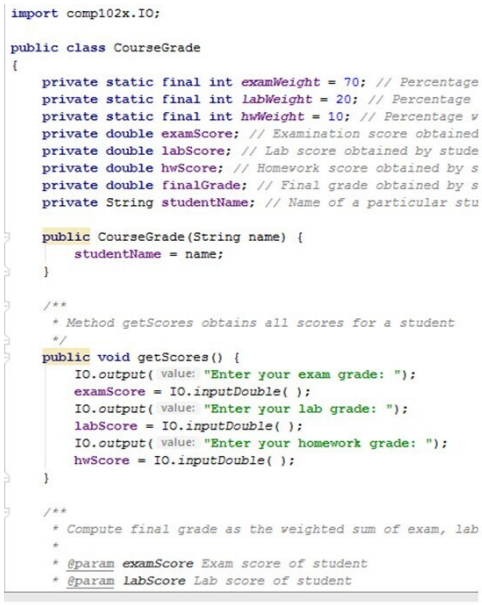
Программа состоит из одной длинной последовательности кода.

Программа также не различает оценки разных студентов.

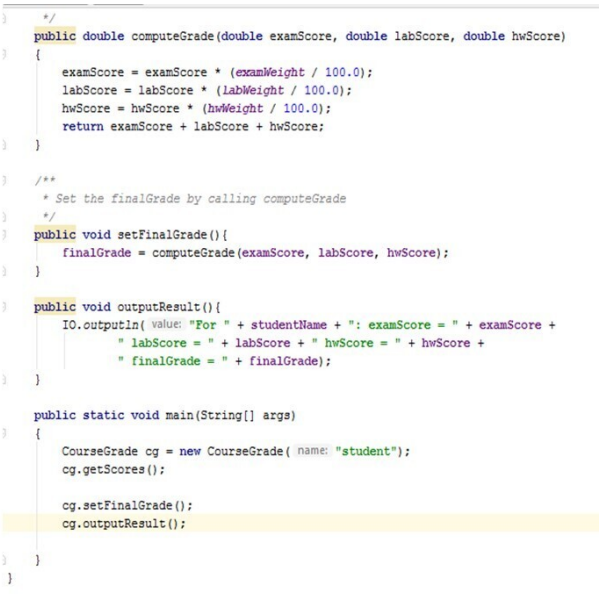
Поэтому хотелось бы, чтобы мы разработали программу, используя объектно-ориентированный подход, которая позволяла бы создавать различные экземпляры оценок для разных студентов.

Например, мы можем думать о создании карты оценок для каждого студента в классе.

Давайте изменим код класса CourseGrade.



```
import comp102x.IO;

public class CourseGrade
{
    private static final int examWeight = 70; // Percentage
    private static final int labWeight = 20; // Percentage
    private static final int hwWeight = 10; // Percentage v
    private double examScore; // Examination score obtained
    private double labScore; // Lab score obtained by stude
    private double hwScore; // Homework score obtained by s
    private double finalGrade; // Final grade obtained by s
    private String studentName; // Name of a particular stu

    public CourseGrade(String name) {
        studentName = name;
    }

    /**
     * Method getScores obtains all scores for a student
     */
    public void getScores() {
        IO.output( value: "Enter your exam grade: ");
        examScore = IO.inputDouble( );
        IO.output( value: "Enter your lab grade: ");
        labScore = IO.inputDouble( );
        IO.output( value: "Enter your homework grade: ");
        hwScore = IO.inputDouble( );
    }

    /**
     * Compute final grade as the weighted sum of exam, lab
     *
     * @param examScore Exam score of student
     * @param labScore Lab score of student
```



```
     */
    public double computeGrade(double examScore, double labScore, double hwScore)
    {
        examScore = examScore * (examWeight / 100.0);
        labScore = labScore * (labWeight / 100.0);
        hwScore = hwScore * (hwWeight / 100.0);
        return examScore + labScore + hwScore;
    }

    /**
     * Set the finalGrade by calling computeGrade
     */
    public void setFinalGrade(){
        finalGrade = computeGrade(examScore, labScore, hwScore);
    }

    public void outputResult(){
        IO.outputln( value: "For " + studentName + ": examScore = " + examScore +
                " labScore = " + labScore + " hwScore = " + hwScore +
                " finalGrade = " + finalGrade);
    }

    public static void main(String[] args)
    {
        CourseGrade cg = new CourseGrade( name: "student");
        cg.getScores();

        cg.setFinalGrade();
        cg.outputResult();

    }
}
```

Здесь есть объявления переменных, которые были у нас раньше, когда мы обсуждали весовые коэффициенты, в том числе вес экзамена, вес лабораторной, и вес домашних заданий.

И они финальные, т.е. они являются константами.

Мы также отметили, что они могут быть одинаковыми для всех студентов в том же классе.

То есть, вместо того чтобы принадлежать к определенному экземпляру класса, они на самом деле относятся ко всему классу.

В Java, это называется переменной класса, или их еще называют статическими переменными.

Ключевое слово static используется для обозначения переменных класса.

Статические переменные создаются без необходимости создания любого экземпляра.

Так вот, мы объявим вес экзамена, вес лабораторной, и вес домашних заданий как статические переменные, поставив ключевое слово static перед объявлением.

И чтобы различать разные экземпляры объектов CourseGrade, мы также вводим переменную экземпляра studentName типа String.

Она объявляется как переменная экземпляра, потому что отчет CourseGrade для одного студента может отличаться от других.

Посмотрим на декларацию конструктора для класса CourseGrade.

Конструктор просто присваивает имя студента с учетом его параметра переменной studentName экземпляра.

Затем мы объявляем метод получения всех баллов для студента.

Выражения ввода / вывода похожи на прошлую программу. Разница здесь в том, что мы группируем набор похожих действий в метод, который называется getScores.

Также здесь есть метод для вычисления итоговых оценок как взвешенная сумма оценок экзамена, лабораторных и домашних. Обратите внимание, что этот метод использует три параметра, все типа double, и возвращает значение, также типа double.

Возвращаемое значение вычисляется как выражение, указанное в return выражении.

Отметим также, что мы используем те же имена – examScore, labScore и homeworkScore для параметров, что и для переменных экземпляра.

Они не должны быть одинаковыми. Я делаю это намеренно, чтобы показать вам, как значения идентификаторов с одним и тем же именем могут быть разрешены с помощью правил видимости.

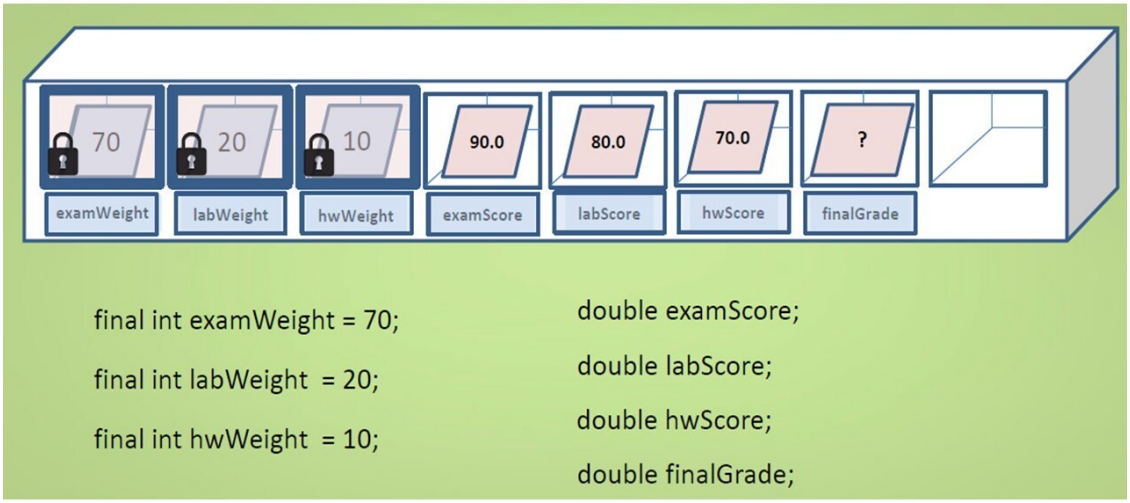
Здесь определены еще два метода.

Метод setFinalGrade вызывает метод computeGrade, который мы определили ранее, чтобы установить значение переменной finalGrade экземпляра.

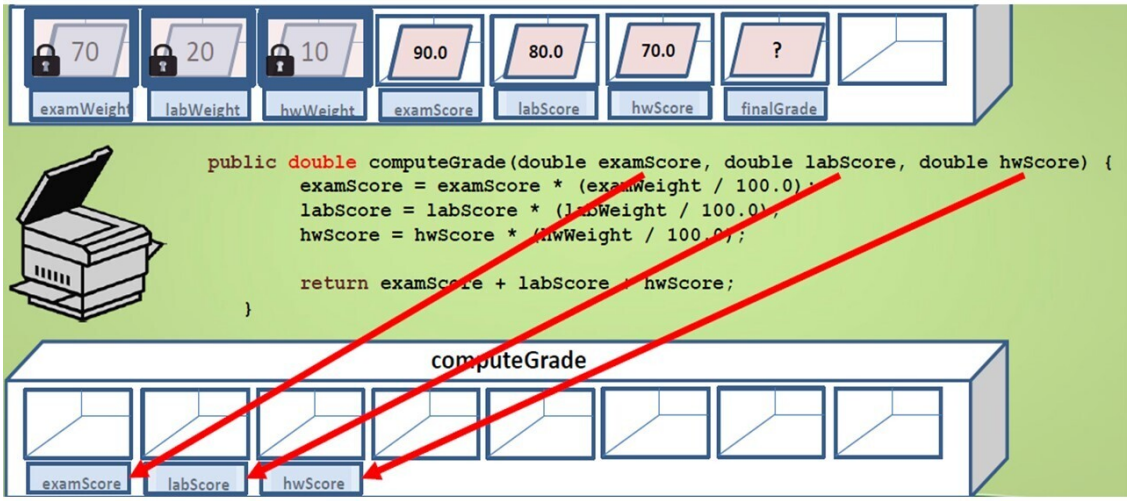
А метод outputResult просто выводит результаты всех оценок и итоговой рассчитанной оценки.

Теперь давайте более внимательно посмотрим на то, что происходит, когда метод computeGrade вызывается в методе setFinalGrade, используя переменные экземпляра examScore, labScore, homeworkScore в качестве аргументов.

Когда мы вводим переменные, мы используем схему хранения бункеров, как показано здесь для иллюстрации распределения памяти для переменных.



Давайте использовать аналогичную диаграмму, чтобы проиллюстрировать, что будет с переменными, когда вызывается метод.



При создании объекта или экземпляра, выделяется объем памяти для переменных класса и переменных экземпляра.

Здесь первые три переменные, это переменные класса, которые объявлены как константы, и следующие четыре переменные, это переменные экземпляра.

Предположим, что examScore, labScore и homeworkScore были инициализированы соответственно со значениями 90, 80 и 70 с использованием метода getScore, а метод finalGrade еще не дал никакого значения.

Давайте посмотрим на процесс, когда метод computeGrade вызывается с использованием переменных экземпляра examScore, labScore и hwScore в качестве аргументов.

Обратите внимание, что computeGrade имеет три параметра, и переменные параметров рассматриваются как один из видов локальных переменных в Java.

Когда вызывается метод, выделяется новое пространство памяти, соответствующее параметрам.

Обратите внимание, что, хотя переменные экземпляра и переменные параметров имеют одни и те же имена, они занимают разные пространства памяти.

Копии значений параметров будут размещены в соответствующих ячейках памяти для examScore, labScore и hwScore.

После того, как параметры были инициализированы, может быть выполнено первое выражение внутри тела метода.

Выражение в скобках на правой стороне оператора присваивания будет выполнено первым, и нужно будет искать значение для переменной examWeight.

Так как значение examWeight не может быть найдено внутри метода computeGrade, оно будет смотреться за пределами метода, и будет обнаружено, что ближе всего определение для переменной examWeight находится там, где она объявлена как переменная экземпляра.

Так что в качестве значения для examWeight будет использоваться число 70.

Следующая операция состоит в умножении examScore на результат деления в скобках.

Но программа найдет, что здесь появляются в программе две версии examScore, одна в качестве переменной параметра внутри метода, computeGrade, а другая как переменная экземпляра с тем же именем.

В соответствии с правилом видимости, будет поиск значения переменной, начиная с ближайшего блока, где она объявлена.

В этом случае будет использоваться переменная параметра.

Результат будет затем передан в пространство памяти, соответствующее параметру examScore, тем самым, изменяя его значение от 90 до 63.

Аналогично, второй оператор присваивания заменяет значение переменной параметра labScore с 80 на 16.

И третий оператор присваивания заменяет значение homeworkScore на 7.

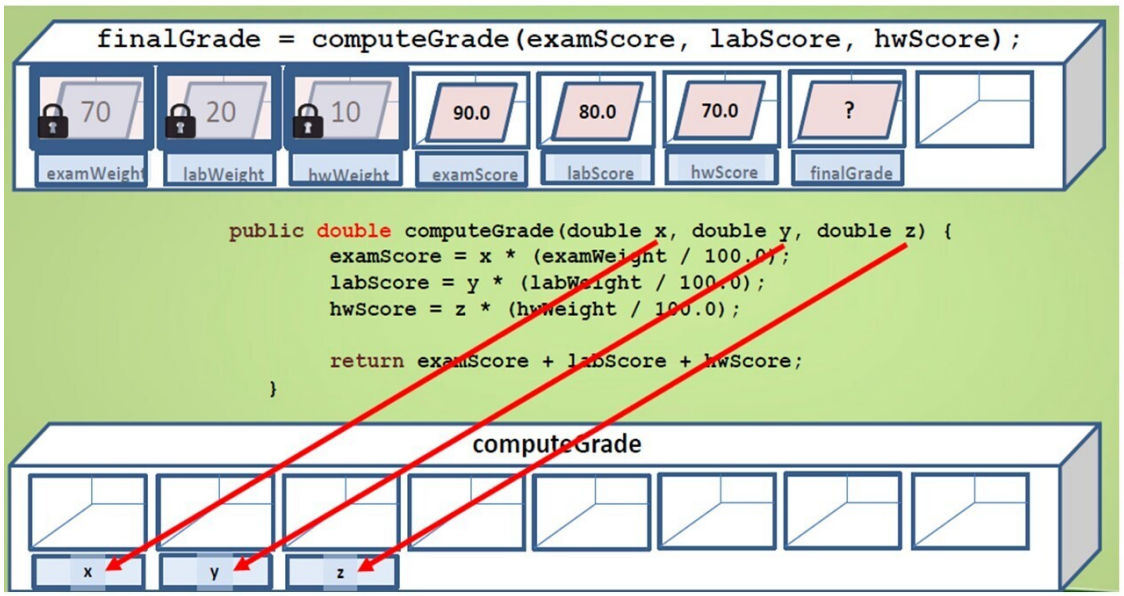
Последнее выражение в методе возвращает значение суммы всех взвешенных баллов и возвращает значение 86.

На самом деле, вы обнаружите, что метод computeGrade будет давать тот же результат, если вы замените переменные параметров examScore, labScore, hwScore набором параметров с другими названиями.

```
/**
 * Compute final grade as the weighted sum of exam, lab and homework scores
 *
 * @param   examScore   Exam score of student
 * @param   labScore    Lab score of student
 * @param   hwScore     Homework score of student
 * @return  Weighted sum of examScore, labScore and hwScore in double type
 */
public double computeGrade(double x, double y, double z)
{
     x = x * (examWeight / 100.0);
     y = y * (labWeight / 100.0);
     z = z * (hwWeight / 100.0);

     return x + y + z;
}
```

Вот пример, показывающий другое объявление computeGrade, где examScore заменяется на x, labScore на y, и hwScore на z, и результат вычислений будет таким же, используя этот модифицированный метод.

Однако, если только изменяются имена параметров, но не имена на левой стороне выражения присваивания, как показано в определении метода, эффект будет совсем другим.



Давайте посмотрим почему.

Как уже говорилось ранее, когда вызывается метод, производится выделение памяти для переменных параметров.

Так вот, области памяти выделяются для параметров с именами X, Y и Z.

Так как переменные экземпляра, examScore, labScore и hwScore используются как аргументы, значения будут скопированы в соответствующие ячейки памяти.

Когда первое выражение обращается к examWeight, будет использоваться переменная класса, как и раньше, и она получит значение 70.

Переменная x будет обращаться к переменной параметра x, но, когда присвоение результата производится в examScore, поскольку определение examScore не может быть найдено внутри метода computeGrade, оно должно искаться за пределами метода и будет найдено объявление переменной экземпляра.

Таким образом значение переменной экземпляра, examScore, будет изменено на 63 вместо 90.

Точно так же, второе выражение будет изменять значение переменной экземпляра, labScore, на 16, и значение переменной экземпляра, hwScore, будет изменено на 7.

Обратите внимание, что значения x, y, z остаются неизменными, что отличается от того, что было раньше, когда значения параметров изменялись, в то время как переменные экземпляра не были изменены.

Хотя два способа выглядят по-разному, оба метода возвращают ту же взвешенную сумму со значением 86 как правильный результат.

Давайте посмотрим на еще одно измененное определение computeGrade, чтобы проиллюстрировать использование ключевого слова this.

```
/**
 * Compute final grade as the weighted sum of exam, lab and homework scores
 *
 * To illustrate the use of the keyward "this" which can be used to make
 * reference to the current object, that is, the object whose method or
 * constructor is being called.
 *
 */
public double computeGrade(double examScore, double labScore, double hwScore)
{
     this.examScore = examScore * (examWeight / 100.0);
     this.labScore = labScore * (labWeight / 100.0);
     this.hwScore = hwScore * (hwWeight / 100.0);

     return this.examScore + this.labScore + this.hwScore;
}
```

Ключевое слово this может быть использовано в методе или конструкторе, чтобы сделать ссылку на текущий объект, который является объектом, чей метод или конструктор в настоящее время вызывается.

В этом методе ключевое слово this используется на левой стороне операторов присваивания, чтобы сделать ссылку на переменные экземпляра, вместо локальных переменных, которые указаны в качестве параметров.

Давайте посмотрим, как работает этот метод.

Как и прежде, выделение памяти и инициализация были сделаны для переменных параметров.

В этом случае, у нас есть имена examScore, labScore, и hwScore, которые такие же, как переменные экземпляра в качестве параметров.

Когда выполняется первый оператор, те же самые ссылки на examWeight и examScore были сделаны, как и раньше, но, когда дело доходит до определения значения переменной на

левой стороне оператора присваивания, this.examScore обращается к examScore для текущего объекта, что является переменной экземпляра вместо переменной параметра.

Аналогичным образом, this.labScore во втором выражении изменяет значение переменной экземпляра на 16.

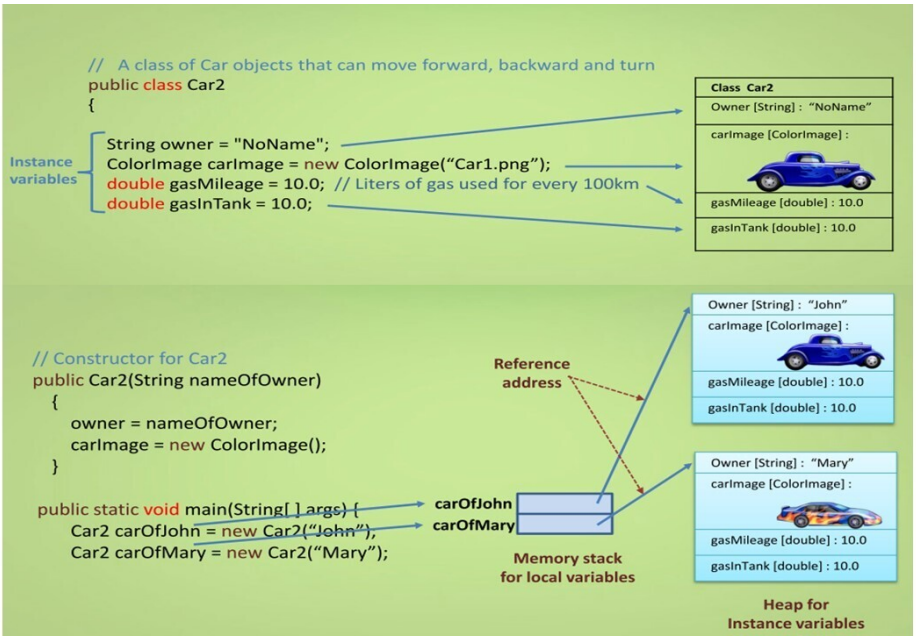
И третье выражение будет изменять переменную экземпляра hwScore на 7.

Переменные в return выражении также относятся к переменным экземпляра, и поэтому возвращается результат путем суммирования 63, 16 и 7, и возвращает значение 86.

Однако, если оператор return будет использовать переменные параметров, то есть без использования ключевого слова this, вычисление даст результат 240, который в корне неверен.

Локальные переменные, переменные класса и экземпляра

Когда мы обсуждали пример с автомобилем, я упомянул, что каждый раз, когда новый объект автомобиля создается с помощью ключевого слова new, необходимо выделить объем памяти для хранения всей необходимой информации об объекте автомобиля.



Здесь диаграмма иллюстрирует информацию, необходимую для представления объекта Car2.

Здесь есть конструктор, который мы определили для Car2.

Когда объявляется переменная Car2, будет выделен объем памяти для переменной carOfJohn.

И когда создается новый объект автомобиля carOfJohn с помощью оператора new, будет выделен объем памяти для всех переменных экземпляра, которые относятся к этому объекту.

Аналогичным образом, когда еще один экземпляр Car2 создается, будет выделен объем памяти для переменной carOfMary.

Когда объект создается с помощью оператора new, отдельное пространство памяти затем будет выделено для объекта carOfMary.

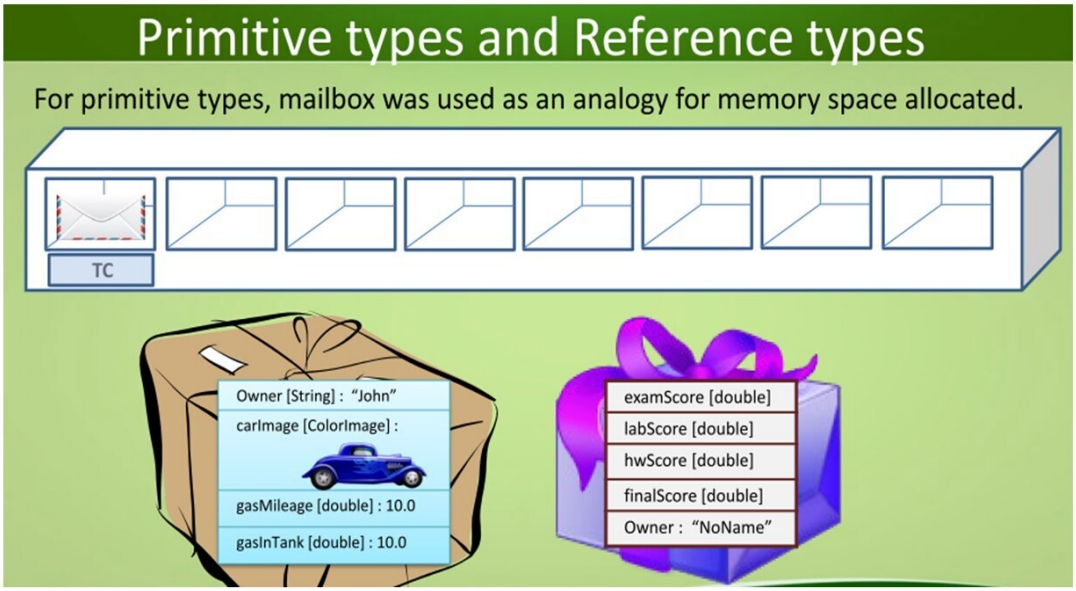
Пространство памяти для локальных переменных называется стеком, и пространство памяти для переменных экземпляра выделяется из кучи.

Обратите внимание, что вместо того, чтобы указать значения в ячейках памяти для carOfJohn и carOfMary, я использую стрелки, чтобы указать что значения, устанавливаемые в ячейках, являются ссылочными адресами.

Позвольте мне объяснить, что это значит.

Когда мы обсуждаем примитивные типы, такие как целочисленные и дробные числа, мы используем почтовый ящик в качестве аналогии для пространства памяти, выделенного для переменных, которые объявлены с примитивным типом.

Например, если объявляется переменная с именем примитивного типа, почтовый ящик будет обозначен как это имя.



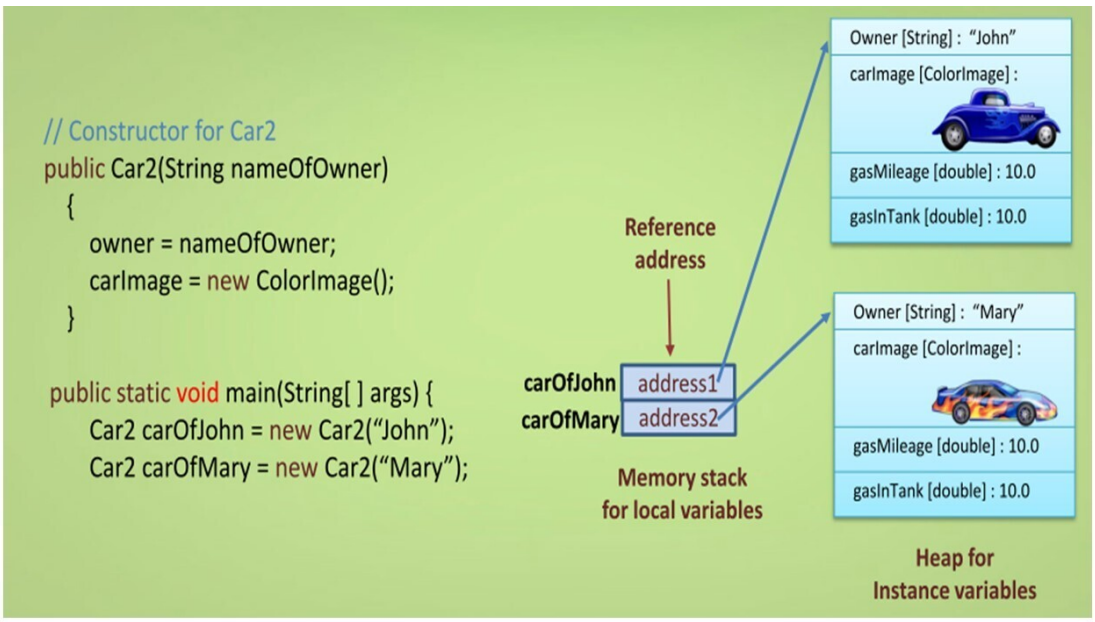
Когда переменная инициализируется с определенным значением, значение будет храниться в соответствующем ящике почты.

Такая схема выделения памяти работает, если тип имеет некоторые предопределенные размеры.

Что делать, если мы получаем большие посылки, которые не будут вписываться в почтовый ящик?

И вот что почтальон будет делать, когда посылка слишком большая, чтобы поместиться в почтовом ящике, он оставляет посылку в почтовой комнате, и оставляет записку в почтовом ящике с инструкциями о том, куда прийти за посылкой.

Таким образом, вместо того чтобы хранить информацию о переменных экземпляра вместе с примитивными типами, память, выделенная для ссылочных типов, будет хранить ссылочный адрес туда, где переменные экземпляра на самом деле находятся.



Я буду использовать таблицу, чтобы подвести итог обсуждения переменных класса, экземпляра и локальных переменных.

	Local variable	Instance variable	Class / static variable
Declaration	Inside a method/constructor or as formal parameters	Inside a class, but outside methods/constructors	Inside a class with 'static', but outside methods/constructors
Usage and Lifetime	Used by methods to hold intermediate results. Created when the method is entered and destroyed on exit.	A separate copy is created for each object using the keyword new and destroyed when no more reference is made by any object.	One single copy of variable shared by all objects is created when the program starts and destroyed when the program ends.

Example: Class of Car2	Public void moveForward(int dist) {
	....
	double gasUsed =
	Math.abs(dist) / 100.0 * gasMileage;
		String owner = "NoName";
		ColorImage carImage =
		new ColorImage("Car1.png");
		double gasMileage = 10.0;
			static final int NUMOFWHEEL = 4;

Initialization	**Must** be initialized before use.	Initialized to default values. E.g. 0 for numbers and null for objects.	Initialized to default values. E.g. 0 for numbers and null for objects.
Scope and Visibility	Visible only in the method/constructor or block in which they are declared. Access modifiers cannot be used.	Can be seen by all methods /constructors in the class. Visibility to other classes depends on access modifiers. They are often declared a private to protect them from being accidentally changed.	Same as instance variables but often declared as constants.

Локальные переменные объявляются внутри метода или конструктора.

Параметры, используемые в методе или конструкторе также локальные переменные.

Переменные класса и экземпляра объявляются внутри класса, но за пределами методов или конструкторов.

В случае переменных класса, используется ключевое слово static.

Локальные переменные объявляются, когда определяется метод. Они используются для хранения немедленных результатов, действуя как кратковременная память. И они будут уничтожены после выхода из метода.

Отдельная копия переменных экземпляра создается для каждого объекта, когда используется ключевое слово new.

Java будет освобождать память посредством процесса, называемого сбором мусора, когда больше не используется ссылка на этот объект.

Только один единственный экземпляр статической переменной создается для класса без необходимости создания экземпляра объекта. Они существуют с начала и до конца программы.

В таблице есть некоторые примеры различных типов переменных, которые могут быть использованы для автомобиля.

Мы указали в примере, что gasUsed это локальная переменная для движения вперед, и не может быть использована в другом методе, таком как addGas.

Здесь же некоторые из переменных экземпляра.

Мы можем объявить статическую переменную, NUMOFWHEEL, чтобы указать, что все автомобили имеют четыре колеса.

Она также объявлена как финальная, которая является постоянной.

Отметим также, что часто используются все заглавные буквы для именования констант.

Как уже говорилось ранее, все локальные переменные должны быть инициализированы, прежде чем они на самом деле могут быть использованы, в противном случае это приведет к ошибке компиляции.

Java обеспечивает значения по умолчанию для переменных экземпляра и переменных класса.

Все типы переменных следуют правилам видимости, которые мы только что обсудили.

Поскольку локальные переменные имеют смысл только в рамках методов, где они объявлены, модификаторы доступа не могут быть использованы для локальных переменных.

Переменные экземпляра часто объявляются как private для того, чтобы защитить их от случайного изменения.

И переменные класса часто объявляются как константы.

Обсуждение правил видимости

Что будет на выходе этой программы?

```
public class Scope1 {
    private int c =1;
    private int a = 2;
     public int one (int a, int b) {
        c = a * 2;
        a = 3 + b;
        b = b + 1;
        IO.outputln("One: " + a + "," + b + "," + c);
        return b;
    }
    public int two (int a, int b) {
        c = one (b, a);
        this.a = a + 1;
        IO.outputln("Two: " + a + "," + b + "," + c);
        return a;
   }
    public static void main(String[ ] args) {
        Scope1 s = new Scope1();
        int a = 4;
        int b = 5;
        b = s.two (a, b);
        IO.outputln("Main: " + s.a + "," + b + "," + s.c);
   }
}
```

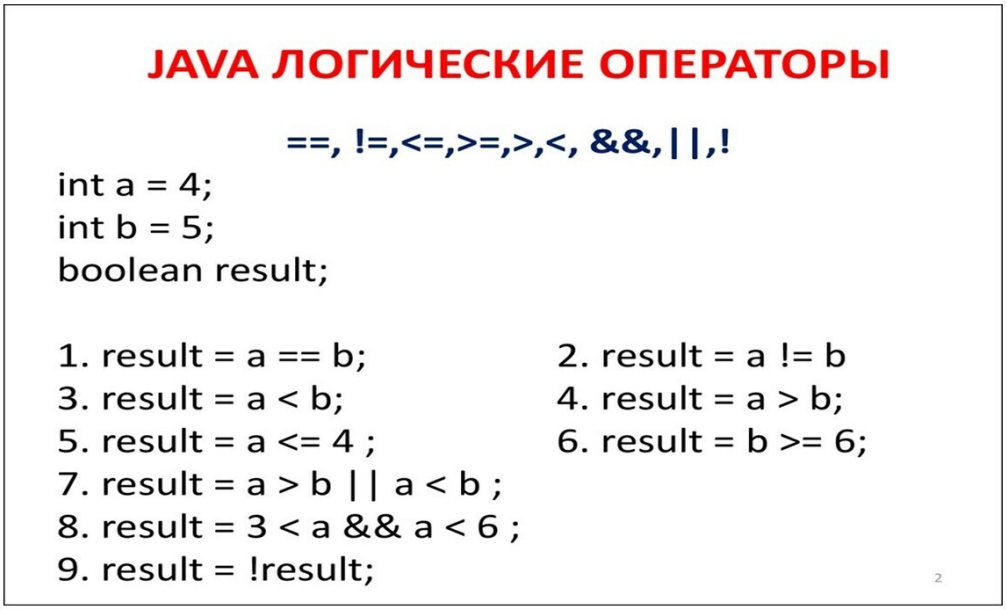
Ответ:
One: 7,5,10
Two: 4,5,5
Main: 5,4,5

Логические выражения

Давайте теперь перейдем к новой теме.

Когда мы обсуждали выражения, я упомянул, что существует два типа выражений, а именно арифметические выражения и логические выражения.

Логические выражения аналогичны арифметическим выражением, за исключением того, что булевы (логические) типы объявляются с помощью ключевого слова Boolean.



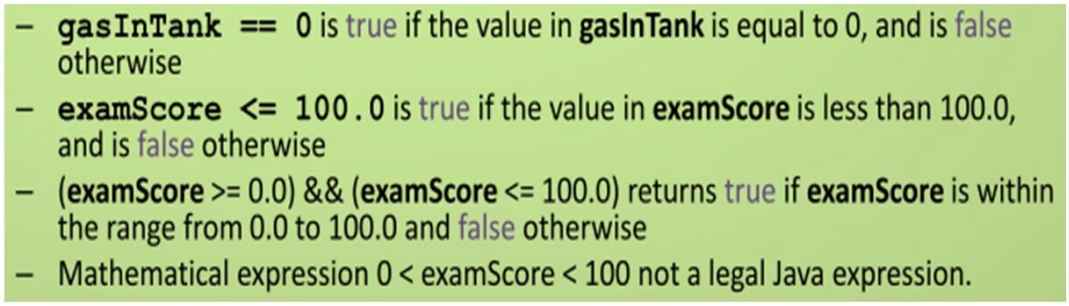
Название Boolean используется, потому что булева логика была введена в 1950 Джорджем Булем, английским математиком.

Булевы переменные имеют только два возможных значения, истина или ложь.

При этом арифметические операторы заменяются операторами отношения или условными операторами.

Прежде чем обсуждать операторы отношения или условные операторы, давайте сначала посмотрим на то, как логические выражения могут использоваться в примерах, которые мы обсуждали ранее.

В классе автомобиля, например, мы можем захотеть проверить, пуст ли топливный бак.



Выражение, gasInTank == 0 дает true, если значение gasInTank равно 0, и false в другом случае.

В примере courseGrade будет полезно проверить, чтобы вводимые баллы были в пределах диапазона.

Выражение examScore <= 100 дает true, если значение examScore меньше, чем 100, и false в другом случае.

Если мы хотим проверить как верхний диапазон, так и нижний диапазон, выражение, которое соединяет два логические выражения, использует условный оператор && (и), и возвращает true, если examScore находится в диапазоне от 0 до 100, и false в противном случае.

Обратите внимание, что для того чтобы протестировать значение examScore между 0 и 100, в математике мы можем использовать математическое выражение 0 < examScore < 100, но это незаконно в Java.

Я поговорю больше о синтаксисе, используя эти условные операторы.

Вот список из операторов отношений.

Operator	Name	Example
<	Less than	2 < 3
<=	Less than or equal to	2 <= 3
>	Greater than	2 > 3
>=	Greater than or equal to	2 >= 3
==	Equal to (Note that, you must have two consecutive '=')	2 == 3
!=	Not equal to	2 != 3

Оператор «<» – это меньше чем.

Выражение «2 < 3» будет оцениваться как true.

Оператор «<=» – это меньше или равно.

Обратите внимание, что касается вопроса порядка, Java не признает оператор «=<».

Выражение «2 <= 3» будет оцениваться как true.

Аналогичным образом, мы имеем операторы «>» и «>=».

Выражения «2 > 3», и «2 >= 3» оба дают ложь.

Оператор равенства состоит из двух последовательных знаков равенства.

Помните, что, если есть только один знак равенства, вы имеете в виду оператор присваивания, а не логический оператор равенства.

И наконец, восклицательный знак вместе со знаком равенства, это оператор не равенства.

Выражение «2 != 3» дает true.

Помните, что восклицательный знак не нужно путать с вертикальной чертой.

Мы можем использовать условные операторы или логические операторы для соединения нескольких логических выражений.

Operator	Name	Example
!	Logical NOT	! (2 == 3)
&&	Logical AND	4 < 5 && 2 < 3
||	Logical OR	4 < 5 || 2 < 3

Это три логических операторов: логическое НЕТ, которое представлено восклицательным знаком, логическое И, представленное символами амперсанда &&, и логическое ИЛИ, представленное двумя вертикальными линиями.

Значения в приведенных примерах в последнем столбце могут быть определены по таблицам истинности, которые мы будем обсуждать далее.

Таблица истинности является математической таблицей, которая используется в логике, чтобы вычислять значения логических выражений.

Вот таблица истинности для логического оператора НЕТ.

p	!p
false	true
true	false

Логический оператор НЕТ делает реверс истинному значению логических выражений, ложь станет правдой, и правда станет ложью.

Вот таблица истинности для логического И.

P	Q	P && Q
false	false	false
false	true	false
true	false	false
true	true	true

```
(examScore >= 0.0) && (examScore <= 100.0)
```

Учитывая два логические выражения, P и Q, логическое И между P и Q будет возвращать истину, если оба верны, иначе результат будет ложью.

В качестве примера, мы уже видели условия для определения, будет ли examScore находиться в диапазоне.

Вот таблица истинности для логического ИЛИ.

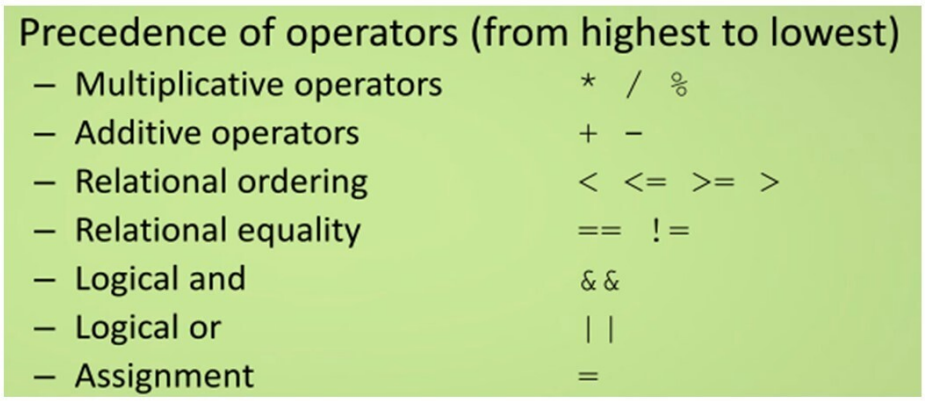
P	Q	P || Q
false	false	false
false	true	true
true	false	true
true	true	true

Учитывая два логические выражения, P и Q, логическое ИЛИ между P и Q будет возвращать ложь, только если оба ложны.

В противном случае результат будет true.

Мы уже рассматривали приоритеты операторов, когда мы обсуждали арифметические выражений.

Теперь, когда у нас есть дополнительные операторы отношения и логические операторы, вот обновленный список приоритетов операторов.



И обратите внимание, что операции отношения выполняются после умножения, деления и вычитание, но прежде, чем логические операторы.

Вопросы

Задача

Каким будет значение следующего логического выражения в Java – true или false?

0.1* 0.1 == 0.01

Ответ: false.

Объяснение

Числа с плавающей точкой не представлены точно в компьютере. Таким образом, результат "0,01", который получают путем умножения 0,1 на 0,1, отличается от "0,01", представленным 0,01 напрямую.

Если мы хотим проверить на равенство два числа с плавающей точкой, мы можем найти абсолютную разницу между двумя числами и проверить, будет ли эта разница меньше, чем порог, например, 0.0001.

В качестве примера, если мы действительно хотим проверить 0,1 * 0,1 == 0,01, можно записать логическое выражение и проверить, будет ли результат истинным или ложным.

Math.abs(0.1 * 0.1 – 0.01) < 0.0001;

Операторы ветвления

if

–

else

–

if

В повседневной жизни есть много ситуаций, которые требуют от вас принимать альтернативные решения.

В качестве простого примера, если вы находитесь в здании, и вы хотите посетить определенный этаж, один из способов сделать это – ходьба по лестнице, или использование лифта.

Если вы решили взять лифт, прежде чем идти в лифт, вы должны решить, следует ли нажать кнопку вверх или вниз, в зависимости от того, требуется ли вам двигаться вверх или вниз.

Когда вы находитесь в лифте, вы должны решить, на какую кнопку нажать в зависимости от этажа, который вы хотите посетить.

До сих пор все программы, которые вы писали, могли выполнять выражения программы только последовательно.

Но компьютерные программы должны быть достаточно умны, чтобы принимать решения или вести себя по-другому в зависимости от ситуации.

Например, когда мы проектируем класс для автомобилей, мы замечаем, что метод moveForward просто продолжает двигать автомобиль, даже если у него нет бензина.

Мы также не обрабатывали оценки, которые вне диапазона, когда мы писали нашу программу Coursegrade.

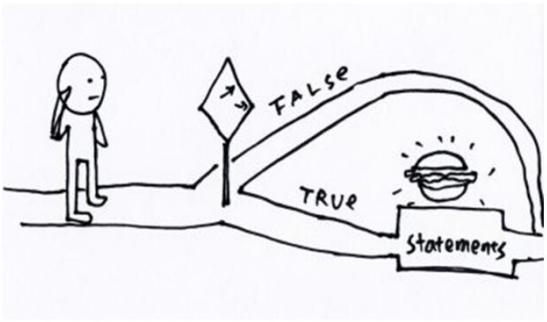
Вы должны быть в состоянии справиться с такого рода проблемами, узнав выражения ветвления.

Мы обсудим три различных типа операторов ветвления. А именно, выражения if, if-else и switch.

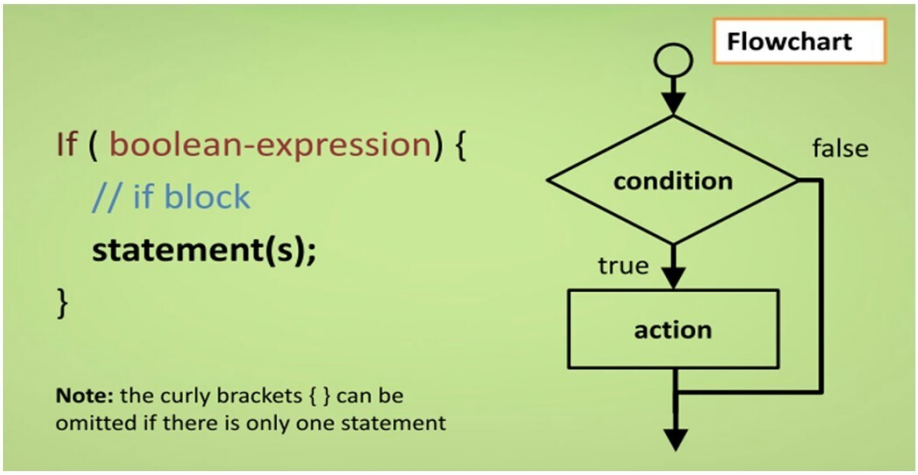
В общем, все эти выражения состоят из двух основных частей.

Первая из них является частью условия, которая проверяет выполнение условия, как правило, представленного логическим выражением.

Вторая является частью действия, которая выполняет соответствующее действие, когда условие выполнено.



Здесь показан синтаксис для выражения if.



Обратите внимание, что if это зарезервированное Java слово.

Как показано на блок-схеме справа, когда вводится if выражение, будет оцениваться условие, которое представлено логическим выражение.

Если условие истинно, тогда действие, представленное выражением, будет выполнено.

Если условие ложно, тогда блок действия пропускается.

Обратите внимание, что, если есть только одно выражение в if блоке, фигурная скобка может быть опущена.

Давайте теперь посмотрим на пример использования if выражения.

```
/**
 * The method absolute takes an integer
 * parameter & returns its absolute value
 *
 * @param value  argument whose absolute value is
 *                to be determined
 * @return       absolute value of the argument
 */
public int absolute (int value)
{
    if(value < 0)
       value = -value;

    return value;
}
```

Когда мы обсуждали пример с машиной, мы использовали абсолютные методы математической библиотеки, чтобы решить проблему для метода moveForward, когда расстояние является отрицательным.

Мы можем легко написать наш собственный метод вычисления абсолютного значения.

Этот абсолютный метод принимает один целочисленный аргумент и возвращает целое число, которое является абсолютным значением фактического аргумента в качестве параметра.

Тело программы использует if выражение, чтобы проверить меньше ли нуля значение.

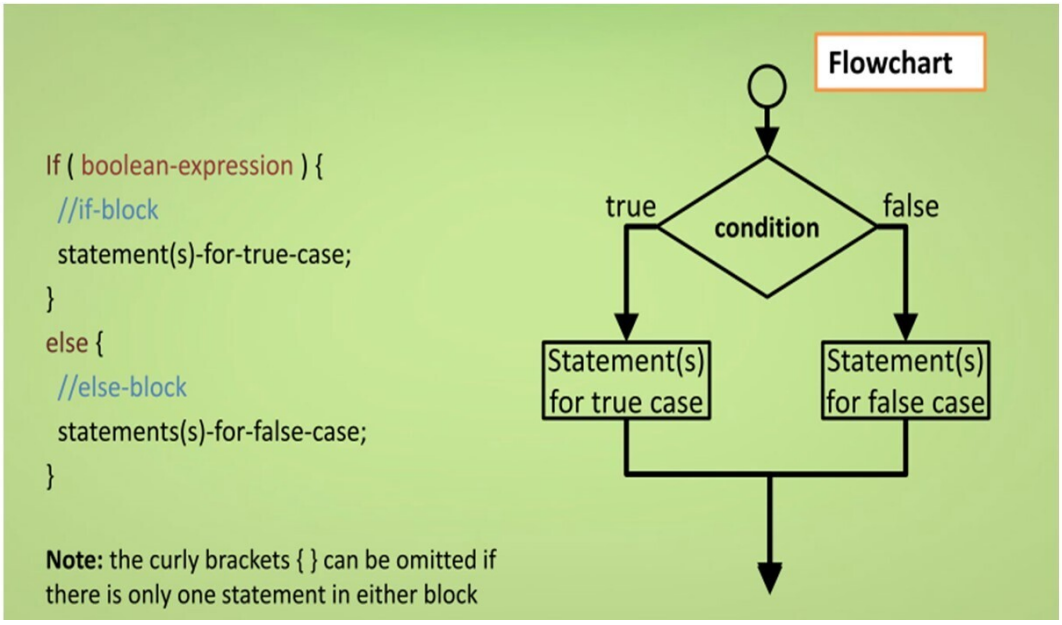
Если это так, то значение будет установлено как отрицательное от первоначального, что сделает его положительным.

Если это условие ложно, то есть исходное значение является нулем или положительным, тогда ничего и не должно быть сделано.

Метод вернет значение как результат в конце.

Теперь рассмотрим следующее выражение ветвления if-else.

Вот его синтаксис.



Первая часть if блока выглядит так же, как и раньше.

Однако здесь есть и вторая часть, которая начинается с ключевого слова else, за которым следует блок выражений.

Поток управления для if-else выражения может быть представлен в виде блок-схемы справа.

В начале if выражения сначала вычисляется условие.

Если if правда, выполняются операторы внутри if блока.

В противном случае, выполняется блок else.

Но не оба блока вместе.

Опять же, фигурные скобки можно опустить, если есть только один оператор либо в блоке if или блоке else.

Давайте сначала посмотрим на простой пример с использованием if-else выражения, а затем мы вернемся к более интересному примеру.

```
/**
 * The method larger returns the larger of two
 * integer parameters
 *
 * @param value1  first argument for comparison
 * @param value2  second argument for comparison
 * @return        larger of the two arguments
 */
public int larger (int value1, int value2)
{
    if(value1 > value2)
       return value1;
    else
       return value2;
}
```

Этот метод, названный larger, как предполагается, находит большее из двух целых чисел, которые обеспечиваются методом в качестве параметров.

Тело метода состоит из одного единственного выражения if-else.

Булево выражение проверяет, будет ли первое значение больше, чем второе значение.

Если это так, это означает, что величина одного является большей из двух, и метод просто возвращает его значение в качестве результата.

В противном случае, возвращается величина второго параметра.

Пример

Давайте посмотрим на другой пример.

```
public void moveForward(int dist)
{
    /*
     * Need to check if there is enough gas to travel the given
     * distance dist
     */

    // move the car by a distance dist in the orientation of the car
    double radian = Math.toRadians(carImage.getRotation());
    double distX = dist * Math.cos(radian);
    double distY = dist * Math.sin(radian);
    carImage.setX(carImage.getX() + (int)distX);
    carImage.setY(carImage.getY() + (int)distY);

    double gasUsed = Math.abs(dist) * gasMileage / 100.0;
    gasInTank = gasInTank - gasUsed;
    IO.outputln("Gas used:" + gasUsed + ",gas remained:" + gasInTank);
}
```

Это метод moveForward, который мы видели в классе автомобиля, после того как мы исправили направление движения вдоль текущей ориентации машины.

Когда мы обсуждали этот метод класса, я также отметил, что еще одной проблемой метода является то, что автомобиль будет просто продолжать двигаться, даже если закончится бензин.

Теперь, когда мы оснащены выражениями if и if-else, давайте исправим это с помощью проверки, достаточно ли бензина для автомобиля, чтобы покрыть расстояние, указанное в параметре.

```
/*
 * Need to check if there is enough gas to travel the given
 * distance dist
 */

// First determine the maximum distance that the car could
// travel with the amount of gas in tank.
int maxDist = (int)(gasInTank / gasMileage * 100.0);

if (Math.abs(dist) > maxDist)

  dist = maxDist;
```

Один из подходов заключается том, чтобы в первую очередь определить максимальное расстояние, которое автомобиль мог бы путешествовать с количеством бензина в баке, так как gasInTank и gasMileage оба даются как переменные экземпляра.

Максимальное расстояние можно вычислить путем деления gasInTank на gasMileage, а затем умножить его на 100.

Мы используем целый тип для максимального расстояния, потому что мы должны сопоставить пройденное расстояние в километрах числу пикселей на холсте.

И мы должны сделать приведение выражения справа, потому что там результат типа double.

Как только мы получим максимальное расстояние, тогда мы сравним расстояние, которое нужно проехать, с максимальным расстоянием.

Мы должны сделать это сравнение с помощью абсолютного значения расстояния, потому что оно может быть положительным или отрицательным.

Если абсолютная величина расстояния больше maxDist, тогда будет недостаточно бензина, чтобы закончить поездку.

В противном случае, ничего лишнего не должно быть сделано, если есть достаточно бензина.

Теперь мы можем думать о том, что следует предпринять, если там не хватает бензина, чтобы проехать данную дистанцию.

Вы можете выбрать, чтобы не совершать поездку вообще, или мы можем переместить автомобиль, пока не закончится бензин, и, надеюсь, должна быть бензоколонка на этом пути.

Возьмем второй вариант.

То есть, чтобы использовать весь бензин, чтобы переместить автомобиль на расстояние maxDist.

Другими словами, мы хотим заменить указанное расстояние на maxDist.

Будет ли это достаточно хорошо?

Ну, это если расстояние всегда положительно.

Но машина может перемещаться вперед, а может и назад, когда расстояние отрицательно.

Таким образом, мы должны проверить, если расстояние отрицательно.

Если это так, то расстояние должно быть установлено как обратное maxDist, таким образом, автомобиль будет двигаться назад, как указано.

В противном случае, расстояние устанавливается в значение maxDist.

Это может решить проблему, как описано, но, если вы рассмотрите это более тщательно, так, как if-else выражение написано, там действительно может быть две интерпретации.

Мы можем определить, что else будет спаренным со вторым if.

```
/*
 * Need to check if there is enough gas to travel the given
 * distance dist
 */

// First determine the maximum distance that the car could
// travel with the amount of gas in tank.
int maxDist = (int)(gasInTank / gasMileage * 100.0);

if (Math.abs(dist) > maxDist)

  if (dist < 0) dist = - maxDist;

  else dist = maxDist;
```

Но исходя из синтаксиса, как указано, if-else выражение еще может также идти с первым if.

```
/*
 * Need to check if there is enough gas to travel the given
 * distance dist
 */

// First determine the maximum distance that the car could
// travel with the amount of gas in tank.
int maxDist = (int)(gasInTank / gasMileage * 100.0);

if (Math.abs(dist) > maxDist)

  if (dist < 0) dist = - maxDist;

else dist = maxDist;
```

Чтобы справиться с неоднозначностью в этой задаче, мы всегда можем вставить фигурные скобки, чтобы сделать наше определение ясным.

```
/*
 * Need to check if there is enough gas to travel the given
 * distance dist
 */

// First determine the maximum distance that the car could
// travel with the amount of gas in tank.
int maxDist = (int)(gasInTank / gasMileage * 100.0);

if (Math.abs(dist) > maxDist)
{
  if (dist < 0) dist = - maxDist;
  else dist = maxDist;

  IO.outputln("Not enough gas to complete the trip!");
}
```

Теперь понятно, что значение расстояния будет заменено только если запрашиваемое расстояние больше, что maxDist.

Возможно, мы могли бы даже распечатать сообщение, чтобы показать, что не хватает бензина, чтобы завершить путешествие, как просили.

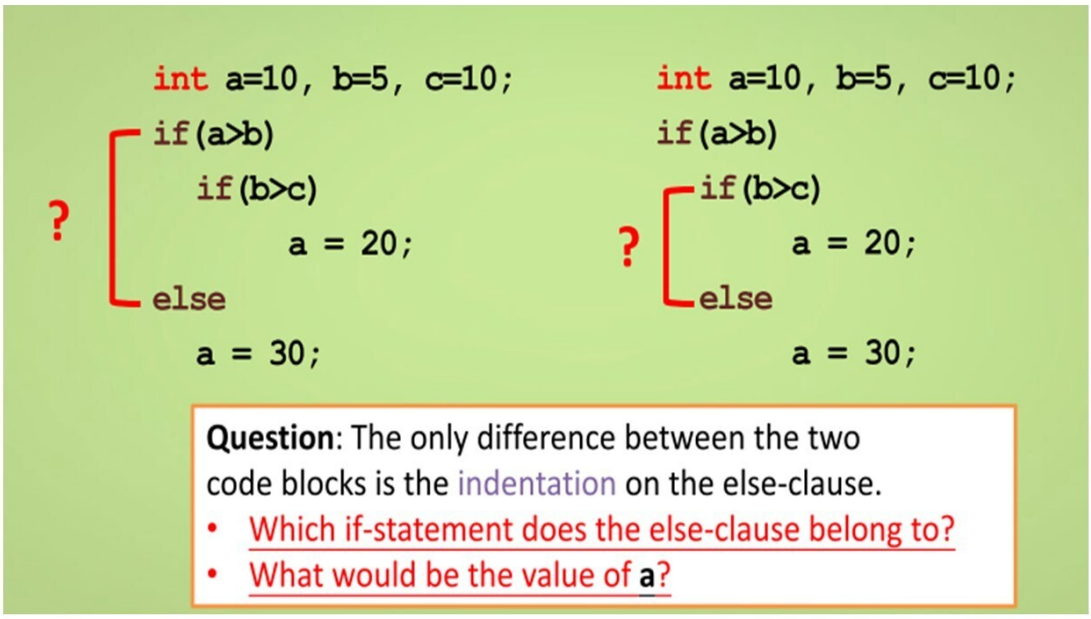
```
public void moveForward(int dist)
{
    int maxDist = (int) (gasInTank / gasMileage * 100.0);
    if (Math.abs(dist) > maxDist)
    {
      if (dist < 0) dist = - maxDist;
      else dist = maxDist;
      IO.outputln("Not enough gas to complete the trip!");
    }

    // move the car by a distance dist in the orientation of the car
    double radian = Math.toRadians(carImage.getRotation());
    double distX = dist * Math.cos(radian);
    double distY = dist * Math.sin(radian);
    carImage.setX(carImage.getX() + (int)distX);
    carImage.setY(carImage.getY() + (int)distY);
```

Так что это часть кода, который мы могли бы вставить в метод moveForward.

Выражение

switch

Для исследования проблемы else, давайте взглянем на эти два блока кода.



Единственное различие между двумя блоками является идентификация принадлежности else.

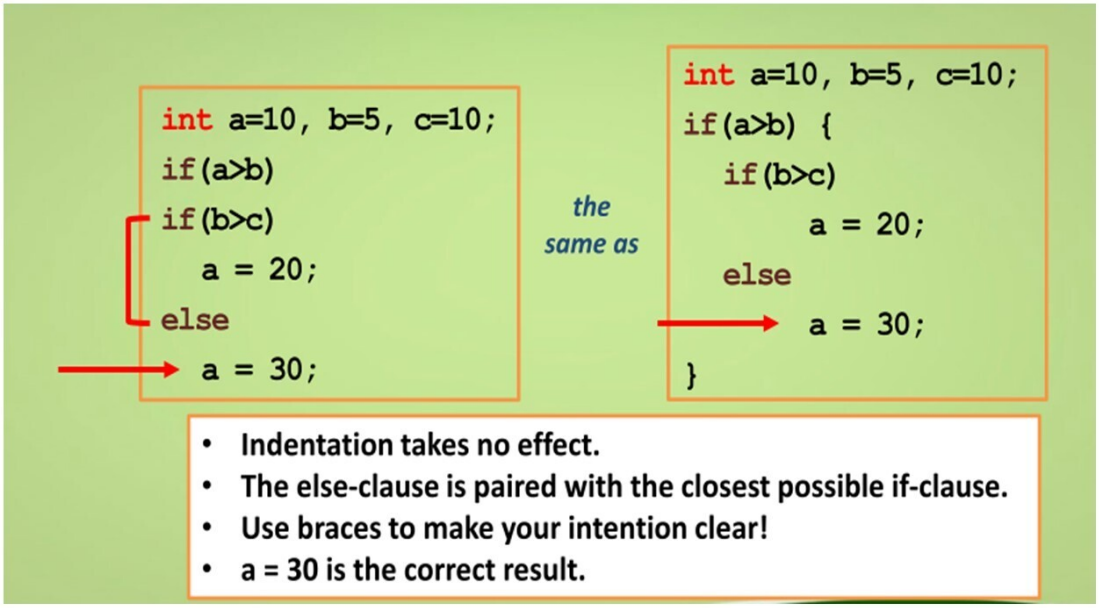
И здесь могут быть два вопроса.

Первый, к какому выражению if выражение else принадлежит?

Второй вопрос, это то, каким будет значение после оценки if выражения?

Идентификация фактически не влияет на то, как компилятор будет интерпретировать блоки кодов.

В Java, else выражение соотносится с ближайшим возможным if выражением. В этом случае, это проверка значения b.



Таким образом, здесь блок кода слева такой же, как код блока справа, с парой вставленных фигурных скобок.

Результат оценки блока кода приведет к установке значения a = 30 в конце выполнения.

Мы можем также использовать комбинацию if-else if.

```
if(score >= 90)
     Grade = 'A';
else if(score >= 80)
     Grade = 'B';
else if(score >= 70)
     Grade = 'C';
else if(score >= 50)
     Grade = 'D';
else
     Grade = 'F';
```

Пример здесь показывает, как эта комбинация может быть использована для определения вашего класса.

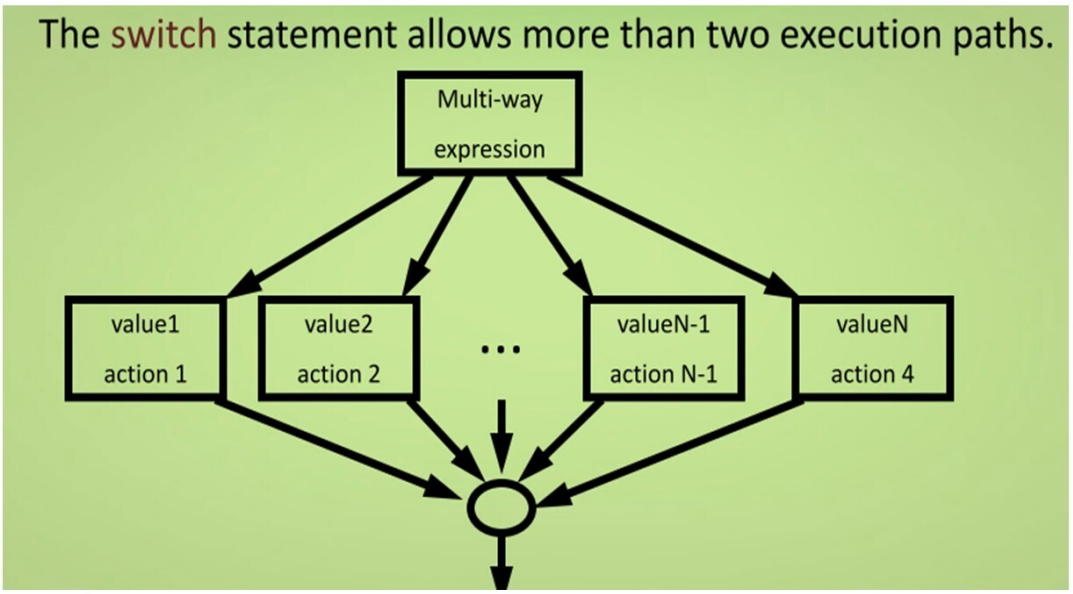
Обратите внимание, что будет иметь большое значение, если ключевое слово else остается перед if.

```
if(score >= 90)
     Grade = 'A';
if(score >= 80)
     Grade = 'B';
if(score >= 70)
     Grade = 'C';
if(score >= 50)
     Grade = 'D';
else
     Grade = 'F';
```

Сравните со случаем, когда else убрано.

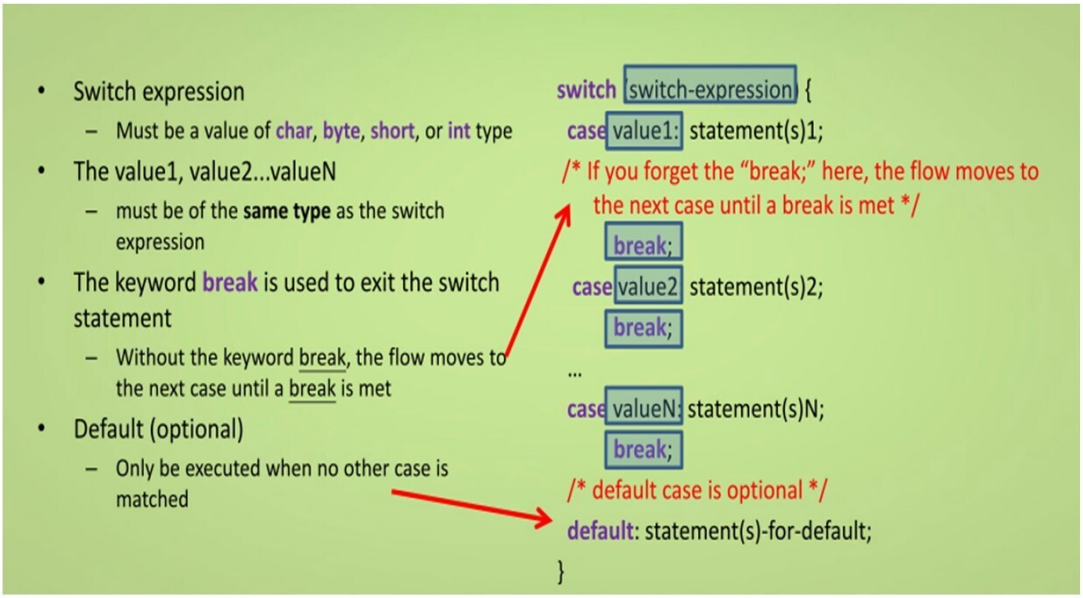
В этом случае поток выполнения прерываться не будет.

Теперь, в то время как if выражение позволяет выбрать из двух возможных путей, switch выражение позволяет более двух путей выполнения.



Вот диаграмма для иллюстрации потока управления switch выражения.

И вот синтаксис switch выражения.



Синтаксис switch выражения начинается с ключевого слова switch.

Выражение switch может иметь тип char, byte, short или int.

Значения case value1, value2 и т.д., должны быть того же типа, что и выражение switch.

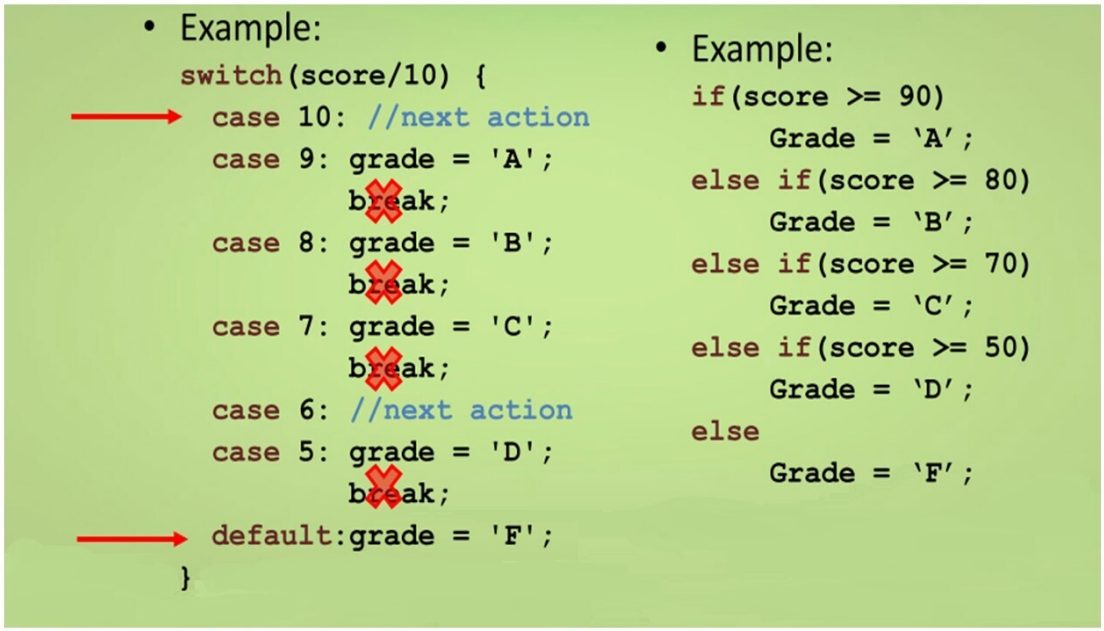
Ключевое слово break используется для выполнения switch выражения.

Важно помнить, что без break, поток будет продолжать двигаться к следующему case, пока break не будет найден.

И наконец, есть опция по умолчанию.

С ключевым словом default, эта часть кода будет выполняться только, когда никакие другие случаи не соответствуют.

Вот пример с использованием switch выражения.



Угадайте, что произойдет, если убрать все ключевые слова break?

Это будет то же самое, как если в примере if-else if убрать ключевое слово else.

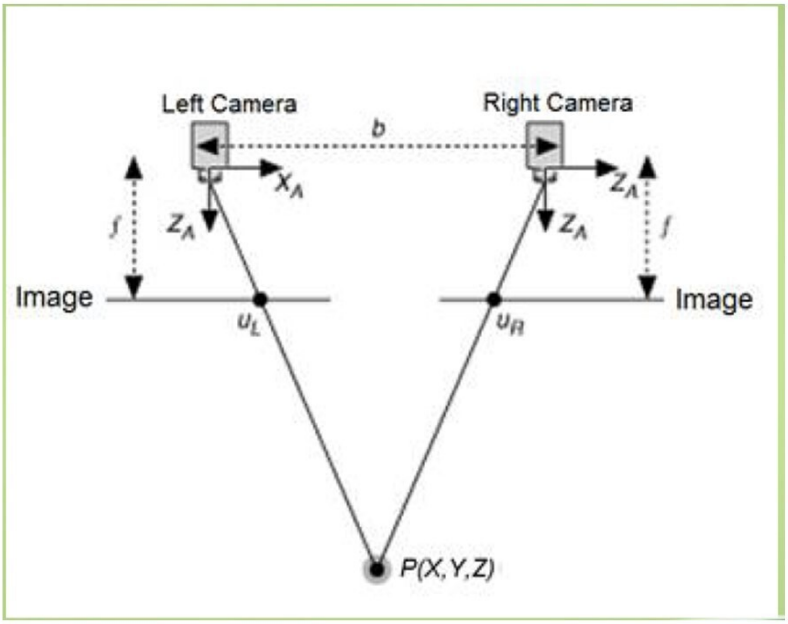
На самом деле, все, что может быть сделано с помощью switch выражения, также может быть сделано с помощью if-else выражения.

Демонстрация примера

Ранее, мы рассматривали демо-программу для создания трехмерных изображений.

В дополнение к созданию 3D-изображений сцены вокруг нас с помощью видеокамер, вы также можете создать искусственно 3D-изображения из 2D-изображений.

Идея состоит в том, чтобы разместить 2 копии одного и того же изображения с небольшим горизонтальным смещением, которое часто называют стерео-диспропорцией.



Поскольку наши два глаза ориентированы более или менее вдоль горизонтальной оси, смещение должно быть ориентировано горизонтально и смещение не может быть слишком большим, в противном случае, наши глаза не смогут легко слить изображения.

Затем мы извлекаем красный канал из левого изображения и синий канал из правого изображения, и добавив их, вы сможете создать 3D изображение, которое всплывает над фоном.

Вот иллюстрация того процесса, который я только что описал.

Если два изображения ставятся рядом, после извлечения красного канала из левого изображения и синего канала из правого изображения, добавляя их, вы должны быть в состоянии видеть плавающее полученное изображение над фоном, используя красно-синие 3D-очки.

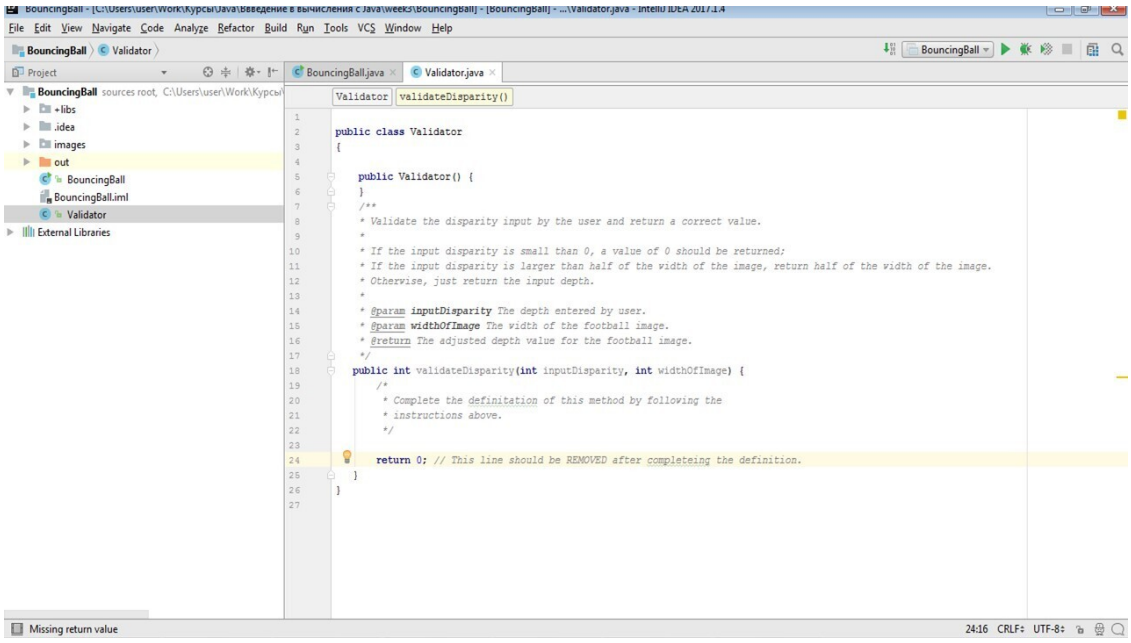
Это результирующее 3D-изображение.

Если вы используете красно-синие очки, как те, которые я уже обсуждал с вами, вы должны увидеть изображение, плавающее над фоном.

Используя различные стерео-диспропорции, вы можете увидеть 3D-изображения с разной глубиной.

Для того чтобы сделать это, вам на самом деле нужно манипулировать пикселями, используя циклы и массивы, которые будут обсуждаться позже.

А сейчас я хочу, чтобы вы завершили простой метод, который просто проверяет, что стерео-диспропорция, введенная пользователем, находится в пределах определенного диапазона, так что разница не слишком велика для полученного изображения, чтобы быть слитым нашими глазами, при этом мы будем использовать выражения ветвления, которые мы только что обсудили.



Это похоже на проверку диапазона баллов для программы CourseGrade, или проверку того, что у вас есть достаточный баланс банковского счета для снятия денег, или проверку на достаточность бензина в вашем автомобиле, чтобы выполнить требуемую поездку.

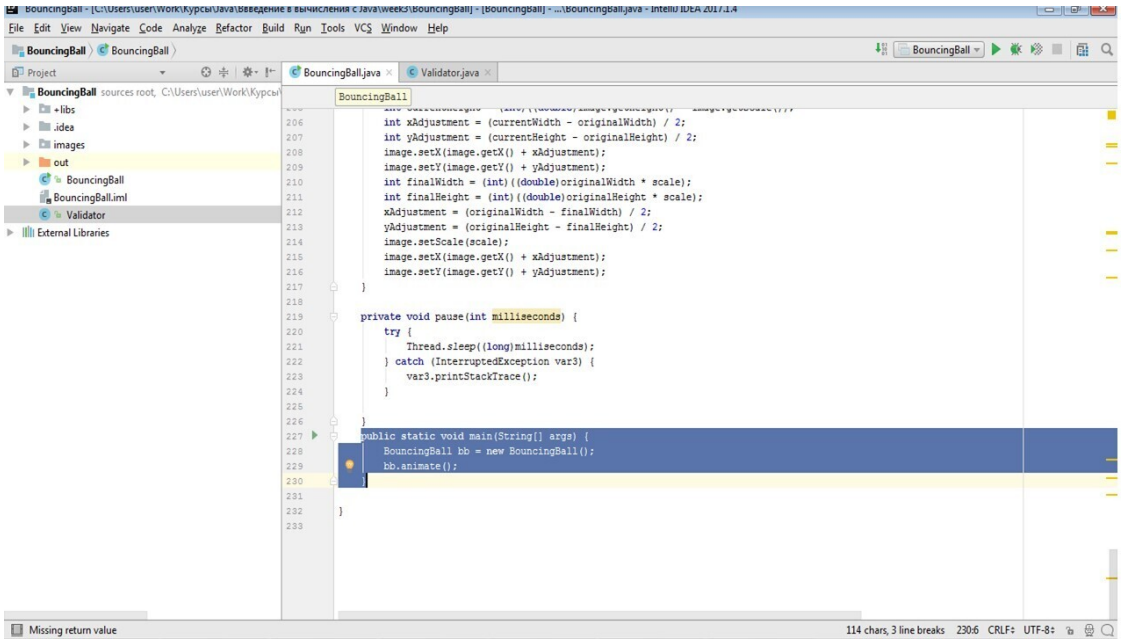
В частности, мы хотим определить диапазон между 0 и половиной ширины изображения.

То есть, если на входе меньше нуля, метод будет устанавливать значение стерео-диспропорции 0, и вы не увидите 3D-эффект.

Если на входе больше, чем половина ширины изображения, стерео-диспропорция устанавливается на половину ширины изображения.

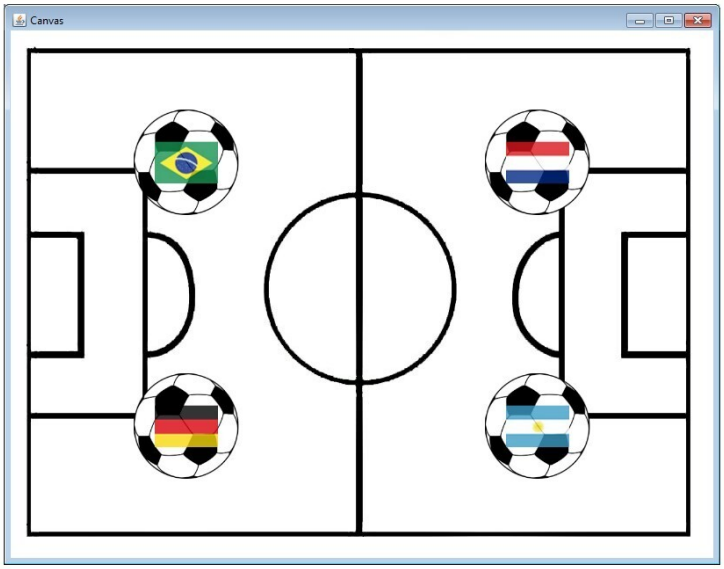
После завершения написания метода, вы можете скомпилировать проект.

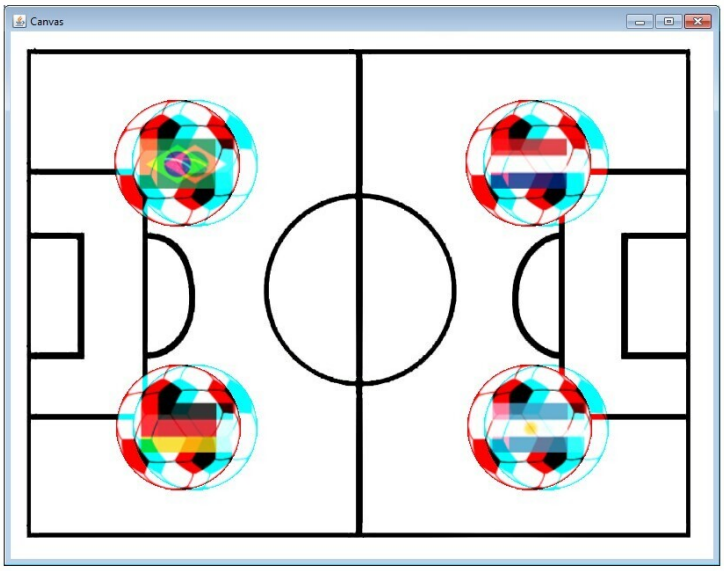
Затем можно создать экземпляр класса объекта класса BouncingBall.



Вы увидите мяч, который отображается на футбольное поле.

Из экземпляра вы можете выбрать метод анимации. Он попросит вас ввести значение стерео-диспропорции.





Давайте сначала попробуем 20, и вы увидите анимацию.

Если вы наденете красно-синие 3D-очки, вы должны увидеть, что футбольные мячи движутся к вам.

Давайте попробуем большую диспропорцию, скажем 200, которая больше, чем ширина изображения.

Программа, как предполагается, вернется к значению половины ширины изображения.

И вы сможете увидеть, что изображение движется выше, чем раньше.

Позже, вы должны быть в состоянии создавать анимацию как эту самостоятельно.

Вопросы

Задача

Учитывая следующий сегмент кода, каким было бы окончательное значение x, если изначально:

i) x = 8, y = 4, z = 2

ii) x = 2, y = 4, z = 8

iii) x = 2, y = 8, z = 4

Код:

```
if(z < y)
if(x > y)
x = 10;
else
x = 3;
```

Варианты ответа:

1. i) 10 ii) 2 iii) 3
2. i) 10 ii) 3 iii) 2
3. i) 10 ii) 3 iii) 3
4. i) 8 ii) 2 iii) 2

Ответ: 1.

Объяснение

В приведенном выше коде, else выражение должно следовать второму if выражению. Таким образом, код должен быть интерпретирован следующим образом:

```
if(z < y) {
if(x > y) {
x = 10;
} else {
x = 3;
}
}
```

В случае i) z меньше y, и x больше, чем y. Поэтому x присваивается значение 10.

В случае ii) z больше, чем y, поэтому присваивание x не производится. Поэтому x сохраняет свое первоначальное значение 2.

В случае iii) z меньше, чем y, и x меньше y. Таким образом, x присваивается значение 3.

Задача

Учитывая следующее switch выражение, каким будет окончательное значение grade, если начальное значение score составляет 52? Что будет, если начальное значение score 89?

```
int score; //initialize value according to question
char grade;
switch(score/10) {
case 10: grade = 'A';
case 9: grade = 'A'; break;
case 8: grade = 'B';
case 7: grade = 'C';
case 6: grade = 'C'; break;
case 5: grade = 'D';
default: grade = 'F';
}
```

// i) initial value of score = 52;
// ii) initial value of score = 89;
Варианты ответа:
1. i) 'A', ii) 'A'
2. i) 'D', ii) 'B'
3. i) 'F', ii) 'C'
4. i) 'F', ii) 'F'
Ответ: 3.
Объяснение
В приведенном выше switch выражении, break выражение доступно только в случае 9 и случае 6.

Поэтому значение grade будет 'A' для оценки от 100 до 90; 'C' для оценки в пределах от 89 до 60 и 'F' для оценки ниже 60.

Циклы. Введение

Ранее мы обсуждали темы области видимости и выражений ветвления.

Правила области видимости имеют важное значение для разрешения имен и ограничения области видимости переменных, так что изменения, внесенные в одной части программы, не могут случайно повлиять на другую часть программы.

Это особенно важно, когда разные команды участвуют в разработке различных частей одной комплексной программы.

Трудно полностью избежать использования идентификаторов с одним и тем же именем в разных частях программы разными программистами.

Даже в реальной жизни, вы не можете заставить разные города не использовать одни и те же популярные названия улиц.

Теперь давайте начнем обсуждение циклов.

while

do-while

for

Давайте сначала обсудим, чем являются циклы и почему циклы полезны в программировании.

Для программ, которые мы писали до сих пор, мы должны были перезапускать программу каждый раз, когда мы хотели запустить программу на другом наборе входных данных.

Я уверен, что вы не были довольны такими ограничениями.

Например, в программе CourseGrade, вы можете захотеть вычислить оценки для разных студентов, при одном выполнении программы.

В примере с автомобилем, вместо того чтобы сделать только один шаг, вы можете захотеть перемещать автомобиль непрерывно в разных направлениях.

И вместо того чтобы создавать только одно единственное трехмерное изображение, вы можете захотеть создать последовательность трехмерных изображений.

Для того, чтобы сделать это, язык программирования должен поддерживать определенные структуры управления, позволяющие выполнять набор одинаковых действий, которые будут осуществляться неоднократно.

Такого рода управляющие структуры называются циклами.

Циклы позволяют выполнять набор одинаковых действий, которые будут выполняться многократно с помощью одного и того же блока кода.

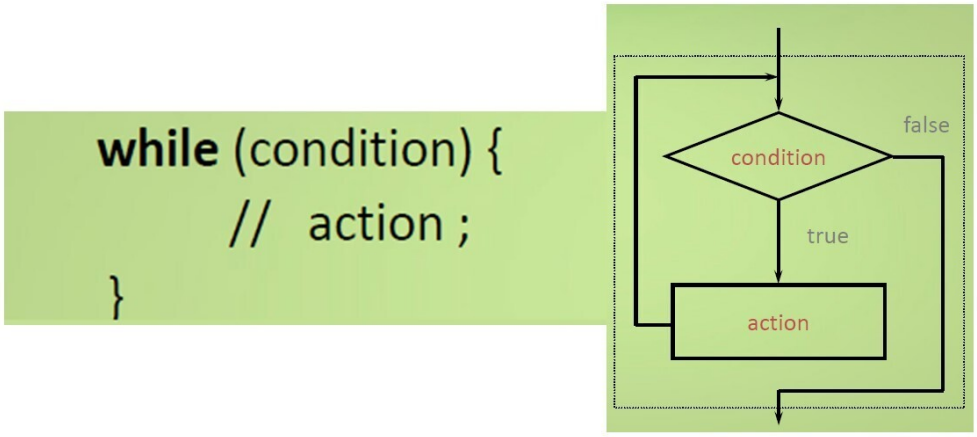
И Java поддерживает три различных типа циклов, а именно while цикл, do-while цикл и for цикл.

Я буду обсуждать каждый из них в деталях.

Подобно if else выражениям, циклы также могут быть вложенными.

И я буду использовать примеры для иллюстрации некоторых полезных применений циклов.

Давайте сначала посмотрим на while цикл.



Цикл while начинается с ключевого слова while, а затем следует выражение условия.

Цикл while выполняется, если выражение условия возвращает true, тогда действие в теле while цикла выполняется.

Действие цикла будет повторяться до тех пор, пока выражение условия не вернет false.

Действием внутри тела цикла может быть группа выражений, или только одно выражение.

Поток управления while циклом может быть проиллюстрирован на показанной блок-схеме.

Давайте посмотрим на эту блок-схему более подробно.

Когда программа входит в цикл, она сначала оценивает выражение условия. И помните, что выражение условия может быть оценено как истина или ложь.

Если условие истинно, блок кода, представляющий действие, будет выполнен.

Действие может быть представлено одним выражением или несколькими выражениями.

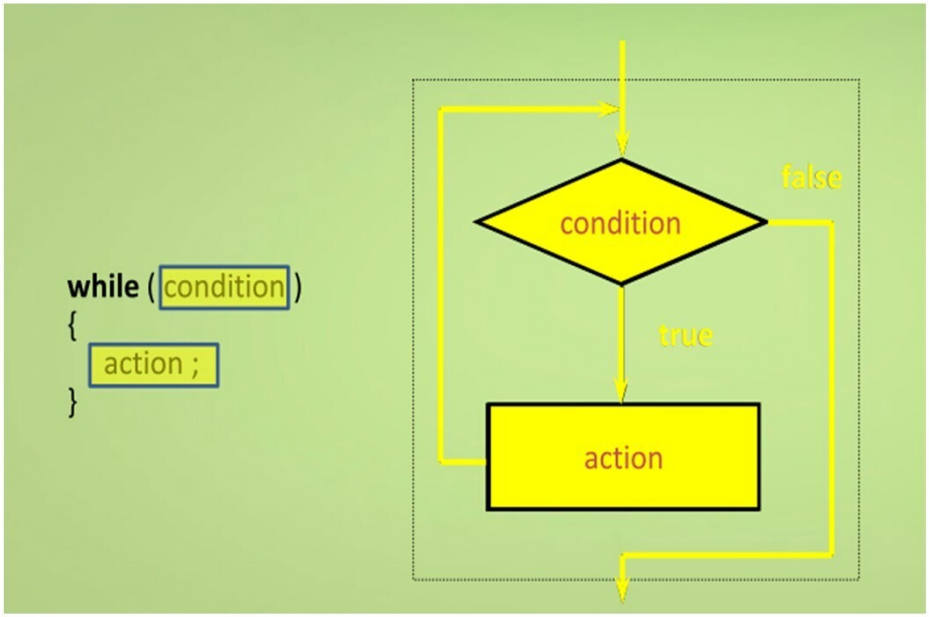
Это может быть даже еще один цикл. И я поговорю больше о вложенном цикле позже.

После окончания выполнения действия, контролируемый поток будет возвращаться в начало цикла, чтобы оценить условие снова.

Обратите внимание, что что-то должно было быть изменено, когда выполнялось действие.

В противном случае, условие всегда будет оценено в то же значение, как и раньше.

Если предположить, что условие по-прежнему возвращает true, действие будет выполнено снова, и процесс будет повторяться.



Вы можете видеть, что вся структура выделена желтым цветом и здесь сформирован цикл.

Программа выйдет из цикла, когда условие окажется ложным, и она продолжит выполнять остальную часть кода.

Примеры

Давайте посмотрим на пару простых примеров, чтобы проиллюстрировать, как циклы могут быть использованы для вычисления чего-то полезного.

Давайте рассмотрим пример вычисления n! факториала.

n! факториал прогоняет ряд способов упорядочить n различных объектов.

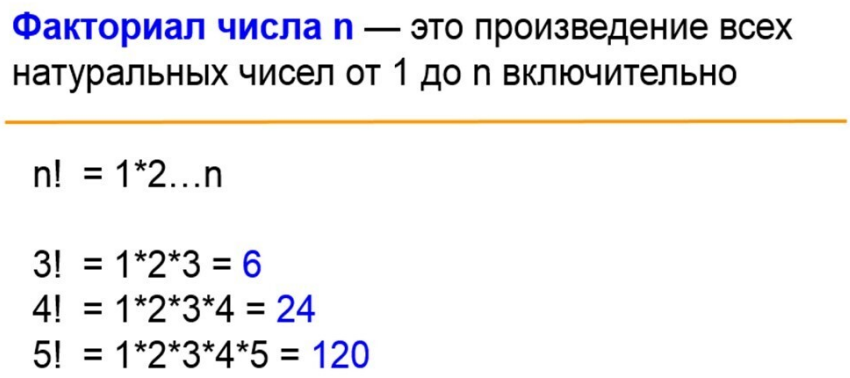
Он также называется перестановкой n различных объектов.

Например, в автомобильной гонке с участием трех автомобилей, любой из трех автомобилей может финишировать первым.

После того, как у нас есть победитель, только две машины могут конкурировать за второе место.

После того, как второе место определяется, только единственный оставшийся автомобиль станет третьим.

Итак, мы имеем 3 * 2 * 1, что равно 6 различных исходов.



Вот некоторые результаты вычисления факториала. Они перечислены здесь на картинке.

В общем, n! вычисляется путем умножения один на два, на три, и так далее, пока не достигнем n.

Также можно записать итеративную форму (n – 1)! * n.

n!
(n-1)!*n

Отметим также, что 0! имеет значение 1.

Так 5! расписывается как 1 * 2 * 3 * 4 * 5, и результат будет 120.

И n! факториал растет очень быстро.

На самом деле, вы обнаружите, что 13! переполнит 32-битный int, и даже long тип может обрабатывать только до 20 факториала.

Теперь, как n! факториал может на самом деле быть вычислен.

Вот краткий обзор алгоритма для вычисления n! факториала.

- 1! = 0!*1 = t*1
- 2! = 1!*2 = t*2
- 3! = 2!*3 = t*3
- 4! = 3!*4 = t*4
- 5! = 4!*5 = t*5

Вы можете начать с начального случая 0!, который имеет результат 1.

Обратите внимание, что в примере 5!, мы должны получить пять промежуточных результатов, а именно: 1!, 2!, 3!, 4!, а затем 5!, который будет результатом.

В общем, для n!, будет n шагов.

Давайте установим счетчик для подсчета количества шагов и инициализируем счетчик в 1. И условие для расчета установим меньше, чем или равно n, из числа шагов n.

Затем мы повторим умножение промежуточного результата t на счетчик.

Этот процесс проиллюстрирован на слайде.

Первый шаг состоит в вычислении 1!, который является t, или 0! * 1.

Затем мы должны увеличить счетчик на 1. То есть, счетчик становится 2.

Второй шаг состоит в вычислении 2! умножением t на 2 или значения счетчика.

Третий шаг состоит в вычислении 3! умножением t на 3. И 3 – это обновленное значение счетчика.

Процесс продолжается до тех пор, пока счетчик не больше, меньше или равен пяти, и конечный результат n!, будет сохранен в t.

Вот реализация n! факториала, используя while цикл.

```
public static int factorial(int number) {
  int t = 1;    // initialize t to 1
  int counter = 1; // initialize counter to 1

  while(counter <= number){
    t = t * counter;
    counter = counter + 1;
  }
  return t;
}
```

После того, как у нас есть алгоритм, перевод алгоритма в программу Java является простым.

Метод получил название факториал, который принимает один аргумент целого типа.

Имя параметра number представляет n, для которого n! должен быть вычислен.

Метод, возвращает значение десятичного типа как результат.

Как уже упоминалось ранее, вы можете изменить тип возвращаемого значения на long, если метод, как ожидается, будет вычислять факториал числа больше, чем 12.

Отметим также, что я объявляю метод как статический. И я вернусь к статическим методам немного позже.

Первые два выражения внутри метода являются инициализацией, соответствующей первой стадии алгоритма.

В то время как while цикл соответствует второму шагу алгоритма.

Обратите внимание, что условие проверяет счетчик на меньше или равно n, или параметра number.

Внутри тела while цикла, промежуточный результат t, обновляется путем умножения t на счетчик, который фактически является счетчиком факториала.

Счетчик затем увеличивается на единицу. Процесс повторяется до тех пор, пока счетчик не больше, чем number, или параметр.

А значение t затем будет возвращено в качестве результата вычислений.

Давайте посмотрим на второй пример.

В этом примере, мы хотим разработать метод вычисления 2 в n степени.

В информатике, мы должны вычислять 2 в n степени очень часто, потому что все в компьютере представлено в двоичном виде.

Например, байт это 2 в восьмой степени, а 32 бит это 2 в 32-й степени.

В общем, 2 в n степени вычисляется с помощью умножения.

$$2^n = 2^{n-1} * 2$$

Обратите внимание, что так же, как и другие числа, два в нулевой степени равно 1.

Далее 2 в 1-й степени является 2, 2 во второй степени это 4 и т.д.

Таким образом, чтобы вычислить 2 в n степени, мы должны разработать алгоритм, чтобы определить, как это значение должно быть вычислено.

Процесс на самом деле очень похож на n!.

Первый шаг состоит в том, чтобы посмотреть на первоначальный случай для первой инициализации промежуточного результата t в исходное состояние, в этом случае, 2 в нулевой степени, которая равна 1.

И установить счетчик, чтобы считать количество шагов в вычислении.

Следующий шаг будет также очень похож на n!, за исключением того, что здесь мы обновляем промежуточный результат путем умножения t на 2 вместо t на счетчик.

Вот шаги для вычисления 2 в 4-й степени.

- $2^0 = 1 = t$ (set t to 1, which is 2^0)
- $2^1 = 2^0 * 2 = t * 2$ (update t to this new value, which is 2^1)
- $2^2 = 2^1 * 2 = t * 2$ (update t to this new value, which is 2^2)
- $2^3 = 2^2 * 2 = t * 2$ (update t to this new value, which is 2^3)
- $2^4 = 2^3 * 2 = t * 2$ (update t to this new value, which is 2^4)

t сначала инициализируется 1.

Следующим шагом является вычисление 2 в 1-й степени, затем 2 во 2-й, 2 в третьей степени, и на последнем этапе вычисляется 2 в 4-й степени.

Вот реализация 2 в n степени, используя while цикл.

```
public static int powerTwo(int number) {
  int t = 1;    // initialize t to 1
  int counter = 1; // initialize counter to 1

  while(counter <= number) {
    t = t * 2;
    counter = counter + 1;
  }
  return t;
}
```

Опять же, так как у нас есть хорошо разработанный алгоритм, легко преобразовать алгоритм в программу Java.

Метод получит имя powerTwo, с одним параметром десятичного типа.

Параметр number это n, для которого должен быть вычислен 2 в n степени.

Первые два оператора внутри метода являются инициализацией, соответствующей первой стадии алгоритма. В то время как цикл соответствует второму шагу алгоритма.

Если условие счетчика меньше или равно n, условие возвращает true, тело while цикла будет умножать промежуточный результат t на 2, а затем счетчик увеличивается на единицу.

И процесс повторяется, пока счетчик не станет больше, чем number, и значение t будет затем возвращаться как результат.

Вы можете подумать о том, как сделать метод более гибким для вычисления степени любого числа. Например, 3 в седьмой степени, или 7 в пятой степени, и т.д.

Вопросы

Задача

В этом упражнении вы должны завершить свою собственную реализацию метода powerN(int, int) в классе PowerCalculator.

Этот метод вычисляет неотрицательную целую степень целого числа.

Метод принимает два десятичных параметра, число и степень.

Затем он вычисляет результат как степень числа и возвращает вычисленный результат.

Если отрицательная степень дана, метод просто возвращает 1.

Для этого упражнения предоставляется проект Java (https://github.com/novts/java-base).

И нужно организовать ввод значений для параметров – число и степень – в консоли.

Рассчитанный результат также нужно показать в консоли.

Ответ:

```
long result = 1;
while (power > 0) {
result *= number;
power–;
}
return result;
```

Статические методы

Как мы видели ранее, оба метода factorial и powerTwo были объявлены как статические методы.

И мы видели статические переменные раньше, когда мы говорили о переменных класса, используя ключевое слово static.

Переменные класса являются общими для всех объектов класса, и могут быть использованы без необходимости создания экземпляра и объекта.

Java поддерживает также статические методы.

Похожие на статические переменные, статические или методы класса объявляются с помощью ключевого слова static.

В то время как методы экземпляра связаны с объектом, статические методы связаны с классом, в котором они определены.

И подобно переменным класса, методы класса могут вызываться без необходимости создания экземпляра класса.

Они могут использоваться за пределами класса, с именем класса с помощью оператора точки, как показано здесь.

```
ClassName.staticMethodName(parameters);
```

Обратите внимание, что в то время, как методы экземпляра могут сделать ссылку на переменные экземпляра или методы, а также статические переменные или методы без явной ссылки на объект с помощью оператора точка, статические методы не могут сделать прямую ссылку на переменные экземпляра.

Тем не менее, они могут сделать ссылку на статическую переменную или другие статические методы.

Статические методы обычно используются для общих расчетов, которые применяются ко всем объектам.

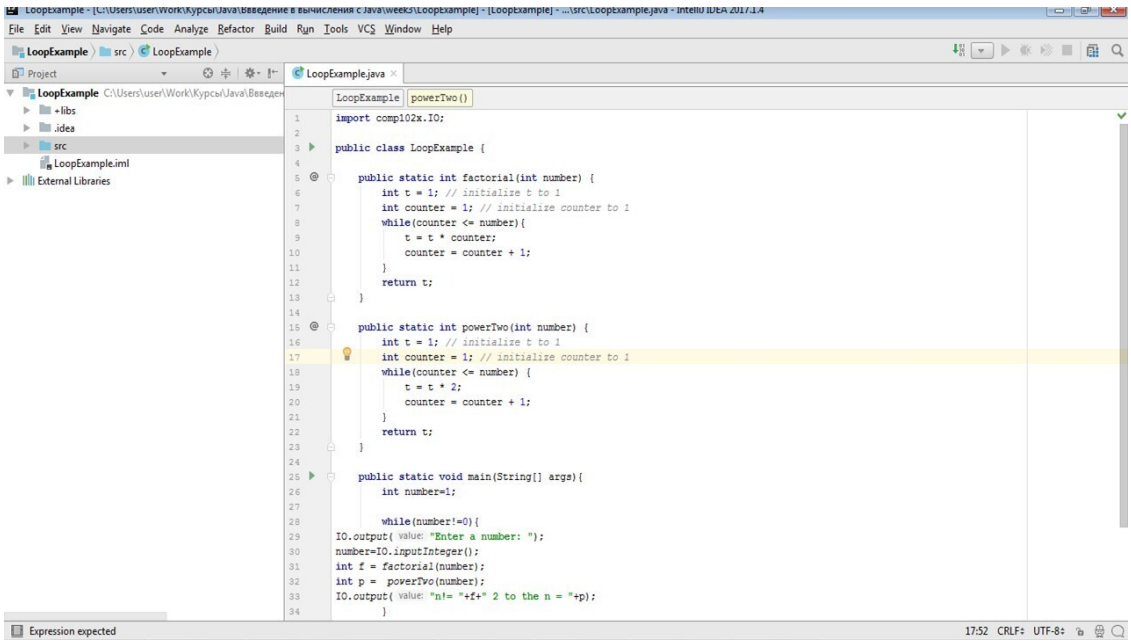
Например, все методы в библиотеке Math, объявлены как статические методы класса Math, и ссылка на метод в библиотеке Math производится с использованием имени класса Math.

Например, абсолютный метод, Math.abs(n), который мы использовали в классе автомобиля, если бы он не был объявлен как статический, мы должны были бы сначала создать объект класса Math перед вызовом метода.

Некоторые из вас, возможно, уже столкнулись с сообщением об ошибке, когда вы сделали ссылку на переменную экземпляра или метода внутри метода, который объявлен как статический.

```
"non-static variable or method cannot
be referenced from a static context"
```

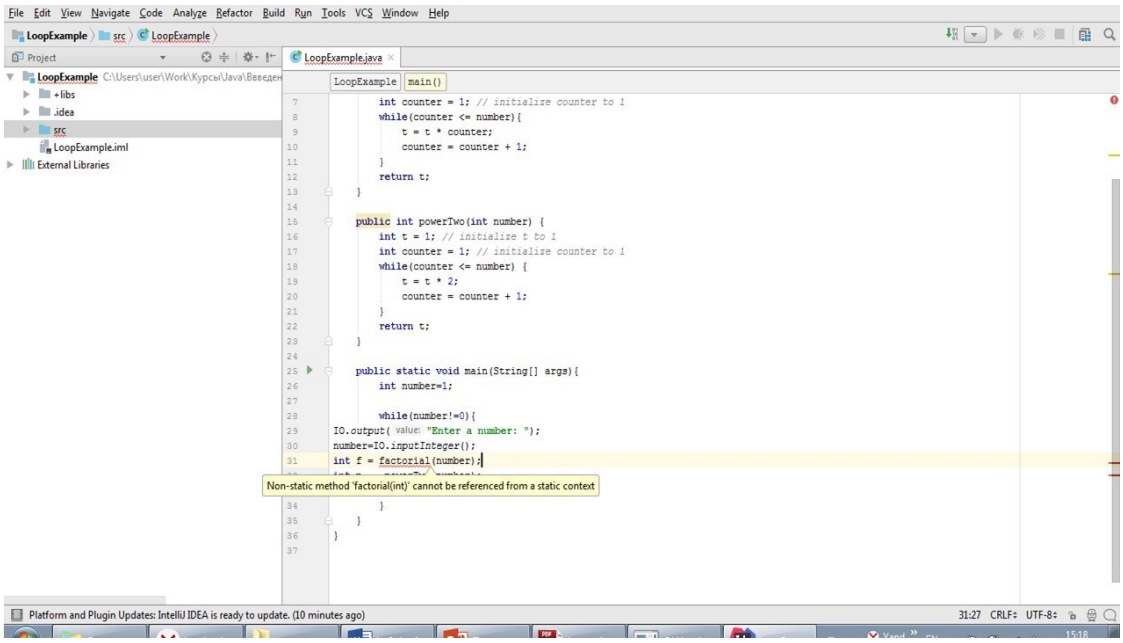
Давайте откроем среду IDEA, чтобы посмотреть на статические методы factorial и powerTwo.



Здесь я загрузил два метода factorial и powerTwo, которые мы только что обсуждали. Давайте посмотрим, что произойдет, если мы уберем ключевое слово "static".

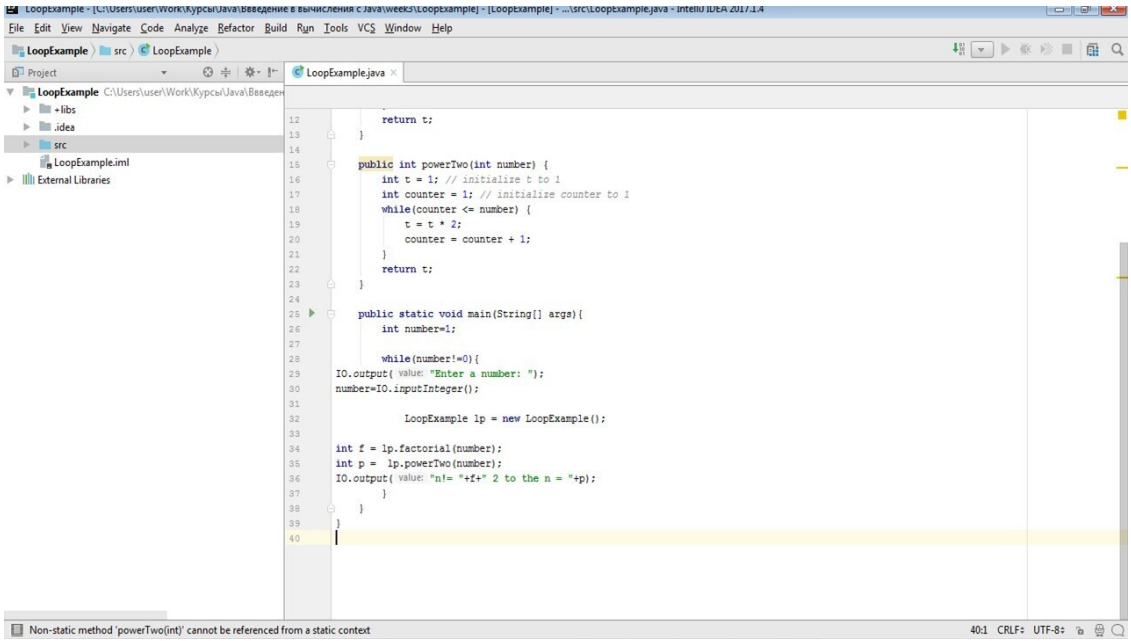
И здесь, я также написал метод main, с помощью которого можно воспользоваться двумя методами, которые мы описали.

Вы можете видеть, что здесь есть еще один while цикл, чтобы мы могли проверить эти методы на разных входных данных.



Даже не компилируя программу, вы увидите, что появится замечание о неверном использовании factorial и powerTwo методов, потому что вы не можете использовать не статические методы в методе main, так как он объявлен как статический.

И еще один способ использовать не статический метод в статическом методе, это на самом деле создать экземпляр объекта.



Например, здесь, если мы определяем объект lp с помощью оператора new, тогда вы можете обратиться к методу factorial с помощью оператора точки.

В этом случае метод компилируется, но тогда lp здесь бессмысленный объект.

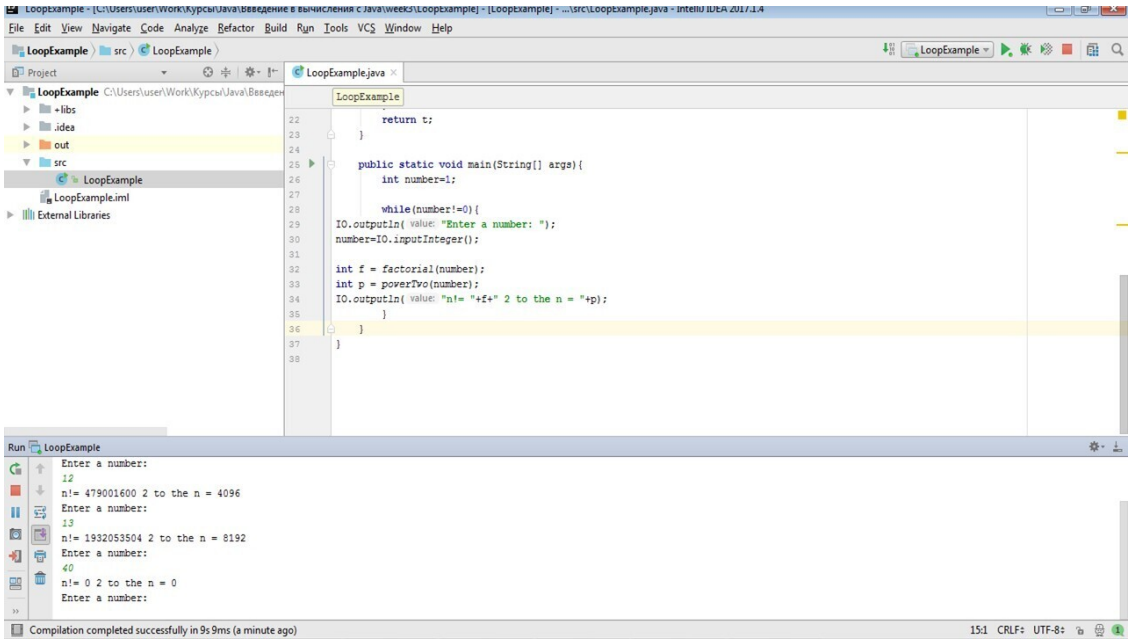
Так что давайте вернемся назад и изменим метод обратно в статический. И программа скомпилируется без проблем.

Давайте теперь попробуем запустить программу.

Программа попросит входные данные. И давайте попробуем вычислить факториал и 2 в n степени.

Я хочу проиллюстрировать, что если у вас есть 12!, вы получите результат, который должен быть правильным.

Но как только мы вводим большее число, скажем 13!, вы обнаружите, что результат здесь не 13!.



Или даже если вы попытаетесь ввести еще большее число, можно увидеть, что для 40! который, как предполагается намного больше, чем 13!, результат будет явно неправильным.

И мы можем ввести 0 здесь, чтобы остановить выполнение программы.

Как вы можете здесь видеть, в методе main условием является проверка, является ли число не равным 0.

Сочетания операторов

Прежде чем мы перейдем к обсуждению двух других структур цикла, рассмотрим сначала пару опций, предоставляемых Java, которые часто полезны в циклах.

Первая опция, это операторы увеличения и уменьшения, представленные двумя последовательными знаками «++» и двумя последовательными знаками «–».

Операторы увеличения и уменьшения были введены в язык программирования C. Это одна из причин, почему более поздняя версия C называется C ++.

Оператор инкремента, ++, увеличивает значение целого числа на 1, а оператор декремента, – -, приведет к уменьшению числа на 1.

```
++a  --a

a++  a--
```

В случае оператора инкремента, для целого a, выражение ++a увеличит значение a на 1, и полученное значение приращения будет передано как значение выражения ++a.

Что касается выражения a++, то здесь разница в том, чтобы поставить оператор ++ после целой переменной a, и выражение a++ будет использовать начальное значение a как значение выражения, и это значение будет увеличено на 1.

Давайте используем таблицу, чтобы суммировать значение и эффект операторов увеличения и уменьшения.

Operator	Name	Description
++a {y = ++a;}	Pre-increment	Increase a by 1 and then use the value of a in the assignment {a = a + 1; y = a;}
a++ {y = a++;}	Post-increment	Use the initial value of a in the assignment and then increase a by 1 {y = a; a = a + 1;}
--a {y = --a;}	Pre-decrement	Decrease a by 1 and then use the value of a in the assignment {a = a - 1; y = a;}
a-- {y = a--;}	Post-decrement	Use the initial value of a in the assignment then decrease a by 1 {y = a; a = a - 1;}

Первое выражение, ++a, сначала будет увеличивать a на 1.

Если выражение используется вместе с оператором присваивания, увеличенное значение затем будет использоваться в присваивании.

Заметим, что это процесс в два этапа, во-первых, увеличиваем, а затем присваиваем полученное значение y.

Второе выражение, a++, которое представляет собой оператор пост-инкремента.

Это выражение будет сначала использовать начальное значение a в присвоении до увеличения на 1.

Обратите внимание на разницу между этим и предыдущим выражением, значение y будет отличаться в них, в то время как результирующее значение a будет таким же, в обоих случаях.

Третье выражение, –a, которое представляет собой оператор пре-декремента подобно оператору пре-инкремента.

––a сначала уменьшает a на 1, и результат будет затем передан y.

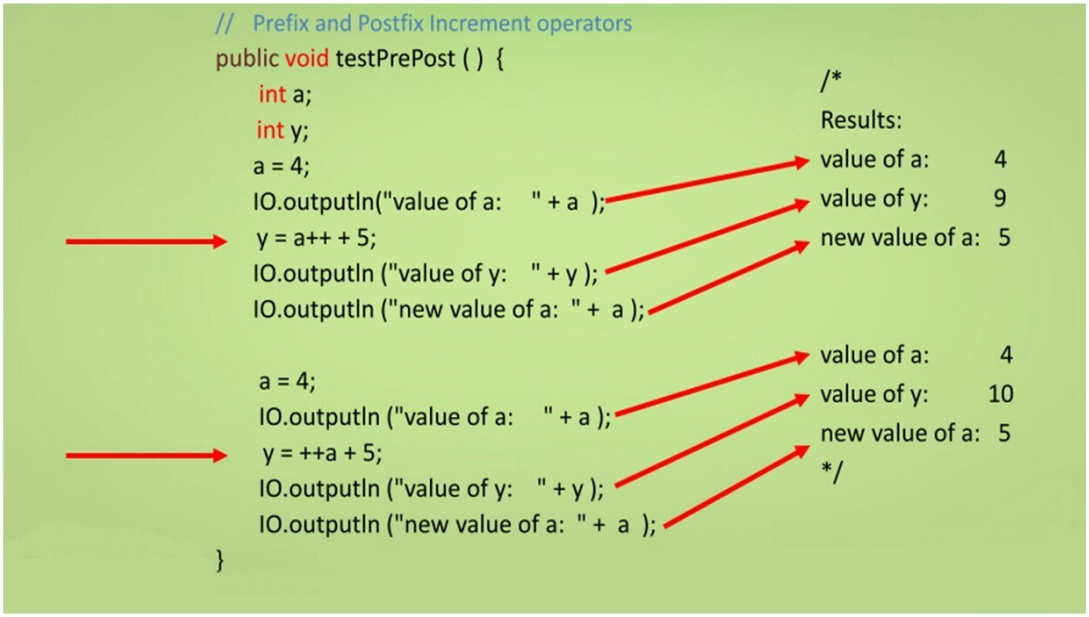
Последнее выражение, a–, является оператором пост-декремента.

Здесь в первую очередь используется начальное значение a в присвоении перед уменьшением на 1.

Таким образом, эти два указанные здесь префикс операторы, которые стоят перед a, сначала увеличивают или уменьшают значение, а затем уменьшенное или увеличенное значение используется в случае необходимости в присвоении.

И эти два постфиксные операторы, которые стоят после переменной a, сначала будут использовать начальное значение в присвоении до приращения, а затем увеличивают или уменьшают значение.

Пример здесь показывает разницу между префиксным и постфиксным оператором.



Здесь объявляются две целых переменные a и y, с инициализацией a = 4.

Первый вывод достаточно простой.

Оператор присваивания будет пытаться присвоить a++ плюс 5 в y.

Подумайте, каким будет результат для этого выражения.

Обратите внимание, что a++ это постфиксный оператор приращения.

Начальное значение a будет использоваться в выражении до увеличения значения, таким образом, a++ даст значение 4, как начальное значение.

Это значение затем будет добавлено к 5, и результат 9 будет затем передан y.

Побочным эффектом a++ будет увеличение значения на 1. Таким образом, значение a будет изменено на 5.

Так что это будет результат, который вы получите от блока кода, о котором мы только что говорили.

Следующая половина кода инициализирует a обратно в 4. Теперь подумайте о том, что произойдет с присвоением здесь.

++a является префиксным оператором приращения.

Здесь будет сначала выполняться приращение, то есть значение a будет увеличено на 1, и мы получим значение 5.

Увеличенное значение a будет использоваться в качестве значения выражения ++a.

То есть 5 будет добавлено к целому 5, и результат 10 будет передан y.

Таким образом, вы можете видеть, что эти данные были бы выведены во второй половине программы.

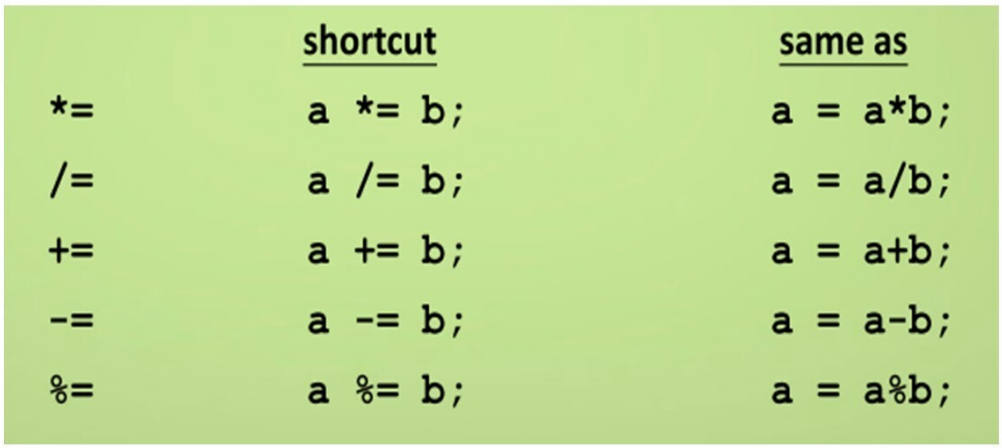
Разница здесь в том, что в первой половине кода, значение y равно 9.

Во второй половине программного блока, значение y равно 10.

В дополнение к операторам увеличения и уменьшения, Java также предоставляет набор сокращенных операторов присваивания.

Эти сочетания операторов применяют соответствующие арифметические операции к переменной, а затем присваивают результат обратно в переменную.

Вот пять таких операторов.



	shortcut	same as
*=	a *= b;	a = a*b;
/=	a /= b;	a = a/b;
+=	a += b;	a = a+b;
-=	a -= b;	a = a-b;
%=	a %= b;	a = a%b;

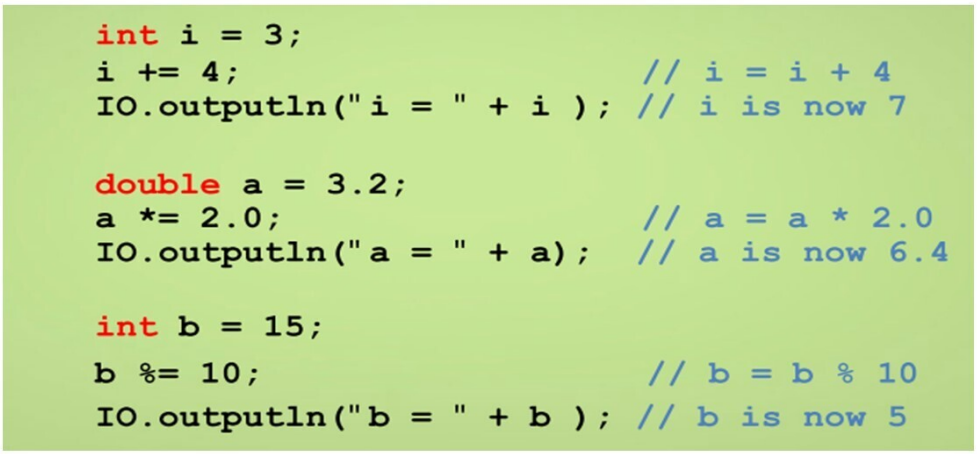
Они начинаются с арифметической операции с последующим знаком =.

Обратите внимание, что порядок символа важен, так как нет соответствующего сочетания операторов в обратном порядке.

Смысл a *= b точно такой же, как a равно a умножить на b, или a * b.

И остальные операторы имеют тот же принцип.

Вот несколько маленьких примеров.



```
int i = 3;
i += 4;                          // i = i + 4
IO.outputln("i = " + i );  // i is now 7

double a = 3.2;
a *= 2.0;                        // a = a * 2.0
IO.outputln("a = " + a);   // a is now 6.4

int b = 15;
b %= 10;                         // b = b % 10
IO.outputln("b = " + b );  // b is now 5
```

Первый сокращенный оператор + = в выражении i + = 4, точно такой же, как i = i + 4, так что значение i будет изменено на 7.

Второй сокращенный оператор * = имеет точно такой же смысл, что и a = a * 2, то есть, 3.2 * 2, таким образом, значение теперь становится 6,4.

Обратите внимание, что сокращенный оператор применяется также к числам с плавающей точкой.

Последний сокращенный оператор для оператора остатка.

Таким образом, b %= 10 является таким же, как принимать оставшуюся часть от деления 15 на 10, и результат 5 будет присвоен b.

Вы можете спросить, зачем Java предоставляет такого рода сокращенные операторы, а также операторы увеличения и уменьшения?

Это опции, которые обеспечивают некоторые удобства для программистов.

Эти особенности действительно не добавляют никакой силы языку программирования, в том смысле, что все, что можно сделать с этими операторами также может быть сделано без использования любого из них.

Однако, поскольку многие Java программисты любят использовать сокращения, вы должны быть в состоянии понять, что они означают.

Вопросы

Задача

Каковы будут значения переменных "result1" и "result2" после выполнения следующего сегмента кода?

```
int a = 6;
int b = 42;
int result1 = b / ++a;
int result2 = b / a++;
```

Варианты ответа:

1. result1: 6 result2: 6
2. result1: 6 result2: 7
3. result1: 7 result2: 6
4. result1: 7 result2: 7

Ответ: 1.

Объяснение

После выполнения первых двух строк кода, a будет иметь значение 6 и b значение 42.

При выполнении третьей строки кода, увеличенное значение a используется при оценке выражения int result1 = b / ++a;.

Значение a увеличивается до 7 перед вычислением выражения.

Поэтому, result1 = 42/7 = 6.

При выполнении четвертой строчки кода, текущее значение a используется при оценке выражения int result2 = b / a++;.

Значение a увеличивается после вычисления выражения.

Поэтому, result2 = 42/7 = 6.

Окончательное значение a=8.

Задача

Каким будет значение переменной “result” после выполнения следующего сегмента кода?

```
int i = 10;
int j = 25;
i += 5;
j /= i;
int result = i + j;
```

Варианты ответа:

1. 35
2. 16
3. 12
4. 11

Ответ: 2.

Объяснение

После выполнения первых двух строк кода, i будет иметь значение 10 и j значение 25.

При выполнении третьей строки кода, значение i увеличивается на 5 и результат присваивается обратно в i.

Таким образом, i становится 15.

При выполнении четвертой строки кода, значение j делится на i и результат присваивается обратно в j.

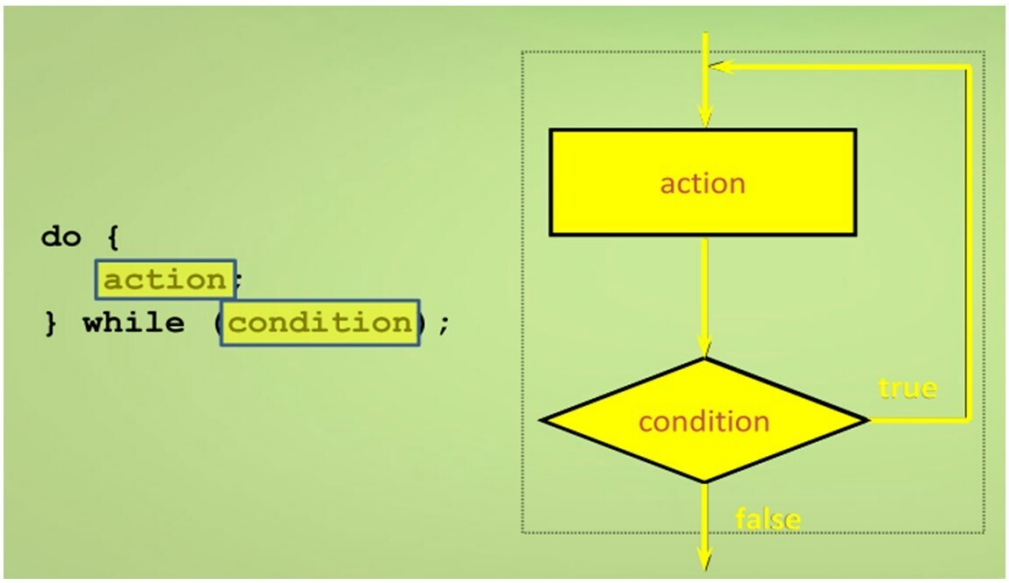
То есть, j = 25 / 15.

Так как это целое деление, j становится 1.

В последней строке, переменной result присваивается значение i + j, которое равно 15 + 1 = 16.

Циклы Do-While

Давайте теперь посмотрим на еще одну структуру цикла.
Она называется do-while цикл.



Синтаксис do-while цикла начинается с ключевого слова do, а затем следует тело цикла.
И тело цикла завершается проверкой условия, используя ключевое слово while.

Основное различие между while циклом и do-while циклом является то, что do-while цикл будет выполнять действие, указанное в теле цикла без предварительной проверки условия, в то время как while цикл условие сначала проверит.

Давайте рассмотрим эту схему, чтобы проиллюстрировать поток управления для do-while цикла.

Как уже упоминалось, do-while цикл будет идти непосредственно в теле цикла без проверки любого условия, то есть, do-while цикл будет всегда выполнять действие по крайней мере, один раз.

После окончания выполнения действия, будет проверяться условие.

Если проверка условия вернет true, поток вернется к началу цикла, и выполнит действие еще раз.

Этот процесс будет повторяться, и программа выйдет из цикла, когда условие окажется ложным.

```
public static int factorial(int number) {
  int t = 1, counter = 1;

  do {
    t *= counter;  // t = t * counter
    counter += 1;  //counter = counter + 1
  } while(counter <= number); //don't forget the ';'
  return t;
}
```

Здесь показана реализация n!, используя do-while цикл. И это выглядит очень похоже на реализацию, с помощью while цикла.

Как упоминалось ранее, метод будет выполнять тело цикла до проверки условия.

Обратите внимание, что я здесь использую сокращенные операторы.

Я хочу также отметить одну общую ошибку в использовании do-while цикла, то есть, когда забывают ставить точку с запятой после условного выражения.

```
public static int powerTwo(int number) {
  int t = 1, counter = 1;

  do {
    t *= 2;                  // t = t*2
    counter += 1;            //counter = counter+ 1
  } while (counter <= number); //don't forget the ';'
  return t;
}
```

Здесь показана реализация 2 в n степени, используя do-while цикл.

И это очень похоже на предыдущий пример, но вы можете проверить это более тщательно, потому что эта реализация не даст правильные результаты для всех случаев.

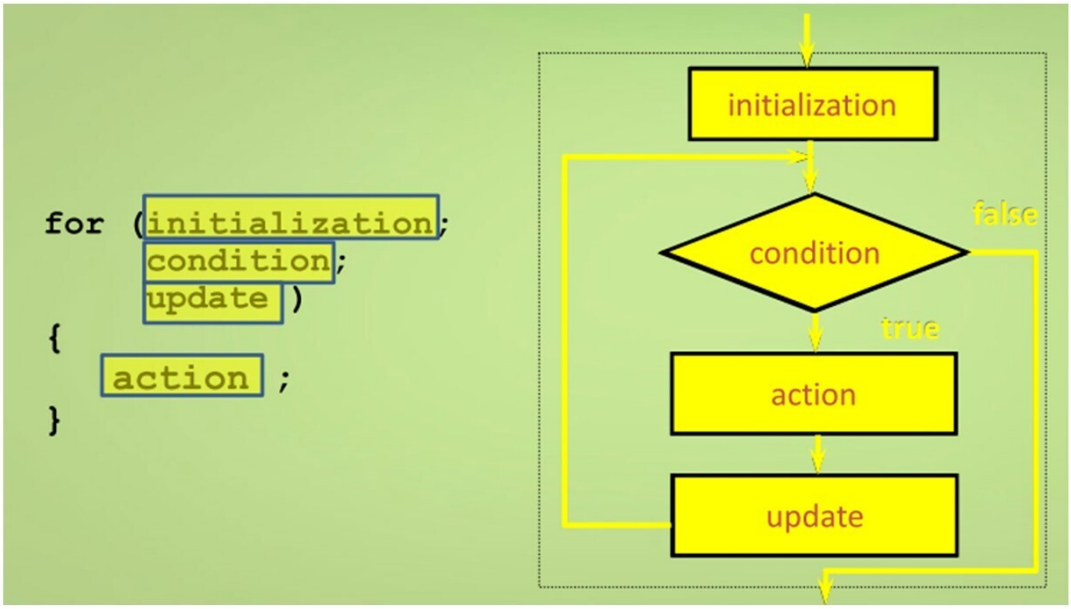
Подумайте, что будет, если number, то есть параметр, равен 0.

Следует помнить, что 2 в 0-й степени, как предполагается, дает 1 в результате. И поскольку do-while цикл всегда выполняется один раз до проверки условия, таким образом, даже если number равно 0, результат t будет изменен на 2 внутри цикла.

Когда счетчик оказывается меньше или равно 0, t уже получил значение 2, и возвращенный результат было бы неправильным для 2 в 0-й степени.

Вы можете подумать о том, как эта проблема может быть исправлена.

Третья структура цикла это for цикл.



Цикл for имеет довольно отличающийся синтаксис, чем while и do-while циклы. Он начинается с ключевого слова for.

Вместо того, чтобы иметь только одно условное выражение, как в while и do-while циклах, здесь есть три выражения, заключенные в круглые скобки после ключевого слова for.

Также следует обратить внимание на точку с запятой, отделяющей эти выражения.

Первое выражение является выражением инициализации, второе является условным выражением, а третье является выражением для обновления состояния.

Давайте посмотрим на поток управления, чтобы понять, как работает for цикл.

Цикл начинается с инициализации состояния, используя счетчик или повторяющуюся переменную.

Итерация выполняется только один раз в начале цикла. Затем проверяется условие, и тело цикла будет выполнено, если условие истинно.

После выполнения действия в теле цикла, затем будет вычисляться выражение обновления.

Здесь часто используется оператор увеличения или уменьшения.

Затем будет еще раз проверено условие. И этот процесс будет повторяться, если условие остается верным.

Цикл будет нарушен, когда условие станет ложным.

```
public static int factorial(int number) {
  int t = 1; int counter;

  //set up counter, condition check, update together
  for(counter=1; counter<=number; counter++) {
    t *= counter;
  }
  return t;
}
```

Здесь показана реализация n! используя for цикл.
Обратите внимание, что счетчик устанавливается в начале цикла.
На самом деле, счетчик может быть даже объявлен в цикле, как показано здесь.

```
public static int factorial(int number) {
  int t = 1;

  //set up counter, condition check, update together
  for(int counter=1; counter<=number; counter++) {
    t *= counter;
  }
  return t;
}
```

Таким образом, область применения переменной счетчика будет ограничена только циклом.

Состояние и действие в теле цикла являются такими же, как и раньше.

Обновление счетчика выполняется в выражении обновления цикла. В этом случае используется оператор приращение.

```
public static int powerTwo(int number) {
  int t = 1;

  //set up counter, condition check, update together
  for(int counter=1; counter<=number; counter++) {
    t *= 2;
  }
  return t;
}
```

Здесь показана реализация 2 в n степени, используя for цикл. И это очень похоже на то, что мы сделали для n!.

Кстати, здесь не будет никаких проблем при попытке вычислить 2 в 0-й степени, потому что условие проверяется перед входом в цикл.

Дополнительные ресурсы

Когда выполняется компьютерная программа, выражения в этой программе, как правило, выполняются по очереди, по порядку.

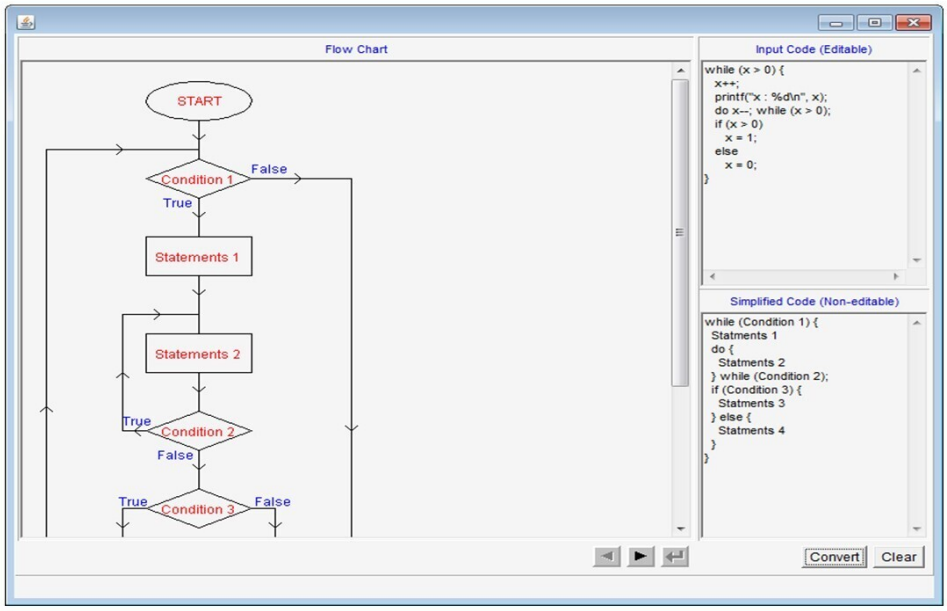
Однако, реальная компьютерная программа часто содержит тип выражений под названием выражение "choice", которое выбирает одно из выражений, которое следует выполнить в зависимости от некоторых условий.

Примерами являются выражения if, if-else и switch в Java.

Другой тип выражений, которые могут изменять нормальный поток управления, называются выражением "iteration".

Выражения итерации вызывают блок операторов для его повторения несколько раз, в зависимости от счетчика или некоторого условия прекращения.

Примерами являются выражения while, do-while и for в Java.



Программа Control позволяет подготовить короткий сегмент программы в правом верхнем окне, содержащем выражения if-else, while и do-while в Java.

Пример приведен в этом окне, который можно изменить или заменить.

Другие формы выражений выбора и итерации не поддерживаются.

Вы можете нажать на кнопку "Convert", чтобы увидеть эквивалентный сегмент псевдокода в правом нижнем углу апплета.

Этот псевдо-код также преобразуется в виде блок-схемы в окне слева.

Затем вы можете использовать кнопки "Next" и "Back", чтобы увидеть анимацию этого сегмента программы.

Для запуска приложения нужно запустить метод main класса ControlFlow.

Вопросы

Задача

Учитывая следующий метод:

```
public static int fact2(int n1, int n2) {
int t = 1; // initialize t to 1
for (int counter = n1; counter <= n2; counter++) {
t = t * counter;
}
return t;
}
```

Каким будет возвращаемое значение "t", если

i) n1 = 3, n2 = 5

ii) n1 = 4, n2 = 2

iii) n1 = -2, n2 = 2

Варианты ответа:

1. i) 60 ii) 24 iii) 0
2. i) 60 ii) 1 iii) infinite loop
3. i) 60 ii) 1 iii) 0
4. i) 12 ii) 1 iii) 0

Ответ: 3.

Объяснение

Цикл идет от итерации номер n1 вплоть до итерации номер n2.

Цикл будет завершен, когда текущий номер итерации больше n2.

Вначале инициализируется переменная t в 1.

В каждой итерации, значение t умножается на текущий номер итерации, и результат присваивается обратно в t. Поэтому результатом программы будет:

n1 * (n1 + 1) * … * (n2 – 1) * n2

Для случая i), n1 = 3, n2 = 5.

Таким образом, t = 3 * 4 * 5 = 60.

Для случая i), n1 = 4, n2 = 2.

Так как первоначально n1 больше, чем n2, цикл будет завершен перед выполнением первой итерации.

Таким образом, t остается в ее исходном значении 1.

В случае iii), n1 = -2, n2 = 2, поэтому t = -2 * -1 * 0 * 1 * 2 = 0.

Задача

Что делает эта программа?

```
public static int fact2(int n1, int n2) {
int t = 1; // initialize t to 1
for (int counter = n2; counter > n1; counter–) {
t = t * counter;
}
return t;
}
```

Варианты ответа:

1. n1 * (n1 + 1) * … * (n2 – 1)
2. n1 * (n1 + 1) * … * (n2 – 1) * n2
3. (n1 + 1) * … * (n2 – 1)

4. (n1 + 1) * … * (n2 – 1) * n2

Ответ: 4.

Объянение

Цикл идет от итерации номер n2 до и исключая итерацию номер n1.

Цикл будет завершен, когда текущий номер итерации равен n1.

Вначале инициализируется переменная t в 1.

В каждой итерации, значение t умножается на текущий номер итерации, и результат присваивается обратно в t.

Таким образом, результатом программы будет n2 * (n2 – 1) * … * (n1 + 1), что может быть записано как (n1 + 1) * … * (n2 – 1) * n2.

Общие ошибки

Теперь рассмотри некоторые распространенные ошибки при использовании циклов.

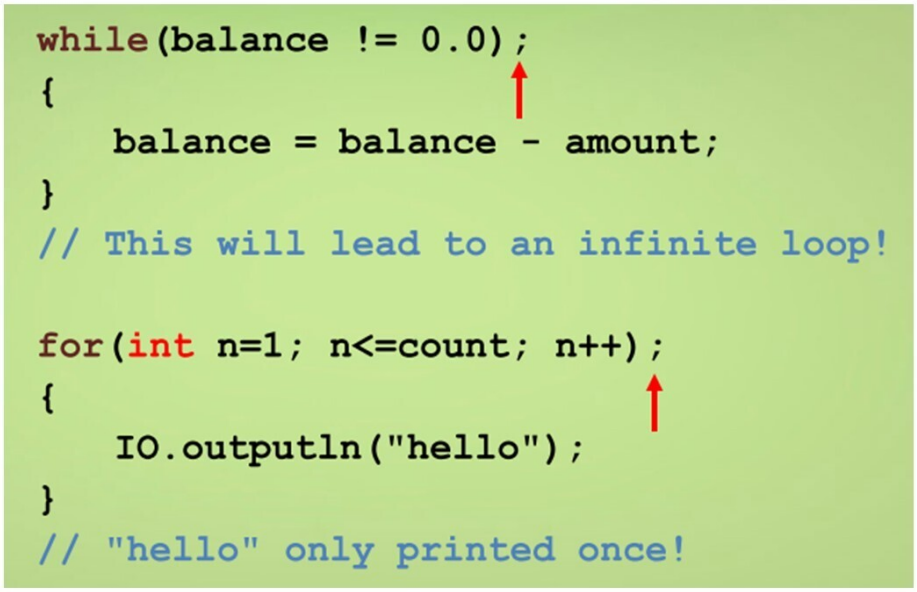
Я уже упоминал, что сравнивание равенства числа с плавающей точкой может привести к проблеме в связи с неточностью в представлении числа с плавающей запятой в компьютерах.

Поэтому использования операторов «==» или «!=» для сравнения чисел с плавающей запятой следует избегать как условий цикла.

Цикл может никогда не остановиться, если эти числа не точно равны или не равны из-за неточности.

```
double item = 1;
double sum = 0;

while (item != 0) {
  sum = sum + item;
  item = item – 0.1;
}
```

В этом примере, бесконечный цикл никогда не закончится, если item не будет никогда в точности равно 0.



Можете ли вы определить проблемы с этим while циклом?

Здесь есть две проблемы.

Первая для сравнения неравенства числа с плавающей точкой, как я только что описал.

Вторая – это точка с запятой после условия.

Это создаст пустое тело цикла, и приведет к бесконечному циклу!

Как насчет for цикла?

Опять же, есть дополнительная точка с запятой после закрытия скобки for выражения.

Цикл будет выполняться подсчет, ничего не делая в теле цикла.

Когда произойдет выход из for цикла, привет будет напечатано только один раз.

Вот некоторые из ошибок, которые вы никогда не должны делать.

Не существует четких правил, по которым следует решать, какой тип цикла лучше для решения определенных задач.

В общем, for цикл является лучшим для случаев, если известно число итераций.

Например, количество студентов в классе, если оно известно.

В то же время while или do-while циклы используются, когда число итераций неизвестно.

Основное различие между while и do-while циклом является то, что do-while цикл всегда будет выполнять тело цикла по крайней мере, один раз.

Что касается while цикла, бывают случаи, что тело цикла не будет выполнено вообще.

Задание

Вы только что увидели несколько распространенных ошибок циклов.

Давайте попробуем определить все ошибки в следующей программе, которая пытается распечатать здание с несколькими квартирами.

```
import comp102x.IO;

/*
 * Expected output of the program:
 *          _____________________
 * 9/F   |_?_|_?_|_?_|_?_|_?_|
 * 8/F   |_$_|_$_|_$_|_$_|_$_|
 * 7/F   |___|___|___|___|___|
 * 6/F   |_@_|_@_|_@_|_@_|_@_|
 * 5/F   |___|___|___|___|___|
 * 3/F   |_?_|_?_|_?_|_?_|_?_|
 * 2/F   |_$_|_$_|_$_|_$_|_$_|
 * 1/F   |___|___|___|___|___|
 */

public class Week05Debugging
{
    public static final int numberOfFloors = 8;

    public static void printBuilding()
    {
        IO.outputln("          _____________________");
        int currentFloor = numberOfFloors;

        do {

            if (currentFloor > 4)
                printFloor(currentFloor + 1);

            else {
                printFloor(currentFloor);
                currentFloor--;
            }

        }; while (currentFloor > 1)
    }

    /** Prints a floor. e.g.  "1/F  |___|___|___|___|___|" */
    public static void printFloor(int floor)
    {
        IO.output(floor + "/F  ");
        String symbol = "_";

        // for floors divisible by both 2 and 3, symbol should be "@"
        // for floors divisible by just 3, symbol should be "?"
        // for floors divisible by just 2, symbol should be "$"
        if (floor % 2 == 0 && floor % 3 == 0)
            symbol = "@";
        if (floor % 3 == 0)
            symbol = "?";
        if (floor % 2 == 0)
            symbol = "$";

        for (i = 0, i < 5, i + 1); {
            IO.output("|_" + symbol + "_");
        }

        IO.outputln("|");
    }
}
```

Ответ:

```
import comp102x.IO;

/*
 * Expected output of the program:
 *          _____________________
 * 9/F   |_?_|_?_|_?_|_?_|_?_|
 * 8/F   |_$_|_$_|_$_|_$_|_$_|
 * 7/F   |___|___|___|___|___|
 * 6/F   |_@_|_@_|_@_|_@_|_@_|
 * 5/F   |___|___|___|___|___|
 * 3/F   |_?_|_?_|_?_|_?_|_?_|
 * 2/F   |_$_|_$_|_$_|_$_|_$_|
 * 1/F   |___|___|___|___|___|
 */

public class Week05Debugging
{
    public static final int numberOfFloors = 8;

    public static void printBuilding()
    {
        IO.outputln("          _____________________");
        int currentFloor = numberOfFloors;

        do {

            if (currentFloor >= 4)
                printFloor(currentFloor + 1);

            else {
                printFloor(currentFloor);
            }

            currentFloor--;
        } while (currentFloor >= 1);
    }

    /** Prints a floor. e.g.  "1/F  |___|___|___|___|___|" */
    public static void printFloor(int floor)
    {
        IO.output(floor + "/F  ");
        String symbol = "_";

        // for floors divisible by both 2 and 3, symbol should be "@"
        // for floors divisible by just 3, symbol should be "?"
        // for floors divisible by just 2, symbol should be "$"
        if (floor % 2 == 0 && floor % 3 == 0)
            symbol = "@";
        else if (floor % 3 == 0)
            symbol = "?";
        else if (floor % 2 == 0)
            symbol = "$";

        for (int i = 0; i < 5; i++){
            IO.output("|_" + symbol + "_");
        }

        IO.outputln("|");
    }
}
```

Подклассы

Давайте посмотрим на другой пример, который связан с классом BankAccount, который мы обсуждали ранее.

Вид счета, который мы собираемся рассмотреть, это сберегательный счет.

Сберегательные счета зарабатывают проценты от остатка на счете.

Давайте предположим, что составные проценты используются при вычислении процентов, получаемых от сберегательного счета.

Для сложных процентов, в дополнение к основной сумме, проценты, начисленные ранее, также будут включены в вычисление новых процентов.

В то время как для простых процентов, только первоначальная основная сумма используется при определении процентов.

Например, если предположить, что проценты составляются ежегодно, если основной счет 1000 рублей с годовой процентной ставкой в 10%, начисляемой ежегодно в течение 5 лет.

Какой будет баланс в конце 5 лет.

В конце 1-го года, проценты, 1000 * 10% от 100 рублей, будут прибавляться к основному счету и получится новый баланс 1100 рублей.

Сумма 1100 руб. будет использоваться при определении процентов, полученных в конце 2-го года, то есть 1100 * 10% или 110 руб.

Новый баланс становится 1100 + 110 или 1210 руб.

В конце 3-го года, проценты будут 121 руб. и новый баланс 1331 руб.

В конце 4-го года, проценты будут 133,1 руб. и новый баланс 1464,1 руб.

В конце 5-го года, баланс будет составлять 1610,51 руб.

Это ужасно хорошо, но это всего лишь пример.

Маловероятно, что вы могли бы получить этот вид заработка, используя сберегательный счет в наши дни.

Во всяком случае, как указано в начале, сберегательный счет является своего рода банковским счетом.

Так как мы уже определили класс банковского счета в предыдущих лекциях, вместо того чтобы начинать с нуля, возможно ли для нас, чтобы использовать то, что мы уже сделали?

В ООП, есть важная концепция, которая называется наследованием.

Наследование является простой, но мощной концепцией.

При создании нового класса, и, если есть уже существующий класс, который содержит многое из кода, который вам нужен для нового класса, вы можете использовать этот код.

Это часто называют повторное использование кода.

Через повторное использование кода вы не должны повторять те усилия, которые были затрачены на проектирование и отладку кода.

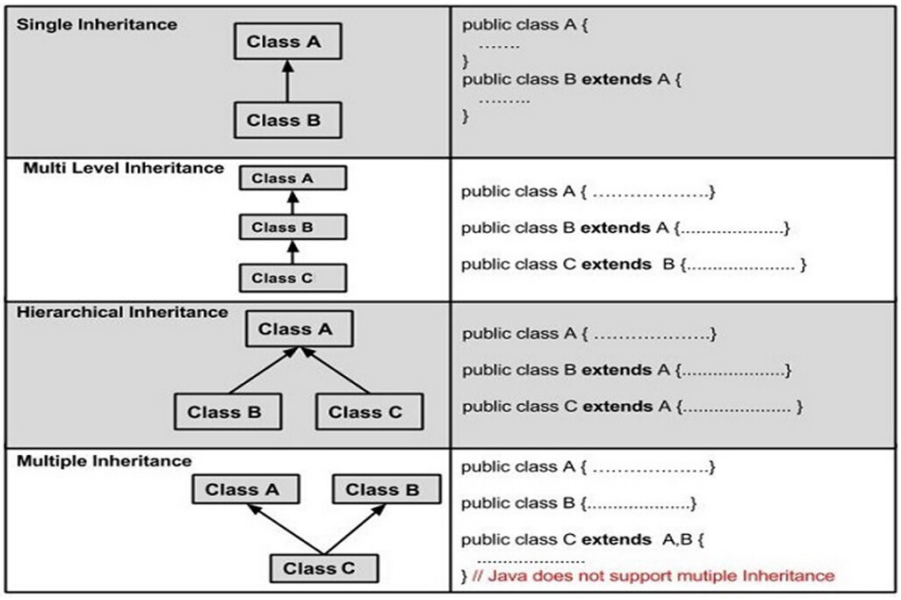
Новый класс может быть получен из существующего класса.

При этом, вы можете использовать поля и методы существующего класса без необходимости переписывать их самостоятельно.

Наследование поддерживается в Java через отношения подкласса и суперкласса, которые я буду обсуждать далее.

Подкласс, это класс, который является производным от другого класса. И класс, из которого подкласс происходит, называется суперкласс.

Подкласс часто называют дочерний класс или расширенный класс. И суперкласс иногда называют родительский класс.



Ключевое слово extends используется в объявлении подкласса для указания отношения подкласса-суперкласса. Я уверен, что вы видели это раньше в других примерах.

В Java, класс Object является корнем иерархии классов, то есть, в каждом классе есть Object как суперкласс.

Такая иерархическая организация является мощной для решения проблем, потому что объекты из реальной жизни часто организуются или классифицированы в некоторой иерархической структуре.

Например, человек является млекопитающим, млекопитающее является позвоночным, и позвоночное является животным.

Для географического образования, у нас есть города, которые разделены на районы, города на страны и страны на разные континенты.

Для физических устройств, телефоны можно классифицировать по двум основным категориям – стационарные телефоны и мобильные телефоны.

Мобильные телефоны классифицируются на айфоны, Android телефоны и другие менее популярные модели, таких как телефоны Windows. И далее, мобильные телефоны и планшеты, такие как iPad, могут быть сгруппированы в классе мобильных устройств.

Таким образом, в реальной жизни могут быть очень сложные иерархические структуры.

То есть, родительский класс может иметь несколько подклассов и один класс может быть подклассом нескольких суперклассов.

В Java, каждый класс может иметь один и только один прямой суперкласс. Это также называется одиночным наследованием.

Позволяя наследование в программировании, подкласс наследует все поля и методы от его суперкласса без необходимости переписывать их самостоятельно, или изобретать колесо.

```
class A
    {
    public int a, b, c;
    }

class B extends A
    {
        public B()
            {
                super.a = 10;
                super.b = 20;
                super.c = su
            }
    }
```

Ключевое слово super можно использовать для подкласса, чтобы ссылаться на конструкторы и методы из своего суперкласса. И ключевое слово super может понадобится, когда метод с тем же именем, объявленный в подклассе, переопределяет код суперкласса.

Когда переопределённый метод вызывается из своего подкласса, он всегда будет ссылаться на версию этого метода, определённую подклассом. А версия метода из суперкласса будет скрыта.

Если нужно получить доступ к версии метода, определённого в суперклассе, тогда нужно использовать ключевое слово super.

Пример подкласса

С точки зрения идеи об отношении подкласса и суперкласса, давайте посмотрим, как класс сберегательного счета может быть реализован как подкласс класса банковского счета, который мы обсуждали раньше.

```
import comp102x.IO;
/**
 * A bank account has a balance and an owner who can make
 * deposits to and withdrawals from the account.
 */
public class BankAccount {
   private double balance = 0.0;     // Initial balance is set to zero
   private String owner = "NoName";     // Name of owner
   /**
    *  Default constructor for a bank account with zero balance
    */
   public BankAccount ( ){ }
   /**
    *  Construct a balance account with a given initial balance and owner's name
    *  @param   initialBalance    the initial balance
    *  @param   name              name of owner
    */
   public BankAccount (double initialBalance, String name) {
      balance = initialBalance;
      owner = name;
   }
```

Вот определение класса BankAccount, где объявляются две переменные экземпляра balance и owner, и два конструктора, один без параметров, а другой с двумя параметрами.

Обратите внимание, что переменные экземпляра объявлены как private.

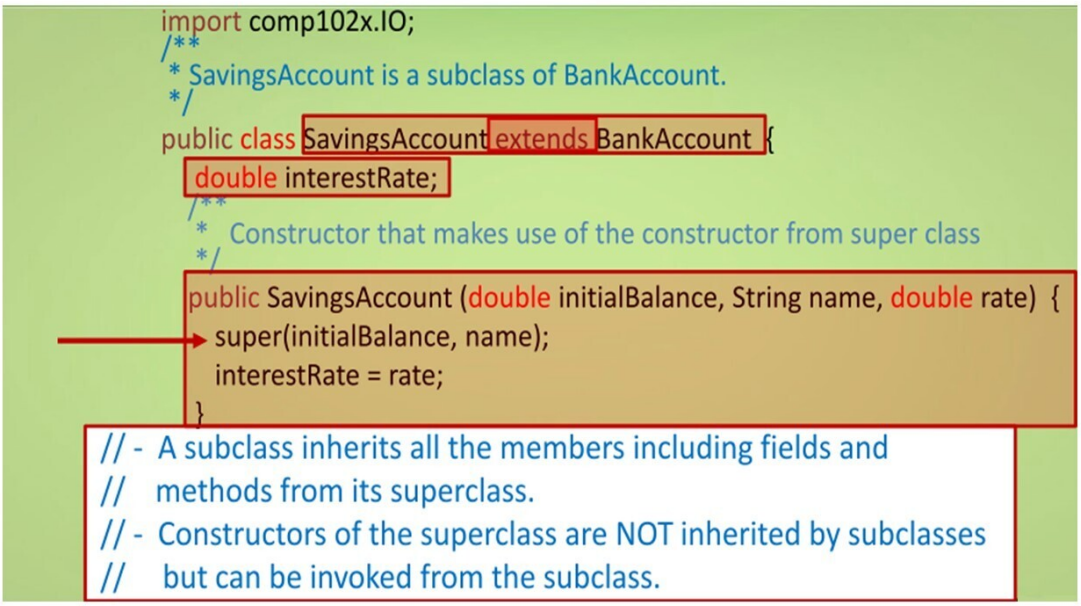
Я вернусь к private полям подкласса позже.

И есть три метода, а именно, deposit, withdraw и getBalance.

```
/**
 *  Method for depositing money to the bank account
 *  @param   dAmount     the amount to be deposited
 */
public void deposit(double dAmount) {
    balance = balance + dAmount;
}
/**
 *  Method for withdrawing money from the bank account
 *  @param   wAmount     the amount to be withdrawn
 */
public void withdraw(double wAmount) {
    balance = balance - wAmount;
}
/**
 *  Method for getting the current balance of the bank account
 *  @return   the current balance
 */
public double getBalance() {
    return balance;
}
```

Все это также полезные методы для класса SavingsAccount, который я собираюсь обсудить.

Вот реализация SavingsAccount класса в Java.



Ключевое слово extends используется для создания подкласса, который наследует поля и методы от существующего класса, который становится суперклассом.

Подклассу дано имя SavingsAccount, который расширяет BankAccount.

Подкласс наследует все члены, включая поля и методы из своего суперкласса.

Одна дополнительная переменная InterestRate экземпляра объявлена в подклассе SavingsAccount, и он также наследует переменные экземпляра balance и owner от BankAccount.

Я поговорю о методах позже.

Конструкторы обрабатываются по-разному, они не наследуются подклассами, но конструктор суперкласса может быть вызван из подкласса.

Здесь объявляется конструктор для SavingsAccount.

Он принимает три параметра, в дополнение к initialBalace и name, как в одном из конструкторов BankAccount, третий параметр является double параметром rate, который используется для инициализации процентной ставки.

Конструктор подкласса может вызывать конструктор суперкласса с помощью ключевого слова super.

Это вызов конструктора с двумя параметрами из BankAccount.

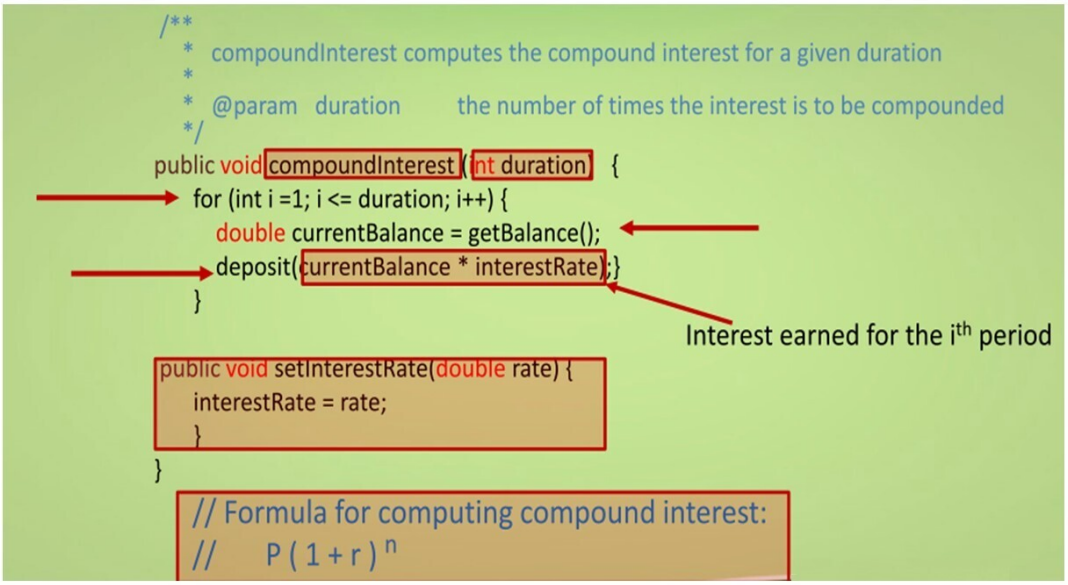
Обратите внимание, что синтаксис вызова конструктора суперкласса – это использование ключевого слова super, после которого следует список параметров в скобках.

Список может быть пустым, если вызывается конструктор по умолчанию суперкласса.

Если конструктор не определен явно в подклассе, конструктор суперкласса по умолчанию без аргументов будет использоваться для подкласса.

Ошибка компиляции возникает, если суперкласс не имеет конструктора по умолчанию без аргументов.

Я проиллюстрирую это в демо, после того как мы завершим определение SavingsAccount.



Это реализация метода вычисления сложных процентов.

Метод называется compoundInterest, который принимает один аргумент duration типа int, который дает число раз, сколько должны быть вычислены проценты.

Здесь используется цикл for, так как мы получаем переменную duration, которая говорит нам точно, сколько раз будут вычисляться проценты.

Если у вас есть время, вы можете подумать об альтернативных реализациях, использующих while или do-while цикл.

Каждый раз внутри цикла текущий баланс извлекается с помощью метода getBalance.

Мы не можем получить доступ к переменной экземпляра balance в BankAccount напрямую, потому что она объявлена как private.

Конструкторы или методы из подкласса имеют не больше прав на private поля суперкласса, чем любые другие методы.

То есть, если метод в SavingsAccount хочет получить доступ к балансу, он должен получить доступ к нему с помощью public метода в BankAccount, в данном случае, getBalance.

Проценты, начисленные за i-й период, получаются умножением текущего баланса на InterestRate, который задан в качестве параметра.

Важным шагом здесь является использование метода deposit, определенным в BankAccount, чтобы внести или добавить новые проценты к текущему балансу.

Важно отметить, что метод deposit наследуется от суперкласса BankAccount и модификация, произведенная методом deposit, является переменной экземпляра balance, которая также наследуется от BankAccount.

Таким образом, баланс теперь обновляется, чтобы включить новые начисленные проценты, и в следующий раз через цикл, метод getBalance будет получать обновленный баланс, и таким образом, вычисляются составные проценты.

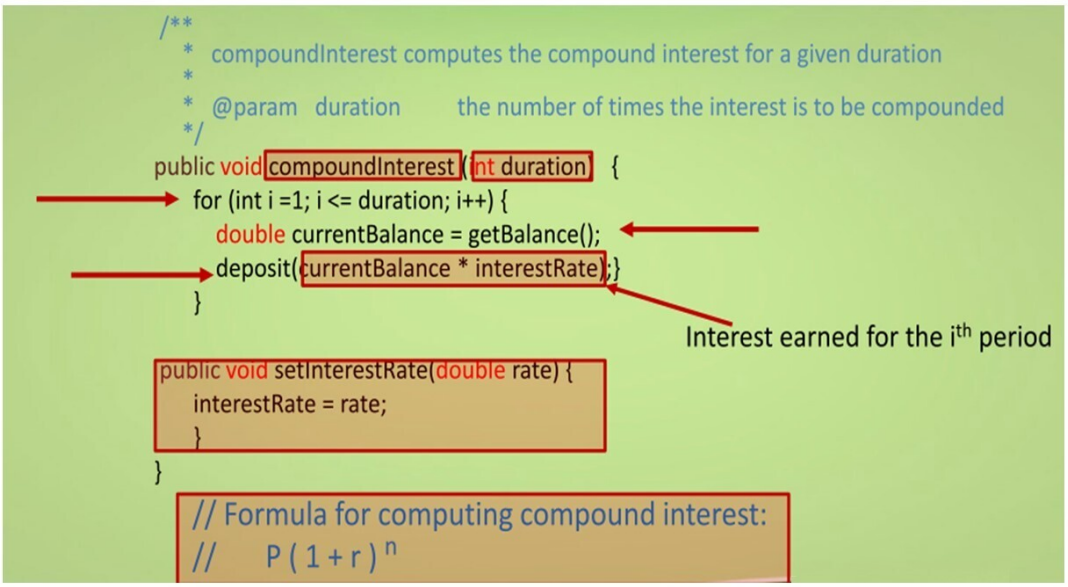
Я хочу отметить, также, как составные проценты могут быть вычислены.

Используется единая формула:

P * (1 + r)^n, где P является базой, r это процентная ставка и n это число раз, сколько проценты начисляются.

P * (1 + r)^n, где P является базой, r это процентная ставка и n это число раз, сколько проценты начисляются.

Не волнуйтесь, если вы не знаете или не помните эту формулу, потому что даже если вы бы узнали об ней, вы, вероятно, забыли бы о том, как она была получена.



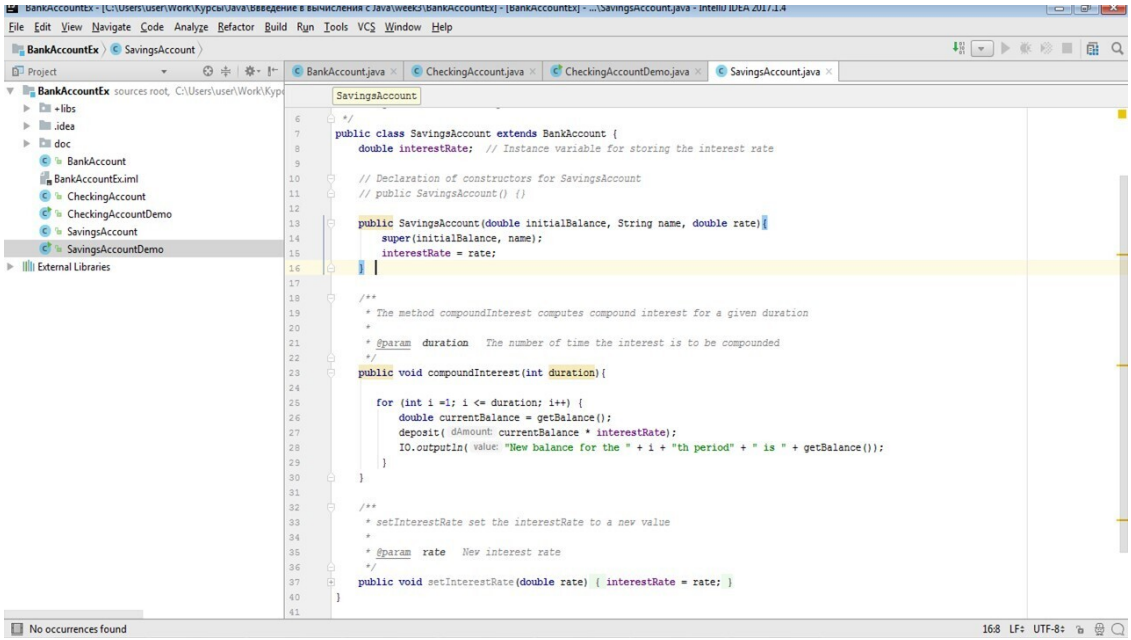
Но если вы посмотрите на то, как составные проценты вычисляются с использованием for цикла, это, будет вероятно, более интуитивным, чем вычисление, следующее тому, как составные проценты определяются.

Обратите внимание также, что существует сеттер метод setInterestRate для установки переменной InterestRate экземпляра в новое значение, указанное параметром rate.

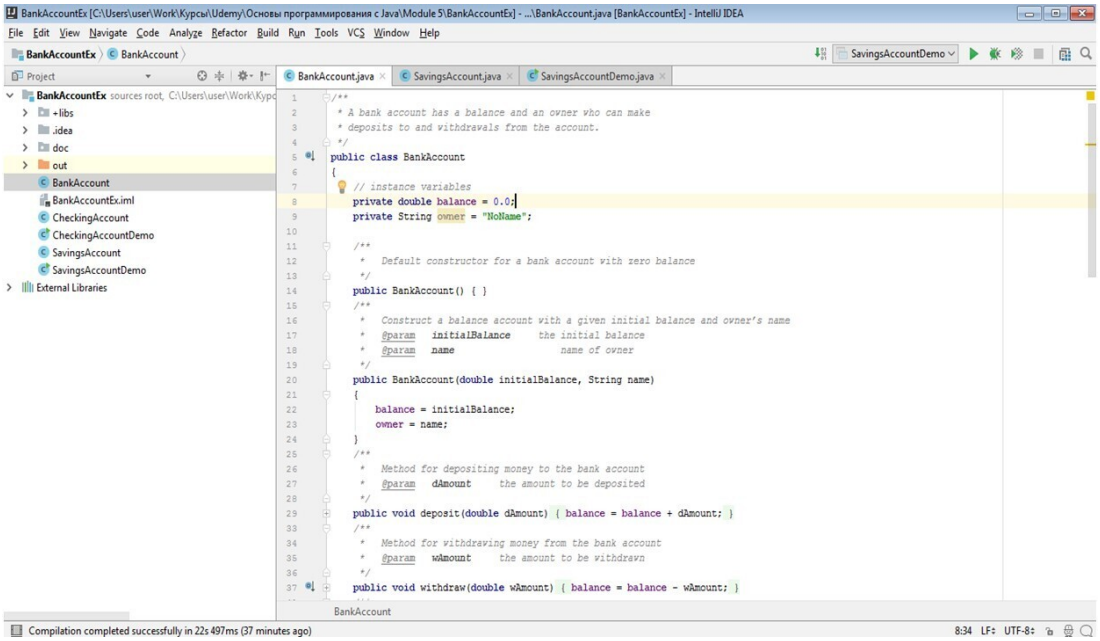
Давайте откроем IDEA, чтобы посмотреть на демо программу SavingsAccount.

Демонстрация подкласса

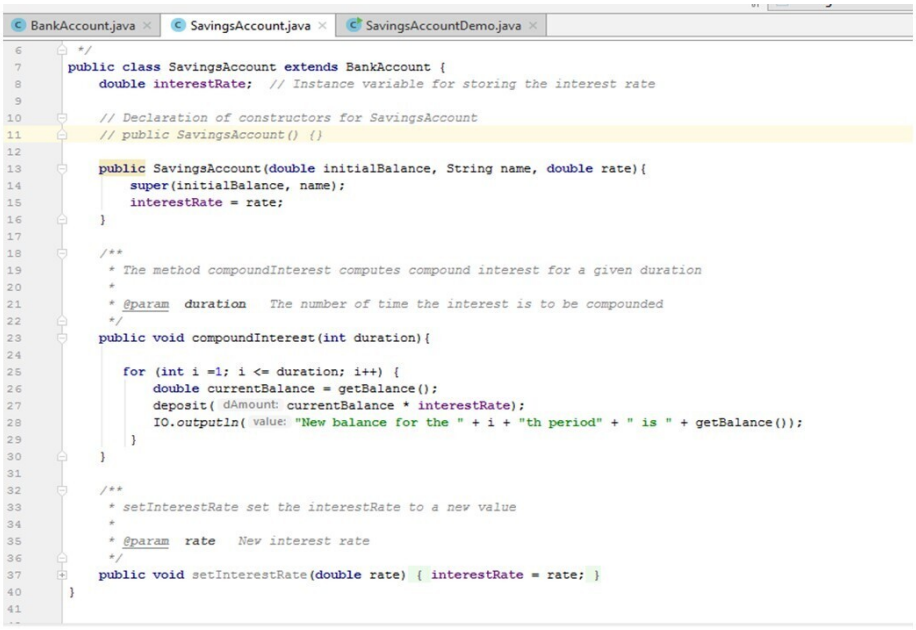
Откроем проект BankAccountEx в среде IDEA.



Как вы можете видеть, здесь есть класс BankAccount, и есть класс SavingsAccount. Есть и другие классы, которые я буду обсуждать позже.



Если вы откроете BankAccount, это тот класс BankAccount, который мы определили ранее.



И вот класс SavingsAccount, который мы только что описали.

Как вы можете видеть, SavingsAccount расширяет BackAccount.

SavingsAccount также объявляет переменную InterestRate экземпляра.

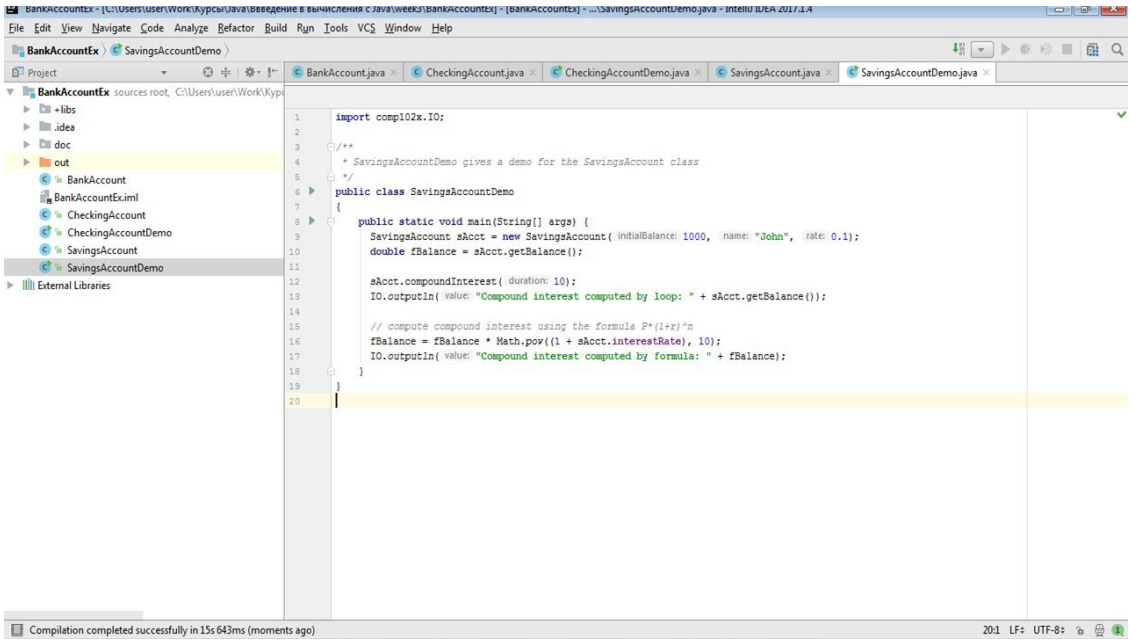
Там также есть один инструктор и два метода, compoundInterest, и сеттер-метод setInterestRate.

Если мы скомпилируем этот метод, мы не увидим никаких синтаксических ошибок.

В классе SavingsAccount представлен один конструктор.

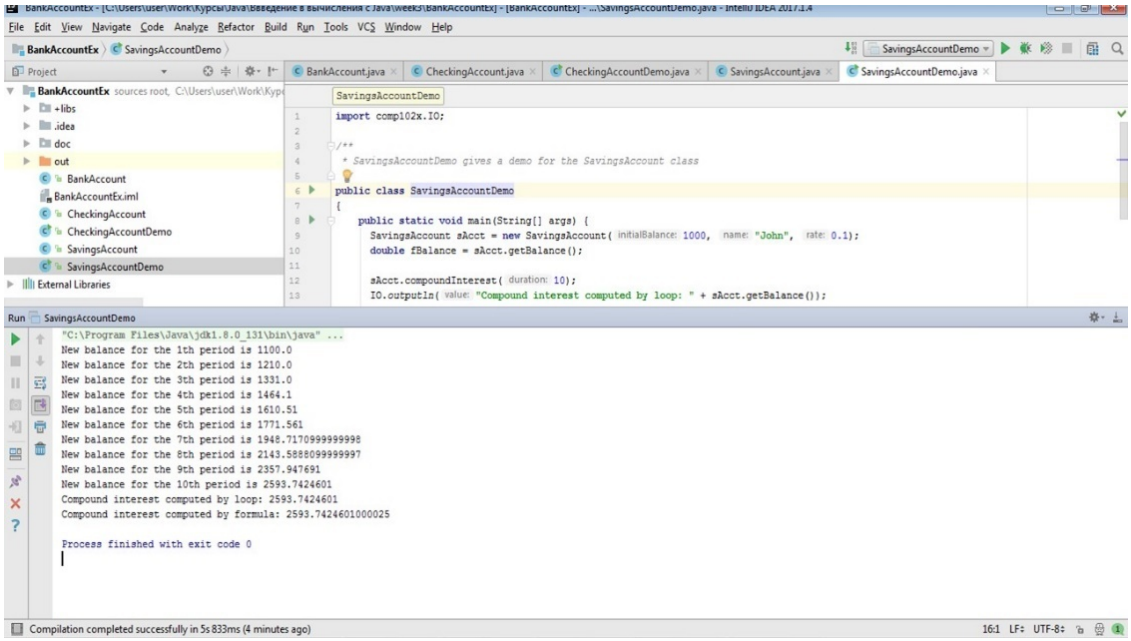
Этот конструктор имеет три параметра, initialBalance, name и rate, как указано в конструкторе.

Обратите внимание, что здесь нет конструктора по умолчанию без аргументов.



Давайте попробуем создать экземпляр класса с помощью этого конструктора, и значений трех параметров, для initialBalance введем, скажем, 1000, а затем имя владельца, и процентную ставку, в обсуждении мы говорили о 10%, и 10% будет 0,1.

Мы можем вычислить составные проценты, вызвав метод compoundInterest.



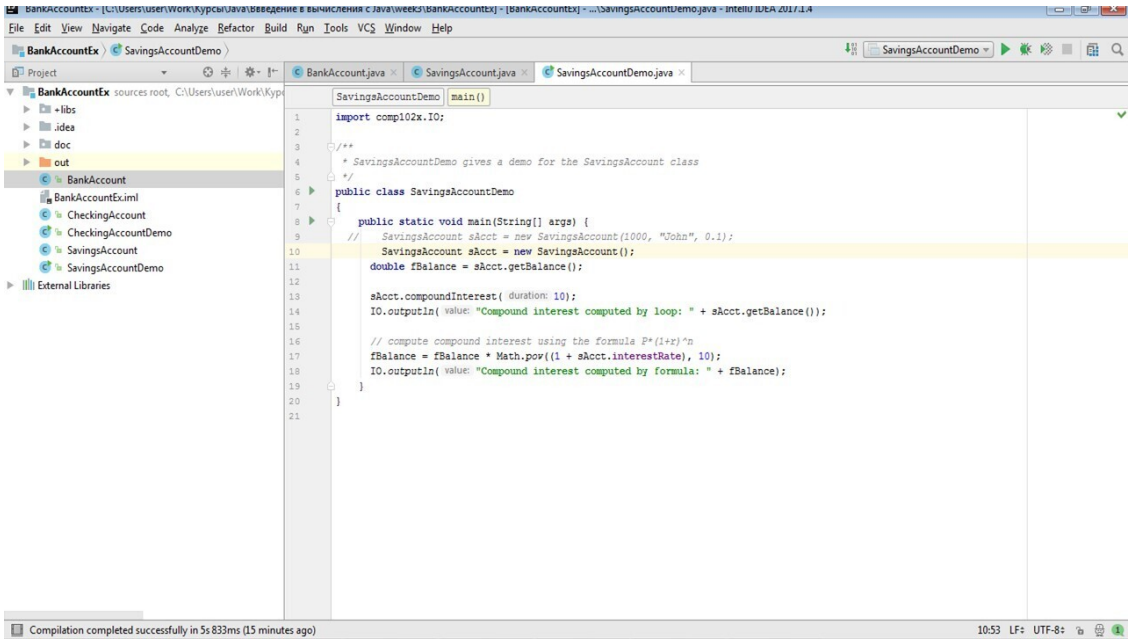
Скажем, если мы установим срок до 10 лет, Вы можете увидеть эти новые балансы по состоянию на конец каждого года.

Теперь давайте вернемся к BankAccount и попытаемся посмотреть на влияние различных способов определения конструктора.

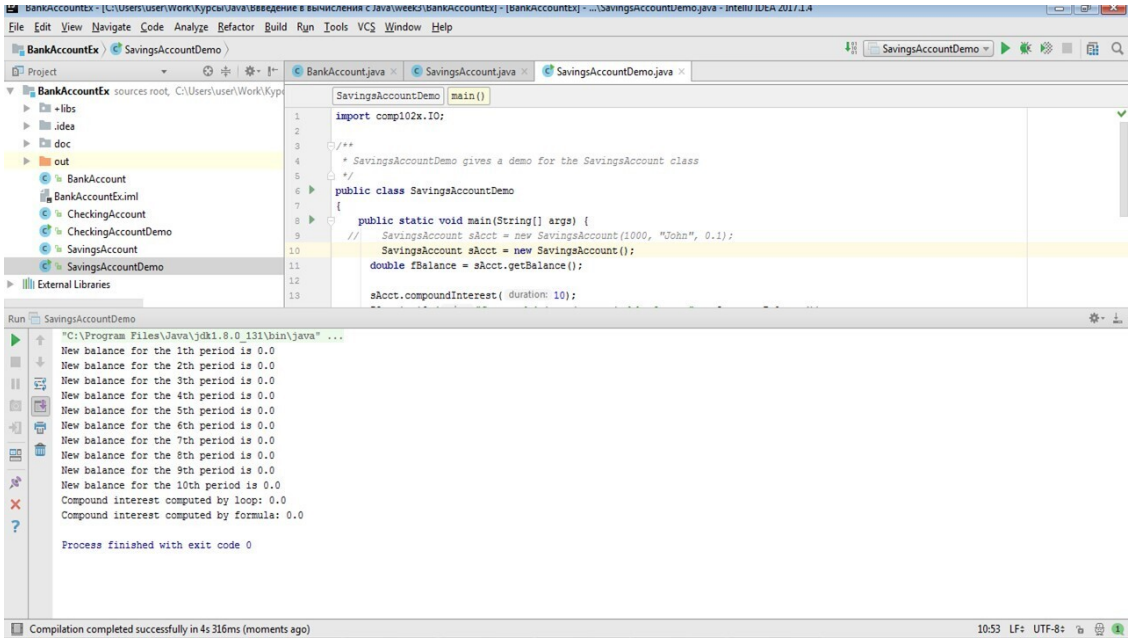
```
*/
public class SavingsAccount extends BankAccount {
    double interestRate;  // Instance variable for storing the interest rate

    // Declaration of constructors for SavingsAccount
    // public SavingsAccount() {}

    /*
    public SavingsAccount(double initialBalance, String name, double rate){
        super(initialBalance, name);
        interestRate = rate;
    }
*/
    /**
     * The method compoundInterest computes compound interest for a given duration
     *
     * @param  duration   The number of time the interest is to be compounded
     */
    public void compoundInterest(int duration){

      for (int i =1; i <= duration; i++) {
          double currentBalance = getBalance();
          deposit( dAmount: currentBalance * interestRate);
          IO.outputln( value: "New balance for the " + i + "th period" + " is " + getBalance());
      }
    }
```

Давайте сначала попробуем удалить определение конструктора в SavingsAccount.

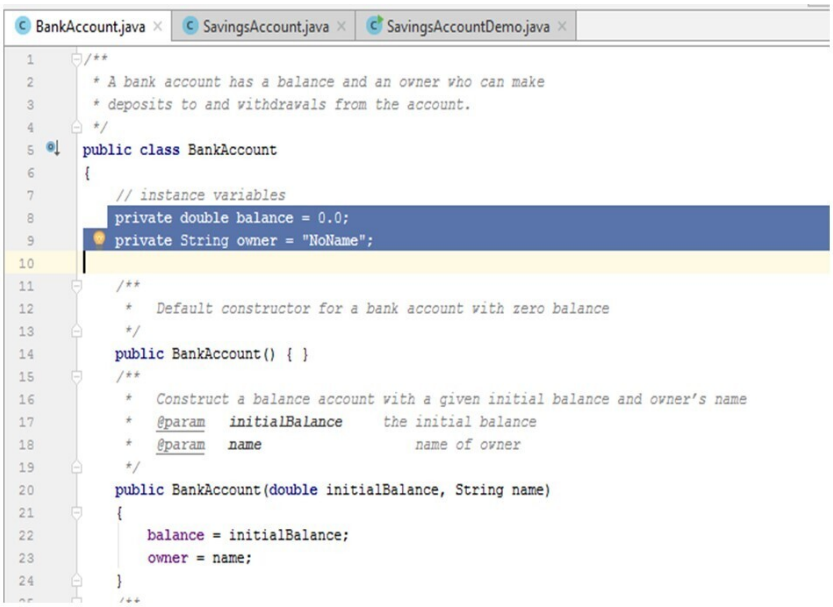
Теперь, если вы пойдете в SavingsAccount, вы обнаружите, что теперь доступен конструктор по умолчанию.



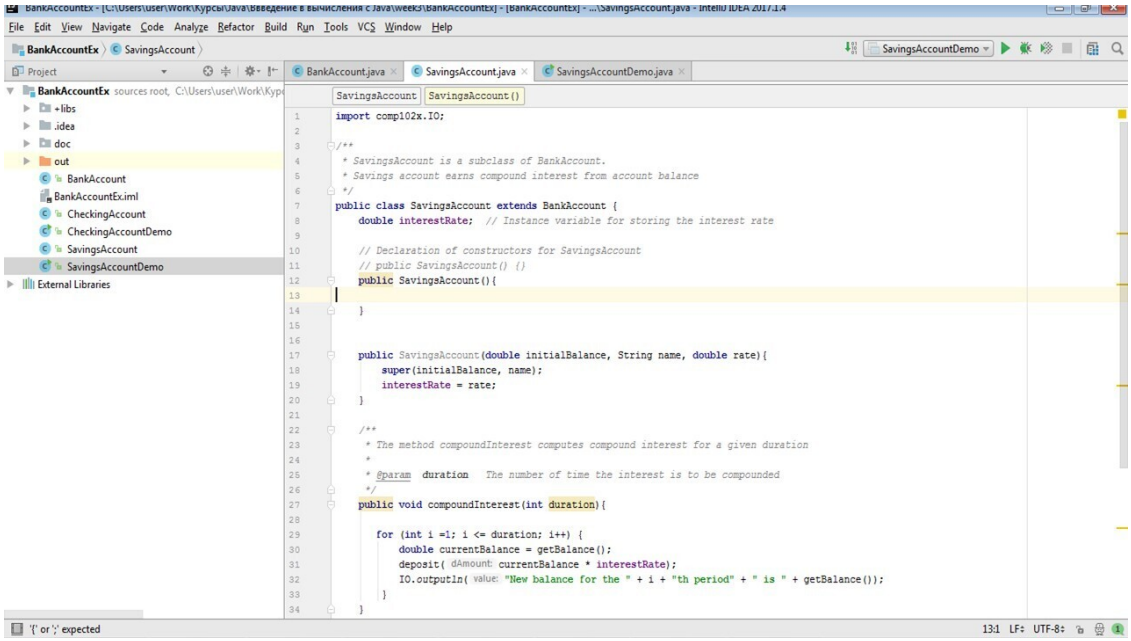
Этот конструктор на самом деле конструктор от суперкласса BankAccount.



Если мы создаем экземпляр, используя этот конструктор, вы увидите, что баланс установлен в 0.0, и владельцу было дано имя NoName.



Если вы пойдете в BankAccount, вы увидите, что конструктор по умолчанию в BankAccount использует это начальное значение для баланса, а также имя владельца NoName.



Давайте вернем конструктор SavingsAccount.

Если мы также создадим здесь, в классе SavingsAccount, конструктор по умолчанию, мы можем скомпилировать программу, и не будет синтаксических ошибок.

```
import comp102x.IO;

/**
 * SavingsAccountDemo gives a demo for the SavingsAccount class
 */
public class SavingsAccountDemo
{
    public static void main(String[] args) {
        SavingsAccount sAcct = new SavingsAccount( initialBalance: 1000,  name: "John",  rate: 0.1);
        SavingsAccount sAcct1 = new SavingsAccount();

        double fBalance = sAcct.getBalance();

        sAcct.compoundInterest( duration: 10);
        IO.outputln( value: "Compound interest computed by loop: " + sAcct.getBalance());

        // compute compound interest using the formula P*(1+r)^n
        fBalance = fBalance * Math.pow((1 + sAcct.interestRate), 10);
        IO.outputln( value: "Compound interest computed by formula: " + fBalance);
    }
}
```

Теперь вы можете видеть, что здесь доступны конструктор по умолчанию и конструктор с тремя параметрами.

На самом деле, если вы хотите быть более точным,

Если вы хотите вызвать конструктор по умолчанию суперкласса, вы также можете сделать это, указав super и пустой список параметров внутри скобок.

Программа SavingsAccountDemo в основном состоит только из main метода.

```
public class SavingsAccountDemo
{
    public static void main(String[] args) {
        SavingsAccount sAcct = new SavingsAccount( initialBalance: 1000,  name: "John",  rate: 0.1);

        double fBalance = sAcct.getBalance();

        sAcct.compoundInterest( duration: 10);
        IO.outputln( value: "Compound interest computed by loop: " + sAcct.getBalance());

        // compute compound interest using the formula P*(1+r)^n
        fBalance = fBalance * Math.pow((1 + sAcct.interestRate), 10);
        IO.outputln( value: "Compound interest computed by formula: " + fBalance);
    }
}
```

Помните, что main метод является точкой входа для всех программ Java?

Здесь мы создали сберегательный счет с помощью конструктора, и мы указали начальный баланс 1000, Джона как владельца счета, и процентную ставку 10%.

Обратите внимание, что в BankAccount переменную экземпляра balance можно получить с помощью метода getBalance, но мы еще не создали метод получения имени владельца.

Поскольку владелец объявлен как private, мы не можем получить доступ к имени владельца в классе SavingsAccount.

Вы можете подумать о том, как определить метод получения имени владельца.

После установки сберегательного счета с именем, мы пытаемся извлечь первоначальный баланс из этого объекта.

То, что мы пытаемся сделать здесь, заключается в попытке сравнить два различных способа при вычислении составных процентов.

Первый заключается в использовании метода compoundInterest, который мы определили для SavingsAccount.

Помните, что мы используем цикл при вычислении составных процентов.

Другой способ заключается в вычислении составных процентов с использованием формулы.

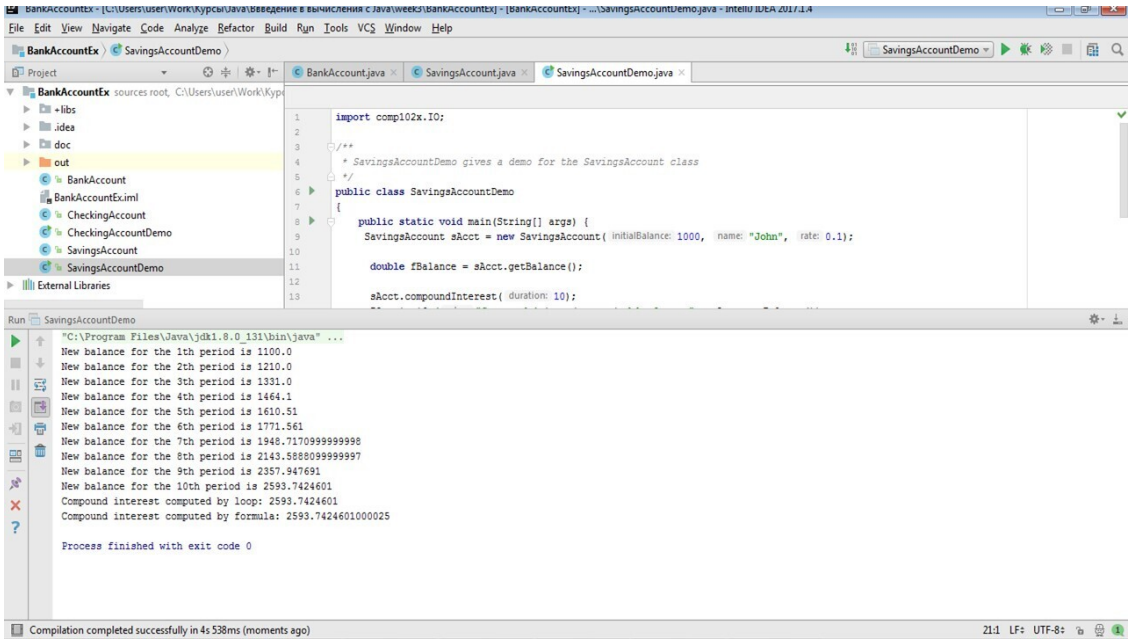
Это формула, которую я кратко уже обсудил.

Чтобы вычислить определенное число в n степени, можно использовать метод pow в библиотеке Math.

Таким образом, здесь приведена реализация этой формулы.

Вы можете видеть, после того как мы вычислили составные проценты, используя метод compoundInterest, мы показываем результат, используя вывод IO.outputln.

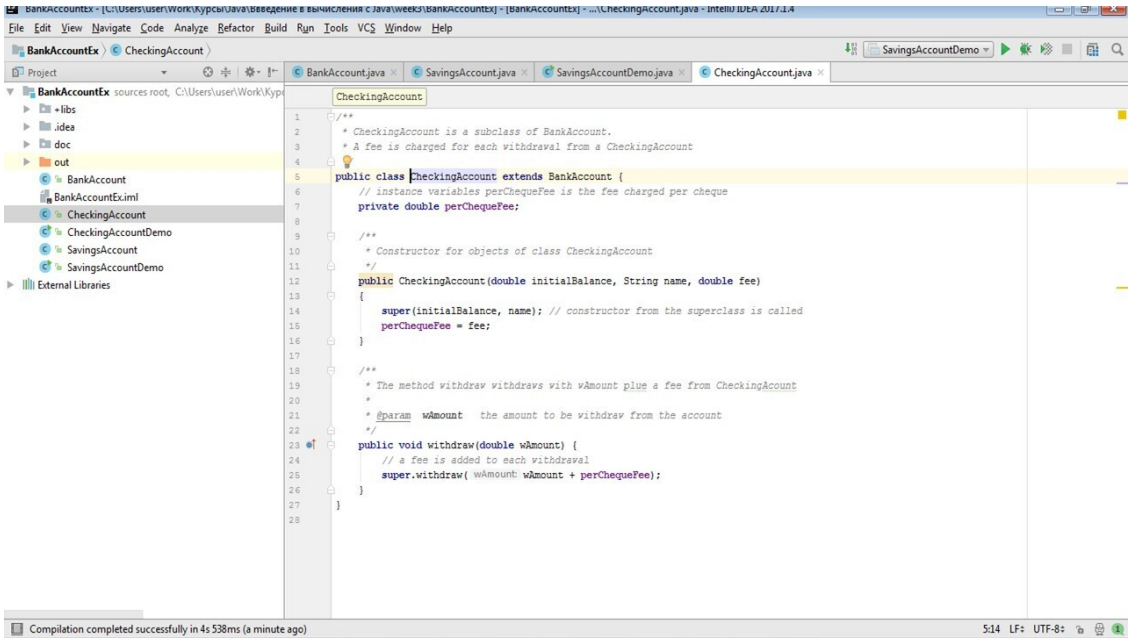
Мы также вычислили составные проценты по формуле, а затем выводим результат.



Вы можете видеть, что эти два числа в основном одинаковые.

Единственное различие состоит в неточности числа с плавающей точкой.

Давайте посмотрим на другой подкласс класса BankAccount, который мы написали здесь.



Он называется CheckingAccount.

CheckingAccount является типом счета, который не зарабатывает проценты.

Существует также плата за каждое снятие с CheckingAccount, в то время как нет комиссии для внесения денег на счет.

И CheckingAccount расширяет BankAccount.

Здесь есть переменная экземпляра, представляющая комиссию, которая будет взиматься за каждый чек снятия денег со счета.

Существует один конструктор, снова с 3 параметрами.

Первый параметр это initialBalance, второй это имя владельца, и третий параметр, это комиссия, чтобы установить значение переменной perChequeFee экземпляра.

Существует только один метод в классе CheckingAccount.

Заметьте, что мы определяем метод withdraw, который имеет то же имя, что и метод withdraw в BankAccount.

Что делает этот метод, это забирает деньги из CheckingAccount.

Но разница здесь в том, что вместо того, чтобы просто снять wAmount, мы должны добавить комиссию perChequeFee в сумму, в дополнение к wAmount.

Поскольку withdraw в CheckingAccount имеет то же имя, как метод withdraw в суперклассе BankAccount, если опустить super, вы вызываете метод с таким же именем внутри метода.

Это называется рекурсия. Я буду говорить о рекурсии позже.

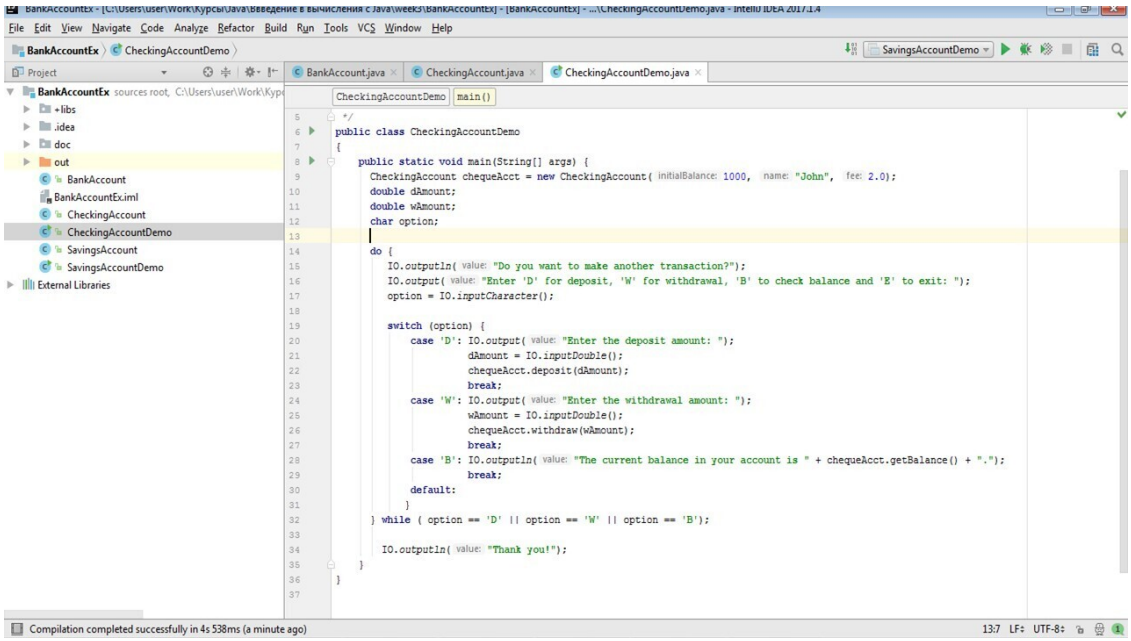
Если вы не будете осторожны с рекурсией, это может привести к бесконечному циклу.

Это означает, что этот метод никогда не завершится.

Таким образом, вместо того, чтобы вызвать метод withdraw класса CheckingAccount, мы можем указать, что мы хотим вызвать метод withdraw в BankAccount, суперклассе, указав ключевое слово super, а затем оператор точки.

Используя этот способ, будет вызываться метод withdraw в BankAccount, вместо метода withdraw, определенного здесь, внутри CheckingAccount.

При компиляции этого метода нет ошибок.



Вы можете создать экземпляр, введя initialBalance 1000, имя, и комиссию 2 рубля.

Этот экземпляр унаследовал три метода, определенных в BankAccount, deposit, getBalance и withdraw.

При попытке вызвать метод withdraw, скажем, если вы хотите снять 100 рублей,

Вместо того чтобы получить баланс 900, обновленный баланс будет 898 рублей, потому что в дополнение к 100 рублям, которые вы снимаете, плата в 2 рубля также добавляется к сумме вывода.

Так что, если вы попытаетесь внести обратно 100 рублей на счет, заметьте, что мы используем метод deposit, который наследуется от класса BankAccount.

Так что, если Вы захотите внести 100 рублей назад, вы получите баланс на 2 рубля меньше, потому что 2 дополнительных рубля были сняты с чеком.

Итак, программа CheckingAccountDemo создает объект класса CheckingAccount.

Мы также определяем как вносимую сумму, так и сумму снятия.

У нас есть еще одна переменная типа character, которая называется option.

То, что вы видите здесь, в теле main метода, это do-while цикл.

То, что мы пытаемся сделать, это позволить пользователю продолжать делать различные транзакции, в том числе вложение денег, снятие денег, проверку баланса счета и выйти из цикла.

Как вы можете видеть, это организовано с помощью do-while цикла, и do-while цикл будет продолжаться до тех пор, пока пользователь хочет сделать еще один ввод, вывод или проверить баланс.

Если пользователь вводит любой другой вариант option в виде символа, он выйдет из цикла.

Если вы посмотрите на цикл более тщательно, он сначала спрашивает пользователя тип сделки, которую он хочет осуществить.

И пользователь должен ввести символ.

Заметьте, что переменная option типа CHAR.

Для переменной char в Java, она определяется внутри одинарных кавычек.

Это важно помнить, потому что это отличается от строки символов.

Как вы можете видеть, символьные строки заключены в двойные кавычки, в то время как символ заключен в одинарные кавычки.

Таким образом, после ввода option пользователем, выражение switch будет проверять опцию, а затем выполнит соответствующие действия соответствующим образом с помощью case выражения.

В случае, когда пользователь вводит D для депозита, у пользователя спросят сумму денег, которую нужно внести на счет.

Здесь на самом деле получается сумма вносимых денег с помощью метода inputDouble.

Вносимая сумма будет присвоена переменной dAmount.

И далее будет вызываться метод deposit.

Заметьте, что метод deposit здесь хотя и принадлежит объекту chequeAccount, он фактически наследуется от класса BankAccount.

И это та сумма, которая внесется на счет.

И помните, что для выражения switch важно поставить break, в противном случае выполнение будет просто проходить через каждый из этих случаев.

В случае ввода символа W, клиент хочет сделать вывод.

Далее получаются входные данные с помощью класса IO.

Сумма изъятия присваивается переменной wAmount.

Теперь, здесь вызывается метод withdraw, withdraw метод, определенный в CheckingAccount.

Концепция в том, что, если метод с тем же именем определен в подклассе, он будет переопределять тот метод, что в суперклассе.

Таким образом, в этом случае, если вы посмотрите на класс CheckingAccount, это withdrawn метод.

И тогда для варианта B, идет просто вызов метода getBalance.

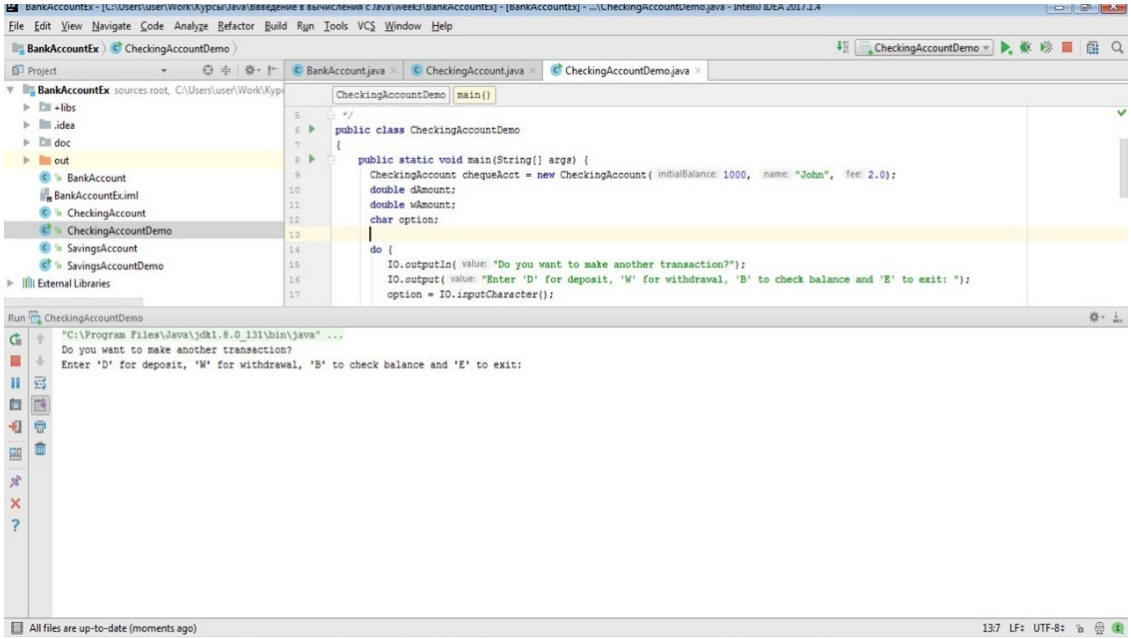
Мы должны использовать метод getBalance, чтобы получить переменную экземпляра balance от BankAccount, потому что там она объявляется как private.

Здесь нет действия для случая по умолчанию.

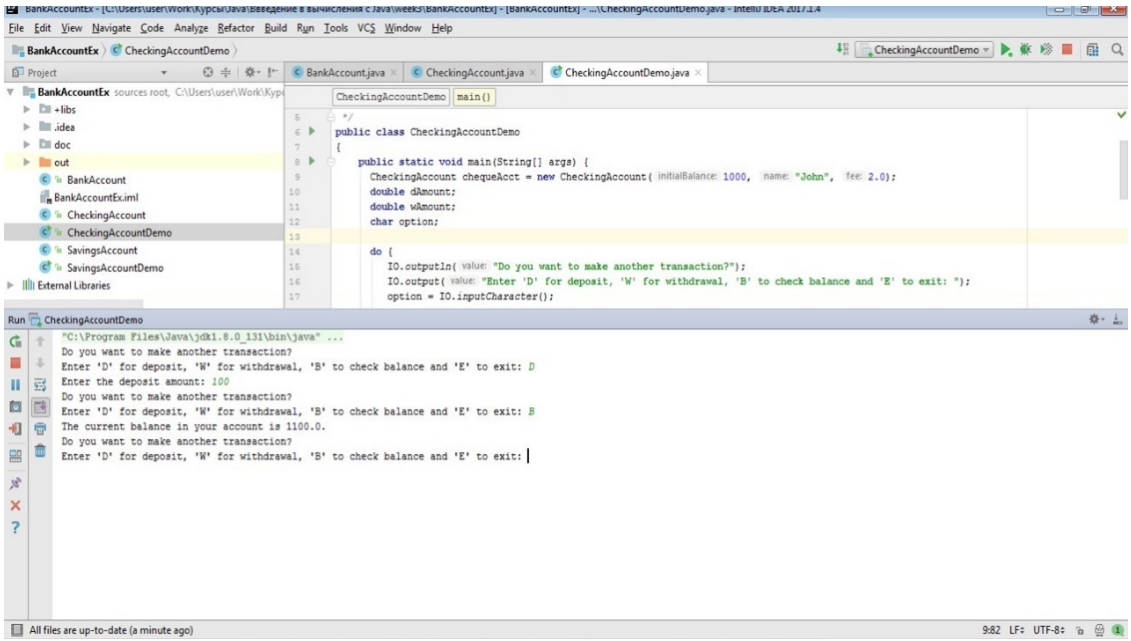
Поэтому этот цикл будет просто продолжаться так долго, как пользователь будет вводить D, W или B.

Цикл здесь на самом деле будет прекращаться при вводе другого символа, не D, W или B.

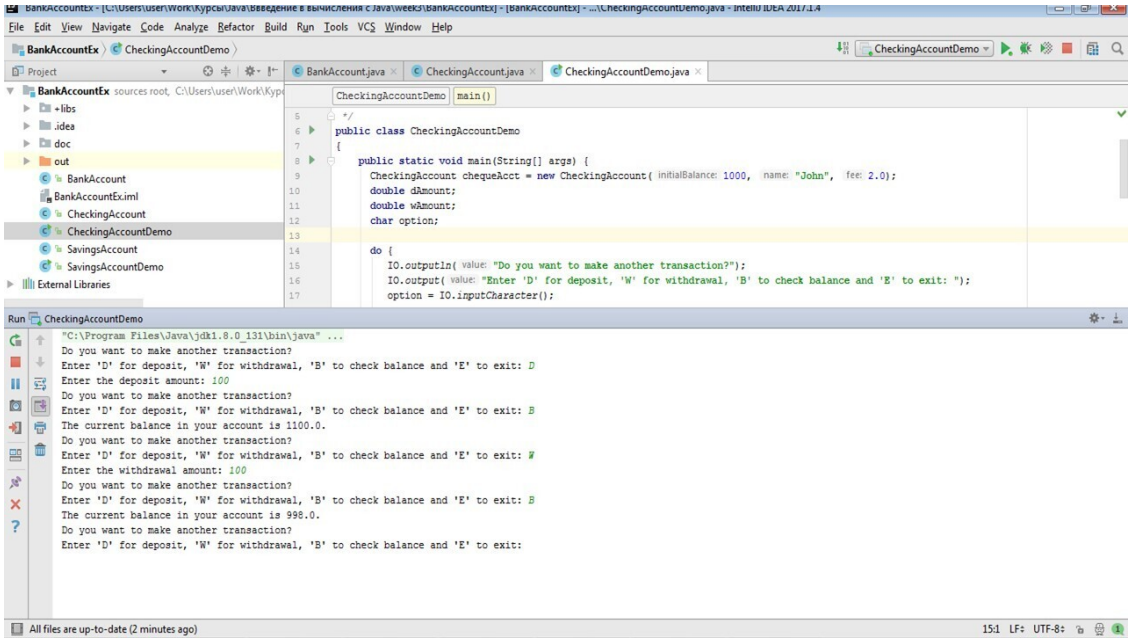
Запустим эту программу.



Здесь вы можете увидеть приглашение на ввод.
Вы должны ввести один из этих вариантов.
Скажем, мы введем D, и сумму вклада, скажем 100 рублей.
Ну, мы действительно не знаем, была ли на самом деле сумма переведена на депозит.
Так что, если мы вводим опцию B, что означает получить баланс, вы можете увидеть, что сумма в 100 рублей была добавлена к исходному балансу 1000.



Если мы хотим сделать вывод, скажем 100 рублей.
И опять же, если вы хотите проверить баланс, вы должны ввести опцию B.
Вы можете видеть, что баланс теперь составляет 998 рублей.



Таким образом, вместо того, чтобы просто вычесть 100 рублей по сравнению с предыдущим балансом, 2 дополнительных рубля были сняты с баланса, потому что есть плата за каждое снятие.

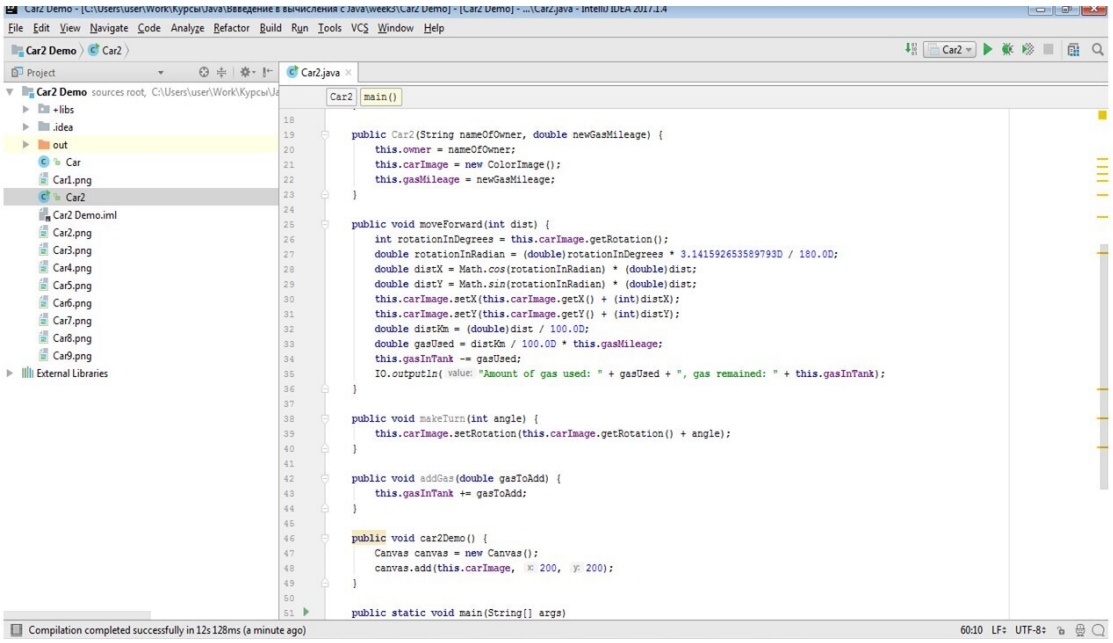
И тогда мы можем выйти из программы, просто введя E.

Существует также благодарственное сообщение. Как определено здесь в конце программы.

Демонстрация цикла

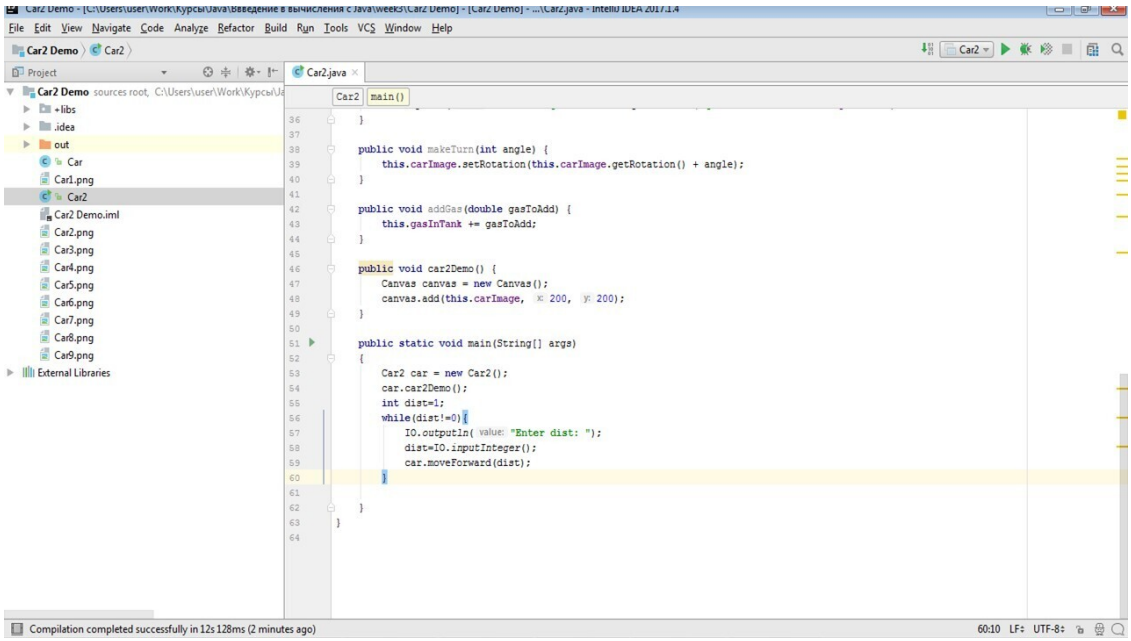
Давайте рассмотрим несколько примеров циклов.

Помните, мы написали программу Java для перемещения автомобиля на холсте вдоль ориентации автомобиля.



Вместо того чтобы автомобиль двигался постоянно вперед, то, что вы увидите, это автомобиль будет прыгать с одного места на другое.

Давайте попробуем запустить программу, чтобы увидеть эффект.



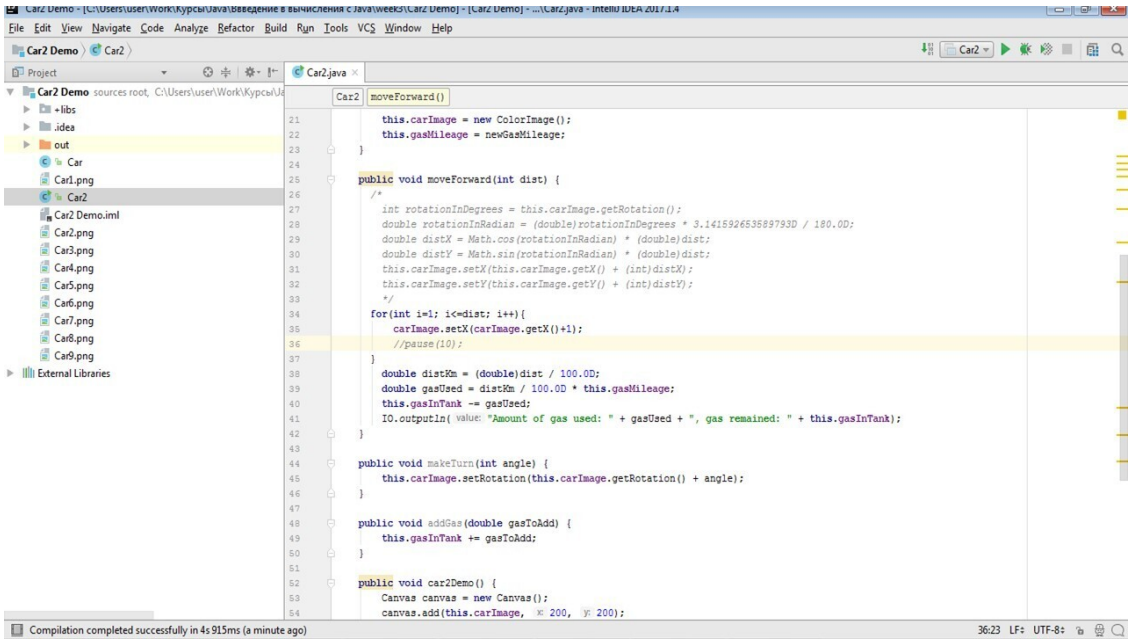
Таким образом, мы можем создать новый автомобиль, просто используя конструктор по умолчанию.

Вызовите метод car2Demo для отображения автомобиля на холсте.

Скажем, если мы хотим переместить автомобиль вперед на определенное расстояние, скажем 400, Вы увидите, что автомобиль просто прыгнет с начального положения в положение конечное.

Если вы хотите, чтобы автомобиль перемещался постоянно, вы должны обновлять его положение шаг за шагом.

Внесем изменения в программу.



В этой программе, вы можете видеть, что здесь есть цикл.

И здесь я упростил программу, просто перемещая автомобиль вперед горизонтально.

Вы можете изменить программу, чтобы попытаться перемещать автомобиль в любом направлении.

В этом модифицированном коде moveForward, я ввел цикл, в этом случае for цикл.

Для заданного расстояния, заданного в качестве параметра, вместо обновления положения изображения на параметр dist, цикл здесь будет двигать автомобиль пиксель за пикселем.

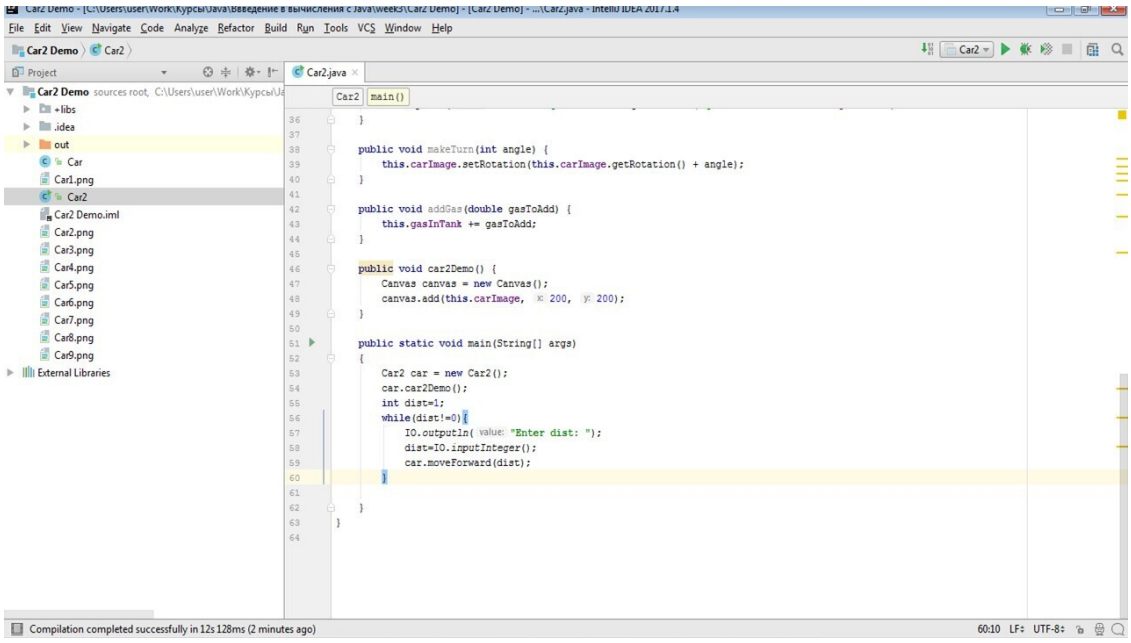
Таким образом, вы увидите, что автомобиль движется вперед на один пиксель за один цикл, на общее количество циклов dist.

Так, в конце, автомобиль переместится вперед на dist пикселей.

Вы можете подумать о том, что сделать, если dist будет отрицательным, то есть для перемещения автомобиля назад.

А также, вы можете рассмотреть перемещение автомобиля по его ориентации.

Рассмотрим эту простую реализацию.



Снова создадим экземпляр класса по умолчанию и отобразим автомобиль на холсте.

Давайте посмотрим, что произойдет, если мы попытаемся переместить автомобиль вперед, скажем, на 400 пикселей.

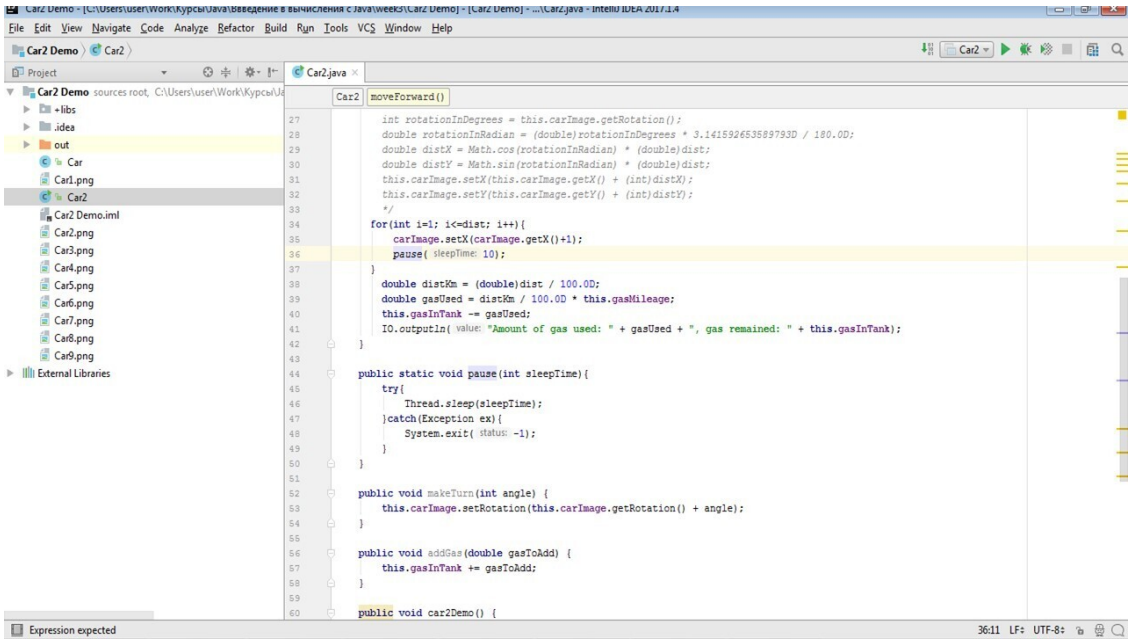
Это все еще выглядит, как будто он прыгнул с одного места на другое.

Это потому, что наш компьютер так быстро работает.

Хотя автомобиль фактически двигался пиксель за пикселем, экран обновлялся так быстро, что вы действительно не видели этого движения.

Для того чтобы увидеть, как автомобиль движется непрерывно, шаг за шагом, вы должны замедлить обработку.

Здесь я создал метод с именем pause, вызывающий системный метод sleep.



Вы не должны беспокоиться о try-catch блоке здесь.

Это в основном для обработки ошибок, в случае если системный вызов создает некоторые ошибки.

Так что с помощью этого метода можно замедлить обработку.

Что вы можете сделать, это ввести выражение pause в цикле, единица здесь миллисекунды.

Так что каждый раз цикл будет останавливаться на 10 миллисекунд.

Давайте скомпилируем программу, и создадим экземпляр автомобиля еще раз, используя конструктор по умолчанию.

Отобразим автомобиль на холсте. Попробуйте переместить его снова на 400 пикселей.

Таким образом, вы можете увидеть, что автомобиль на самом деле движется непрерывно вперед.

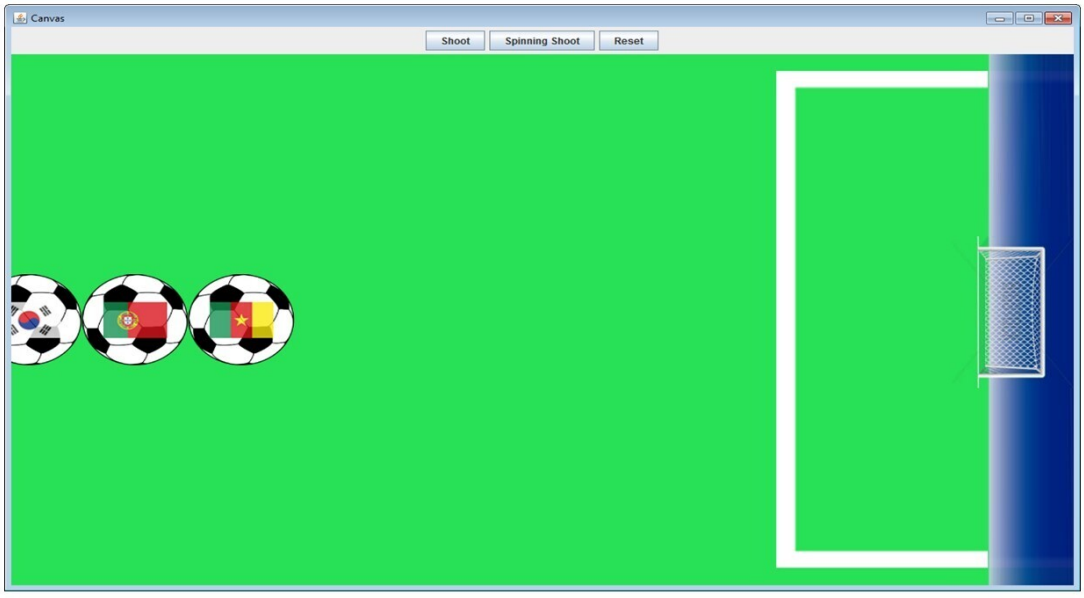
Вы можете изменить скорость автомобиля.

Скажем, если мы хотим замедлить его, сделайте паузу 30 миллисекунд.

Запустите еще раз, и вы увидите, что это намного медленнее.

Мы также создали простую демонстрационную программу для вас.

Вместо того чтобы двигать автомобиль, здесь вы будете двигать футбольный мяч.



Хотя чемпионат мира уже закончился, вы можете создать свой собственный чемпионат мира.

Здесь можно увидеть, что создается футбольное поле, и есть различные опции.

Первая опция просто забросить мяч вперед.

Это похоже на перемещение автомобиля вперед, как то, что мы только что видели. И здесь можно увидеть, как мяч движется вперед.

Но здесь это не выглядит слишком естественно, потому что обычно, когда вы бьете по мячу, мяч вращается.

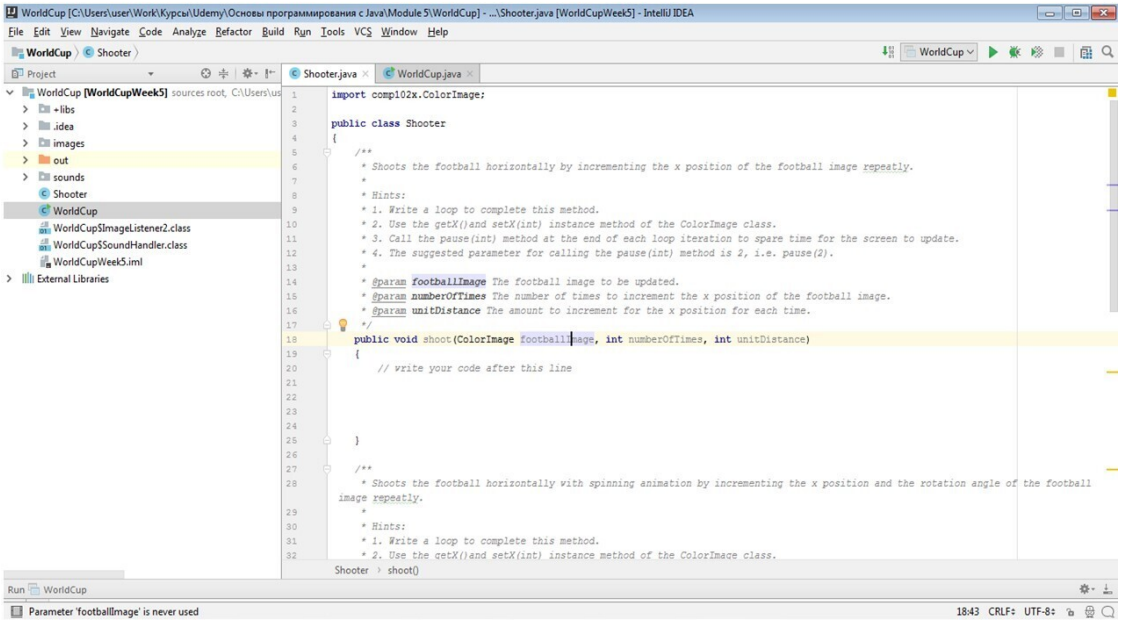
Но здесь этого нет.

Вы можете изменить программу, введя там вращение.

Здесь есть два метода.

Метод shoot просто забрасывает мяч вперед без кругового движения.

И метод spinningShoot забрасывает мяч вперед с круговым движением.



Чтобы запустить программу, Вы можете сделать следующие шаги.

1. Открыть проект.

2. Завершить методы shoot() и spinningShoot() в соответствии с инструкциями, написанными в классе Shooter.

3. Запустите программу, щелкнув правой кнопкой на классе WorldCup, а затем выполнив метод main.

4. Затем можно проверить вашу программу, нажимая на кнопки управления, расположенные в верхней части окна программы.

Массивы

Последнее время мы обсуждали различные структуры управления.

И я хочу обратить ваше внимание опять на то, что отступы кода игнорируются в Java, и ключевое слово else соотносится с ближайшим if.

Если вы хотите соотнести else с другим if, вы должны использовать фигурные скобки.

Использование фигурных скобок также улучшает читаемость кода.

Однако другие языки программирования могут следовать другому синтаксису, например, в языке Python отступы кода могут использоваться для связывания if и else.

Теперь давайте теперь обсудим новую тему – массивы.

Циклы используются часто вместе с массивами.

Во-первых, давайте поговорим о том, почему нам нужны массивы.

До этого, мы использовали переменные, которые представляли только один объект данных. Поэтому каждое имя, которое мы использовали, могло представлять только одну порцию примитивных данных или один экземпляр объекта.

Хотя у объекта могло быть несколько полей, вам нужно было давать различным экземплярам объектов различные имена.

При этом возникают трудности, когда у вас есть большое число объектов одного типа.

Например, когда у вас есть большое число студентов, скажем 100 студентов в классе, и для хранения экзаменационных оценок каждого студента, вы должны создать имена переменных Score1, Score2, Score3, и так далее до Score100.

Score1, Score2, Score3, и так далее до Score100

Score1=Score1 + 1, Score2 =Score2 + 1, и так далее

Score1++, Score2++, и до Score100++

Сделайте объявление 100 различных переменных и представьте, что вам нужно сделать что-нибудь с оценками, скажем, добавить бонус к каждой оценке, для этого вам нужно будет написать 100 выражений в форме Score1=Score1 + 1, Score2 =Score2 + 1, и так далее.

И даже если вы используете оператор инкремента, вы все равно должны написать Score1++, Score2++, и до Score100++.

И нет таких программистов, которые бы пожелали выполнять такую утомительную работу.

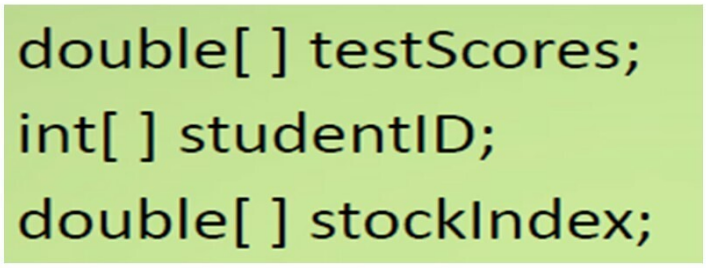
Поэтому большинство высокоуровневых языков программирования включают в себя встроенные структуры данных, называемые массивами.

Использование массива упрощает выполнение таких утомительных задач, как, например, определение средней оценки, максимальной и минимальной оценки, вывод списка оценок, сортировка списка, когда список имен выстраивается в алфавитном порядке.

В реальной жизни иногда вам нужно обрабатывать даже многомерные данные, например, работать с 2-х мерными цифровыми изображениями. И вы найдете, что массивы будут полезны для таких приложений.

Массив – это коллекция однородных объектов данных, при этом данные должны быть одного типа.

Синтаксис объявления массива – это, во-первых, определить тип данных, затем идут квадратные скобки.



Здесь тип данных – это тип данных, которые должны храниться в массиве.

Это может быть примитивный тип данных, int, double, или пользовательский тип – класс, как мы рассматривали раньше, CourseGrade или Car.

И помним, что здесь используются квадратные скобки, а не круглые или фигурные скобки.

Имя массива, как и имя переменной, может быть любым действительным именем для переменной.

На слайде показаны примеры оформления массивов.

Первый пример, это объявление массива double типа с именем testScores.

Второй пример, это массив целых чисел с именем studentID.

И последний пример, это double массив с именем stockIndex.

Эти объявления на самом деле не создают массив. Они только создают ссылочную переменную массива.

Для того чтобы реально создать массив, вы можете использовать оператор new и определить размер массива в квадратных скобах, который вы хотите создать.

```
testScores = new double[100];
studentID = new int[50];
double[ ] stockIndex = new double[365];
```

И в этом случае вы создаете массив с именем testScores, типа double, с 100 элементами.

Во втором примере создается массив studentID типа int, с 50 элементами.

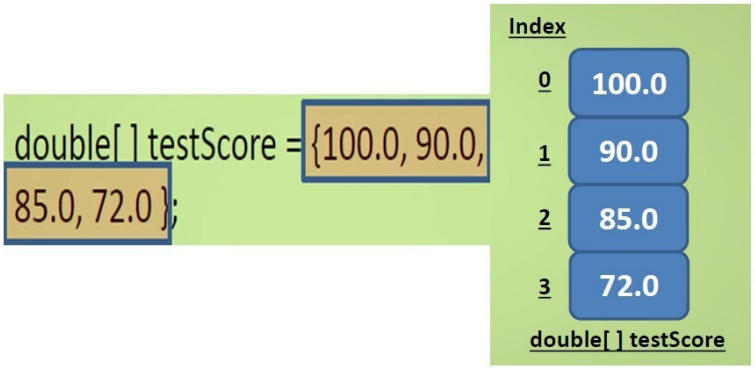
И последний пример немного отличается. В нем объявляется и сразу создается массив в одном выражении. В этом выражении создается массив с именем stockIndex, типа double, с 365 элементами.

Нужно отметить, что размер массива не может быть отрицательным.

Java позволяет создавать массивы с нулевым размером, но такой пустой массив не может что-либо хранить.

Java также позволяет объявлять массивы, создавать их и инициализировать в одном выражении.

В этом примере создается массив с именем testScore, типа double.




```
double[ ] testScore;
testScore = {100.0, 90.0, 85.0, 72.0};
```

Четыре числа внутри фигурных скобок затем используются для инициализации четырех элементов массива.

И результат проиллюстрирован здесь диаграммой.

Заметьте, что размер testScore определяется, когда массив создается, используя 4 числа при инициализации.

Я хочу указать, что такая короткая инициализация должна быть сделана в одном выражении.

Например, если массив уже объявлен, вы не можете инициализировать массив с помощью списка начальных значений в фигурных скобках. Java выдаст вам ошибку в этом случае.

Java инициализирует элементы массива в их значения по умолчанию. Но хорошая практика программирования не полагаться на это.

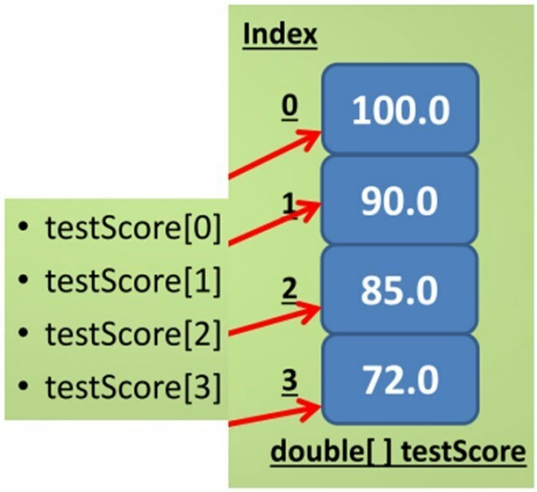
Отдельные элементы массива могут быть доступны с помощью индекса.

Рассматривая последний пример, действительные индексы находятся в диапазоне от 0 до 3 для массива размером 4.

Важно помнить, что индекс массива начинается с 0, а не с 1. Общая ошибка, индексировать первый элемент, используя 1, вместо 0.

Поэтому, действительные индексы в этом примере 0, 1, 2, 3.

Сослаться на отдельный элемент массива можно, указав имя массива, а затем индекс в квадратных скобках.

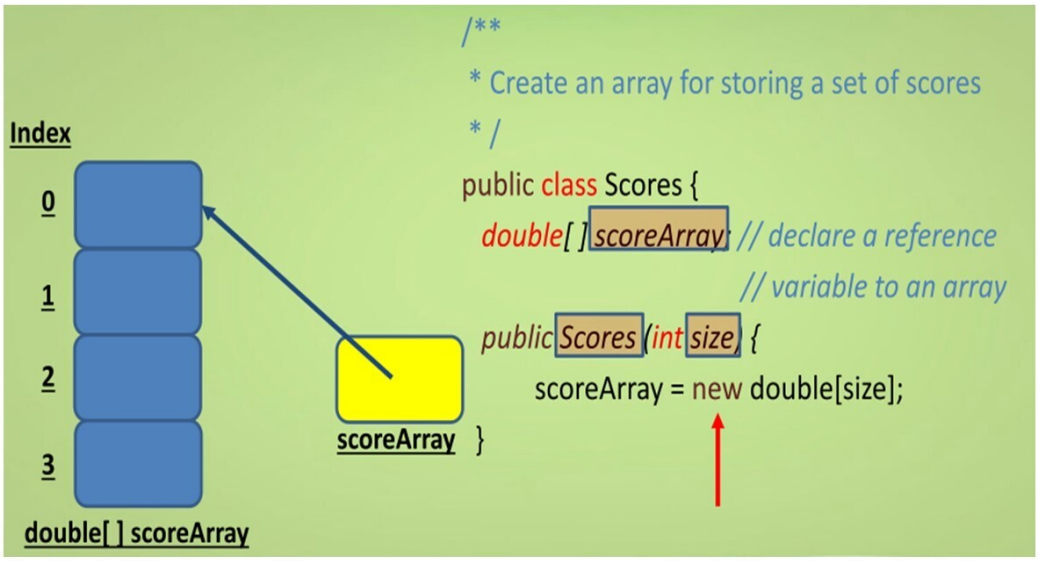


В этом примере первый элемент testScore с 0 индексом имеет значение 100.

testScore с индексом 1 даст значение 90.

testScore с индексом 2 имеет значение 85.

Последний элемент массива с индексом 3 имеет значение 72.



В этом примере определяется класс Scores.

Внутри класса объявляется переменная экземпляра scoreArray double типа, но она еще не создается.

Конструктор определяется с 1 параметром, который дает размер массива.

Надо заметить, что в Java размер массива может быть определен динамически, т.е. во время выполнения.

Когда вызывается конструктор, создается массив.

Предположим, размер имеет значение 4, и диаграмма здесь иллюстрирует создание массива.

Ссылочная переменная scoreArray даст адрес памяти, где расположен первый элемент массива scoreArray.

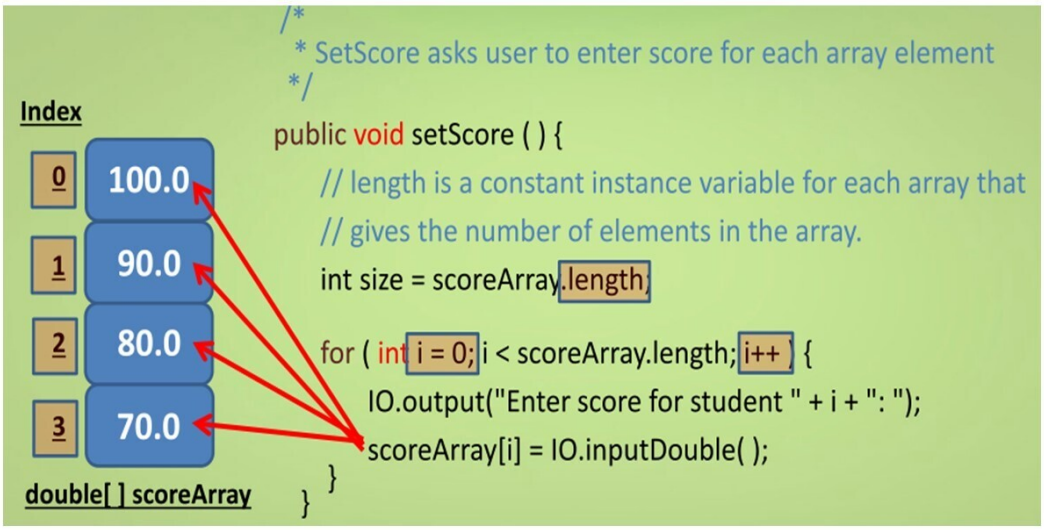
Как я упомянул, Java будет инициализировать элементы массива значениями по умолчанию. В случае целых чисел, эти значения будут 0.

Но рекомендовано самостоятельно инициализировать массив перед его использованием.

Заметьте, что первый элемент массива имеет индекс 0, а последний индекс имеет значение 3, которое меньше чем размер массива.

Примеры

Давайте посмотрим на метод setScore, который работает с массивом scoreArray, который мы только что создали.



Метод setScore спрашивает пользователя ввести оценки, одну за другой, в соответствующие элементы массива.

Перед тем как начать ввод, мы должны определить количество элементов массива.

В Java размер массива может быть получен из финальной переменной экземпляра с именем length.

Поэтому, поместив имя массива, в этом случае, scoreArray, перед .length, это даст нам размер массива scoreArray.

Помним, что размер массива на единицу больше, чем максимальный индекс массива.

Цикл for внутри метода в первую очередь инициализирует счетчик-переменную i в 0.

При этом, ссылаясь на первый элемент массива scoreArray, внутри цикла, пользователь будет приглашен для ввода.

Если пользователь введет 100 как оценку, IO.inputDouble возьмет введенное значение и присвоит его элементу массива с индексом 0, который является первым элементом.

Индекс и позиция в массиве отличаются, но мы должны принять это.

Когда действие внутри тела цикла for завершится, счетчик i увеличится на 1.

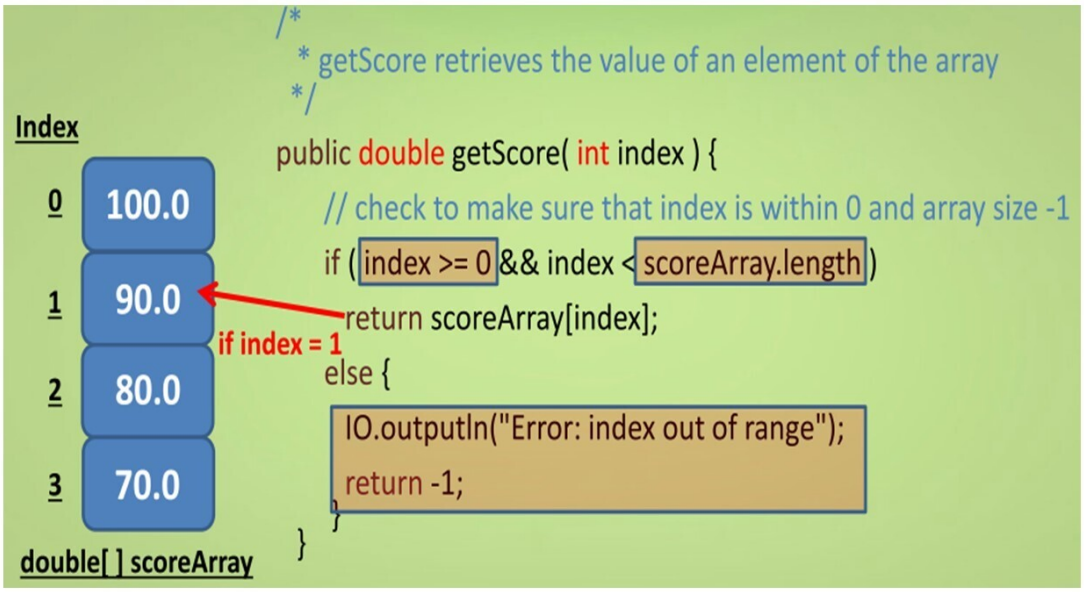
Пользователь введет другую оценку, скажем 90, которая присвоится индексу 1 массива.

Далее процесс продолжится и i++ увеличит i до 2.

Следующее введенное значение, скажем 80, будет присвоено третьему элементу массива.

Выражение i++ затем увеличит i до 3 и последний элемент массива примет значение 70.

Когда условие цикла будет проверено снова, i++ даст значение 4, которое будет больше, чем размер массива, и цикл будет завершен.



Метод getScore будет возвращать значение элемента массива, который имеет индекс, указанный параметром метода.

Это может быть достигнуто, используя простое if-else выражение, где мы делаем первую проверку индекса в диапазоне.

Помним, что действительный диапазон индексов от 0 до размера массива минус 1.

Первое условие здесь проверяет индекс больше или равный 0, и второе условие проверяет индекс меньше чем размер массива, который дается переменной экземпляра length.

Второе условие строго меньше, потому что максимальный индекс – это размер массива минус 1.

Если оба условия возвращают true, getScore возвратит значение, хранимое в индексе массива.

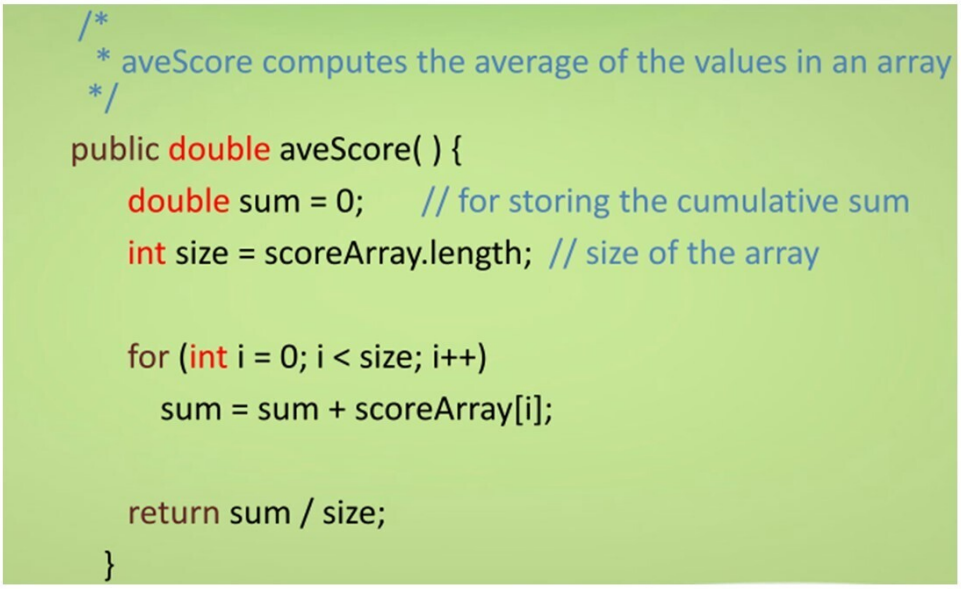
Например, если индекс 1, будет возвращен второй элемент scoreArray.

Если одно из условий возвращает false, условное выражение возвратит false и выражение else будет выполнено.

Сообщение об ошибке будет выведено на экран и будет возвращено значение -1.

Возвращаемое значение здесь ничего не означает, но что-то должно быть возвращено, потому что getScore метод определен как возвращающий значение типа double.

Так как значения хранятся в массиве, мы можем выполнять различные операции над массивом, перемещаясь по массиву один элемент за другим.



```
/*
 * aveScore computes the average of the values in an array
 */
public double aveScore( ) {
    double sum = 0;      // for storing the cumulative sum
    int size = scoreArray.length;  // size of the array

    for (int i = 0; i < size; i++)
       sum = sum + scoreArray[i];

    return sum / size;
 }
```

Следующий метод, который я собираюсь здесь обсудить – это aveScore, он вычисляет среднее всех значений, хранимых в массиве.

Предположим, массив определяется переменной экземпляра, в противном случае, мы можем передать массив в метод как параметр.

Я вернусь к этому позже.

Метод начинается с определения двух переменных.

Одна переменная типа double для хранения суммы всех значений массива и вторая для размера массива, который может быть определен из переменной экземпляра length.

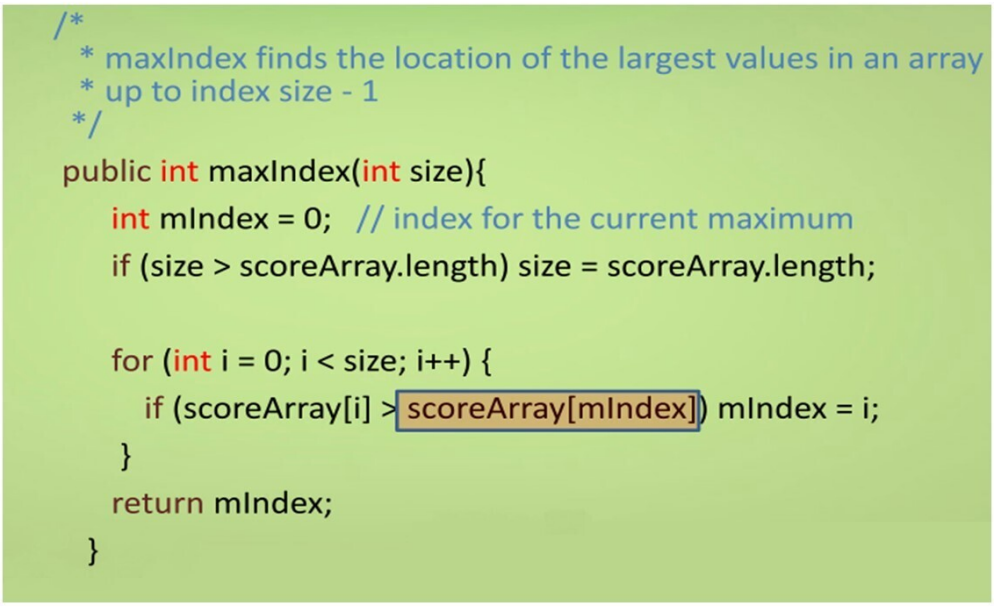
Главная часть программы обновляет общую сумму, добавляя значение каждого элемента массива, один за другим, используя цикл for.

Мы уже видели подобный цикл for в методе setScore, так что это общий подход для перебора элементов массива фиксированной длины.

И помним снова, что индекс начинается с 0 и он меньше чем размер массива.

Последнее выражение метода aveScore делит общую сумму на размер, что даст среднее всех оценок, и среднее затем будет возвращено как результат.

Другая общераспространенная операция для массива, это нахождение минимальных и максимальных значений в массиве.



```
/*
 * maxIndex finds the location of the largest values in an array
 * up to index size - 1
 */
public int maxIndex(int size){
    int mIndex = 0;   // index for the current maximum
    if (size > scoreArray.length) size = scoreArray.length;

    for (int i = 0; i < size; i++) {
       if (scoreArray[i] > scoreArray[mIndex]) mIndex = i;
    }
    return mIndex;
}
```

Этот метод maxIndex, который вместо нахождения максимального значения, находит позицию или индекс элемента массива, который дает максимальное значение.

Метод maxIndex имеет один параметр, это размер целого типа, и возвращает целое значение, которое дает индекс максимального элемента.

Вы может быть удивлены, почему мы определяем размер как параметр, а не используем переменную экземпляра length для определения размера массива.

Это делает метод более гибким, так как в некоторых приложениях вы, может быть, захотите обработать только часть массива, вместо перебора всего массива.

Я дам вам больше примеров использования этой опции позже.

Наш метод объявляет локальную переменную mIndex, для хранения позиции максимального элемента.

Так как размер передается в метод как параметр, вместо использования переменной length, мы должны убедиться, что размер не больше реального размера массива.

Если размер будет слишком большим, мы его сбросим до реального размера.

Цикл for будет проверять элементы массива, один за другим, до индекса равного размеру минус 1, и будет проверять текущий элемент массива больше чем текущий максимум.

Заметьте, что текущий максимум получается с помощью переменной mIndex, потому что это позиция найденного максимума.

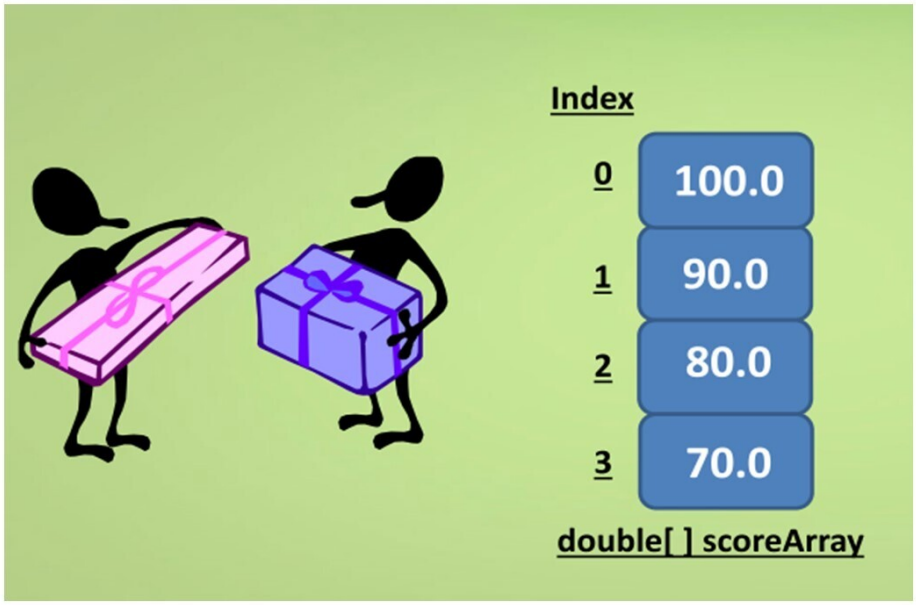
Если условие верно, новый максимум найден и переменную mIndex нужно обновить в текущий индекс.

Когда цикл for завершится, mIndex будет содержать позицию максимального элемента массива, и его значение будет возвращено как результат.

Вы можете подумать о том, что произойдет, если максимальное значение встретится несколько раз в массиве.

Перестановка элементов

Одна операция, которая часто встречается при работе с массивами, это перестановка двух элементов.

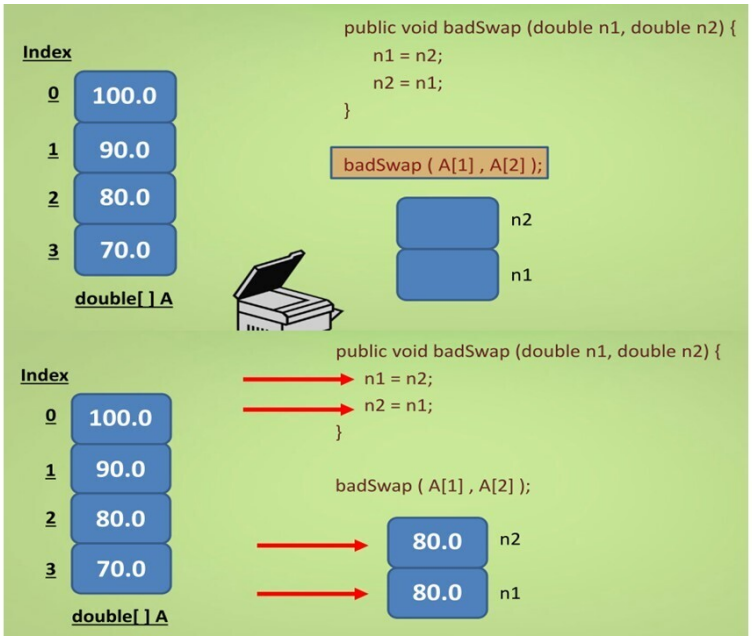


В этом примере значения второго и третьего элементов массива scoreArray будут переставлены.

Это похоже на то, как вы обмениваетесь подарками на Рождество.

Давайте сделаем нашу первую попытку написать метод перестановки двух элементов в массиве A.

Я назову этот метод badSwap, так что вы поймете, что этот метод не предназначен для работы.



Метод badSwap имеет два параметра типа double.

При вызове этого метода мы хотим переставить элементы с индексами 1 и 2 в массиве A, и они будут переданы в метод как параметры.

Эти параметры будут обрабатываться как локальные переменные, и память будет выделяться для них из стека памяти.

Так как отдельные элементы массива A[1] и A[2] имеют тип double, который является примитивным типом, их значения будут скопированы в область памяти для параметров.

Теперь n1 имеет значение 90, и n2 имеет значение 80.

Когда в badSwap выполняется первое выражение n1 равно n2, значение n2, которое равно 80, присваивается n1, и старое значение n1 стирается и заменяется на 80.

Теперь и n1 и n2 имеют одинаковое значение 80.

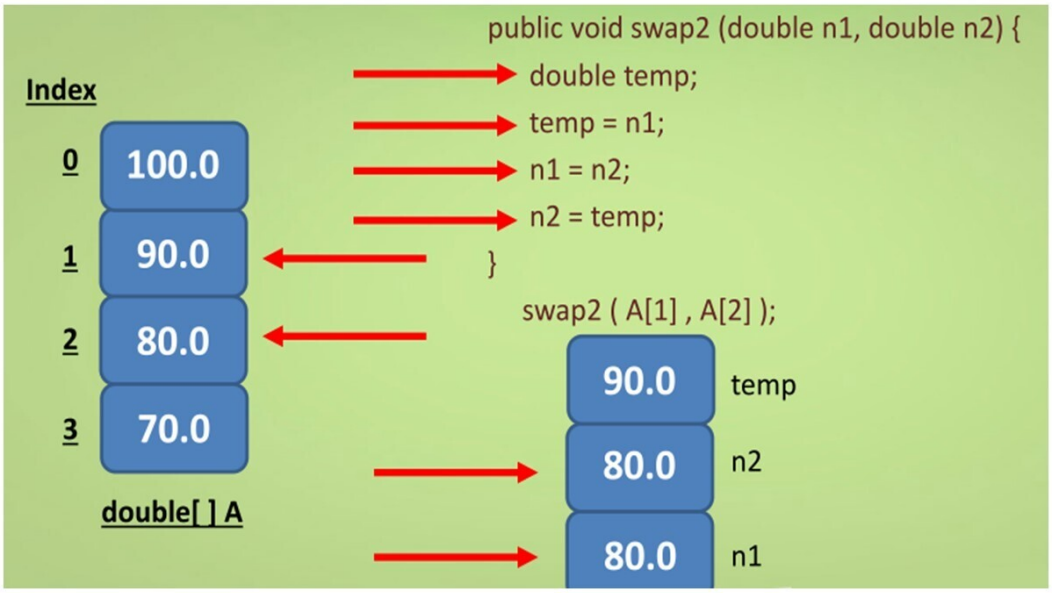
Когда выполняется второе выражение n2 равно n1, значение n1, которое равно 80, будет присвоено n2.

В результате обе переменные n1 и n2 получат значение 80 в конце выполнения метода.

Ясно, что это не то, для чего этот метод предназначен.

Один способ решить эту проблему, это в начале где-нибудь сохранить значение n1, перед выполнением первого выражения n1 равно n2, так что старое значение n1 может позже быть присвоено n2.

Давайте посмотрим на другую версию метода перестановки с именем swap2.



Локальная переменная temp объявлена в методе swap2.

Поэтому в дополнение к переменным параметров n1 и n2, еще одна дополнительная ячейка памяти должна быть выделена для переменной temp.

Когда выполняется первое выражение temp равно n1, значение n1 или 90 присваивается temp.

Так что сохраняется временная копия n1.

Когда выполняется второе выражение n1 равно n2, значение n1 удаляется и заменяется на 80.

Заметьте, что теперь обе переменные n1 и n2 имеют одинаковое значение 80.

Вместо того, чтобы присвоить n1 в n2, как мы делали перед этим в методе badSwap, присваивать мы теперь будем temp, которая хранит старое значение n1, в n2.

Так что в конце выполнения метода значения n1 и n2 будут переставлены.

Но решили ли мы действительно проблему перестановки значений элементов массива A[1] и A[2]?

Элементы A[1] и A[2] все еще имеют те же значения, как и раньше.

Помним, что после выполнения метода swap2, вся информация, содержащаяся в локальных переменных, будет удалена и никакой перестановки не будет сделано в элементах массива.

Это потому что отдельные элементы массива A[1] и A[2] имеют тип double, который является примитивным типом.

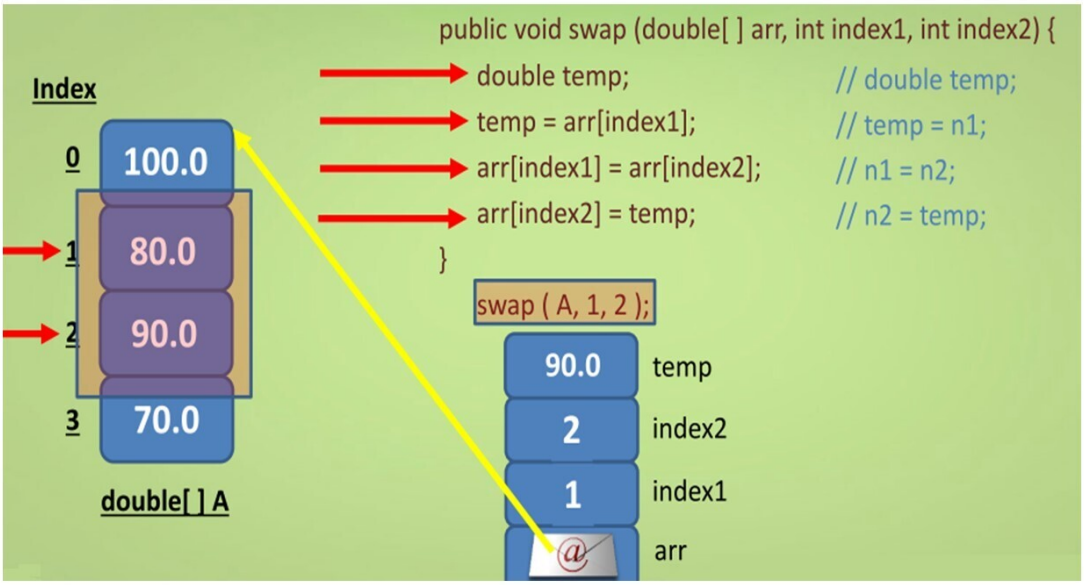
Поэтому только копии значений передаются в метод swap2, и все что изменяется в методе swap2, не влияет на значения массива.

Так как мы можем решить эту проблему?

Если массив и метод определены внутри одного класса, мы можем получить доступ к элементам массива напрямую без передачи их в метод как параметры.

Это легко сделать, и вы можете подумать, как это можно реализовать.

Давайте напишем более общий метод для перестановки двух элементов в массиве, даже если массив и метод определены не в одном и том же классе.



Метод swap здесь имеет три параметра.

Первый параметр – это массив типа double.

Далее index1 и index2, это индексы элементов массива для перестановки.

Заметьте, что тело этого метода похоже на предыдущий метод.

Для иллюстрации, как метод работает: массив A и индексы 1 и 2 передаются в метод.

Далее метод переставляет второй элемент или 1 индекс и третий элемент или 2 индекс массива A.

Когда метод вызывается, память для трех локальных переменных, соответствующих трем параметрам, должна быть выделена.

Массив A используется как первый аргумент.

Так как A – это переменная ссылочного типа, помним, что мы используем аналогию больших посылок для объектов, которые не помещаются в слот памяти.

Вместо того чтобы хранить все переменные экземпляра в стеке памяти, указывается адрес в куче памяти.

Так что здесь переменная параметра arr дает адрес начала области памяти для массива A.

Параметры index1 и index2 дают значения 1 и 2, как аргументы вызова метода.

Объявление локальной переменной temp выделит память типа double.

Далее будет вычисляться правая сторона этих выражений.

При этом будет сделана ссылка на элемент массива arr, имеющий индекс index1, и это 1.

Для массива arr, его ссылочная переменная указывает на начало массива A.

Эта переменная знает, как обнаружить массив A, а затем извлечь элемент с индексом 1 или второй элемент массива.

Так как второй элемент A имеет значение 90, это значение будет присвоено переменной temp.

Когда будет вычисляться правая часть второго выражения, значение элемента A с индексом index2, равным 2, будет извлечено.

Это даст значение 80, которое будет присвоено элементу массива с индексом index1, равным 1.

Заметьте, что значение второго элемента A теперь изменилось с 80 на 90.

Последнее выражение метода присвоит значение переменной temp элементу массива A с индексом index2, и так как temp имеет значение 90, начальное значение третьего элемента массива A заменится на 90.

Теперь в конце выполнения метода, второй и третий элементы массива A обменяются значениями.

Сортировка

Следующие три слайда резюмируют все, что мы обсуждали при использовании массива.

```
 * Demostrate the use of array for storing and manipulating a set of scores
 */
public class Scores
{
   double[] scoreArray; // declare a reference variable to an array

   public Scores(int size) {
       scoreArray = new double[size];
    }
   /**
    * SetScore asks user to enter score for each array element
    */
   public void setScore( ) {
       // length is a constant instance variable that gives the number
       // of elements in an array.
       int size = scoreArray.length;

       for (int i = 0; i < size; i++) {
           IO.output("Enter score for student " + i + ": ");
           scoreArray[i] = IO.inputDouble();
        }
```

Здесь объявляется массив, затем он инициализируется в конструкторе класса.
И затем в методе setScore инициализируются элементы массива.

```
public double getScore(int index) {
    if (index >=0 && index < scoreArray.length)
       return scoreArray[index];
    else {
       IO.outputln("Error: index out of range");
       return -1;
     }
}
```

```
/**
* aveScore computes the average of the values in an array
*/
public double aveScore(){
    double sum = 0;  // for storing the cumulative sum
    int size = scoreArray.length; // size of the array
    // update the cumulative sum by going all elements in the array
    for (int i = 0; i < size; i++)
        sum = sum + scoreArray[i];
    // compute and return the average score
    return sum / size;
 }
```

Метод getScore возвращает элемент массива по его индексу.
А метод aveSore возвращает среднее элементов массива.

```
* maxIndex finds the location of the largest values in an array up to index size - 1
*/
public int maxIndex(int size){
    int maxI = 0;  // index of the current maximum
    // check if size is larger than the maximum size of the array
    if (size > scoreArray.length) size = scoreArray.length;
    for (int i = 0; i < size; i++) {
        // check if there is a new maximum and update the index accordingly
        if (scoreArray[i] > scoreArray[maxI]) maxI = i;
    }
    // return the index of the maximum element
    return maxI;
 }
/**
 * swap two elements of the array arr indexed by index1 and index2
 */
public void swap (double[] arr, int index1, int index2) {
    double temp;
    temp = arr[index1];
    arr[index1] = arr[index2];
    arr[index2] = temp;
 }
```

Здесь методы возвращают индекс наибольшего элемента в массиве и переставляют два элемента массива.

Заметьте, что этот метод находит позицию максимального элемента, а не само максимальное значение.

Мы также обсуждали в деталях как переставить два элемента в массиве и механизм передачи ссылочной переменной массива в другой метод.

Теперь давайте соберем все это вместе и разработаем важную операцию для работы с массивами, которая называется сортировкой.

Сортировка – это процесс расположения элементов списка в определенном порядке, например, в порядке убывания или возрастания, используя числовой или алфавитный порядок.

Множество приложений требует сортировки, например, упорядочивание группы студентов по их имена или оценкам, упорядочивание списка событий в хронологическом порядке для облегчения поиска информации.

Например, в словарях было бы тяжело искать значение слова, если бы слова не были бы упорядочены в алфавитном порядке.

Или смотреть телефонные номера в списке контактов гораздо проще, если они упорядочены.

Или показ списка веб страниц на основе их популярности, который вы видите каждый день в поиске Google.

Сортировка – это наиболее изучаемая область в компьютерной науке.

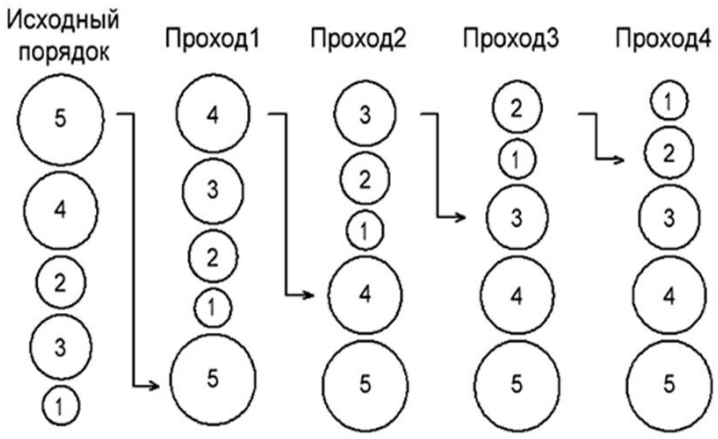
Существует много алгоритмов сортировки, но я здесь только дам вам идею как сортировка может быть выполнена и простой алгоритм для понимания.

Алгоритм сортировки, который мы будем рассматривать, называется выборочной сортировкой.

Идея выборочной сортировки простая и следует способу, которым многие из нас выполняют сортировку.

Предположим, что мы сортируем элементы в возрастающем порядке от небольших элементов к большим элементам.

При этом выборочная сортировка разделит сортируемый массив на две части, отсортированную часть и несортированную часть.



Первоначально весь массив считается несортированным.

Затем вы выполняете сортировку, периодически находя больший элемент в не отсортированной части массива, и помещаете его в конец не отсортированной части массива до тех пор, пока весь массив не отсортируется.

Для описания алгоритма в арифметической форме, первый шаг, это установить весь массив как не отсортированный.

Следующий шаг, это итеративный процесс, когда сначала находится больший элемент в не отсортированной части, а затем найденный элемент переставляется с последним элементом не отсортированной части массива.

Например, ищем в массиве самый большой элемент 10, и переставим его с последним элементом 5 в необработанной части массива, и эти две части становятся разными.

Теперь найдем самый большой элемент 9 в необработанной части массива и обменяем его с последним элементом 1 в той же части массива.

Заметим, что этот шаг идентичен предыдущему, за исключением того, что мы не включили в поиск последний элемент, так как мы уже знаем, что он больший.

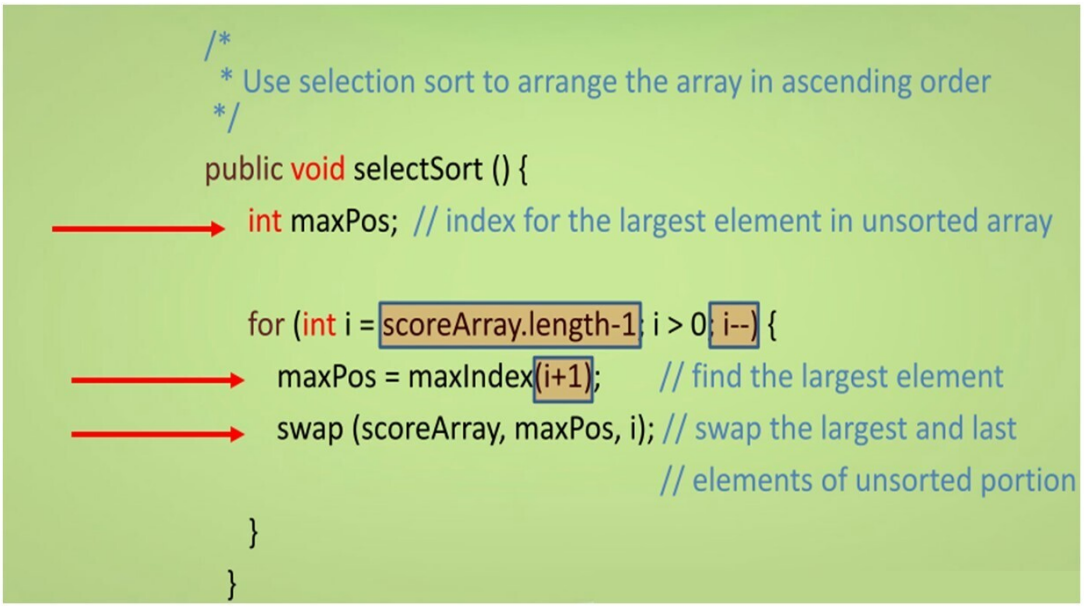
И эта часть массива уже больше не изменяется.

И с каждой итерацией она становится больше на один отсортированный элемент.

И так весь массив становится отсортированным.

Теперь попробуем реализовать выборочную сортировку в Java программе. Вы можете подумать, что это будет сложная программа.

Вы удивитесь, но это полная реализация выборочной сортировки.



Мы начинаем с объявления переменной maxPos, которая используется для отслеживания большего элемента в не отсортированной части массива.

Мы будем использовать методы maxIndex и swap, которые мы разработали ранее, так как в реализации метода maxIndex вместо возвращения максимального значения, нам нужен метод, возвращающий позицию максимального значения, так что мы сможем сделать перестановку, используя позицию или индекс.

Установим цикл for для повторения шага нахождения большего элемента и его перестановки.

Заметим, что i в цикле for начинается с (scoreArray.length – 1), потому что это последний индекс всего массива, который рассматривается в начале как не отсортированный.

И «i – -» здесь чтобы уменьшать размер не отсортированной части массива на 1.

Заметьте также, что аргумент, который используется при вызове maxIndex, это i + 1, потому что, если вы посмотрите на метод maxIndex, его параметр, это размер массива, в котором метод находит больший элемент.

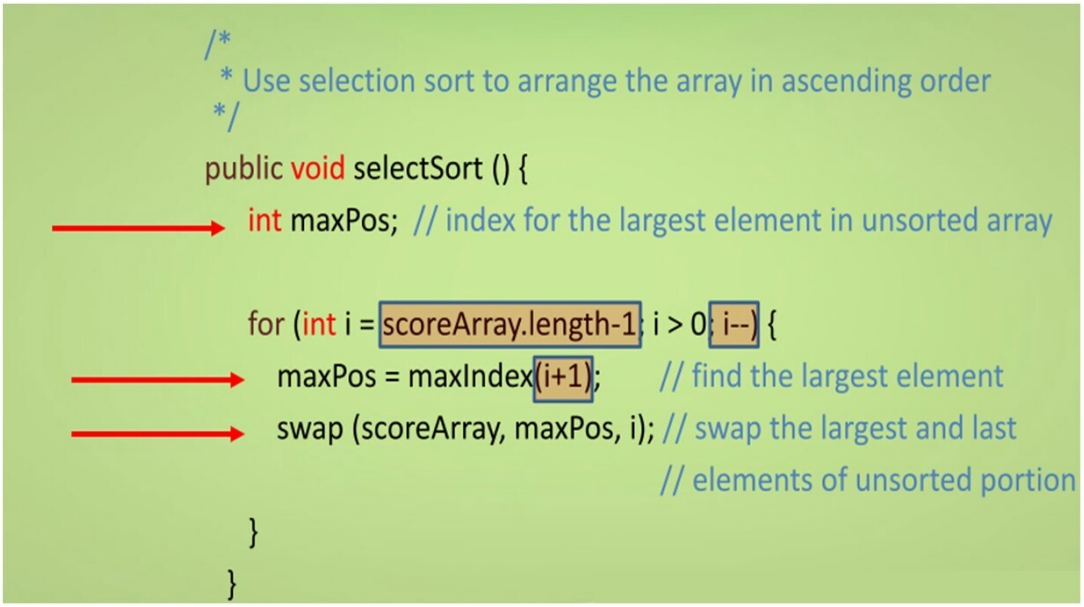
И размер на 1 больше, чем максимальный индекс.

И для метода swap мы передаем адрес массива в метод.

И второй и третий параметры, это индексы элементов массива для перестановки.

Сортировка изображений

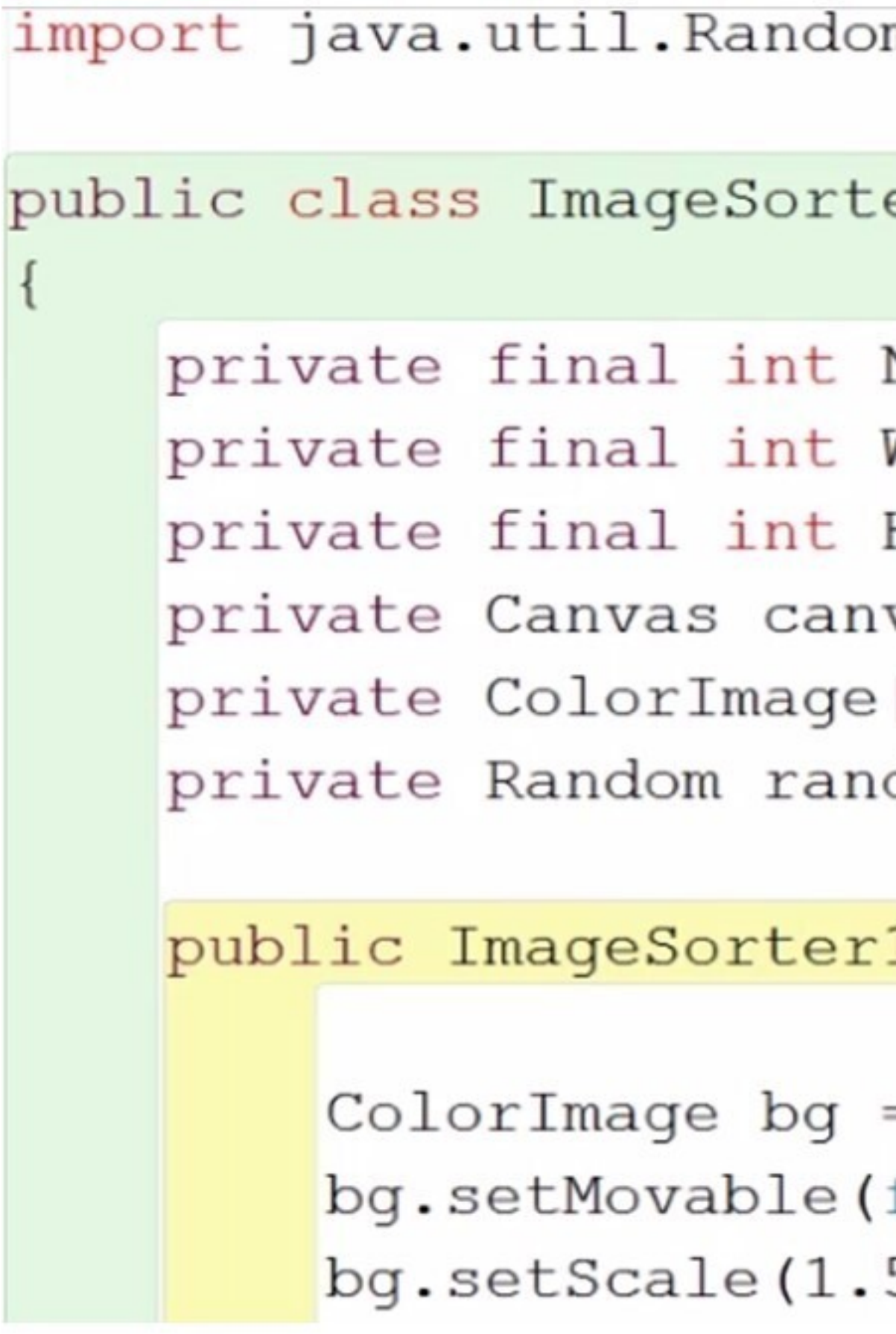
Давайте посмотрим на метод выборочной сортировки, который мы только что обсуждали.



В этом примере мы создали массив оценок, но как я уже говорил, мы также можем создавать массивы различных типов, а не только примитивных типов.

Например, вместо создания массива типа double, как в примере с массивом оценок, мы можем создать массив изображений.

При этом тип массива будет ColorImage.


import java.util.Rando
public class ImageSorte
{
private final int
private final int
private final int
private Canvas can
private ColorImage
private Random ran
public ImageSorter
ColorImage bg
bg.setMovable(
bg.setScale(1.

Здесь пара квадратных скобок указывает, что это массив.

Это объявление создает переменную экземпляра типа массив ColorImage. При этом выделяется память для массива ColorImage с размером NUMBER.

```
public ImageSorter1() {

    ColorImage bg = new ColorImage("background.png");
    bg.setMovable(false);
    bg.setScale(1.5);

    imageArray = new ColorImage[NUMBER];
    imageArray[0] = new ColorImage("Alfred.png");
    imageArray[1] = new ColorImage("Jane.png");
    imageArray[2] = new ColorImage("Mary.png");
    imageArray[3] = new ColorImage("Peter.png");
    imageArray[4] = new ColorImage("Sarah.png");
```

Переменной NUMBER присвоено значение 8.

Затем изображения присваиваются различным элементам массива.

```
/**
 * Sorts the image array by image area in ascending order.
 */
private void selectSort() {

    int maxPos;

    for (int i = imageArray.length - 1; i > 0; i--) {
        maxPos = maxIndex(i + 1);
        swap(maxPos, i);
    }
}
```

И здесь вы также видите метод selectSort, который похож на предыдущий метод, за исключением длины массива, которая определяется как imageArray.length.

```
/**
 * Swaps two color images in the images array.
 */
private void swap(int index1, int index2) {

    ColorImage temp = imageArray[index1];
    imageArray[index1] = imageArray[index2];
    imageArray[index2] = temp;
}
```

У нас также есть метод swap.

Отличие от предыдущего метода здесь в том, что вместо передачи массива в swap метод, так как массив объявлен как переменная экземпляра, он доступен внутри метода.

Перестановка же здесь похожа на предыдущую реализацию.

```
private int maxIndex(int size) {

    int mIndex = 0;
    if (size > imageArray.length) size = imageArray.length;

    /*
     * Complete the for loop below to find the index of the
     * image with the maximum height
     */

    for (int i = 0; i < size; i++) {



    }
```

Для метода maxIndex я сознательно опустил детали, так что для вас будет хорошим упражнением завершить его.

Теперь посмотрим, что этот класс должен делать.

Если вы создаете экземпляр класса ImageSorter, вы можете видеть, что изображения отображаются на холсте.

И мы пытаемся отсортировать эти изображения по их высоте.

Так что упорядочивание должно начаться с небольшого изображения и закончиться большим изображением.

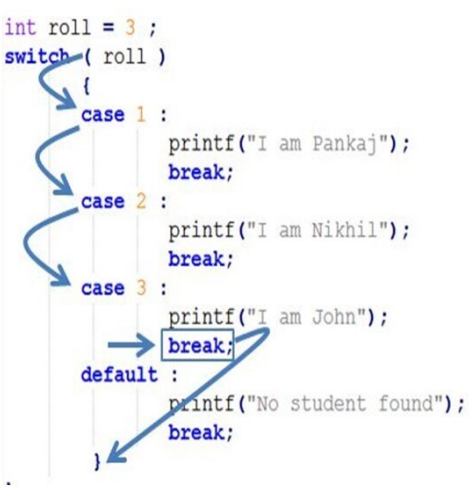
Вызовом метода moveInPosition, вы можете увидеть изображения, расположенные в возрастающем порядке по их высоте.

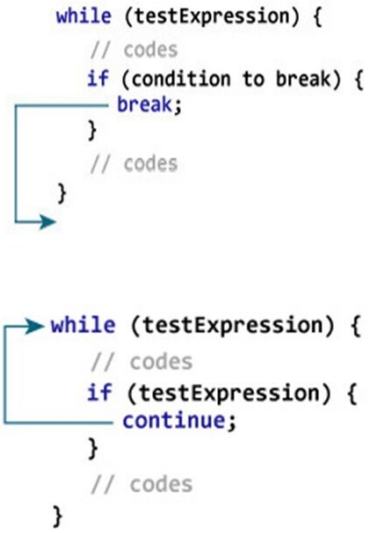
Break и continue

Перед тем как я перейду к многомерным массивам, я хочу рассмотреть пару выражений, которые могут быть полезны для циклов и массивов.

Это выражения break и continue.

Помните ли вы, где мы уже видели выражение break? Это было в выражении switch.





Когда break используется в выражении switch, выполнение кода прерывается.

Выражение break может также использоваться в любом типе цикла.

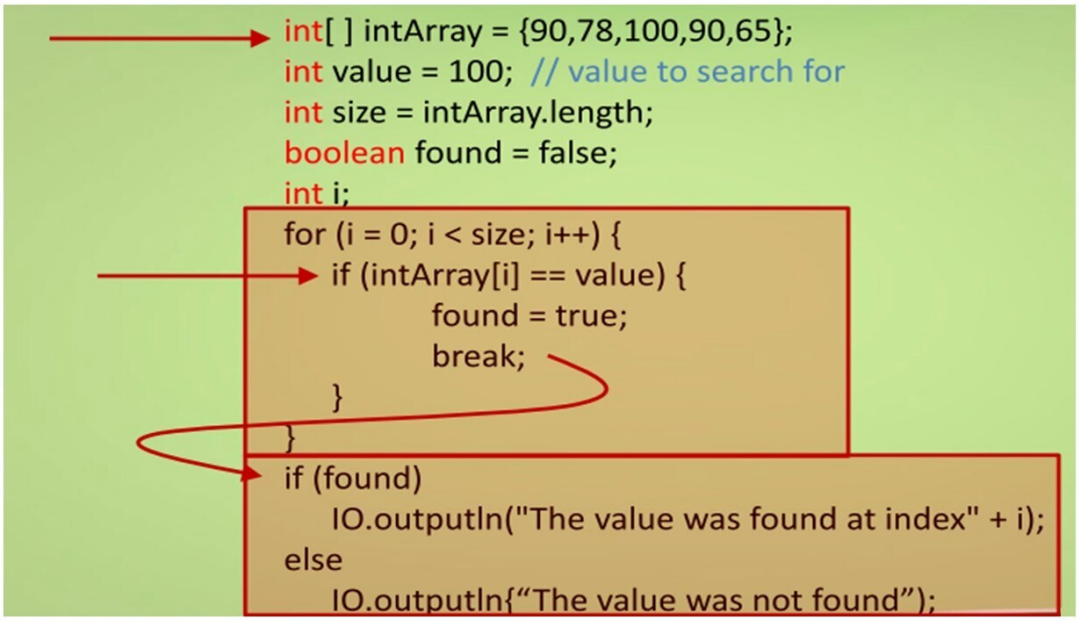
Когда break выражение встречается в цикле, выполнение цикла завершается.

Я приведу соответствующий пример позже.

В случае continue выражения, когда оно встречается в цикле, оставшийся код тела цикла пропускается и начинается новая итерация цикла.

Одно важное различие между выражениями break и continue состоит в том, что continue выражение не завершает выполнение цикла, а вместо этого только пропускает оставшийся код действия цикла и код продолжается со следующей итерации.

Давайте рассмотрим пару примеров.



В этом примере, в дополнении к иллюстрации использования break выражения, также показана важная операция, которая часто используется в компьютерных приложениях.

Сегмент программы здесь ищет определенный элемент в массиве.

Я уверен, что все вы искали какую-либо информацию в Интернете, используя поисковые машины, например, Google.

Вместо выполнения сложного поиска, программа здесь просто пытается искать определенное значение в данном массиве.

Программа начинается с инициализации целого массива, состоящего из 5 элементов.

Целая переменная value затем устанавливается в значение, для которого будет выполняться поиск.

Размер устанавливается как длина массива intArray, используя переменную экземпляра length.

Логическая переменная found используется, чтобы отслеживать, найдено ли значение.

В начале эта переменная устанавливается как false, потому что значение еще не найдено в самом начале.

Затем объявляется переменная i, которая используется как индекс для цикла.

Возникает вопрос, почему она не объявлена внутри цикла.

Вы видите здесь, что индекс, также, используется для отслеживания позиции значения в массиве, если оно будет найдено.

Если индекс объявить внутри цикла for, его область видимости будет ограничена циклом for, и соответственно, значение индекса не будет доступно вне цикла.

Цикл здесь анализирует каждый элемент в массиве один за другим, проверяя соответствует ли его значение искомому значению.

Если соответствие найдено, логическая переменная found будет установлена как true.

И выражение break завершит выполнение цикла.

Далее будет выполнено следующее выражение после цикла.

В этом случае выражение if-else проверит, равна ли переменная found значению true.

Тогда индекс i или позиция найденного значения в массиве будет выведена пользователю.

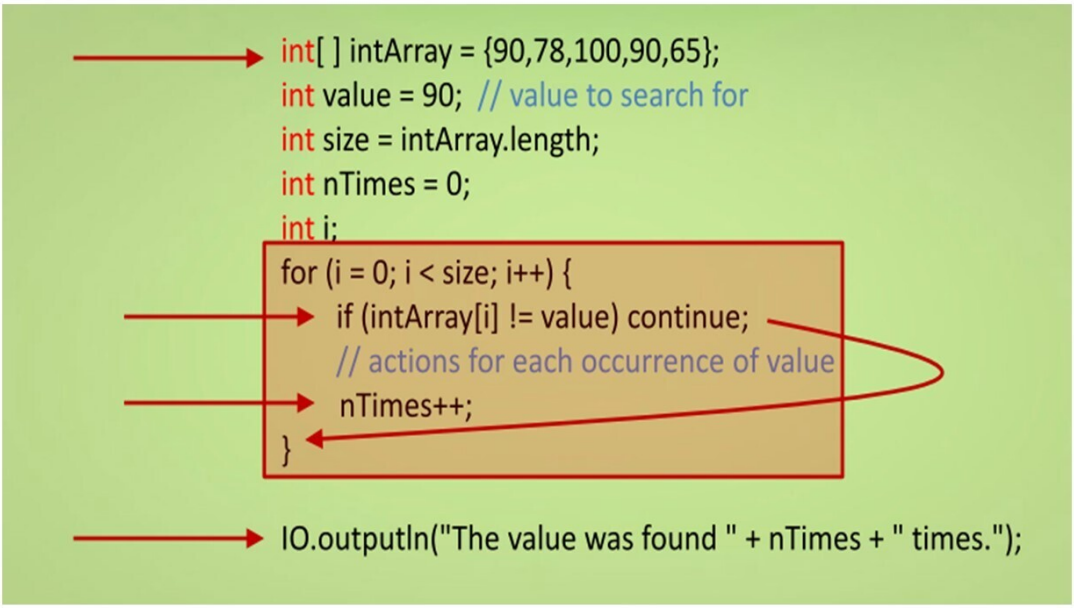
Как я уже говорил ранее, если переменная i будет объявлена внутри цикла for как индекс, она не будет доступна здесь, вне цикла for.

Выражение else выводит сообщение, что значение не найдено.

Заметьте, что метод находит только первое вхождение значения.

Поэтому, даже если значение встречается в массиве несколько раз, будет использоваться только первый индекс.

Вы найдете, что даже если выражение break удалить, программа все равно будет находить значение, однако использование выражения break может экономить вычислительное время для больших массивов.



Теперь этот пример показывает использование выражения continue, и считает число вхождений искомого значения в массиве.

Массив здесь инициализируется так же, как и в предыдущем примере.

Искомое значение устанавливается как 90.

Здесь также есть одна дополнительная переменная nTimes, которая дает число вхождения значения.

Заметьте, что переменная i здесь используется только как индекс цикла, поэтому здесь не важно, где эта переменная будет объявлена, внутри цикла или вне цикла.

Цикл здесь анализирует элементы массива один за другим.

Внутри цикла, если элемент не равен искомому значению, выражение continue выполняется и пропускает окончание цикла, в отличие от выражения break, которое завершает цикл.

Далее начинается новая итерация и продолжается анализ следующего элемента массива.

Если же элемент равен искомому значению, переменная nTimes увеличится на 1.

Можно также увидеть, что такой же результат может быть получен без использования выражения continue, например, используя выражение if-else.

Но в некоторых приложениях выражение continue может существенно упростить код.

Когда цикл завершается, будет выведено число вхождений искомого значения.

Два метода поиска, которые мы только что рассмотрели, анализируют элементы массива один за другим, и такой подход называется линейным поиском.

Однако существуют и более эффективные алгоритмы, по сравнению с линейным поиском, особенно если элементы сортируются.

Я вернусь к этой теме позже.

2D Массивы

Идея одномерного массива может быть легко расширена до двумерного массива, или массива даже более высокой размерности.

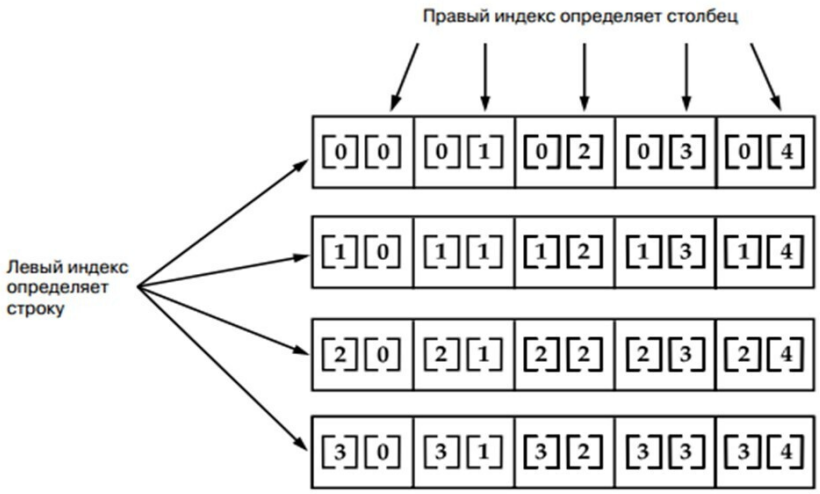
Способ, с помощью которого реализуются многомерные массивы, это рассматривать двумерный массив как массив одномерных массивов, 3D массив в виде массива 2D массивов и так далее.

Сначала сосредоточимся на двумерных массивах.

Итак, в 2D массиве у вас есть массив 1D массивов, то есть, для каждого элемента массива, вместо того чтобы держать данные определенного типа, каждый элемент массива представляет собой одномерный массив.

Я буду использовать схему для дополнительной иллюстрации, что это значит, позже.

В общем, двумерный массив имеет определенное количество строк, давайте назовем это число R, и определенное количество столбцов C, при этом R и C не обязательно должны быть одинаковыми.




```
int twoD[][] = new int[4][5];
```

Для тех, кто изучал матрицу линейной алгебры, 2D массив похож на 2D матрицу, а 1D массив, как вектор.

Но не беспокойтесь об этом, если вы не изучали матрицы раньше, потому что вам не нужно это для понимания массива.

В повседневной жизни, многие приложения требуют обработки многомерных данных.

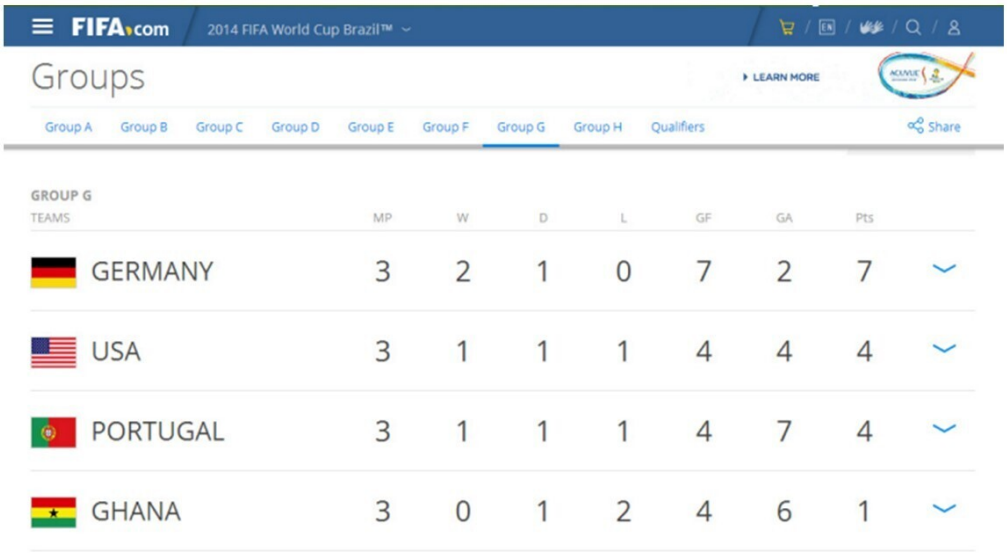
Например, вы можете использовать 2D массив, чтобы представлять разные оценки, в том числе оценки за экзамены, оценки за домашние задания, оценки за лабораторные работы и окончательную оценку для каждого студента на курсе.

В таком представлении, каждая строка массива может представлять различные оценки, полученные студентом, и каждый столбец представляет собой различные виды оценки, скажем, столбец один для оценок за экзамены, столбец 2 для оценок за домашние работы, и так далее.

В частности, оценки всех студенты, полученные за различные работы на курсе, могут быть представлены в виде 2D массива, и для тысяч студентов, это был бы очень большой массив.

Результаты матчей в различных видах спорта также могут быть представлены в виде 2D таблиц или массивов.

Вот результаты матчей кубка мира в разных группах.



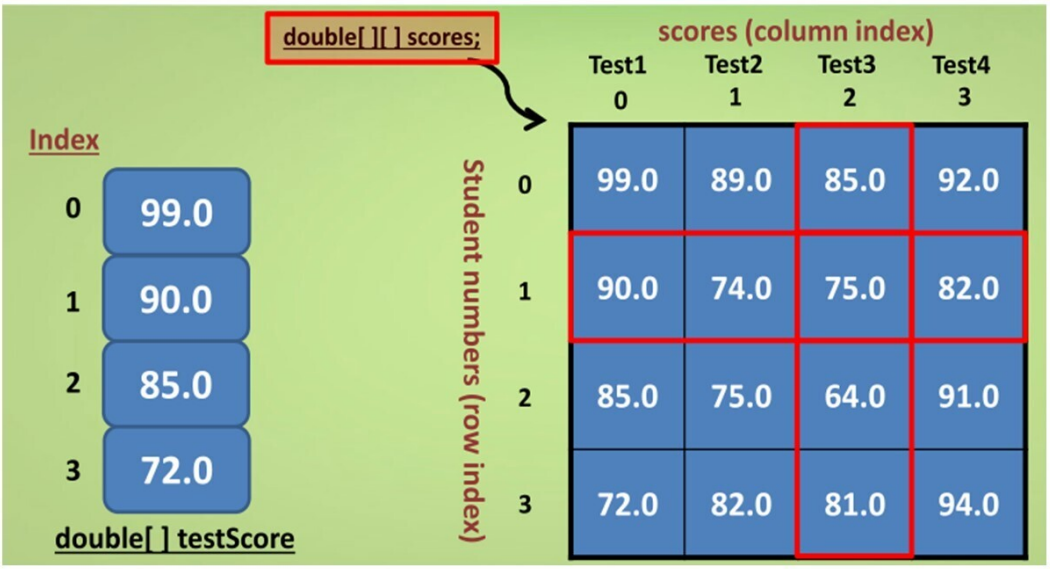
Каждая строка представляет результаты для разных команд и каждый столбец для различных атрибутов.

Коллекция из этих таблиц может быть представлена в виде 3D массива.

Я уверен, что вы можете придумать много других примеров.

Давайте, используем результаты тестов студентов в качестве примера для дальнейшего объяснения идеи 2D массивов.

Когда мы обсуждали 1D массив ранее, мы использовали массив оценок для хранения оценок для группы студентов.



Когда каждый студент берет несколько тестов, предполагая, что есть только четыре студента, и каждый студент взял 4 теста, результаты могут быть представлены массивом 4x4, как этот, где каждая строка представляет результаты тестов для каждого студента, а каждый столбец представляет результаты, полученные различными студентами в том же тесте.

Массив не обязательно должен быть квадратным, так как, как правило, гораздо больше студентов, чем число тестов.

Обратите внимание, что так же, как в одномерном массиве, индексы для студентов и номеров тестов начинаются с 0.

```
double[ ][ ] scores;
```

Это синтаксис объявления 2D массива.

Далее этот вопрос будет обсуждаться более подробно.

```
public class Scores {
   /* 1. A 2D array instance variable */

   /* 2. Initialize a 2D array */

   /* 3. Access a 2D array element */

   /* 4. Traverse a 2D array
              using a nested loop */

}
```

Здесь я буду использовать класс оценок, чтобы проиллюстрировать различные аспекты 2D массивов, в том числе объявление и инициализацию, а также доступ и перемещение отдельных элементов 2D массивов с помощью вложенных циклов.

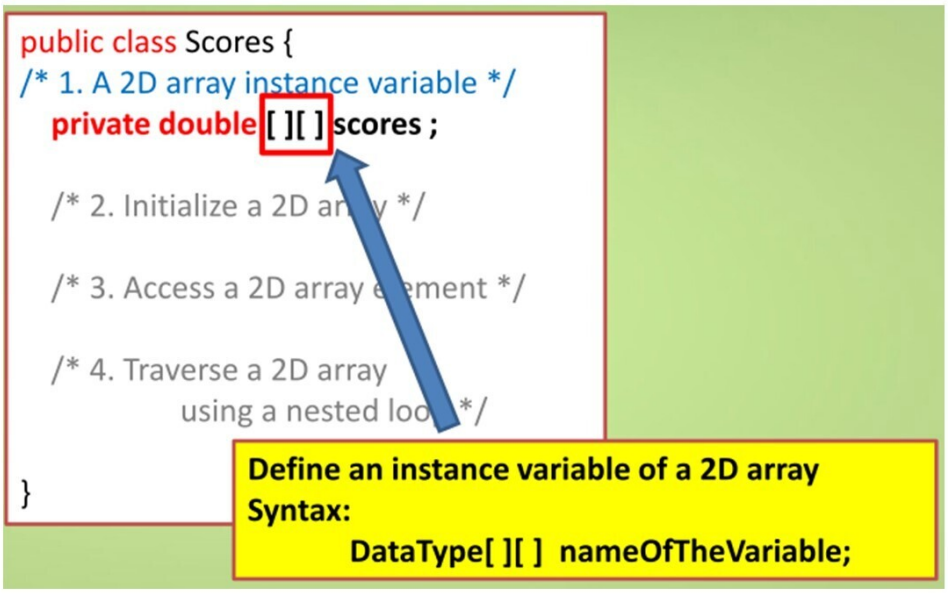
Здесь показан синтаксис объявления 2D массива оценок типа double как переменной экземпляра для класса оценок.

Подобно одномерному массиву, это объявление должно указать тип данных, которые будут храниться в массиве.

Независимо от того, какая используется размерность, массивы являются однородными по типам, то есть, содержат данные одного и того же типа.

Отличие здесь в том, что есть две пары квадратных скобок для указания размера 2D массива, по одной для каждого из 2-х измерений.

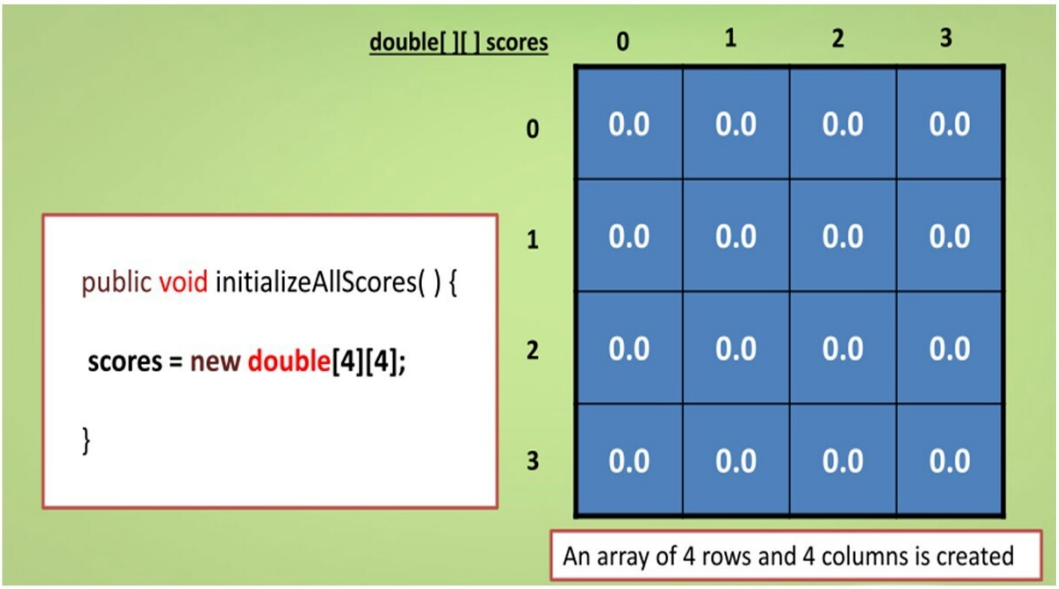
Объявление, подобно этому, просто определяет ссылочную переменную для массива, но при этом память не была еще выделена для элементов массива.



Чтобы инициализировать массив, давайте напишем метод с названием initializeAllScores.

```
public class Scores {
   /* 1. A 2D array instance variable */
   private double [ ][ ] scores ;

   public void initializeAllScores( ) {

        /* 2. Initialize a 2D array */

   }

   /* 3. Access a 2D array element */
   /* 4. Traverse a 2D array using a nested loop */
}
```

Также как мы делали в одномерном массиве, двумерный массив создается с помощью оператора new с последующим указанием типа данных, а затем указания размера каждого из измерений для 2D массива.



Это объявление выделит достаточно памяти для хранения всех элементов создаваемого массива.

Как упоминалось ранее, Java автоматически инициализирует каждый элемент массива на значение по умолчанию для соответствующего типа.

В этом случае, 0.0 используется в качестве значения по умолчанию для типа double.

Я хочу отметить, также, что, как только массив создан, размер массива является фиксированным и не может быть изменен позже в программе.

Здесь я хочу дать вам некоторую концептуальную идею о том, как этот двумерный массив представлен.

Когда делается объявление, создается ссылочная переменная scores.

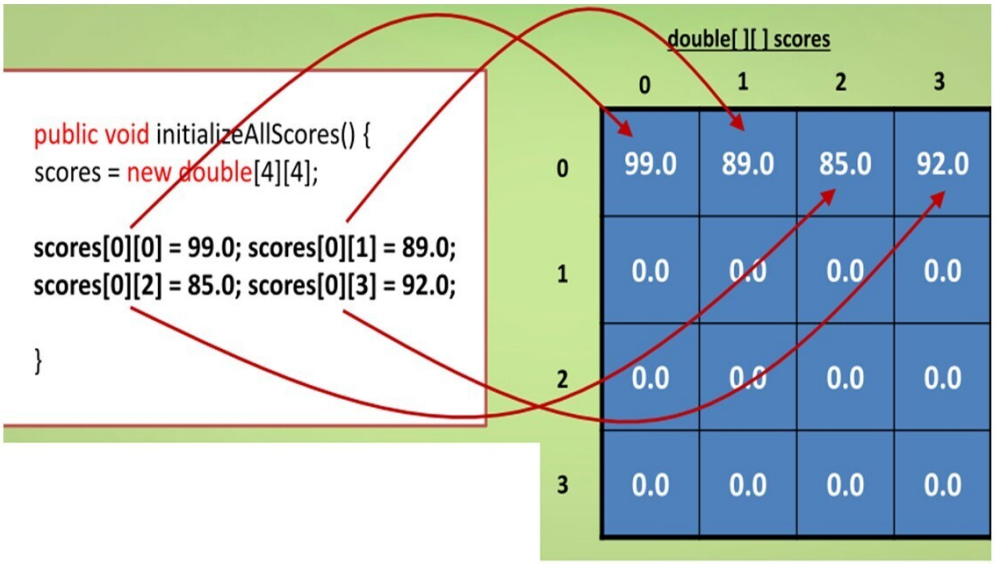
Когда оператор new используется для создания массива, первое измерение, указанное здесь, создаст одномерный массив из 4 элементов.

И каждый элемент этого одномерного массива также ссылочная переменная, указывающая на другой одномерный массив.

В этом примере элементы в одномерных массивах синего цвета имеют тип double.

В общем, каждый элемент массива можно использовать для более сложных объектов до тех пор, пока они одного и того же типа.

После того, как объявлен массив, также, как и для одномерного массива, можно использовать операторы присваивания, чтобы фактически инициализировать массив.



Отличие здесь в том, что индексы для 2D массива указаны в двух парах квадратных скобок, где первый индекс – это индекс строки и второй это индекс столбца.

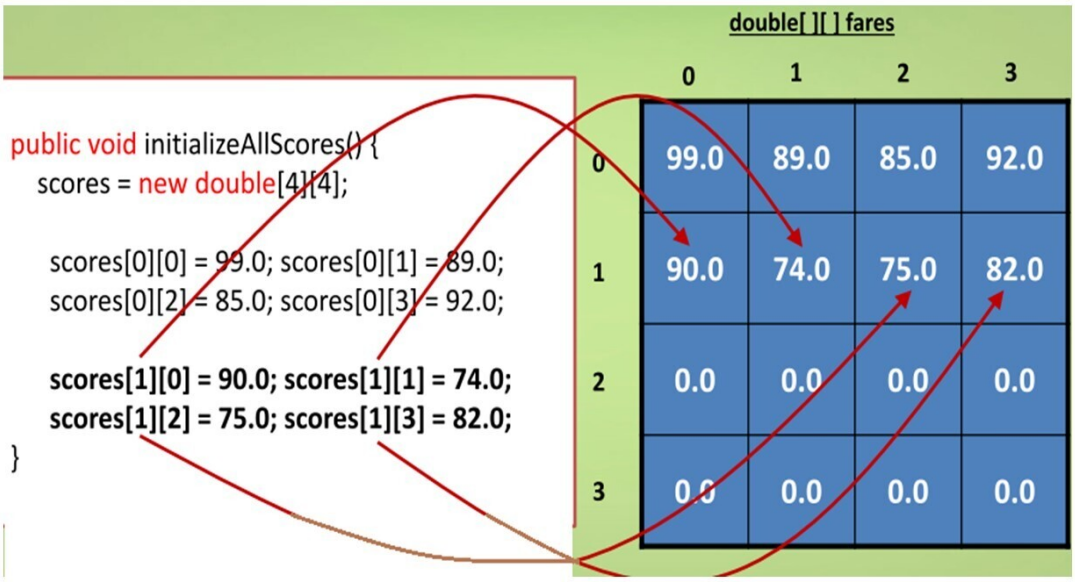
В операторах присваивания первый индекс, 0, массива относится к первой строке массива.

Таким образом, расположение с индексами 0,0 дает значение 99.

Расположение 0,1 дает значение 89.

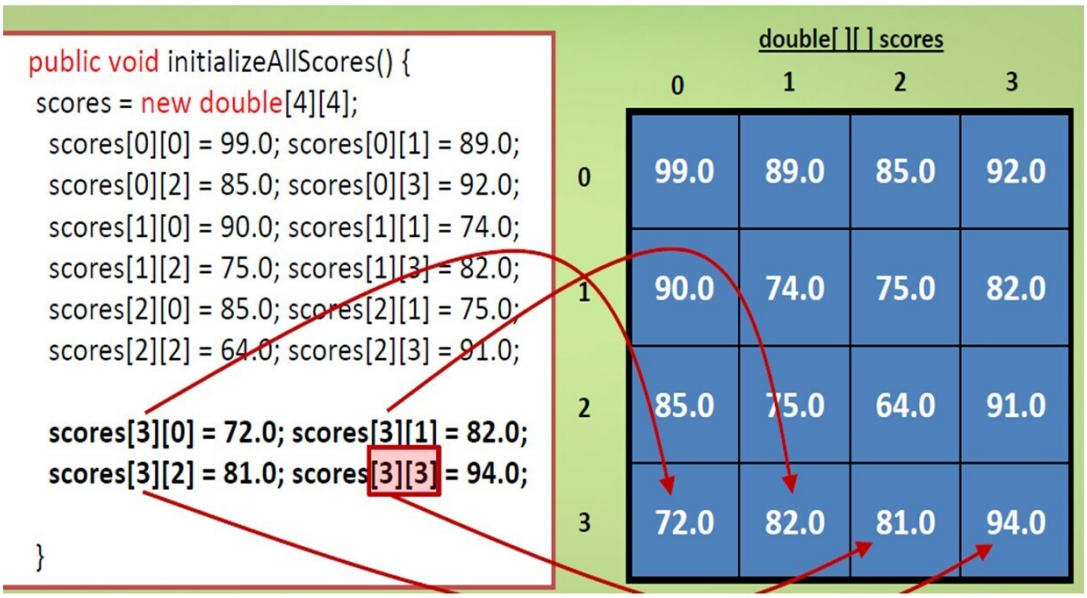
Расположение 0,2 дает значение 85.

Последний элемент этой строки, с индексами 0,3 дает 92.



На второй строке, первый индекс равен 1, а 2-й индекс меняется от 0 до 3.

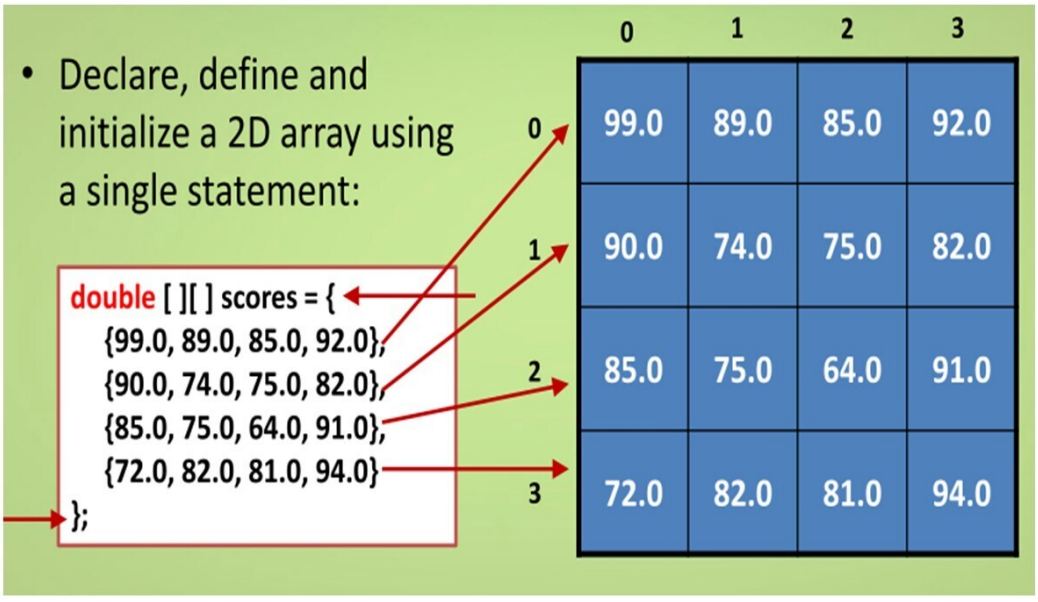
На 3-й строке, первый индекс 2.



Для последней строки, 4-й, обратите внимание, что индекс строки 3, что на единицу меньше, чем размер строки.

И последний элемент в этом 2D массиве индексируются как 3,3.

Подобно одномерному массиву, еще один способ инициализации двумерного массива – это сделать объявление и инициализацию в одном выражении, задав начальные значения в паре фигурных скобок.

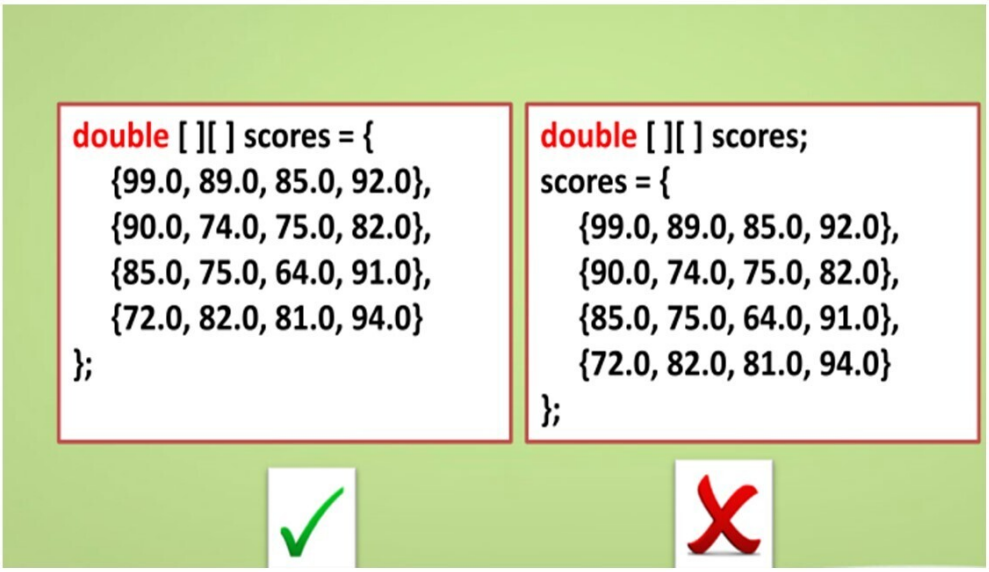


В случае двумерного массива, значения для каждой строки указываются в отдельном списке в фигурных скобках, то есть, в виде одномерного массива.

Эта инициализация также определяет размер массива 4 на 4.

Подобно одномерному массиву, этот формат инициализации должен быть сделан в то же время, когда производится объявление массива.

Компилятор даст вам ошибку, если это делается в отдельном выражении, после того как был объявлен массив.



3-й шаг в определении класса оценок это определение метода с названием getScoreByIndices для доступа к элементам в 2D массиве.

```
public class Scores{

   /* 1. A 2D array instance variable */
   private double[ ][ ] scores;
   public void initializeAllScores() {  /* 2. Initialize a 2D array */  }

   public double getScoreByIndices(int rowIndex, int colIndex) {
         /* 3. Access a 2D array element */
    }

   /* 4. Traverse a 2D array
               using a nested loop */

}
```

Этот метод принимает два параметра, которые определяют индексы строки и столбца элемента, чтобы быть найденным, и возвращает значение типа double потому, что это тип данных, хранящихся в массиве.

```
public double getScoreByIndices(int rowIndex, int colIndex)
{

    int numOfRows = scores.length;
    int numOfCols = scores[0].length;

}
```

Внутри метода, мы должны сначала узнать размер массива.

В случае одномерного массива, можно получить размер массива с помощью переменной экземпляра length.

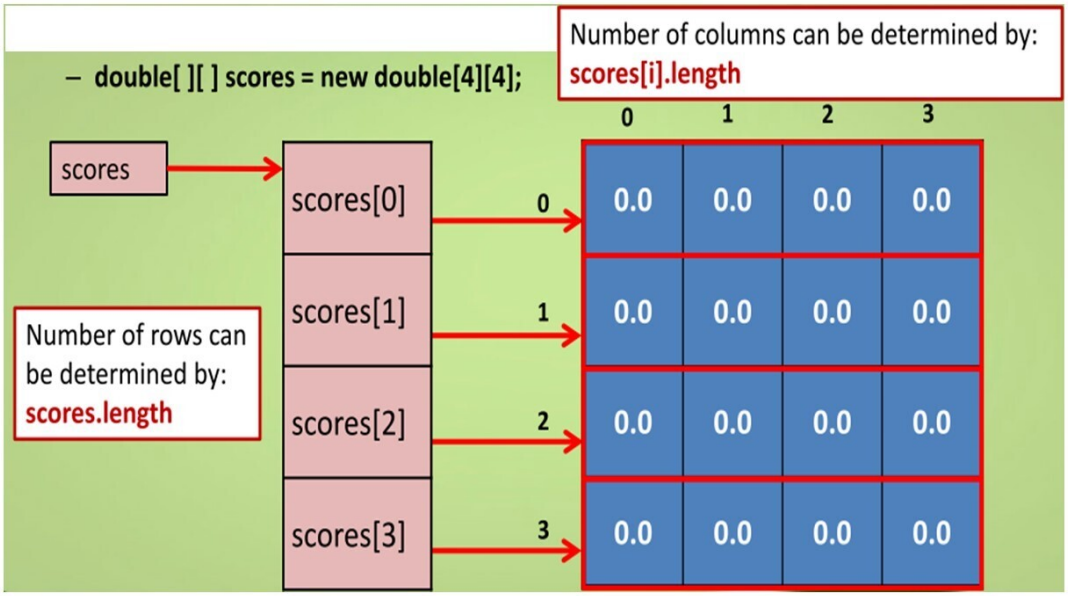
Помните, что length является переменной экземпляра, а не методом, поэтому не содержит пару скобок после length.

В случае двумерного массива, его размер определяется количеством строк, а также числом столбцов.

Так как же нам определить количество строк и количество столбцов?

Два выражения здесь выполнять эту работу.

Давайте использовать схему, чтобы проиллюстрировать, как это работает.



Помните, что scores на самом деле является одномерным массивом с 4-я элементами, а каждый из его элементов является другим массивом.

Таким образом, чтобы определить размер первого индекса, или количество строк, вы можете использовать scores.length.

В этом примере, scores.length даст значение 4, поскольку scores является массивом с 4-я элементами.

Теперь, что насчет количества столбцов?

Как вы можете видеть, каждый элемент scores, "scores [0]", "scores [1]" и так далее это одномерные массивы.

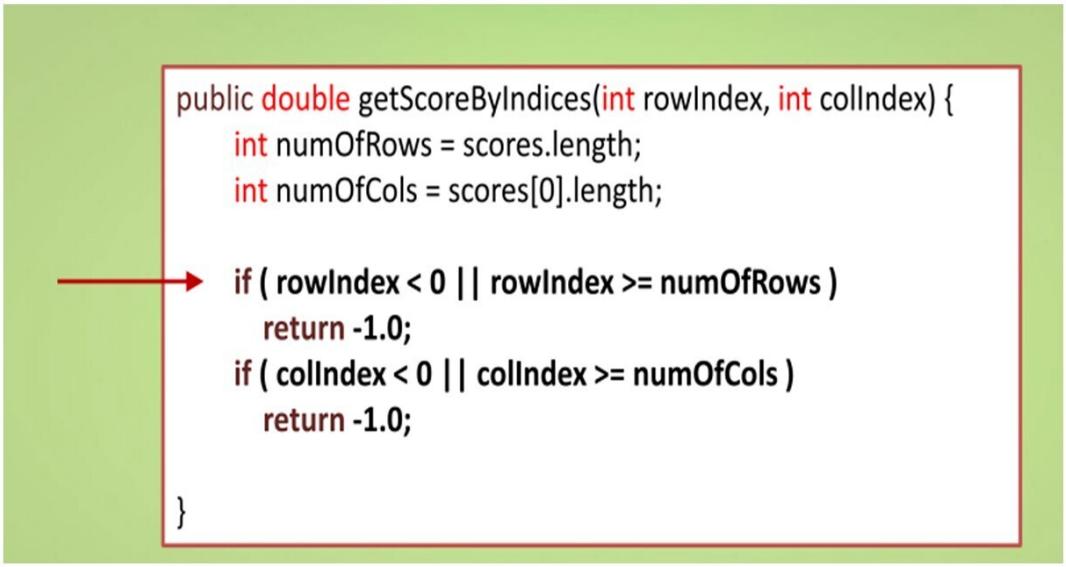
Таким образом, чтобы определить размеры этих массивов, можно использовать scores[0].length для первой строки, scores[1].length для 2-й строки и так далее.

В этом примере, все они дают то же значение 4, то есть количество столбцов.

Так что первое выражение определяет количество строк, а затем присваивает переменной numOfRows, и второе выражение – число столбцов и присваивает переменной numOfCols.

В Java, вы на самом деле можете определить 2D массив с различным числом столбцов для разных строк, но в этом примере, будем считать, что все строки имеют одинаковое число столбцов, так что вы можете использовать scores с индексом 0 или любой другой индекс для определения количества столбцов.

Теперь, когда у нас есть размер 2D массива, мы можем проверить, находятся ли индексы, определяемые в качестве параметров, в пределах диапазона.



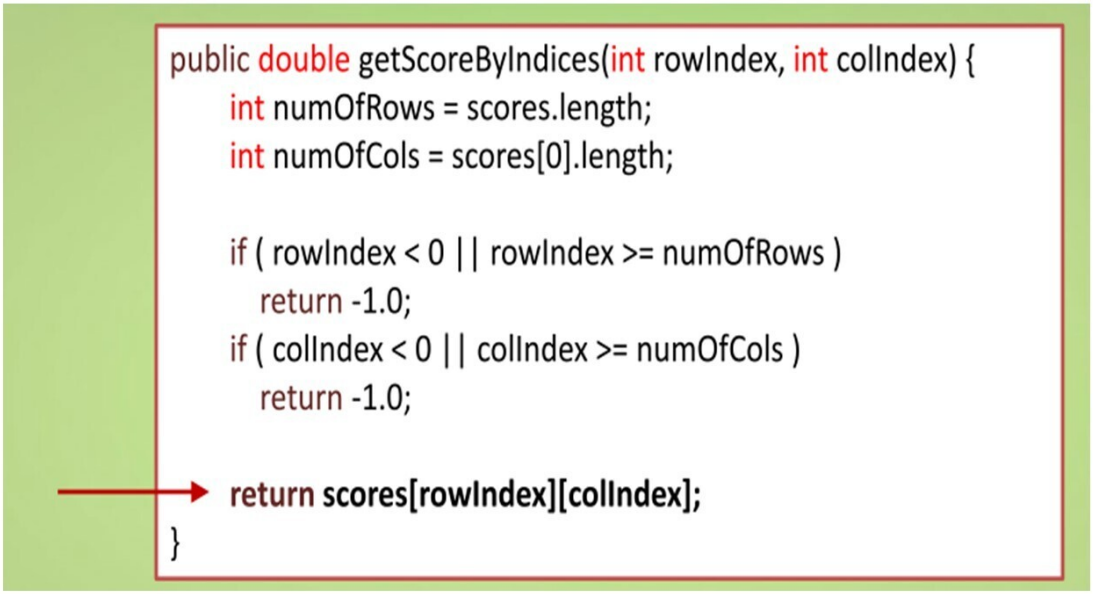
Первое if выражение проверяет, находится ли индекс строки в пределах диапазона для числа строк.

Обратите внимание, что первое условие проверяет, является ли индекс точно меньше 0, так как 0 является допустимым индексом, и второе условие, чтобы проверить, является ли индекс больше или равен numOfRows, так как даже равно numOfRows недопустимо, потому что размер на единицу меньше, чем число строк.

И второе if выражение проверяет индекс столбца.

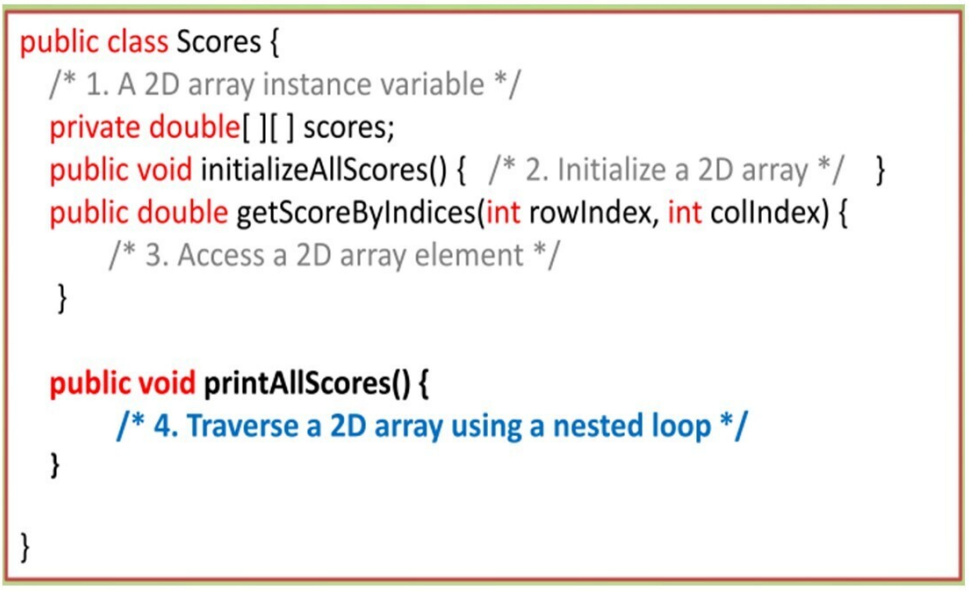
Значение -1 возвращается, если любой из индексов вне диапазона, предполагая, что все значения в массиве неотрицательны.

Если индекс строки и индекс столбца в пределах диапазона, значения scores, индексируемые индексом строки и индексом столбца, будут возвращены в результате методом getScoresByIndices.



```
public double getScoreByIndices(int rowIndex, int colIndex) {
    int numOfRows = scores.length;
    int numOfCols = scores[0].length;

    if ( rowIndex < 0 || rowIndex >= numOfRows )
       return -1.0;
    if ( colIndex < 0 || colIndex >= numOfCols )
       return -1.0;

    return scores[rowIndex][colIndex];
}
```

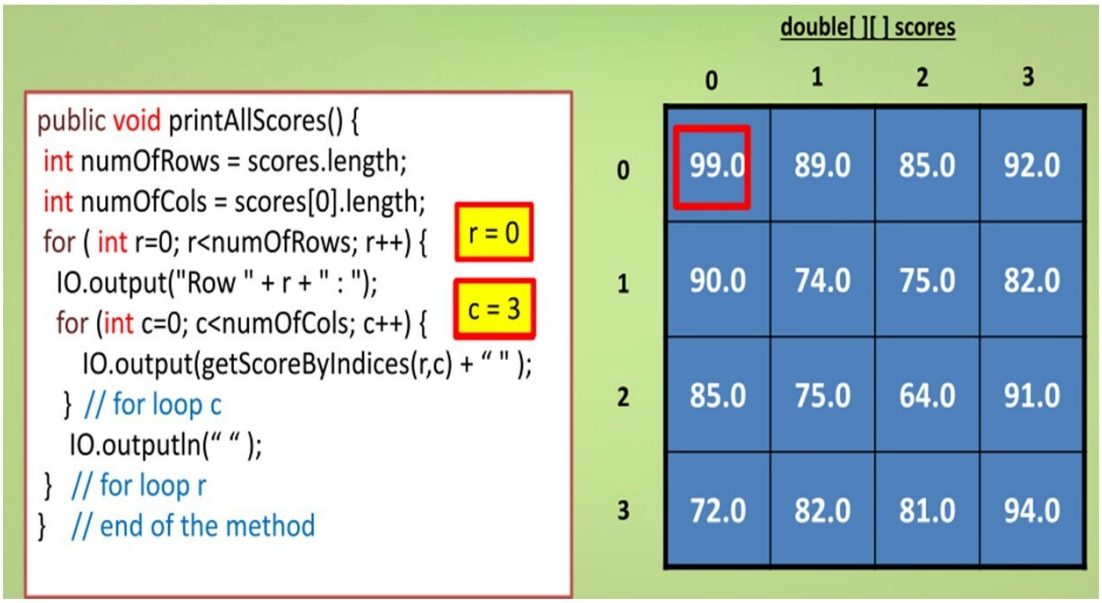
Последняя часть определения scores это печать отдельных элементов в массиве scores путем обхода 2D массива, используя метод printAllScores.



```
public class Scores {
   /* 1. A 2D array instance variable */
   private double[ ][ ] scores;
   public void initializeAllScores() {  /* 2. Initialize a 2D array */  }
   public double getScoreByIndices(int rowIndex, int colIndex) {
        /* 3. Access a 2D array element */
    }

   public void printAllScores() {
        /* 4. Traverse a 2D array using a nested loop */
   }

}
```

Первые шаги в printAllScores это получить количество строк и количество столбцов из массива, аналогично тому, что мы сделали в предыдущем методе.

Это полная реализация метода.



Обратите внимание, что здесь есть два for цикла, или два вложенных цикла, один для управления индексом строки r и другой для индекса столбца c.

Использование вложенных циклов является довольно распространенным в обходе многомерных массивов.

Когда выполнение начинается с внешнего цикла, индекс строки r установлен в 0, и выражение IO.output распечатает номер строки в консоли.

Когда внутренний цикл выполняется, индекс столбца c будет установлен в 0.

Внутри второго for цикла, элемент массива с индексом 0, 0 будет найден с помощью метода getScoreByIndices, который мы только что описали, и значение 99 будет получено и выведено на консоль.

Выполнение внутреннего цикла будет продолжаться за счет увеличения C на 1, элемент с индексом 0,1 затем будет извлечен и выведен.

Этот процесс будет повторяться, пока C не станет равным 3.

И цикл завершится, когда C становится равным 4.

Оператор outputln во внутреннем цикле будет помещать строку вывода на новой линии.

Когда выполнение перейдет к началу внешнего цикла, r будет увеличен на 1, а элементы из второй строки будут получены один за другим с помощью внутреннего цикла.

Этот процесс будет продолжаться, пока последний элемент массива с индексом 3,3 не будет достигнут.

Опять же, я хочу отметить, что последний элемент этого массива индексируются как 3,3, хотя размер массива составляет 4 на 4, потому что индексы начинаются с 0.

Пример работы с двумерным массивом

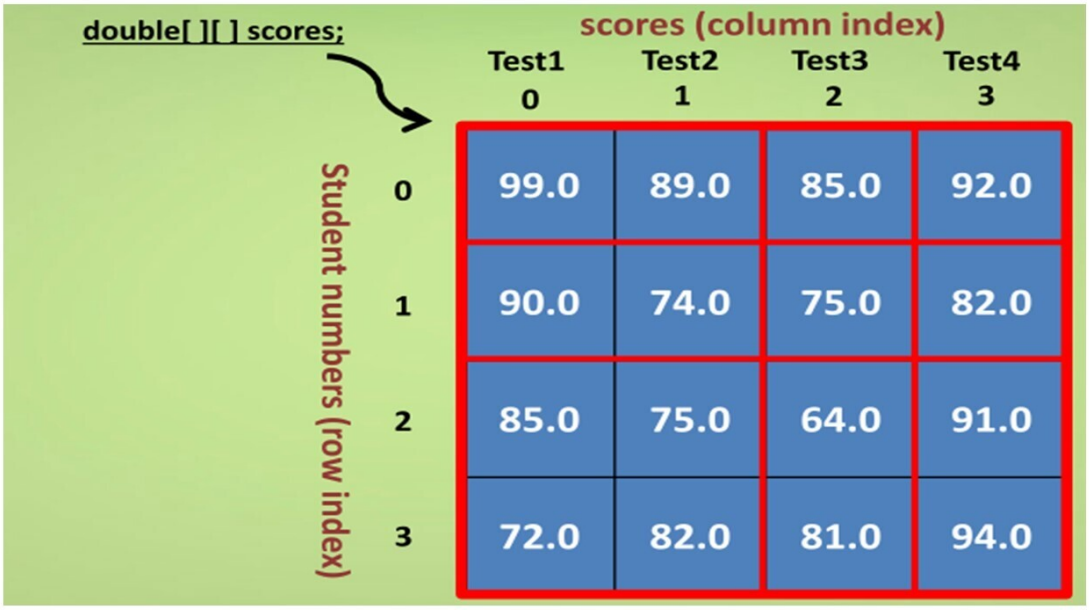
Давайте посмотрим снова некоторые из методов, которые мы написали для одномерных массивов.

```
/*
 * aveScore computes the average of the values in an array
 */
public double aveScore( ) {
    double sum = 0;      // for storing the cumulative sum
    int size = scoreArray.length;  // size of the array

    for (int i = 0; i < size; i++)
       sum = sum + scoreArray[i];

    return sum / size;
 }
```

Это метод для вычисления среднего значения элементов в одномерном массиве.

Когда мы имеем дело с двумерными массивами, подумайте о том, что будет означать среднее для двумерного массива.

Подумайте о том, какой тип усреднения вы хотели бы получить для score массива, такого как этот.



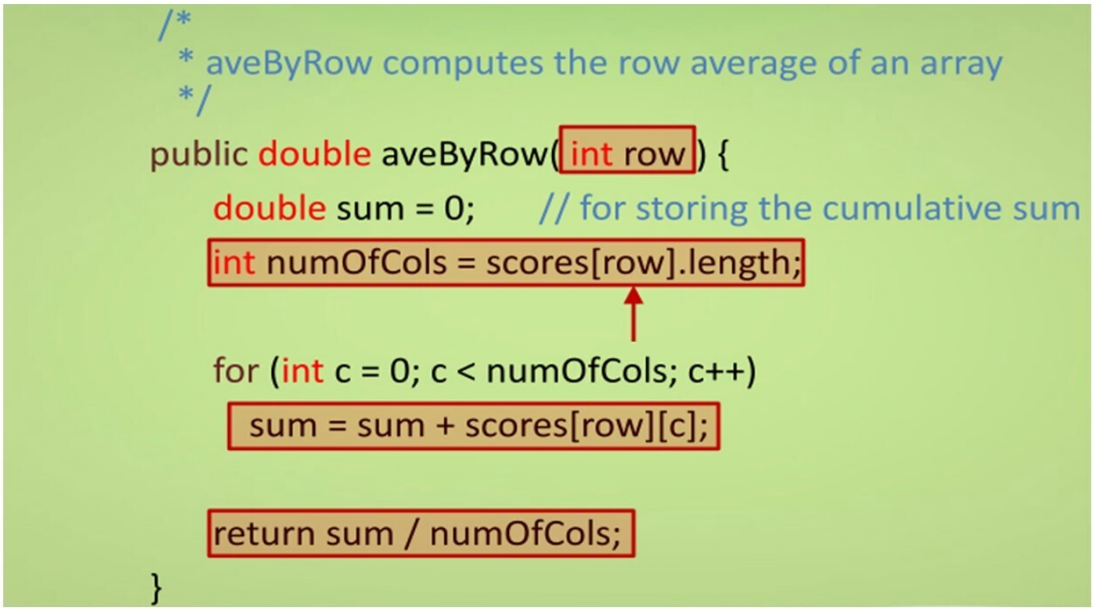
Если предположить, что все тесты имеют одинаковый вес, вы можете вычислить среднее значение четырех тестов для каждого из студентов.

Вы можете вычислить взвешенное среднее, если они имеют разные веса.

Таким образом, вы хотите вычислить среднее строки.

Если вы хотите вычислить средний балл для конкретного теста, значит, вы хотите вычислить среднее столбца.

Также вы можете вычислить среднее значение для всех оценок, полученных всеми студентами.



Этот метод aveByRow вычисляет среднее строки 2D массива.

В то время как в предыдущем методе aveScore не требовались параметры, один дополнительный параметр требуется при расчете среднего строки.

Вы должны указать номер строки, для которой среднее будет рассчитано и номер строки указан в качестве параметра.

Затем вы должны определить количество столбцов для этой конкретной строки с помощью scores[row].length.

Вы можете также использовать scores[0].length, если все строки имеют одинаковое количество столбцов.

Внутри цикла, оценки извлекаются с помощью двух индексов вместо только одного, как в одномерном случае.

Первый индекс – это строка, которая задается в качестве параметра, и 2-й индекс «c» это индекс цикла, который проходит данную строку.

После выхода из цикла, среднее строки вычисляется путем деления итоговой суммы, используя количество столбцов, а затем возвращается в качестве результата для метода aveByRow.

Среднее столбца может быть вычислено таким же образом, за исключением того, что параметр, передаваемый методу aveByCol является индексом столбца.

```
/*
 * aveByCol computes the column average an array
 */
public double aveByCol( int col ) {
    double sum = 0;      // for storing the cumulative sum
    int numOfRows = scores.length;

    for (int r = 0; r < numOfRows; r++)
     sum = sum + scores[r][col];

    return sum / numOfRowls;
}
```

Число элементов, подлежащих включению в расчет среднего, это количество строк или scores.length.

Обратите внимание, чем это отличается от предыдущего метода, мы просто используем scores здесь без индекса.

Внутри цикла первый индекс – это индекс цикла, который можно варьировать с помощью цикла, в то время как второй индекс фиксируется как индекс столбца, заданный в качестве параметра.

Среднее столбца затем вычисляется и возвращается.

Вы можете подумать о том, что нужно будет сделать, если вы хотите вычислить среднее значение для всего массива.

Это метод maxIndex, который мы обсуждали для нахождения местоположения максимального значения в одномерном массиве.

```
/*
 * maxIndex finds the location of the largest values in an array
 * up to index size - 1
 */
public int maxIndex(int size){
    int mIndex = 0;   // index for the current maximum
    if (size > scoreArray.length) size = scoreArray.length;

    for (int i = 0; i < size; i++) {
       if (scoreArray[i] > scoreArray[mIndex]) mIndex = i;
     }
    return mIndex;
  }
```

Подобно нахождению среднего, при попытке найти максимум или минимум для двумерного массива, Вы должны определить, выполняется ли это вычисление для строки, столбца или всего массива.

Этот метод maxRowIndex предназначен для нахождения местоположения максимального элемента для заданного столбца, который приведен в качестве параметра.

```
/*
 * maxRowIndex finds the location of the largest values for a
 * given column in a 2D array
 */
public int maxRowIndex(int col, int size){
    int mIndex = 0;   // index for the current maximum
    if (size > scores.length) size = scores.length;

    for (int i = 0; i < size; i++) {
        if (scores[i][col] > scores[mIndex][col]) mIndex = i;
    }
    return mIndex;
}
```

В примере оценки теста, вы можете подумать об этом, как найти студента, который получил наибольшее количество баллов в данном тесте, и индекс строки, возвращаемый методом, это номер студенческого билета с наибольшим количеством баллов.

Так как мы будем вести цикл по строкам, максимальный размер будет scores.length.

Внутри цикла, единственная разница между этим и предыдущим методом, это то, что здесь есть два индекса для scores.

Для прохождения строки, второй индекс или индекс столбца является фиксированным, и первый индекс будет изменяться как индекс цикла.

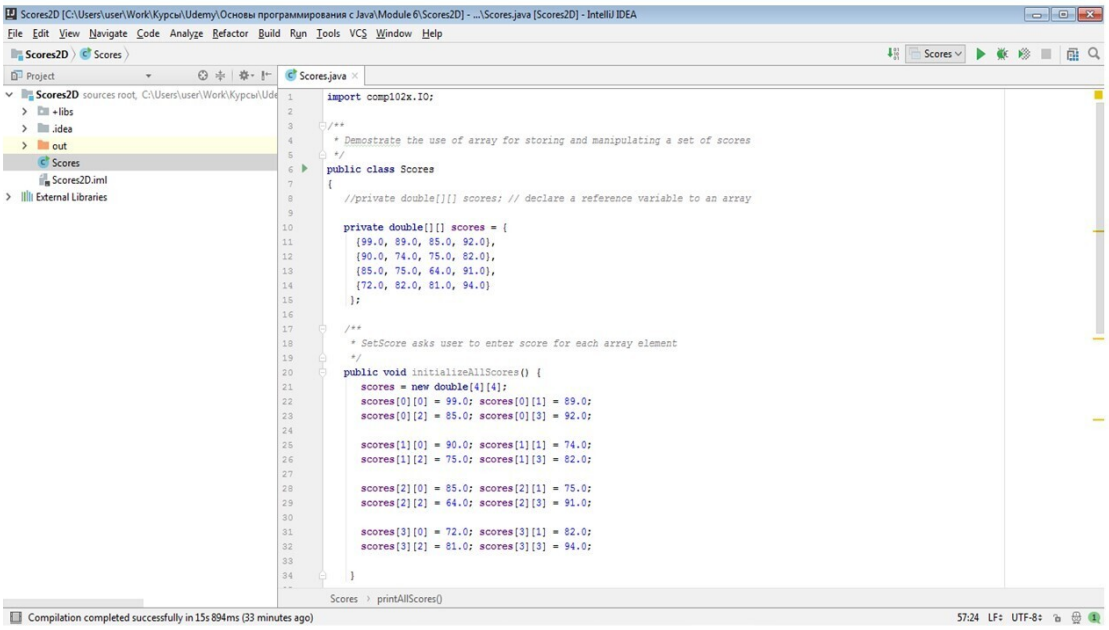
mIndex будет возвращен как результат в конце.

Вы можете подумать о том, как найти максимальное значение для данной строки и для всего массива.

Демонстрация работы с двумерными массивами

Давайте рассмотрим краткую демонстрацию класса Scores, который мы только что обсуждали.

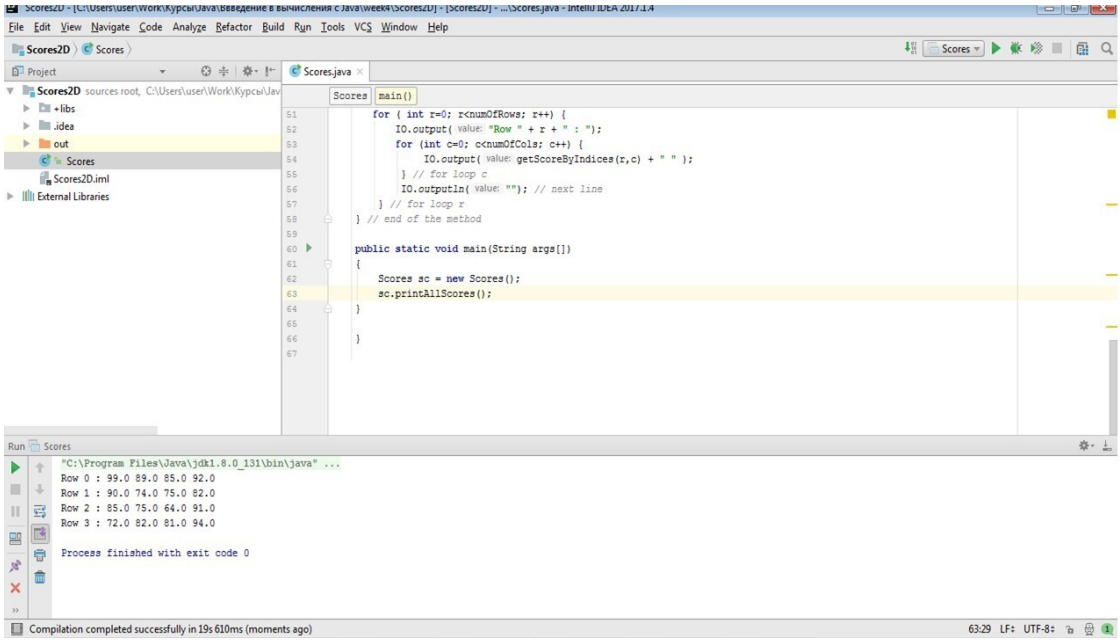
Откроем проект в IDEA.



Вот метод initializeAllScores, метод getScoresByIndices и метод printAllScores.

Так это должно быть таким же, как то, что мы только что обсуждали.

Вместо того чтобы использовать метод initializeAllScores для инициализации массива, я буду делать инициализацию в то же время, когда массив объявлен.



Это было бы то же самое, как вызов метода "initializeAllScores".

Скомпилируйте и запустите программу путем создания экземпляра, вызывая метод printAllScores, при этом отобразится вывод.

```
private double[][] scores = {
   {99.0, 89.0, 85.0, 92.0},
   {90.0, 74.0, 75.0},
   {85.0, 75.0},
   {72.0}
  };
```

Значения, которые здесь печатаются, должны быть такими же, как начальные значения, определенные в массиве.

Как я уже говорил, в Java, вы на самом деле можете определить двумерный массив с различным числом столбцов для разных строк.

Давайте изменим программу здесь, чтобы проиллюстрировать это.

```
Row 0 : 99.0 89.0 85.0 92.0
Row 1 : 90.0 74.0 75.0 82.0
Row 2 : 85.0 75.0 64.0 91.0
Row 3 : 72.0 82.0 81.0 94.0
Row 0 : 99.0 89.0 85.0 92.0
Row 1 : 90.0 74.0 75.0
java.lang.ArrayIndexOutOfBoundsE
        at Scores.getScoreByInd
        at Scores.printAllScore
```

Таким образом, вместо инициализации двумерного массива с одинаковым числом столбцов для каждой строки, давайте оставим первую строку ту же самую с 4 элементами, но уменьшим число элементов во второй строке на единицу.

Это означает, что здесь есть только 3 элемента, а затем в третьей строке на 2 элемента.

Последняя строка, или 4-я строка имеет только 1 элемент.

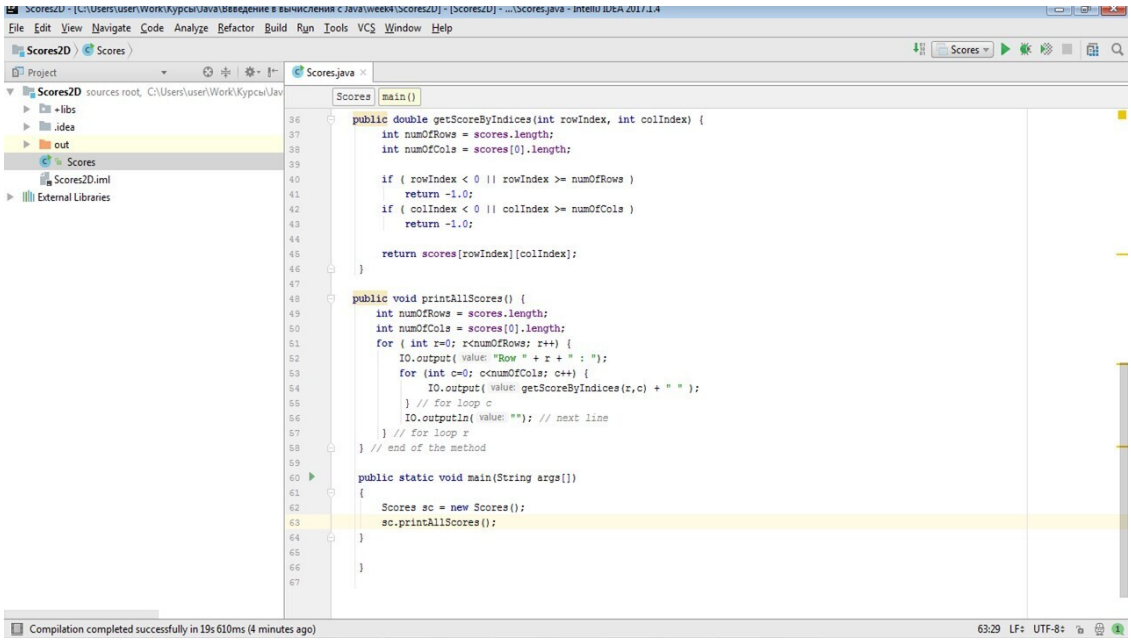
Так что этот массив, если он работает, будет иметь разные столбцы для разных строк.

Давайте посмотрим, позволит ли Java нам сделать это.

Давайте скомпилируем программу; и она компилируется без проблем.

Так что теперь давайте попробуем запустить программу. При этом создается экземпляр объекта.

Если вы запустите программу с помощью метода printAllScores, будет ошибка.



Ошибка ArrayIndexOutOfBounds, и она появляется именно в том выражении, когда мы вызываем getScoreByIndices.

Мы получаем эту ошибку, потому что мы предполагаем, что все строки имеют одинаковое число столбцов, получаемое как scores[0].length.

Но на самом деле, массив имеет разное количество столбцов для разных строк.

Так как же нам это исправить?

```
public void printAllScores() {
    int numOfRows = scores.length;
    int numOfCols = scores[0].length;
    for ( int r=0; r<numOfRows; r++) {
        IO.output("Row " + r + " : ");
        for (int c=0; c<scores[r].length; c++) {
            IO.output(getScoreByIndices(r,c) + " " );
        } // for loop c
        IO.outputln(""); // next line
    } // for loop r
```

Одним из способов исправить это, изменить numOfCols здесь, во внутреннем цикле для printAllScores, вместо использования numOfCols для всех строк, мы можем изменить это на scores[r].length.

```
Row 0 : 99.0 89.0 85.0 92.0
Row 1 : 90.0 74.0 75.0
Row 2 : 85.0 75.0
Row 3 : 72.0
```

```
/**
 * Demostrate the use of array fo
 */
public class Scores
{
   //private double[][] scores; /

   private double[][] scores = {
      {99.0, 89.0, 85.0, 92.0},
      {90.0, 74.0, 75.0},
      {85.0, 75.0},
      {72.0}
```

С этим изменением, внутренний цикл будет ограничивать количество доступа к элементам в разных строках по количеству столбцов в этой строке, которое дается как scores[r].length.

И это даст нам число столбцов, с учетом индексации r, который является индексом строки.

Давайте снова скомпилируем и запустим программу путем создания экземпляра.

Если вы вызовете метод printAllScores снова, вы сможете увидеть, что результат печатается и имеет различное число элементов для разных строк, что то же самое, как массив был инициализирован.

Вопросы

Задача

Какой будет вывод следующего сегмента кода?

```
int[] array = {2, 1, 0, 2, 3, 4};
IO.output(array[2]);
IO.output(array[3]);
IO.output(array[4]);
IO.output(array[1]);
```

Варианты ответа:

1. 1022
2. 2341
3. 0231
4. Error

Ответ: 3.

Объяснение

Индексы массивов начинаются с 0. Поэтому array[2], array[3], array[4] и array[1] представляют третий, четвертый, пятый и второй элемент массива соответственно. Это дает результат 0231.

Задача

```
public static int q2(int[] array) {
int index = 0;
for (int i = 0; i < array.length; i++) {
if (array[i] < array[index]) index = i;
}
return index;
}
```

Учитывая метод "q2" выше, каким будет вывод следующего сегмента кода?

```
int[] array = {22, 99, 11, 99, 33};
IO.output(q2(array));
```

Варианты ответа:

1. 2
2. 11
3. 99
4. 3

Ответ: 1.

Объяснение

Код просматривает массив и находит индекс целого с наименьшим значением в массиве.

Задача

```
public static void change(char[] array, int index1, int index2) {
char temp = array[index2];
array[index1] = array[index2];
array[index2] = temp;
}
```

Учитывая метод " change " выше, каким было бы содержимое массива после выполнения следующего сегмента кода?

```
char[] array = {'H', 'K', 'U', 'S', 'T'};
change(array, 1, 3);
```

Варианты ответа:

1. {'H', 'S', 'U', 'K', 'T'}
2. {'U', 'K', 'U', 'S', 'T'}
3. {'U', 'K', 'H', 'S', 'T'}
4. {'H', 'S', 'U', 'S', 'T'}

Ответ: 4.

Объяснение

В первой строке метода, char переменная temp хранит четвертый символ, 'S' в массиве.

Во второй строке метода, второй символ в массиве присваивает четвертый символ в массиве. Содержание массив становится {'H', 'S', 'U', 'S', 'T'}.

В третьей строке метода, четвертый символ в массиве присваивает значение переменной temp, которое является «S». Тем не менее, четвертый символ уже «S» до присвоения. Поэтому содержание массива остается {'H', 'S', 'U', 'S', 'T'}.

Задача

Каким будет вывод следующего сегмента кода?

```
int i = 0;
int j = 0;
for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
if (j == 1) break;
}
}
IO.outputln(i + " " + j);
```

Варианты ответа:

1. 0 1
2. 0 2
3. 3 1
4. 3 2

Ответ: 3.

Объяснение

В Java, выражение break вызовет прерывание выполнения внутреннего for цикла, когда переменная j равна 1.

Внешний for цикл будет продолжать исполняться.

Таким образом, итоговые значения i и j будут 3 и 1 соответственно.

Задача

99.0	89.0	85.0	92.0
90.0	74.0	75.0	82.0
85.0	75.0	64.0	91.0
72.0	82.0	81.0	94.0

Учитывая scores 2D массив выше, какое из следующих выражений может установить выделенный элемент как 0,0?

Варианты ответа:

1. scores[2][3] = 0.0;
2. scores[1][2] = 0.0;
3. scores[3][2] = 0.0;
4. scores[2][1] = 0.0;

Ответ 2.

Объяснение

Выделенный элемент находится во второй строке третьего столбца 2D массива. Индекс строки и столбца для доступа к элементу 1 и 2 соответственно.

Таким образом, scores[1][2] = 0.0; установит выделенный элемент 0.0.

Задача

Дан 4 x 4 целый 2D массив arr. Каким будет значение, хранящееся в arr[1][3] после выполнения следующего кода для инициализации?

```
int size = 4;
for (int i = 0; i < size; i++) {
for (int j = 0; j < size; j++) {
arr[i][j] = size * j + i;
}
}
```

Варианты ответа:

1. 7
2. 2
3. 8
4. 13

Ответ: 4.

Объяснение

После инициализации, содержание arr становится:

```
0    4    8    12
1    5    9    13
2    6    10   14
3    7    11   15
```

arr[1][3] относится к элементу второй строки и четвертого столбца.

Таким образом, значение 13.

Упражнение

Теперь, когда вы узнали об основах массивов, можете ли вы определить вывод для каждого из сегментов кода ниже?

Code segment 1

```
int[] a;
a[1] = 1;
a[2] = 2;
a[3] = 3;
IO.outputln(a[0]);
```

Code segment 2

```
int[] a = {1, 2, 3, 4};
int[] b = a;
b[0] = 3;
b[1] = 2;
for (int i = 0; i < 4; i++)
IO.output(a[i]);
```

Code segment 3

```
int() a = new int(20);
if (a(10) == 0)
IO.output("zero");
```

Code segment 4

```
int[] a = {4, 3, 2, 1};
for (int i = 0; i < 4; i++)
IO.output(i);
```

Code segment 5

```
int[] a = {1.5, 2.5, 3.5};
double avg = 0;
for (int i = 0; i < a.length; i++)
avg += a[i];
avg /= 3.0;
IO.output(avg);
```

Ответ:

Q1. (Error, array not initialized)

```
int[] a;
a[1] = 1;
a[2] = 2;
a[3] = 3;
IO.outputln(a[0]);
```

Q2. (3234)

```
int[] a = {1, 2, 3, 4};
int[] b = a;
b[0] = 3;
b[1] = 2;
for (int i = 0; i < 4; i++)
    IO.output(a[i]);
```

Q3. (Syntax error, should use [])

```
int() a = new int(20);
if (a(10) == 0)
    IO.output("zero");
```

Демонстрация сортировки изображений

Загрузите проект Sorting Images.

Завершите метод maxIndex() в соответствие с инструкциями, данными в классе ImageSorter.

Запустите программу, создав новый экземпляр ImageSorter. Холст отобразится, показывая 8 различных изображений в случайных позициях.

Вы можете затем щелкнуть правой кнопкой мыши на созданном экземпляре, запустить метод moveInPosition(). Изображения будут отсортированы от самого короткого до самого высокого на холсте.

Вы также можете проверить свой результат сортировки, запустив метод isAscendinglySorted(). Значение trueshould будет возвращено, если изображения сортируются от короткого до самого высокого.

Символьные строки

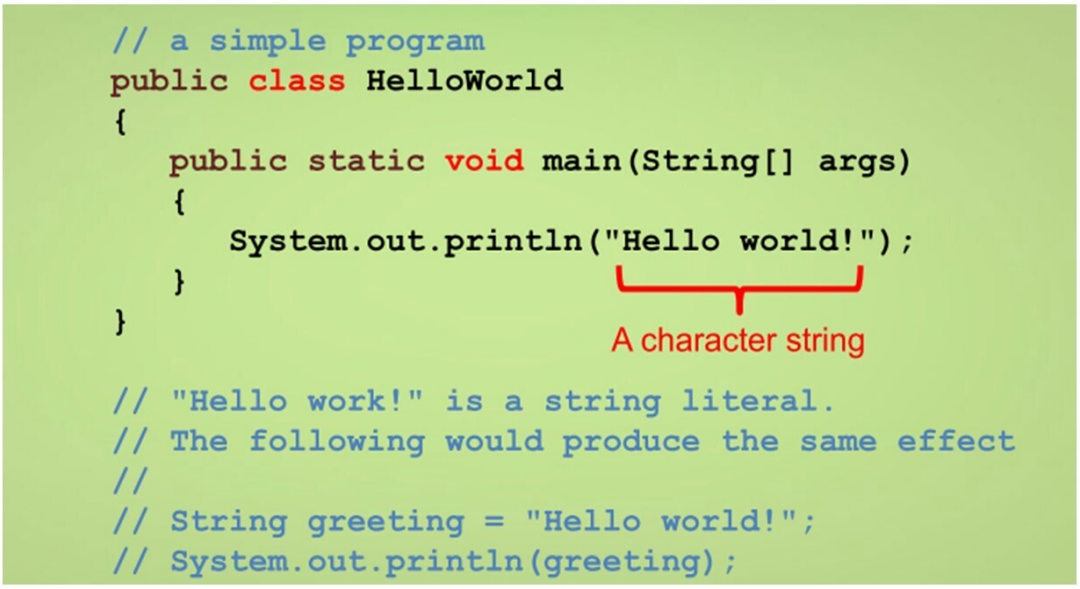
Тема, которую мы будем далее обсуждать, представляет последовательности символов и файловый ввод/вывод.

На самом деле, строки и файлы должны быть хорошо знакомы всем вам, так как мы использовали их, довольно широко в течение последнего времени.

Мы начали использовать строки с самого начала.

Помните, что наша первая программа Java распечатала сообщение "Hello World!".

И "Hello World!" представляет собой строку символов.



И мы использовали изображения, хранящиеся в файлах, и программы Java, которые мы написали, все были сохранены в файлах.

Наша первая программа Java вывела приветственное сообщение, "Привет, мир", и с тех пор мы прошли долгий путь.

Всякий раз, когда компилятор Java встречает строковый литерал, заключенный в двойные кавычки, он создает объект String, с соответствующей последовательностью символов.

Вы можете также объявить строку с помощью класса String.

String greeting = "Hello world!"

Так, String greeting = "Hello world!" создаст объект типа String с "Hello world!" в качестве своего значения.

Использование System.out.println с переменной – строкой символов "greeting" в качестве параметра будет распечатывать сообщение "Hello world!" в консоли.

Если вы захотите, вы можете также использовать IO.outputln, что будет выдавать тот же результат.

Строка представляет собой последовательность символов. И Java предоставляет класс String для создания и манипулирования строками.

Этот класс String предоставляет методы для работы с отдельными символами в строке.

Например, когда вы диктуете по буквам слово, вы должны смотреть на отдельные символы.

Также необходима такая важная операция при сравнении строк, как организация слов или имен в алфавитном порядке.

Другой важной операцией является поиск определенных строк. Это в основном то, что вы делаете, когда вы ищете информацию в Интернете.

Другие полезные операции со строками включают в себя извлечение подстрок и преобразование строк в прописные и строчные буквы.

Я далее объясню, как эти операции могут быть выполнены со строками, используя больше примеров.

Важно отметить, что строки являются неизменяемыми.

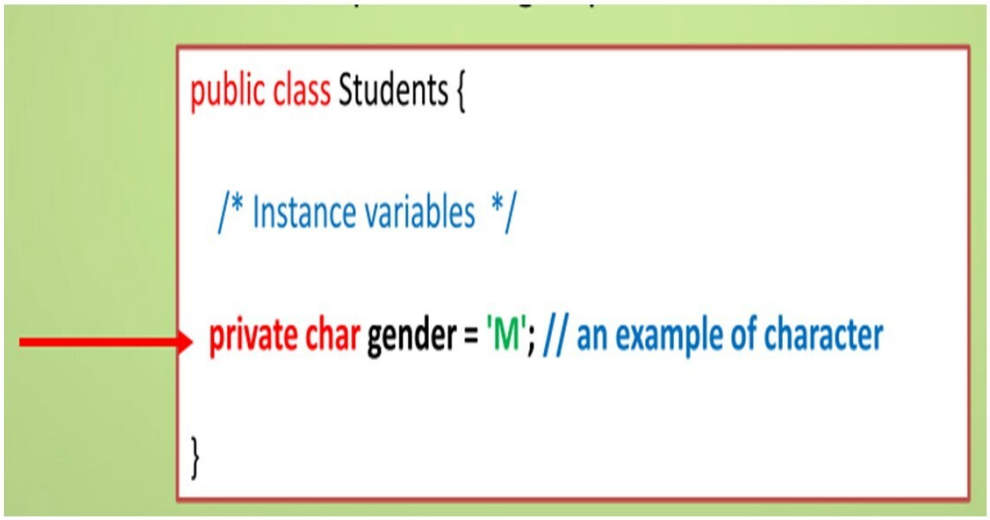
То есть, как только строковый объект создан, он не может быть изменен. Я поговорю об этом позже.

Мы ввели символьный тип char, когда мы говорили о примитивных типах данных.

Так char является одним из восьми примитивных типов данных. И тип char используется для хранения только одного любого символа.

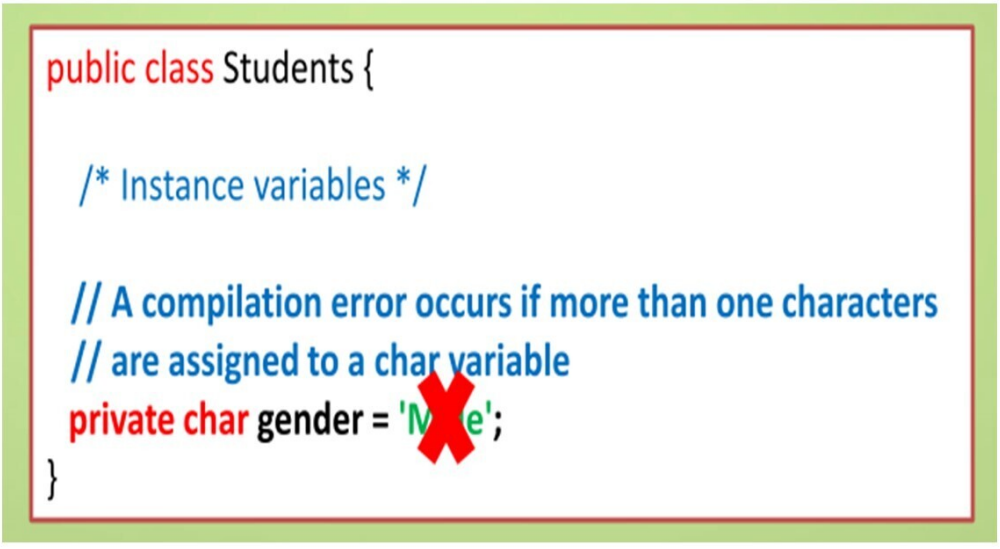
Java использует 16-бит, чтобы представлять символы и кодировать их с помощью 16-битного Unicode формата, который используется для представления многих других языков, помимо английского языка.

И помните, что для символа используется пара одинарных кавычек.



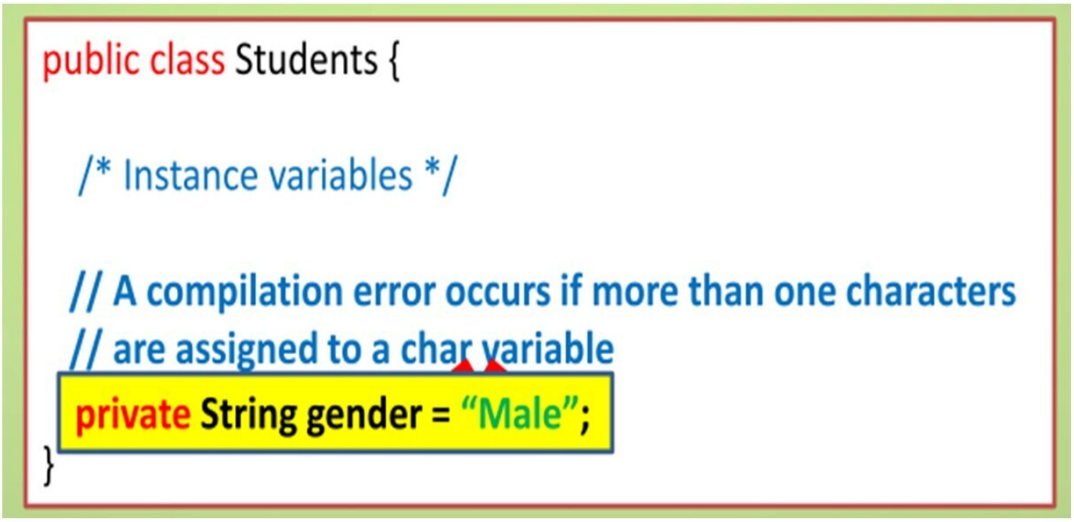
Например, здесь мы объявляем класс студентов. Если мы хотим включить в качестве переменной экземпляра пол студента, мы можем использовать "M" для представления мужчин и "F" для студенток.

Если вместо того, чтобы использовать только один символ, мы хотим написать все слово, male или female, char не может быть использован в качестве типа данных.



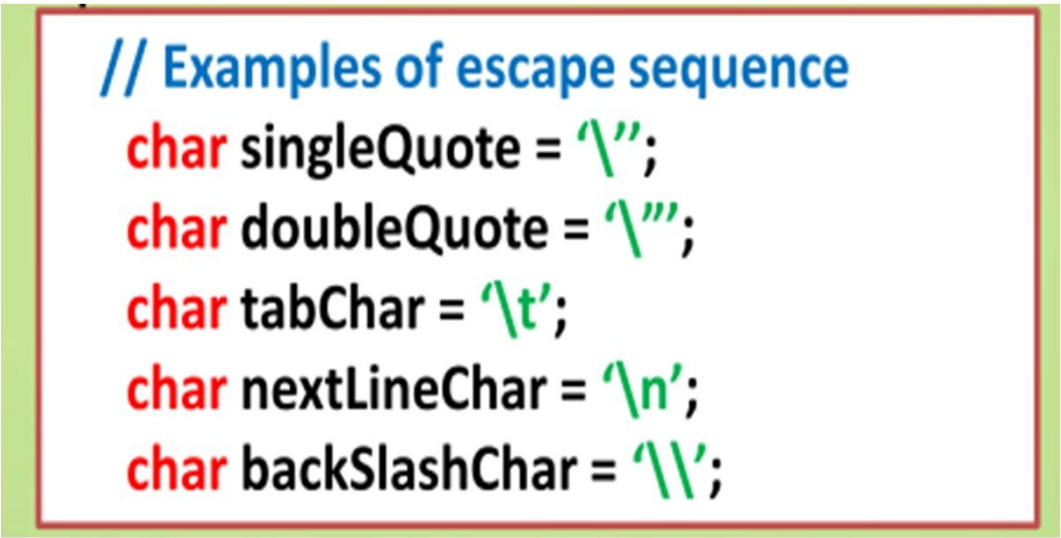
Например, если мы укажем male в паре одинарных кавычек, это приведет к ошибке компиляции.

Мы должны использовать класс String, для того чтобы использовать слово male или female для представления пола в этом примере.



Далее, можно спросить, как можно создать символ, использующий одинарные кавычки или строку с двойными кавычками как один символ.

Сделать это можно с помощью использования дополнительной обратной косой черты перед одинарной или двойной кавычкой, или любых других специальных символов.



Например, если человек хочет распечатать сообщение,
"She received an "A"" с A, вписанной в двойные кавычки.

```
IO.outputln(“She received an \"A\" in COMP1022P.");
```

Обратный слэш перед двойными кавычками выполнит эту распечатку.

Очень часто, мы хотим сохранить более одного символа вместе, как единое целое.

Например, некоторые имена, идентификаторы, которые содержат больше, чем просто цифры, имена файлов, ссылки на веб-страницы, и многое, многое другое.

Один из возможных подходов заключается в создании строки как массива символов типа char.

На самом деле, один из конструкторов класса String позволяет создать строку, указав массив символов.

```
char[ ] studentName;
char[ ] nameArray = {'M', 'a', 'r', 't', 'i', 'n' };
String nameString = new String(nameArray);
```

В этом примере, nameArray это массив символьного типа, который инициализируется как последовательность символов, формирующая имя, Мартин.

Используя nameArray в качестве аргумента конструктора String, создается строка с именем, nameString.

И обратите внимание, что String начинается с заглавной буквы "S".

String является абстрактным типом данных. Абстрактный тип данных является типом данных, внутреннее представление которого скрыто от пользователя.

Мы знаем, что строка должна представлять последовательность символов, и нам нужны методы, которые могут быть использованы для работы со строками, но нам действительно не нужно знать, как строки представлены внутри в компьютере.

И некоторые из операций, которые мы будем обсуждать, включают в себя:

– Методы определения длины строки, определения символа в указанном месте.

– Выделение подстроки заданной строки.

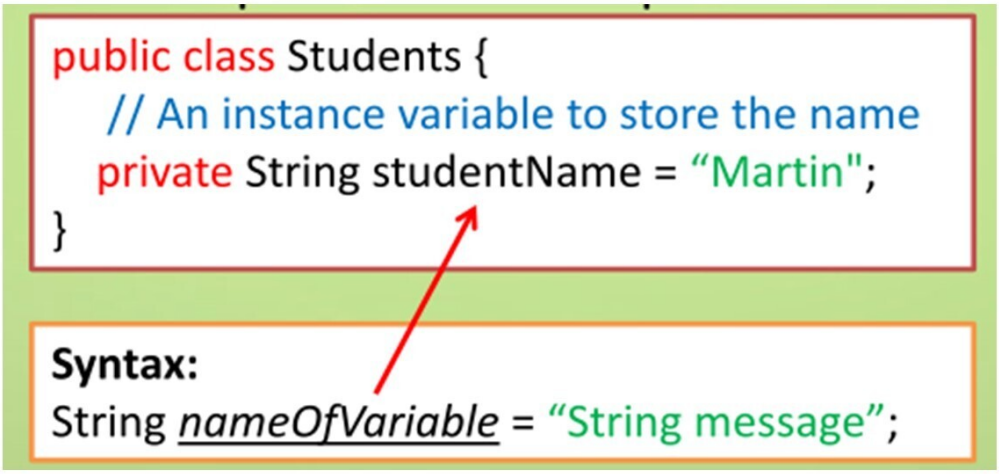
– Сравнение двух строк на равенство или в определенном порядке.

– Объединение строк для формирования длинных строк и многое другое.

Как уже обсуждалось, строка хранит последовательность символов.

Для строк с длиной N, символы, хранящиеся в строке, могут быть проиндексированы от 0 до N-1.

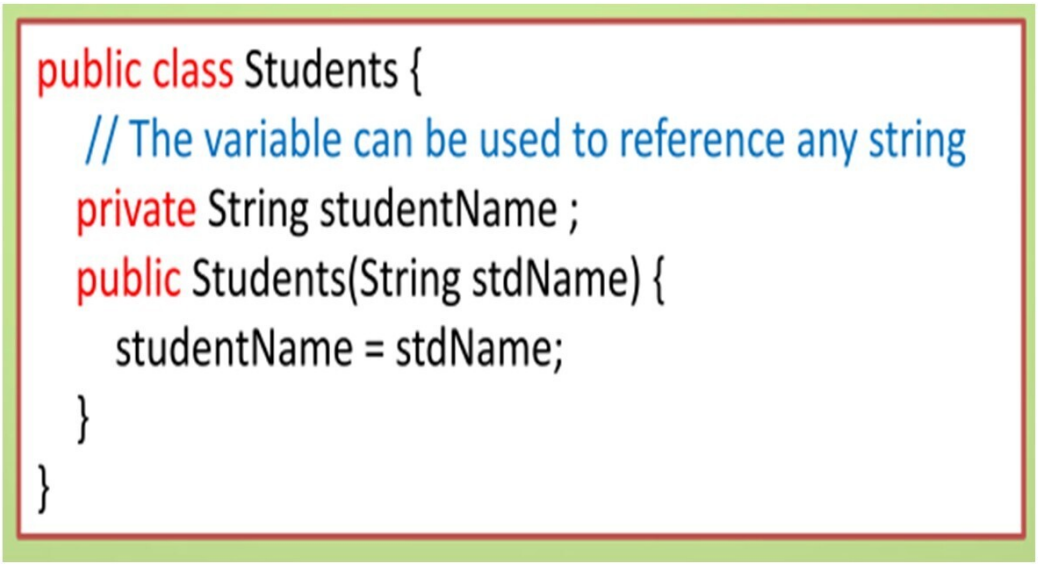
Как мы уже видели много раз прежде, самый простой способ указать строку является использование пары двойных кавычек.



В этом примере, мы определяем класс студентов, и одна из переменных экземпляра это имя студента типа String, и объявление здесь присваивает символьную строку Мартин переменной studentName.

Здесь показан синтаксис объявления и инициализации переменной типа String.

String имеет ссылочный тип и экземпляры, созданные из класса String, являются объектами.



```
public class Students {
   // The variable can be used to reference any string
   private String studentName ;
   public Students(String stdName) {
      studentName = stdName;
   }
}
```

В этом примере, ссылочная переменная studentName типа String объявлена, но еще не инициализирована.

Переменные можно инициализировать в конструкторе, когда параметр конструктора присваивается переменной studentName.

Работа со строками

Метод length() используется для подсчета количества символов в строке.

Обратите внимание, что length() строки является методом, так как имеется пара скобок, в отличие от length для массива, которая является переменной экземпляра.

```
String studentName = “Martin”;
int size = studentName.length();
```

В этом примере целой переменной size будет присвоено значение 6, потому что символьная строка "Мартин" имеет 6 символов.

В отличие от массива, нам не нужно использовать пару квадратных скобок для доступа к символам в строке.

```
String studentName = “Martin”;
char firstChar = studentName.charAt(0);
```

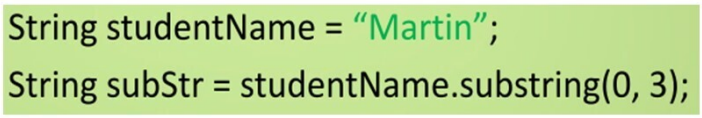
Метод charAt можно использовать для доступа к символу по индексу местоположения, который указан в параметре.

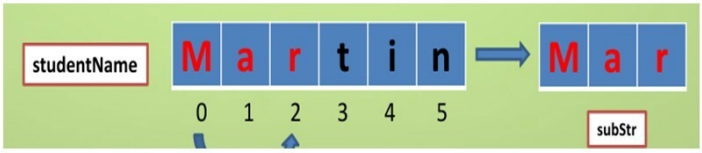
Важно помнить, что также, как и в массиве, здесь индекс начинается с 0.

Так studentName.charAt(0) с параметром 0 вернет один символ «M».

Это первый символ в строке, который затем присваивается переменной firstChar.

Вместо того чтобы просто получить один единственный символ, можно также получить подстроку из заданной строки с помощью метода substring, который принимает два параметра.





Первый параметр "i" указывает на начальный индекс для извлечения подстроки, а второй параметр "j" указывает, что j-1 является индексом последнего символа строки, который должен быть извлечен.

Это немного запутанно, но Вы просто должны помнить, что подстрока, которая должна быть извлечена, начинается с индекса i и до индекса j-1, а не j.

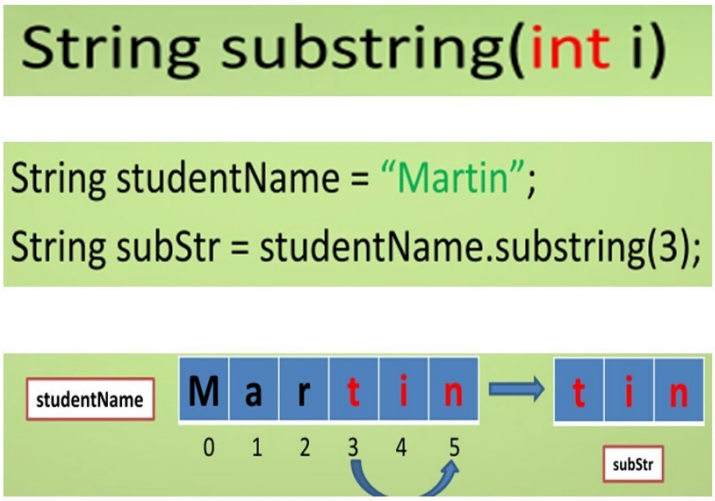
Следует помнить, что последний индекс не j, а j-1.

В этом примере, вызывая метод substring с аргументами 0 и 3, извлекается подстрока от индекса 0 до 3-1 или 2.

Результаты – новая строка с тремя символами.

Затем результат присваивается новой String переменной subStr.

Альтернативный способ использовать метод substring, это указать только один параметр целого типа.



Эта версия substring будет извлекать подстроку, включая символы от индекса i и до конца исходной строки.

В этом примере, вызывая метод substring в studentName с одним аргументом 3, будет извлечена подстрока, которая начинается с индекса 3 и до конца строки.

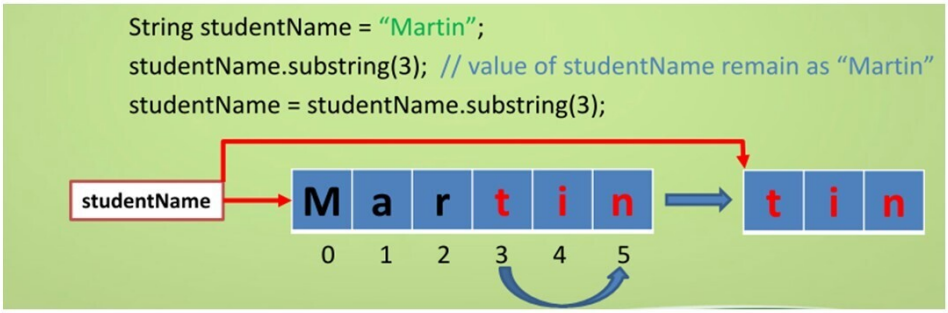
Результатом является новая строка с тремя символами. Результат будет затем присвоен переменной subStr.

Как я уже упоминал, строки являются неизменными.

Это означает, что они не могут быть изменены, как только они будут созданы.

Так substring не изменит значение исходной строки, из которой метод substring извлек подстроку.

Вместо этого, он создаст новую строку.



Чтобы проиллюстрировать это, используя предыдущий пример, переменной studentName присваивается строка "Мартин".

Как и раньше, когда метод substring с одним параметром, целым числом 3, применяется к studentName, вместо изменения памяти для символьной строки "Мартин" на "tin", что не допускается, так объект String является неизменяемым, создается новая строка "tin".

Так что, все, что делает этот код – это новая строка "tin" просто исчезнет после того, как создастся.

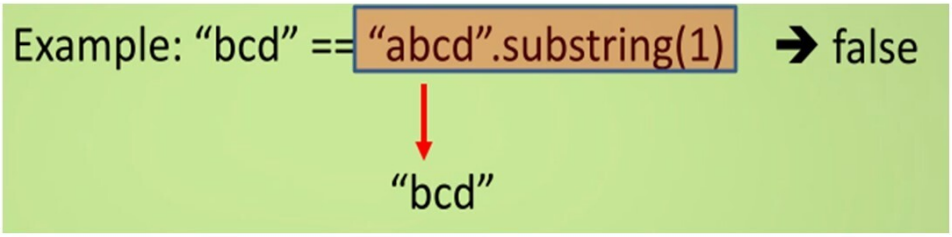
Предположим, что если эта новая строка снова присвоится переменной studentName, это будет допустимо, учитывая то, что строка является неизменной?

Вы можете подумать об этом.

Ответ: да, так как переменная studentName является ссылочной переменной, и она не является финальной.

Так что оператор присваивания назначит переменной studentName указывать на новый объект String со значением "tin", вместо изменения значения исходной строки «Мартин».

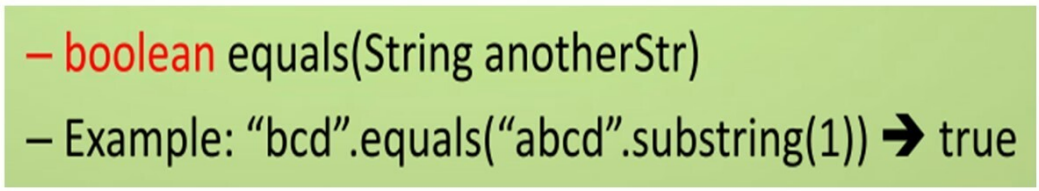
Поскольку объекты String представлены ссылочными переменными, сравнение двух строк на равенство с помощью оператора == может вернуть false, даже если две строки имеют одинаковую последовательность символов, потому что две последовательности символов, определяемых двумя строковыми переменными, могут занимать различные области памяти.



Например, применение substring(1) к символьной строке "abcd", начиная с индекса 1, вернет символьную строку "bcd", как показано здесь.

Но когда сравнение производится с использованием логического равно с другой символьной строкой "bcd", сравнение вернет значение false.

Поэтому следует использовать метод equals() для сравнения двух строк.



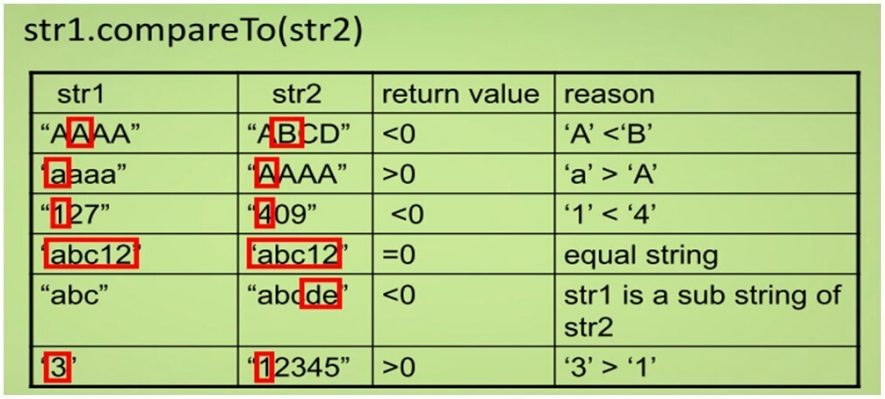
Этот пример аналогичен предыдущему примеру, за исключением того, что используется метод equals() и значение true будет возвращено в качестве результата.

Операция, которая очень часто используется для строки, это упорядочить ее в алфавитном порядке.

В информатике, упорядочение делается с помощью лексикографического порядка, который похож на алфавитный порядок, но это не совсем то же самое, потому что набор символов в строках может включать в себя другие символы, отличные от алфавита, и сравнение фактически выполняется на основе ASCII или Unicode.

Метод compareTo сравнивает две строки лексикографически и возвращает целое число, больше 0, если str1 больше, чем str2 в соответствии с кодировкой ASCII, он возвращает 0, если они одинаковы, и возвращает отрицательное значение, если str1 меньше, чем str2.

Я буду использовать примеры в таблице, чтобы показать, как метод compareTo работает.



str1.compareTo(str2)

str1	str2	return value	reason
“AAAA”	“ABCD”	<0	‘A’ <‘B’
“aaaa”	“AAAA”	>0	‘a’ > ‘A’
“127”	“409”	<0	‘1’ < ‘4’
“abc12”	“abc12”	=0	equal string
“abc”	“abcde”	<0	str1 is a sub string of str2
“3”	“12345”	>0	‘3’ > ‘1’

Что этот метод будет сделать, так это сравнивать символы в двух строках по одному.

Он будет возвращать соответствующий результат, если символы в соответствующих местах не равны друг другу.

В первом примере, первые символы являются одинаковыми, A, так что вы двигаетесь к следующему символу, и метод находит, что A меньше B, таким образом, возвращается отрицательное значение.

Во втором примере, первые символы не одинаковые, хотя они оба «a», один в нижнем регистре, а другой в верхнем регистре, и в представлении ASCII, заглавные буквы стоят перед строчными буквами, и так как строчный символ больше верхнего регистра, возвращается целое положительное число.

При сравнении строк 127 и 409, они сравниваются как строки, а не как числа.

Поскольку представление ASCII для 1 предшествует 4, возвращаемое значение меньше 0.

В четвертом примере, поскольку все символы одинаковы, возвращается значение 0.

Когда первая строка является подстрокой второй строки, как в 5-м примере, возвращается значение меньше 0.

В последнем примере, 3 сравнивается с 12345, и если бы они являлись числами, 12345 явно больше, чем 3, но для сравнения строк, начиная с первого символа, 3 больше, чем 1, и строка 3 рассматривается как больше, чем строка 12345 и возвращается положительное целое число.

Объединение строк – это операция, которая часто используется для String.

И объединение строк означает присоединение одной строки в конец другой строки.

Мы использовали конкатенацию строк много раз, когда мы выводили результаты на консоль с помощью добавления или оператора плюс.

Например, если name инициализируется как строка символов со значением, равным TC.

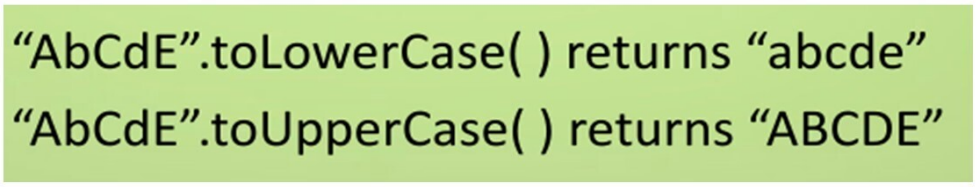
Когда "My name is " добавляется к name, вывод будет «My name is TC».

Java класс String, также обеспечивает метод с именем concat для объединения строк.

Метод concat будет присоединять строку, указанную в качестве аргумента, к концу объекта String, который используется при вызове метода.

Так что, если метод concat применить к объекту String " My name is " с name, будет возвращена комбинированная строка "My name is TC".

Еще двумя широко используемыми методами являются toLowerCase и toUppercase, которые преобразуют все символы в строках в нижний регистр или верхний регистр соответственно.



В этом примере, строка символов "AbCdE" имеет три заглавные буквы A, C и E и две строчные буквы b и d. И применением метода toLowerCase к этой строке, будет возвращена строка со всеми строчными abcde.

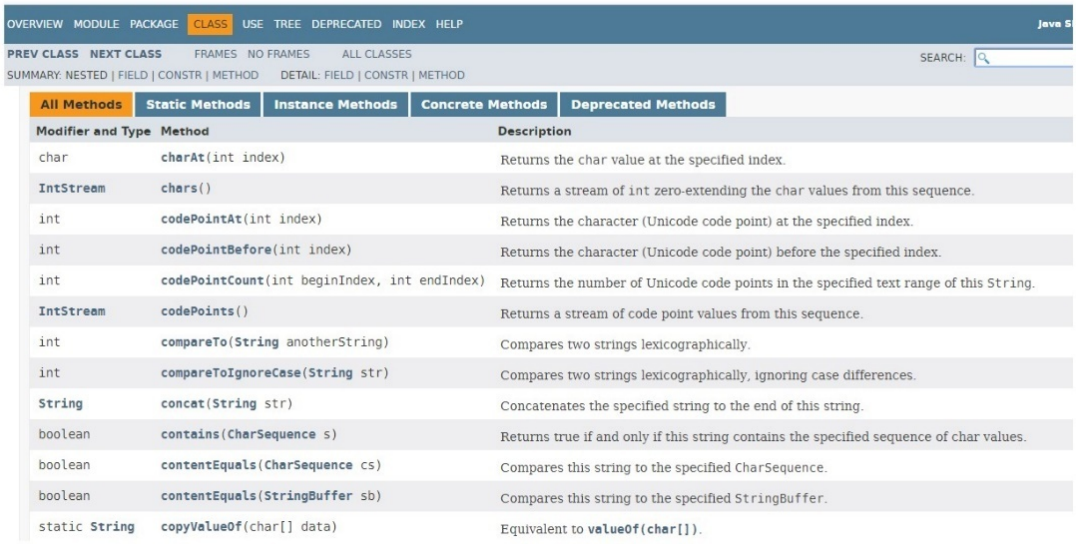
Точно так же, если применить toUpperCase к той же строке, будет возвращена строка со всеми заглавными ABCDE.

Я использую больше примеров позже, чтобы проиллюстрировать, как могут быть использованы эти методы.

Мы только что обсудили ряд полезных методов класса String.

На самом деле существует гораздо больше методов, и вы можете узнать более подробную информацию об этих методах на сайте Java, который поддерживается Oracle.

Вот список методов, которые вы можете здесь увидеть (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html).



OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

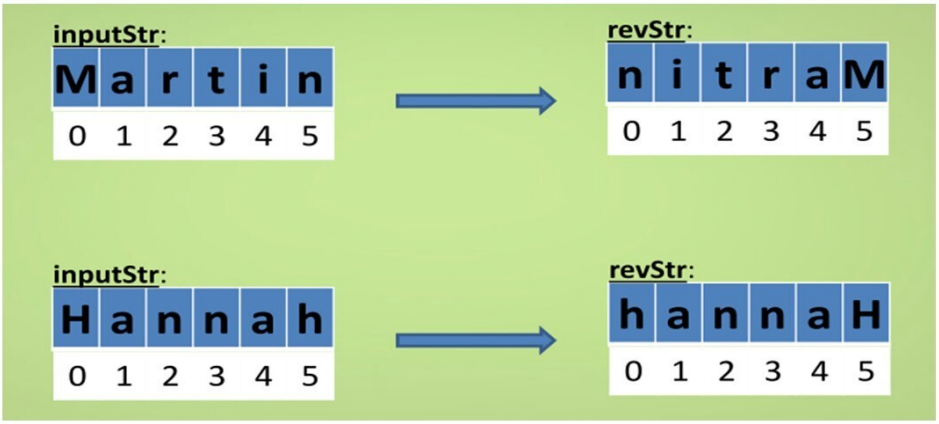
SEARCH:

All Methods | Static Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method	Description
char	charAt(int index)	Returns the char value at the specified index.
IntStream	chars()	Returns a stream of int zero-extending the char values from this sequence.
int	codePointAt(int index)	Returns the character (Unicode code point) at the specified index.
int	codePointBefore(int index)	Returns the character (Unicode code point) before the specified index.
int	codePointCount(int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range of this String.
IntStream	codePoints()	Returns a stream of code point values from this sequence.
int	compareTo(String anotherString)	Compares two strings lexicographically.
int	compareToIgnoreCase(String str)	Compares two strings lexicographically, ignoring case differences.
String	concat(String str)	Concatenates the specified string to the end of this string.
boolean	contains(CharSequence s)	Returns true if and only if this string contains the specified sequence of char values.
boolean	contentEquals(CharSequence cs)	Compares this string to the specified CharSequence.
boolean	contentEquals(StringBuffer sb)	Compares this string to the specified StringBuffer.
static String	copyValueOf(char[] data)	Equivalent to valueOf(char[]).

Это методы charAt(), compareTo(), concat(). Это метод length(), который предназначен для выяснения длины строки, это две различные версии метода substring(), методы toLowerCase (), toUpperCase (), и многое другое.

Давайте использовать некоторые из методов, которые мы обсудили, чтобы разрабатывать наши собственные методы работы со строками, например, метод для реверса строки.



Например, возьмем 6-ти символьную строку Мартин, помня, что индекс элементов строки начинается с 0.

В этом примере, последний индекс, это 5, то есть, на единицу меньше, чем длина строки.

Чтобы изменить строку, первый элемент будет помещен в последнюю позицию и последний в первую позицию, и так далее.

Так обратное Martin является Nitram.

Обратите внимание, что регистр символов сохраняется, так что M в конце является заглавной буквой.

Давайте посмотрим на другой пример, здесь Ханна является строкой, также с 6 символами.

Когда вы измените эту строку, то результатом будет, так или иначе, такое же написание, как в исходной строке.

Хотя большая буква H будет теперь в конце.

Такая строка, как эта, называется палиндромом.

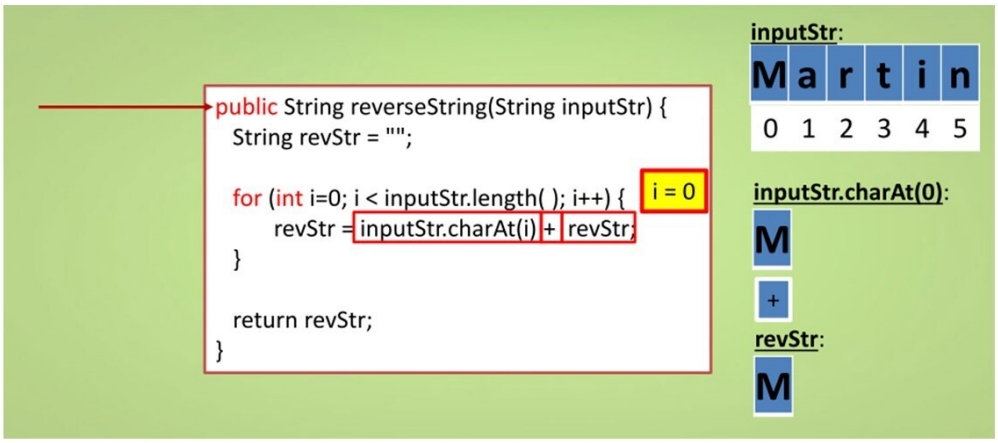
Палиндром – это слово или предложение, которое читается одинаково в обоих направлениях.

Я вернусь к этому позже при определении, является ли строка палиндромом.

Давайте подумаем о том, как мы можем обратить вспять строку? Существует много способов это сделать.

Подход, который мы собираемся использовать, это создавать строку по одному символу за раз, начиная с первого символа исходной строки.

Можно на самом деле также начать с последнего символа, если вы хотите.



Вот метод для реверса строки.

Метод называется reverseString и принимает один параметр типа String и возвращает строку в качестве результата.

Так как мы должны создать перевёрнутую строку, локальная переменная revStr типа String необходима для хранения промежуточных результатов.

Переменная revStr инициализируется пустой строкой.

Обратите внимание, что нет ничего между парой двойных кавычек, даже нет пробела.

Пустая строка не то же самое, как строка с пробелом, так как пробел представляется в виде символа.

Основная работа в этом методе осуществляется for циклом.

Индекс for цикла будет проходить через каждый элемент строки по одному, начиная с индекса 0 или первого символа.

Цикл закончится, если i станет не меньше, чем длина строки, которая определяется методом length.

Следует помнить, что метод length класса String, в отличие от переменной экземпляра как в массивах, имеет пару круглых скобках после length.

Давайте рассмотрим пример, чтобы проиллюстрировать, как работает этот цикл.

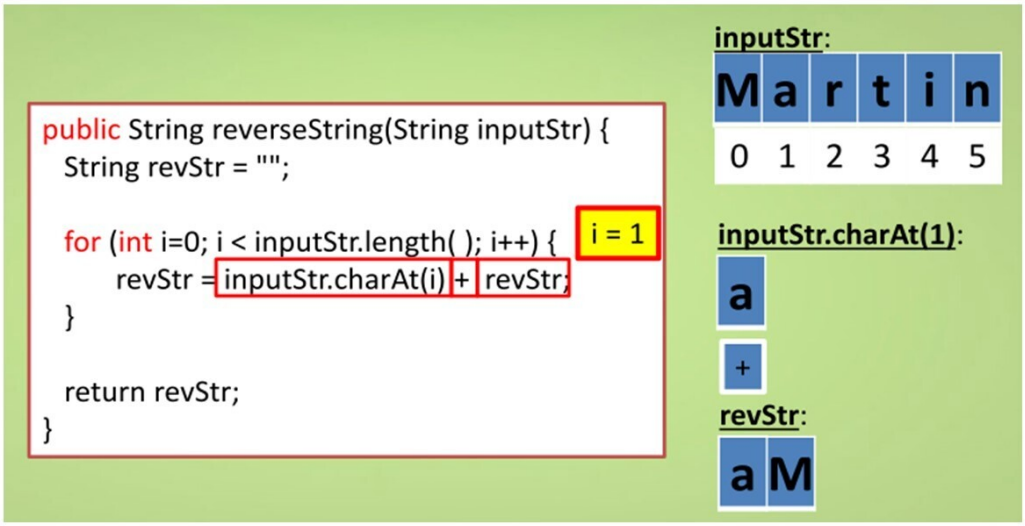
Исходная строка имеет «Martin» в качестве своего значения и revStr является пустой строкой.

Когда цикл имеет первое вхождение, i = 0, тогда метод charAt в операторе присваивания будет извлекать 0-й элемент, от входной строки или заглавную букву M, которая затем объединится с revStr с помощью оператора плюс.

Вы также можете использовать метод concat, если вы хотите.

Объединение приведет к строке, состоящей из заглавной буквы M.

В следующий раз в цикле, переменной i присваивается значение 1.



Внутри цикла, метод charAt будет извлекать символ входной строки, индексированный 1 или символ «a», который затем объединяется с revStr или M.

Обратите внимание, что порядок конкатенации важен в данном случае, так как символ «a» ставится перед M, потому что мы строим перевёрнутую строку.

Полученная revStr в этой итерации цикла будет «aM».

Этот процесс будет продолжаться и конечный результат Nitram присвоится переменной revStr.

Индекс i равный 6 будет проверен, меньше ли он, чем длина строки, и результат будет в этом случае false, так что цикл завершится.

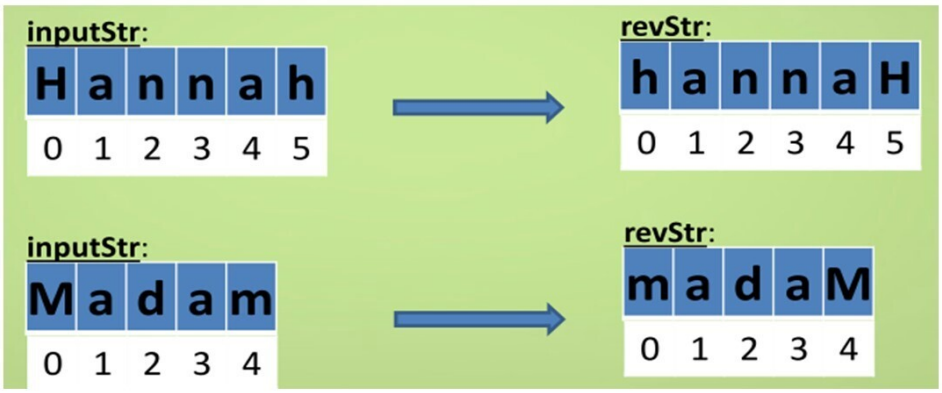
Когда происходит выход из цикла, значение переменной revStr возвращается как результат этого вызова метода.

Это очень короткая программа, вы можете ввести ее в и протестировать ее в среде IDEA, если вы хотите.

Вы можете также поэкспериментировать с использованием метода concat вместо оператора +.

Как я упоминал ранее, палиндром – это слово или фраза, которая читается одинаково в обоих направлениях.

Это пример того, что мы видели ранее, Hannah читается одинаково вперед и назад, так что это палиндром.



Еще один пример, Madam также читается одинаково в обоих направлениях.

В качестве упражнения, я хочу, чтобы вы использовали метод реверса строки, который мы только что обсуждали, чтобы написать логический метод isPalindrome, который возвращает true, если входной параметр является палиндромом и возвращает false в противном случае.

```
public boolean isPalindrome(String str) {

    // hint:  use the method reverseString
}
```

Обратите внимание, что мы можем игнорировать регистры при определении того, является ли входной параметр палиндромом.

В предыдущем примере Ханна, заглавную H можно рассматривать так же, как маленькую букву h.

Вы можете также проверить, какие другие методы доступны в классе String.

Если вы используете методы approach, это может быть реализовано в одном выражении return.

File IO

Программы, которые мы писали до сих пор, в основном полагались на консольный ввод-вывод. И это часто не очень эффективно, если нужно иметь дело с большим количеством входных данных.

Например, информация о всех студентах в классе, где может быть более 100 студентов.

Также это очень раздражает, если нужно обеспечить одинаковый набор входных данных каждый раз, когда вы хотите запустить программу, особенно на стадии разработки программы.

Бывает и так, часто, когда при завершении работы программы, которая производит некоторые результаты, результаты должны быть где-нибудь сохранены.

Все это является хорошей причиной для использования файлов, которые могут быть сохранены на некотором постоянном носителе, например, жестком диске в нашем компьютере.

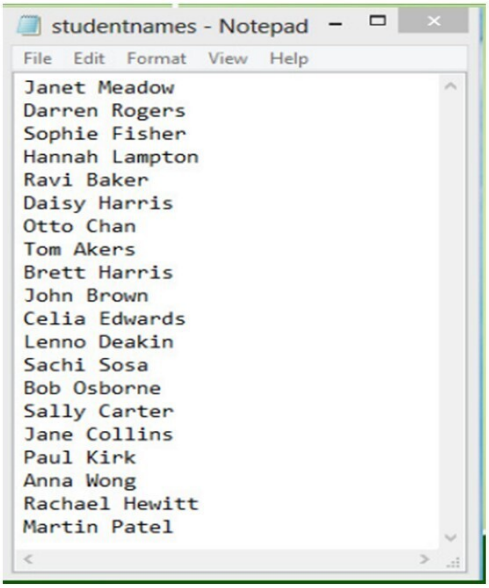
Итак, что представляет собой файл?

Мы используем файлы все время.

Мы используем программное обеспечение для обработки текста, чтобы создать документы и отчеты, которые будут храниться в виде файлов.

Изображения, которые мы использовали, хранятся в файлах.

Программный интерфейс File IO, который мы собираемся использовать здесь, имеет дело только с простыми текстовыми файлами.



Это пример текстового файла, содержащего список имен.

Это может быть список имен студентов в классе.

Тип файлов здесь – .txt, и имя конкретно этого файла studentnames.txt.

Как вы можете видеть из этого примера, данные, хранящиеся в этом файле, организованы построчно и на каждой строке содержится последовательность символов.

Для этого вида файлов, один из способов обработки состоит в чтении строк файла построчно за раз по строке символов, а затем манипулировать строкой символов, используя методы, которые мы только что обсудили.

Когда имеешь дело с файлом, первый шаг, как правило, это открыть файл, если файл уже существует, или создать файл, если файл еще не создан.

После того, как файл открыт, тогда следующим шагом является чтение данных или запись данных в файл.

Важным шагом является закрыть файл, когда использование файла завершено.

Способом открыть файл в Java является связывание некоторого объекта с файлом.

```
import java.io.File;
import java.io.PrintWriter;
import java.util.Scanner;
```

Чтобы установить соответствие между объектом и файлом, мы будем использовать классы File/PrintWriter/Scanner.

Это заранее определенные классы в Java, и для того, чтобы использовать их, мы должны включить соответствующие библиотеки.

Пакет java.io содержит много классов, которые можно использовать для программ с вводом и выводом данных.

Это пакеты, которые должны быть импортированы в начале вашей программы.

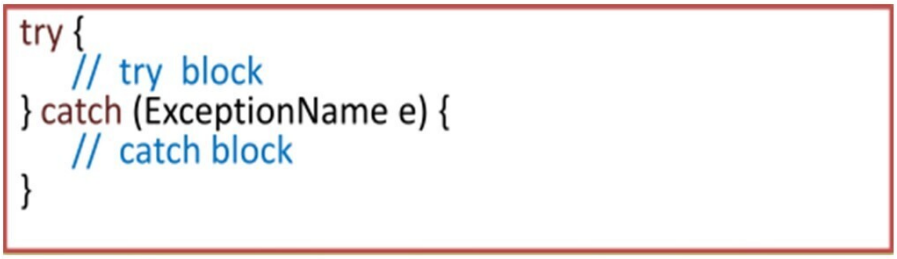
Работа с File IO может сопровождаться ошибками.

Пользователь может ввести неправильное имя файла или может не иметь права доступа к файлу.

Также может быть проблема с устройствами хранения данных, например, диск может быть полностью заполнен.

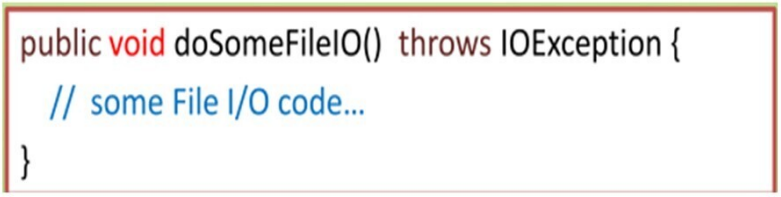
Поэтому, при выполнении File IO, необходима обработка исключений или ошибок.

И правильным способом обработки исключений является использование блока try-catch в виде, приведенном здесь.



Однако, обработка исключений, выходит за рамки данного курса.

Одним из способов обхода обработки исключений является добавление «throws IOexception» после заголовка метода, когда файловый ввод/вывод выполняется в методе.



Это будет информировать вызывающего метод, что вызываемый метод может выбрасывать исключение, и позволить вызывающему методу принять решение о том, как с ним обращаться.

Давайте сначала обсудим ввод данных из файла с помощью класса Scanner.

https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

boolean	`hasNextDouble()` Returns true if the next token in this scanner's input can be interpreted as a double value using the `nextDouble()` method.
boolean	`hasNextFloat()` Returns true if the next token in this scanner's input can be interpreted as a float value using the `nextFloat()` method.
boolean	`hasNextInt()` Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the `nextInt()` method.
boolean	`hasNextInt(int radix)` Returns true if the next token in this scanner's input can be interpreted as an int value in the specified radix using the `nextInt()` method.
boolean	`hasNextLine()` Returns true if there is another line in the input of this scanner.
boolean	`hasNextLong()` Returns true if the next token in this scanner's input can be interpreted as a long value in the default radix using the `nextLong()` method.
boolean	`hasNextLong(int radix)` Returns true if the next token in this scanner's input can be interpreted as a long value in the specified radix using the `nextLong()` method.
boolean	`hasNextShort()` Returns true if the next token in this scanner's input can be interpreted as a short value in the default radix using the `nextShort()` method.
boolean	`hasNextShort(int radix)` Returns true if the next token in this scanner's input can be interpreted as a short value in the specified radix using the `nextShort()` method.
`IOException`	`ioException()` Returns the `IOException` last thrown by this `Scanner`'s underlying `Readable`.
`Locale`	`locale()` Returns this scanner's locale.
`MatchResult`	`match()` Returns the match result of the last scanning operation performed by this scanner.
`String`	`next()` Finds and returns the next complete token from this scanner.
`String`	`next(Pattern pattern)` Returns the next token if it matches the specified pattern.
`String`	`next(String pattern)` Returns the next token if it matches the pattern constructed from the specified string.
`BigDecimal`	`nextBigDecimal()` Scans the next token of the input as a `BigDecimal`.
`BigInteger`	`nextBigInteger()` Scans the next token of the input as a `BigInteger`.

Способ, с помощью которого Java открывает файл, это связать объект с файлом.

То есть установить соответствие между объектом файла и файлом, указанным с помощью имени файла.

```
File file = new File("file name");
Scanner name = new Scanner(file);
```

– Example:

```
File file = new File("mydata.txt");
Scanner input = new Scanner(file);
```

После того, как устанавливается соответствие, объект Scanner может быть создан с помощью объекта файла.

После того, как файл открыт, следующим шагом является чтение данных из файла.

Мы сначала сосредоточимся на чтении данных из файла, строка за строкой.

Важно закрыть файл с помощью метода close, когда вы закончите работать с файлом, в противном случае, могут быть некоторые нежелательные последствия, такие как утечка памяти.

Scanner прерывает свой ввод последовательностей лексем в соответствии с конкретными правилами, используя шаблон разделителя, по умолчанию, пробел используется в качестве разделителя для лексем.

Scanner обеспечивает методы для преобразования вводимого текста в соответствующие типы Java, в том числе примитивные типы и тип String.

Вот некоторые из полезных методов в классе Scanner.

```
nextLine()

next()

nextInt(), nextDouble()

hasNextLine()

hasNext()

hasNextInt(), hasNextDouble()
```

Здесь есть методы для чтения содержимое из файла, в том числе метод nextLine(), который возвращает следующую линию строки.

Метод next() возвращает следующую строку.

Вернее next() может читать ввод только до пробела.

Этот метод не может читать два слова, разделенных пробелом.

Кроме того, метод next() помещает курсор в ту же строку после прочтения ввода.

Метод nextLine() считывает ввод, включая пробел между словами (то есть, он читает до конца строки \n). После считывания ввода метод nextLine() позиционирует курсор в следующей строке.

Методы nextInt() и nextDouble() читают следующую строку, а затем преобразовывают ее в целое, если nextInt() используется,

и число с плавающей точкой, если метод nextDouble() используется.

Другие методы здесь проверяют наличие контента в файле.

Метод hasNextLine() возвращает true, если следующая строка существует.

Метод hasNext() возвращает true, если еще есть следующая строка.

Методы hasNextInt() и hasNextDouble() возвращают true, если еще есть целое или число с плавающей запятой в файле.

Пример использования Scanner

Чтобы показать, как использовать Scanner для выполнения файлового ввода, давайте рассмотрим простую программу, которая открывает текстовый файл, читает файл построчно, а затем распечатывает строки ввода по одной в консоли.

```
public void readStudentNamesFromFile() throws Exception {
  // 1. Create a File and Scanner objects
  File inputFile = new File("studentnames.txt");
  Scanner input = new Scanner(inputFile);
  // 2. read the content using a loop
  for (int i=0;  input.hasNextLine(); i++) {
      String inputStudentName = input.nextLine();
      IO.outputln("Student #" + i + ": " + inputStudentName);
  }
  // 3. close the file and print the result
  input.close();
}
```

Метод называется readStudentNamesFromFile.

Как я упоминал ранее, метод обрабатывает исключение ввода-вывода с помощью throws в сигнатуре метода, объявляя возможное исключение.

Внутри метода, объект файла inputFile объявляется чтобы быть связанным с файлом, с именем, заданным в качестве параметра в конструкторе класса File.

В этом примере, именем файла является studennames.txt. Это тот же файл, который мы видели ранее.

После того, как соответствие между объектом файла и самим файлом устанавливается, создается объект Scanner с именем input для входного файла.

При этом открывается файл. И содержимое можно прочитать из файла.

В этом методе, цикл for используется для чтения по одной строке за один раз из файла.

Обратите внимание, что в то время, как инициализация и обновление индекса цикла i сделаны, используя обычный способ, условие цикла проверяется без ссылки на индекс цикла.

Вместо этого, цикл проверяет, есть ли еще любая оставшаяся новая строка в исходном файле для чтения с помощью метода hasNextLine.

Поэтому, цикл будет продолжаться, пока не достигнет конца файла.

Я хочу отметить, что пустая строка в файле, это не то же самое, что конец файла, поскольку файл, вероятно, содержит пустые строки, которые разделяют различные разделы файла.

Внутри цикла, метод nextLine используется для чтения следующей строки, и результат присваивается переменной String inputStudentName.

Эта программа будет просто выводить каждую строку, извлеченную из файла, добавляя ее к строке Student # i.

Когда программа завершит цикл, входной файл должен быть закрыт перед выходом из метода.

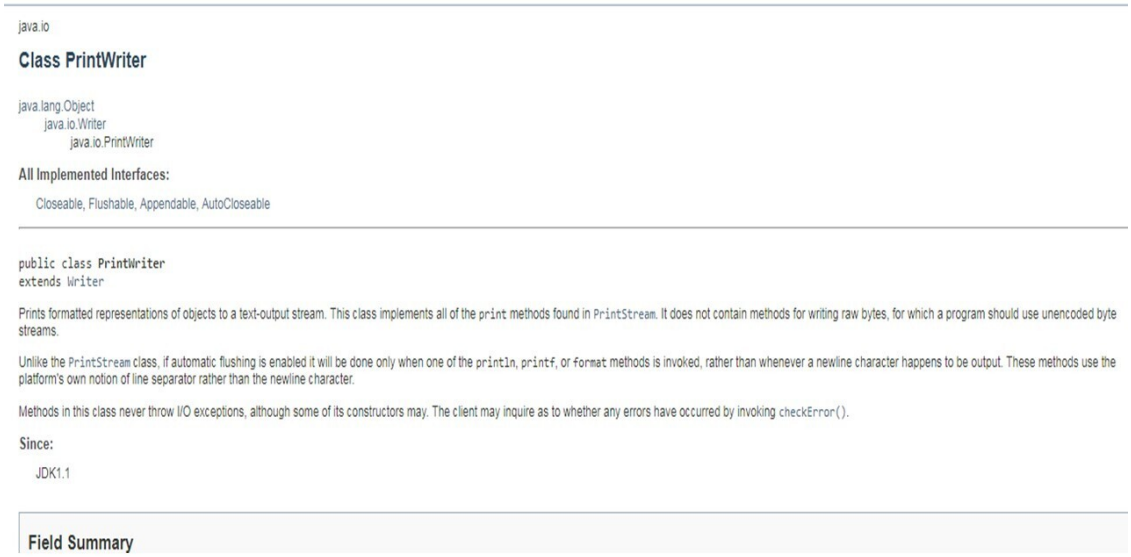
```
Student #0: Janet Meadow
Student #1: Darren Rogers
Student #2: Sophie Fisher
Student #3: Hannah Lampton
Student #4: Ravi Baker
Student #5: Daisy Harris
Student #6: Otto Chan
Student #7: Tom Akers
Student #8: Brett Harris
Student #9: John Brown
Student #10: Celia Edwards
Student #11: Lenno Deakin
Student #12: Sachi Sosa
Student #13: Bob Osborne
Student #14: Sally Carter
Student #15: Jane Collins
Student #16: Paul Kirk
Student #17: Anna Wong
Student #18: Rachael Hewitt
Student #19: Martin Patel
```

```
Janet Meadow
Darren Rogers
Sophie Fisher
Hannah Lampton
Ravi Baker
Daisy Harris
Otto Chan
Tom Akers
Brett Harris
John Brown
Celia Edwards
Lenno Deakin
Sachi Sosa
Bob Osborne
Sally Carter
Jane Collins
Paul Kirk
Anna Wong
Rachael Hewitt
Martin Patel
```

Здесь показаны скриншоты двух окон, в том числе вывод программы, с соответствующим входом.

Пример использования PrintWriter

Подобно тому, что мы сделали с входными файлами, для того чтобы сделать теперь вывод файла, мы должны сначала связать объект File с файлом, указанным в имени файла.



java.io

Class PrintWriter

java.lang.Object
  java.io.Writer
    java.io.PrintWriter

All Implemented Interfaces:

Closeable, Flushable, Appendable, AutoCloseable

```
public class PrintWriter
extends Writer
```

Prints formatted representations of objects to a text-output stream. This class implements all of the `print` methods found in `PrintStream`. It does not contain methods for writing raw bytes, for which a program should use unencoded byte streams.

Unlike the `PrintStream` class, if automatic flushing is enabled it will be done only when one of the `println`, `printf`, or `format` methods is invoked, rather than whenever a newline character happens to be output. These methods use the platform's own notion of line separator rather than the newline character.

Methods in this class never throw I/O exceptions, although some of its constructors may. The client may inquire as to whether any errors have occurred by invoking `checkError()`.

Since:

JDK1.1

Field Summary

После того, как устанавливается соответствие, объект PrintWriter может быть создан с помощью объекта File.

```
public void readWriteStudentNames () throws IOException {
  // 1.1 Create a File and Scanner objects
  File inputFile = new File("studentnames.txt");
  Scanner input = new Scanner(inputFile);
  // 1.1 Create a File and PrintWriter objects
  File outputFile = new File("output.txt");
  PrintWriter writer = new PrintWriter(outputFile);
  // 2. read and outputthe content using a loop
  for (int i=0;  input.hasNextLine(); i++) {
      String inputStudentName = input.nextLine();
    writer.println("Student #" + i + ": " + inputStudentName);
  }
  // 3. close the file and print the result
  input.close();
  writer.close();
}
```

После того, как файл открыт, мы можем начать вывод данных в файл.

Вы можете использовать print или println метод для печати значений различных типов, в том числе примитивных типов и строк в файле.

Разница между print и println заключается в том, что print будет оставаться на той же строке после печати, в то время как println будет двигаться к следующей позиции печати на новой строке.

Опять же, важно закрыть файл с помощью метода close, когда вы закончили работу с файлом.

В противном случае, вы можете потерять все данные в файле.

Здесь мы используем метод, который мы уже обсуждали, и который использовал Scanner для считывания данных из входного файла и выводил результаты в консоль.

В этом примере, мы изменим предыдущую программу, так что, вместо вывода результатов в окно терминала, вывод будет храниться в выходном файле.

Следуя четырехшаговому процессу, который мы только что описали, объект File и объект PrintWriter будут созданы для файла с именем output.txt, который используется в качестве параметра для конструктора файла, когда объект outputFile создается.

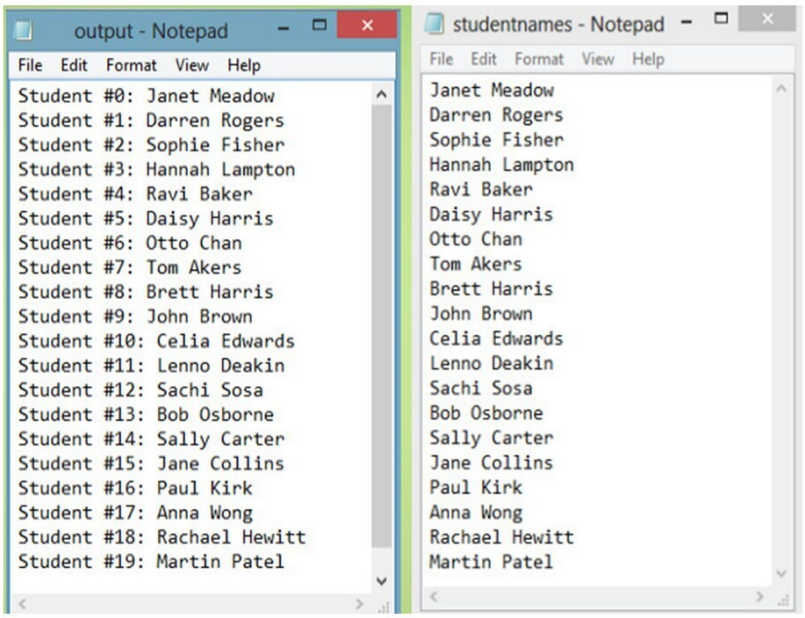
После создания outputFile, объект PrintWriter может использовать метод println для вывода результатов.

Таким образом, здесь, метод writer.printIn() используется для замены IO.outputln, так что результаты будут напечатаны в файл, а не в консоль.

Когда вся работа сделана, перед выходом из программы, как выходные, а также входные файлы должны быть закрыты с помощью метода close.

Чтобы убедиться, что программа действительно закончила работу в соответствии с требованиями, вот скриншот содержания двух файлов с output.txt и studentnames.txt.

Оба открыты с помощью блокнота.



Вы можете видеть, что, вместо того чтобы отображать только имена, как в оригинальном studentnames файле, в выходном файле также печатается счетчик.

Таким образом, с помощью вывода результатов в файл, в отличие от предыдущего примера, где результаты просто печатаются на экране, результаты теперь могут быть сохранены и могут быть использованы, если это необходимо.

Оптическое распознавание символов

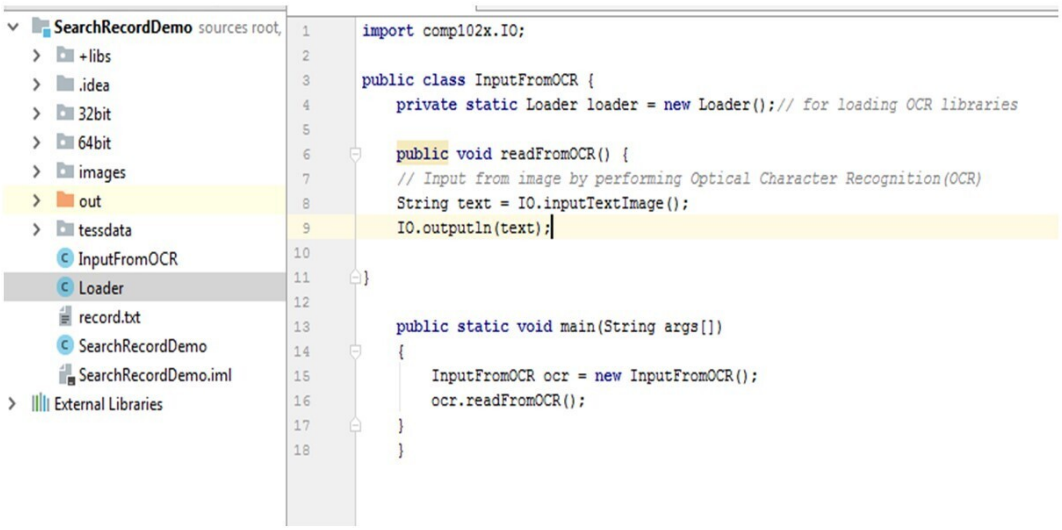
Мы уже видели, что числа могут быть представлены в виде штрих-кодов. И в настоящее время, 2D штрих-коды, такие как QR-коды, могут быть использованы для представления гораздо более богатого набора символов.

Другая важная технология ввода основана на оптическом распознавании символов, или optical character reader OCR.

OCR относится к автоматическому преобразованию печатного или рукописного текста в машиночитаемую форму.

Технология OCR используется для многих приложений, включая чтение почтовых адресов на конвертах, обработку банковских операций, чтение документов для слабовидящих, распознавание рукописного ввода с портативных устройств, например, для проверки подписи, чтение номерных знаков автомобилей, и многое другое.

Я не буду обсуждать здесь технологии, лежащие в основе OCR, потому что это выходит за рамки этого курса. Я просто дам вам простую демонстрацию того, как OCR может быть использовано.



Вот простая программа, которая принимает изображение некоторого текста, которое может быть получено различными средствами, включая цифровую камеру, сканер или скриншот экрана (https://github.com/novts/java-base).

В дополнение к принятию штрих-код в качестве входных данных, класс ввода-вывода IO может также принимать текстовые изображения в качестве входных данных.

Для программы сначала потребуется импортировать класс IO comp102x пакета.

Класс IO имеет встроенный метод для OCR, использующий пакет Java, который доступен в открытом доступе для бесплатного скачивания.

На самом деле есть две версии реализации OCR, одна для 32-битной архитектуры и другая для 64-битной, именно это мы должны проверить здесь с помощью класса Loader, для определения того, следует ли загружать 32-разрядную или 64-разрядную версию библиотеки.

Вам не нужно понимать, как это на самом деле сделано.

К сожалению, эта библиотека будет работать только для Windows.

Я просто даю вам картину здесь того, как OCR может работать.

Здесь метод readFromOCR будет выполнять работу, состоящую в чтении входного изображения с использованием OCR.

Это очень простой метод, он будет вызывать метод inputTextImage из класса IO.

Это похоже на то, что мы делали для целого числа с использованием inputInteger или double типа с помощью inputDouble.

Опять же, мы используем идею процедурной абстракции здесь, так как подробности того, как метод inputTextImage реализуется, скрыты от пользователя.

Все, что вам нужно знать, это то, что метод, как предполагается, выполняется.

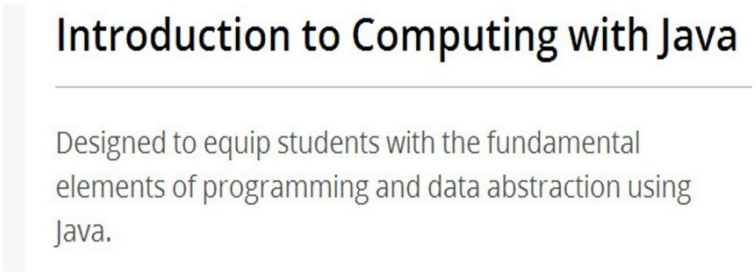
В этом случае он будет рассматривать изображение, и пытаться извлечь текстовые данные из изображения. И результат присваивается символьной строке.

Важно отметить, что этот метод не может все время правильно выполнять задачу.

Я дам вам демонстрацию позже, чтобы показать вам, что это значит.

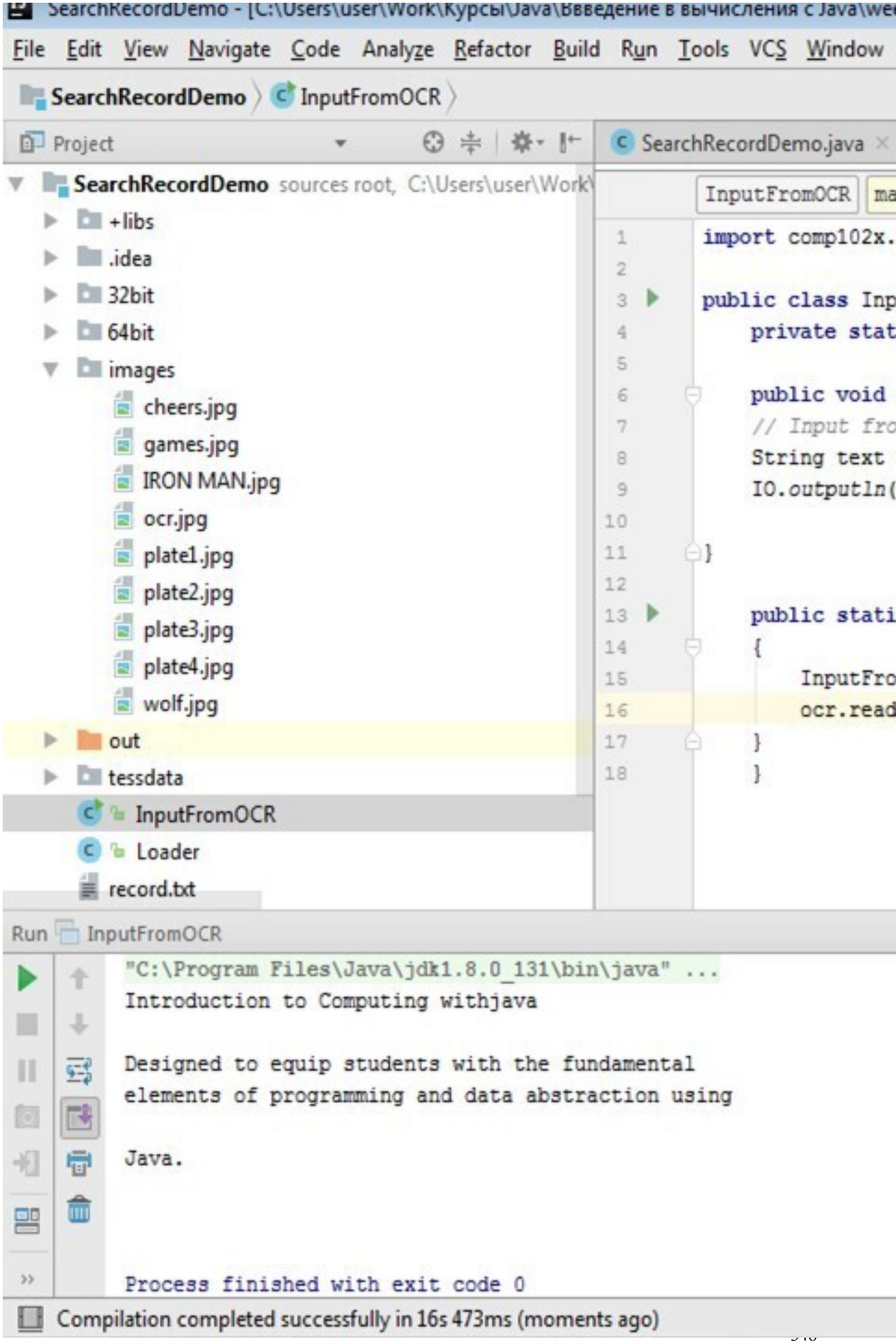
Затем результат отображается в консоли с использованием IO.outputln.

Прежде чем я покажу вам демонстрацию, сначала, я покажу вам некоторые изображения, которые я собираюсь использовать при тестировании программы.



Первое изображение, которое я собираюсь использовать, получается с помощью снимка экрана.

Откроем программу в IDEA.


SearchRecordDemo - [C:\Users\user\Work\Курсы\Java\Введение в вычисления с Java\we
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window
SearchRecordDemo
InputFromOCR
Project
SearchRecordDemo.java
SearchRecordDemo sources root, C:\Users\user\Work\
+libs
.idea
32bit
64bit
images
cheers.jpg
games.jpg
IRON MAN.jpg
ocr.jpg
plate1.jpg
plate2.jpg
plate3.jpg
plate4.jpg
wolf.jpg
out
tessdata
InputFromOCR
Loader
record.txt
InputFromOCR
ma
1 import comp102x.
2
3 public class Inp
4 private stat
5
6 public void
7 // Input fro
8 String text
9 IO.println(
10
11 }
12
13 public stati
14 {
15 InputFro
16 ocr.read
17 }
18 }
Run InputFromOCR
"C:\Program Files\Java\jdk1.8.0_131\bin\java" ...
Introduction to Computing withjava
Designed to equip students with the fundamental
elements of programming and data abstraction using
Java.
Process finished with exit code 0
Compilation completed successfully in 16s 473ms (moments ago)

Это метод, который мы только что видели.

Метод readFromOCR использует imputTextImage() из класса IO для выполнения OCR изображения текста.

Поместим в папку images файл изображения.

Вы можете создать экземпляр объекта и выбрать метод readFromOCR, затем выбрать файл изображения, и вы можете видеть, что результат отображается здесь в консоли, «Introduction to Computing with Java».

Все, кажется, в порядке. На самом деле вы заметили, что J здесь с маленькой буквы, а не с заглавной буквы.

Метод не может легко отличить маленькую букву j и заглавную букву J. И если вы будете распознавать другие изображения с текстом, вы также будете сталкиваться с ошибками. Но результат не так уж плох.

Я покажу вам здесь еще одно интересное применение OCR.

Посмотрим на номерные знаки этих двух автомобилей.

Наша зрительная система настолько хороша, что мы принимаем распознавание символов как должное, но такая задача является гораздо более трудной для компьютера.

Компьютер должен сначала найти положение номерного знака в изображении, а затем иметь дело с различными атрибутами изображения, такими как цвет, размер и условия освещения.

Распознавание автомобильных номеров используется во многих приложениях, в том числе мониторинге трафика и выявлении угнанных автомобилей.

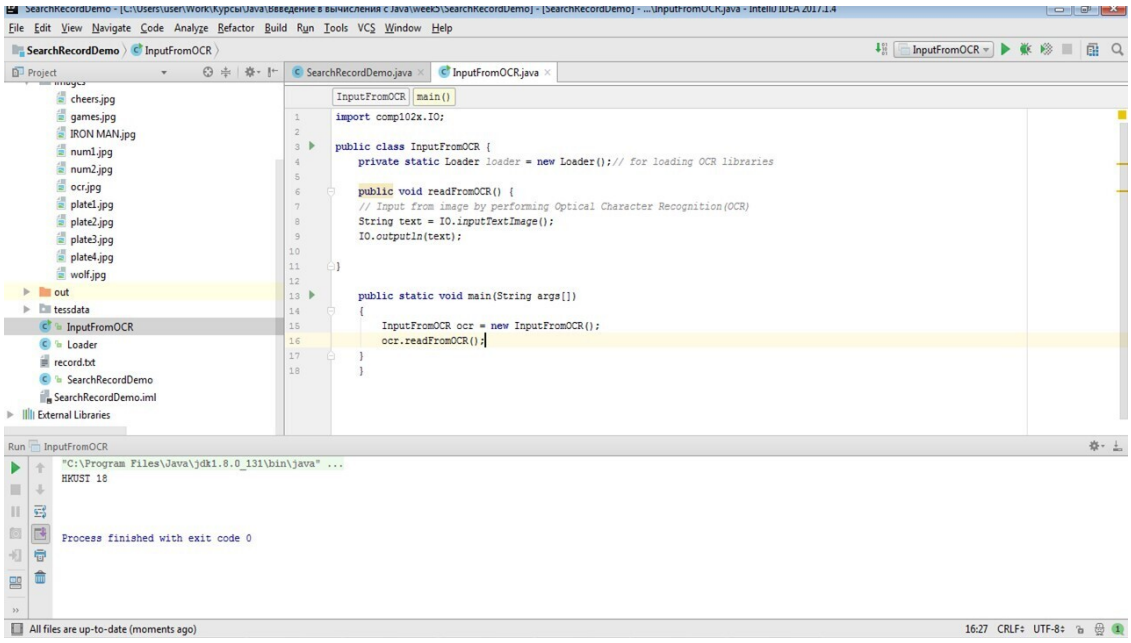
Следующее демо, которое я собираюсь показать, не такое сложное, как то, что я только что описал.

Необходимые номерные знаки должны быть извлечены вручную, и результаты также не очень надежные, потому что наша реализация основана на бесплатной технологии.

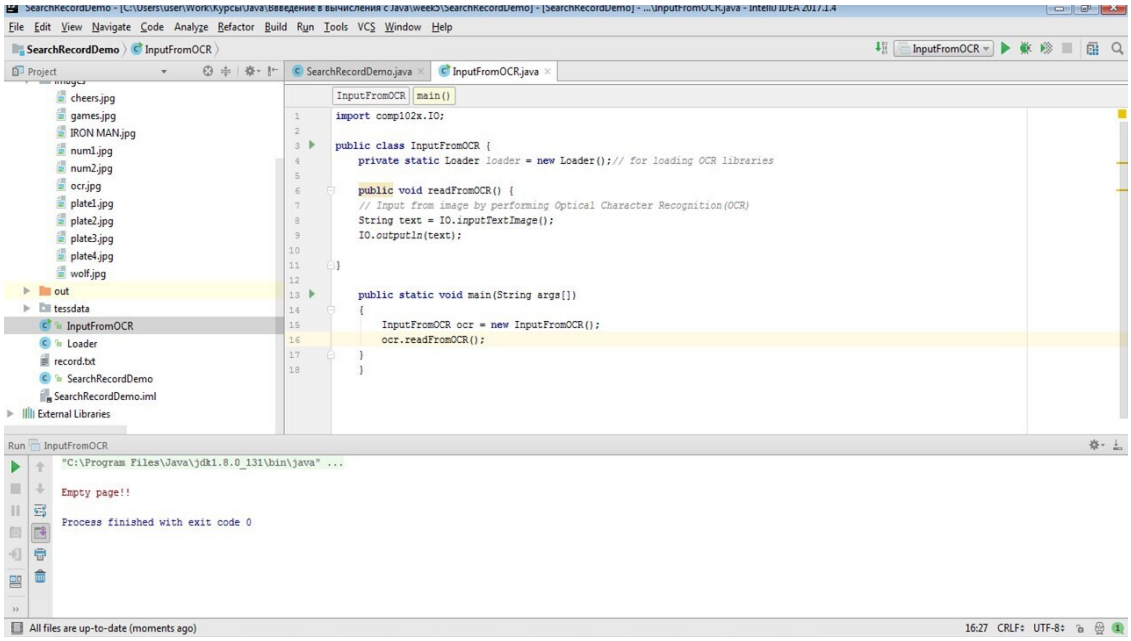
Давайте запустим программу снова.

Давайте сначала попробуем использовать программу для одного из номеров.

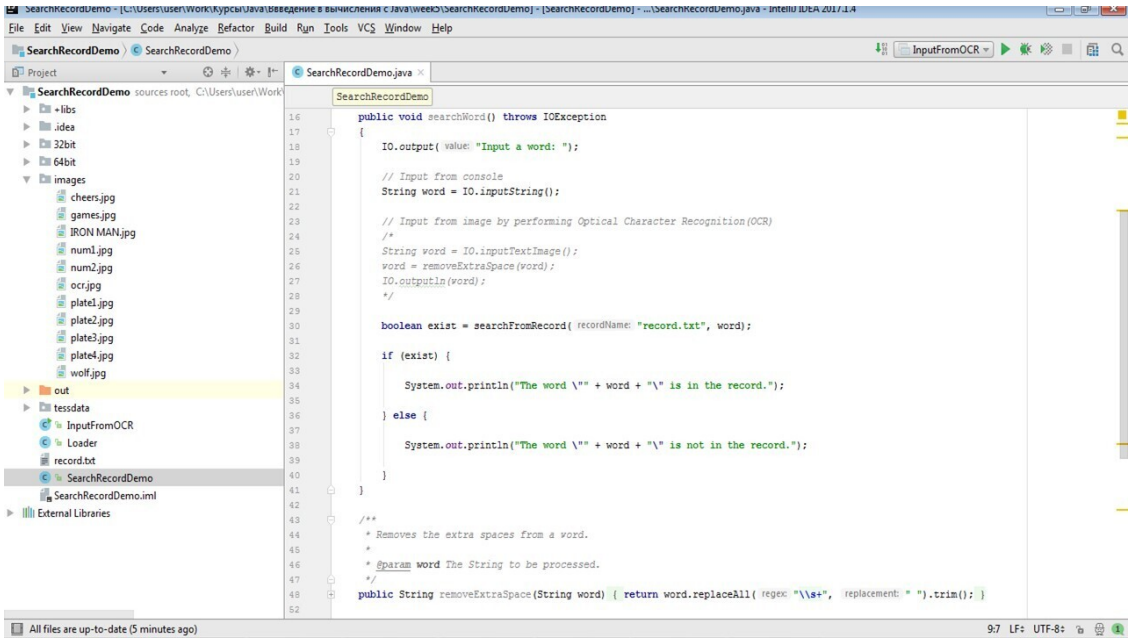
Вы можете увидеть, что результат отображается здесь правильно.



Теперь давайте попробуем другой номер. Результат будет не правильный.



В домашнем задании, вы должны будете завершить реализацию класса SearchRecordDemo, который будет искать файл записи, в котором любая из строк записи соответствует строке ввода.



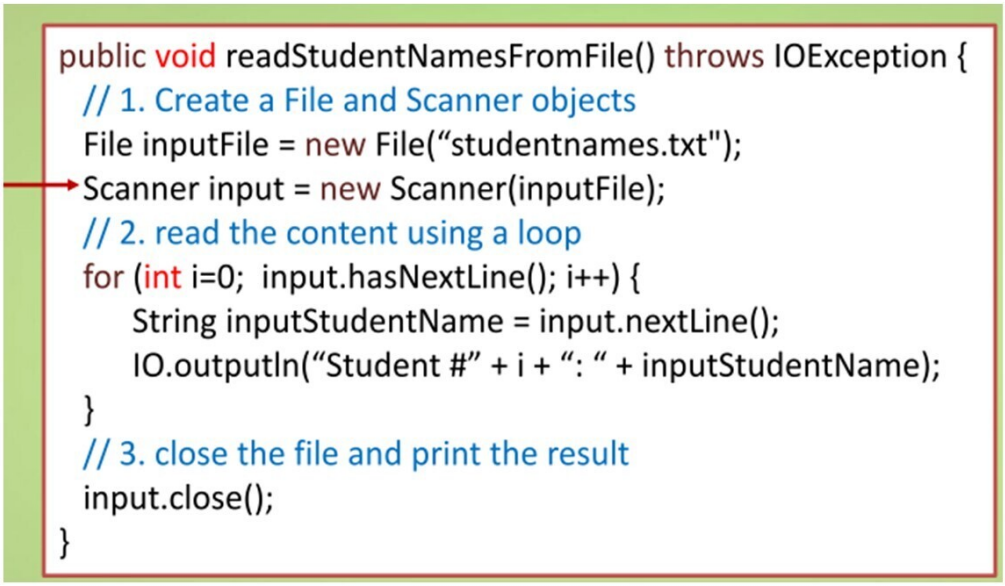
И этот метод searchFromRecord является методом, который вы должны завершить.

Текущая реализация позволяет пользователю вводить строку символов для поиска с консоли в методе searchWord ().

Но вы можете изменить программу, чтобы получать эту строку с помощью OCR.

Демонстрация чтения ввода с консоли

Давайте вернемся к примеру, который читает имена из входного файла, и посмотрим более детально на класс Scanner.



```
public void readStudentNamesFromFile() throws IOException {
  // 1. Create a File and Scanner objects
  File inputFile = new File("studentnames.txt");
  Scanner input = new Scanner(inputFile);
  // 2. read the content using a loop
  for (int i=0;  input.hasNextLine(); i++) {
      String inputStudentName = input.nextLine();
      IO.outputln("Student #" + i + ": " + inputStudentName);
  }
  // 3. close the file and print the result
  input.close();
}
```

Как уже говорилось, сканер разбивает свой вход на лексемы с использованием некоторых разделителей, как правило, пробела.

В этом примере вход объекта сканер указан как входной файл.

Фактически, сканер может обработать ввод различных типов.

На самом деле, сканер обычно используется, чтобы взять входные данные, введенные с клавиатуры в консоли, и это часто называют стандартным вводом.

Давайте изменим этот метод, чтобы вместо того, чтобы брать данные из файла, вход будет введен с консоли.

```
public void readNamesFromConsole() throws IOException {
    // 1. Create Scanner objects for standard input
    Scanner input = new Scanner(System.in);
    int nStudents = 0;
    // 2. read the content using a loop
    while (true) {
            String inputName = input.nextLine();
            if (inputName.equals("")) break;
            IO.outputln("Student #" + nStudents + ": " + inputName);
            nStudents++;
        }
}
```

Имя метода изменится с "readStudentNamesFromFile" на "readNamesFromConsole".

Вместо того чтобы создавать объект сканера для входного файла, объект сканера теперь связан с стандартным входным потоком, System.in.

При вводе из файла, условие для прекращения for цикла, это когда hasNextLine ложно, то есть, когда нет больше следующих строк.

В случае ввода с консоли, мы не можем предопределить, когда пользователь прекратит ввод данных. Так что здесь используется while цикл, который имеет условие, установленное всегда в true.

Внутри цикла, условием для выхода из цикла является случай, когда вводится пустая строка.

Обратите внимание, что используется метод "equals" для класса String, вместо «==», потому что логичное равно не будет работать должным образом, даже если две строки выглядят одинаково.

Подобно чтению из входного файла, метод nextLine вызывается для ввода объекта сканера для чтения всей строки, но в этом случае из консоли вместо входного файла, и результат присваивается переменной inputName типа String.

Если входная строка не является пустой строкой, она будет распечатана на консоли вместе со счетчиком студентов, nStudents.

Обратите внимание, что переменная nStudents объявляется перед входом в цикл для отслеживания числа студентов.

Отметим также, что здесь нет необходимости закрывать стандартный ввод.

Возникает вопрос, какая разница между методами ввода, которые мы использовали в классе IO, такими как inputString, inputInteger, inputDouble и классом сканера.

На самом деле, IO является классом-оболочкой для стандартных методов ввода и вывода.

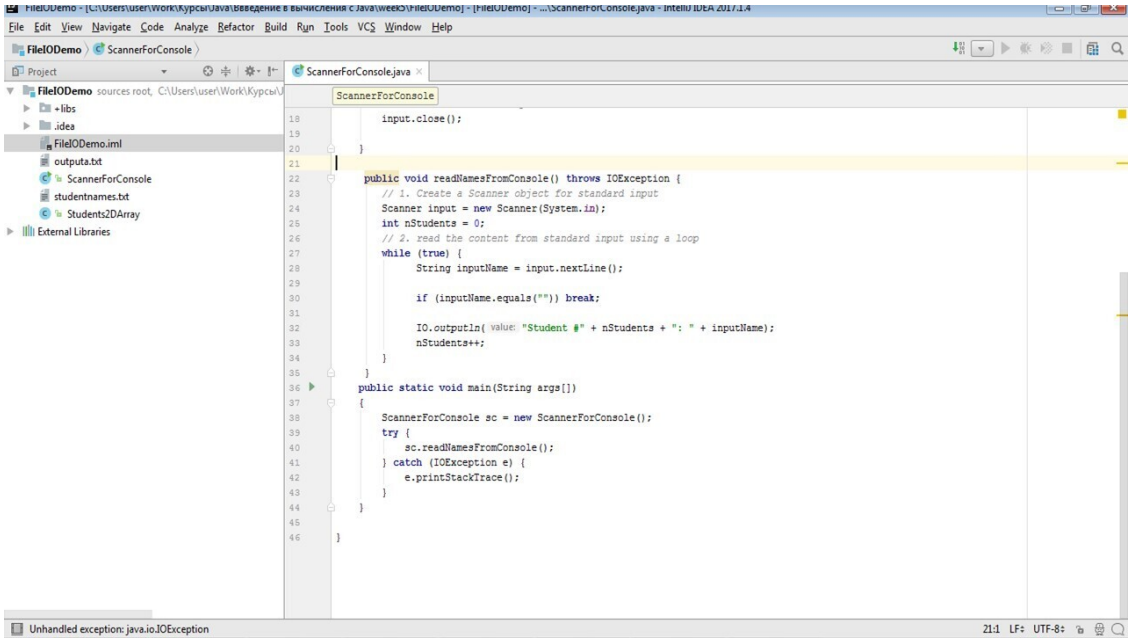
Внутри определения класса IO, создан объект сканер, и соответствующие методы вызываются для разных типов ввода.

Класс IO создан в пользовательской библиотеке специально, потому что было бы трудно объяснить объект сканера в начале курса, когда вы не знали основные понятия о классах и объектах.

Класс IO также имеет некоторые дополнительные возможности, такие как распознавание штрих-кода, а также оптическое распознавание символов.

Далее, вы можете продолжать использовать класс IO или если вы хотите, вы можете вернуться к System.in для стандартного ввода или System.out для вывода.

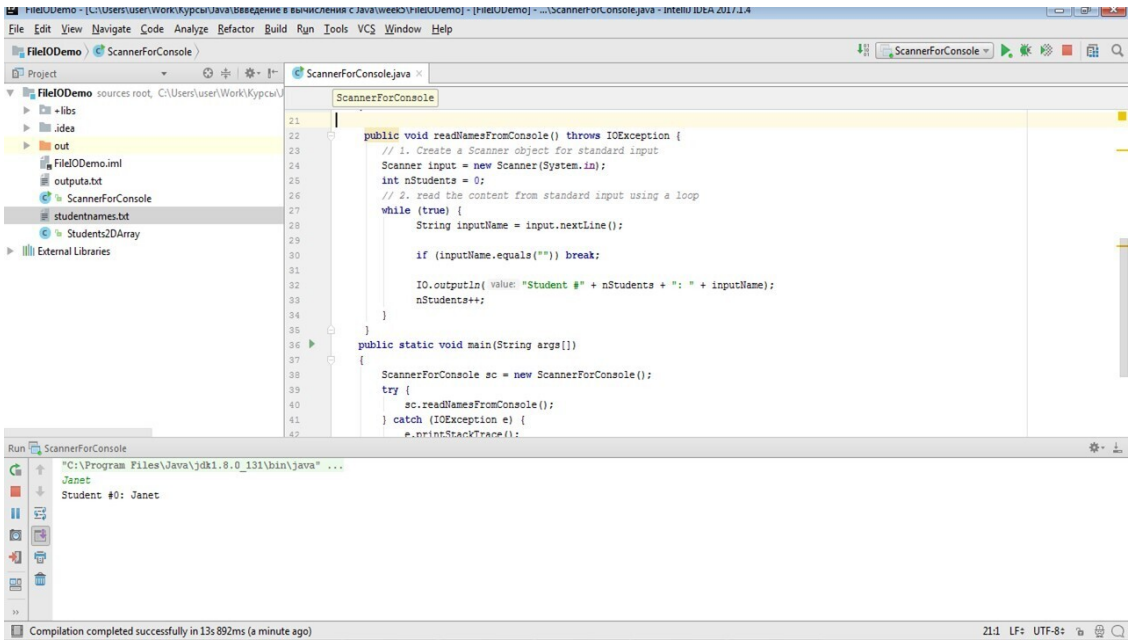
Давайте посмотрим на быструю демонстрацию программы (https://github.com/novts/java-base).



Это метод, который мы только что обсудили.

Вы можете скомпилировать программу и запустить программу путем создания экземпляра объекта, а затем вызвать метод readNameFromConcole,

И вы можете увидеть окно консоли.



Если ввести некоторые имена, и, если вы просто нажмете возврат, что означает пустая строка, вы можете увидеть, что программа останавливается.

Давайте изменим программу немного, и вместо использования метода equals, давайте изменим на оператор ==.

Скомпилируем, а потом создадим экземпляр снова.

Запустим тот же метод, и давайте введем тот же набор входных данных.

Программа будет работать так же хорошо с тем же набором входных данных.

Теперь давайте попробуем завершить программу, введя возврат, или пустую строку.

Однако, программа продолжает просить дополнительный ввод, это потому, что сравнение здесь привело к ошибке, хотя обе строки здесь, как предполагается, это пустые строки.

Программа здесь попала в бесконечный цикл; мы должны завершить программу путем принудительного сброса.

Оператор == не работает, потому что строка имеет ссылочный тип, то есть сравниваются адреса, где эти строки находятся, а не значения строк.

Это как сравнивать Лондон в Великобритании и Лондон в Канаде, хотя они оба называются Лондон, их расположения отличаются.

Таким образом, должен использоваться метод equals, а не логический оператор равно при сравнении строк.

Демонстрация использования Scanner и PrintWriter

Класс Scanner может делать больше, чем просто чтение входных данных из консоли и файлов.

Как я уже упоминал, он может также разбивать вход на лексемы. Это полезно, когда строка содержит несколько полей.

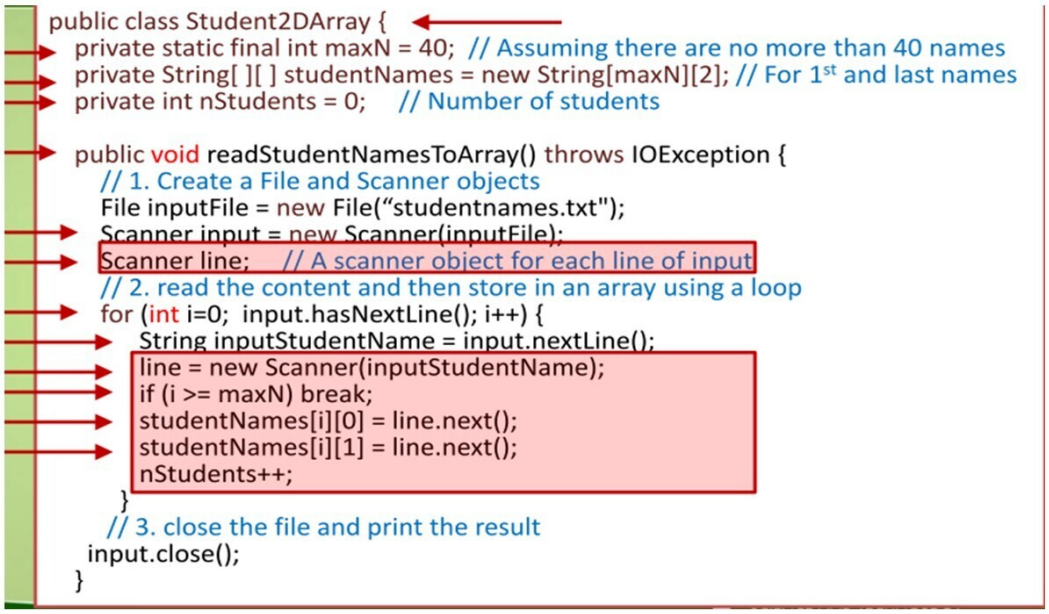
Например, если вы посмотрите на предыдущий входной файл, который мы использовали для studentnames.

Обратите внимание, что каждое имя содержит два компонента, имя и фамилию.

Вместо того, чтобы помещать все имя в одну строку символов, в некоторых случаях может быть полезно, чтобы разделить имя и фамилию.

Например, было бы более вежливо обратиться к вашим клиентам по фамилии с г-ном как префиксом или к вашим друзьям просто по именам или дорогой.

В этом примере класс Student2DArray будет читать имена, содержащиеся во входном файле, а затем извлекать из них имена и фамилии.



Имена и фамилии затем будут храниться в двумерном массиве, где каждая строка массива представляет собой имя для одного студента и столбцы – для хранения имен и фамилий.

Для этого объявлены три переменные экземпляра класса.

Поскольку размер массива должен быть предустановлен, предполагается, что существует не более чем 40 имен.

2D массив с именем studentNames размера maxNx2 объявлен для хранения имен и фамилий для каждого студента.

Переменная nStudents предназначена для отслеживания числа студентов.

Метод readStudentNamesToArray определяется для чтения имен из входного файла в массив.

Вы можете увидеть, что этот метод очень похож на предыдущий метод readStudentNamesFromFile. Основные отличия выделены в этих двух блоках.

Метод создает объект сканера для входного файла studentnames.txt, как и раньше.

Вместо того чтобы использовать только один объект Scanner, другой объект Scanner объявляется здесь для строки ввода, извлеченной из входного файла.

Ниже вы сможете увидеть, как это используется.

Условие для for цикла здесь точно такое же, как и раньше, цикл будет читать каждую строку из входного файла по одной до тех пор, пока не будет больше следующей строки, как указано в методе hasNextLine.

Внутри for цикла, первое выражение такое же, как и раньше, которое считывает строку ввода, используя метод nextLine для объекта сканера, связанного с входным файлом, и присваивает результат переменной типа String inputStudentName.

Следующий сегмент кода отличается от предыдущего метода.

В выражении здесь будет создан новый объект сканера, используя строку символов inputStudentName.

Здесь показана интересная особенность класса Scanner.

В дополнение к созданию объектов сканера из стандартного ввода и входного файла, один из его конструкторов принимает строку символов в качестве параметра и создает объект сканера.

В этом случае результирующий объект Scanner присваивается переменной, которую мы объявили ранее.

Выражение if просто проверяет, что индекс в массиве не больше максимального размера массива.

В противном случае, цикл будет завершен.

Следующие два выражения используют метод next для объекта сканера, чтобы извлечь лексему или слово, которое разделено пробелом.

Чтобы проиллюстрировать, что это значит, помните, что во входном файле studentNames, первая строка содержит имя Janet Meadow.

Таким образом, после того как первая строка считается из входного файла, переменной inputStudentName будет назначена строка символов Janet Meadow.

Обратите внимание, что существует пробел между Janet и Meadow.

В выражении здесь используется метод next для объекта сканера, чтобы извлечь первую лексему, в данном случае, Janet, и присвоить элементу с индексом i,0 из studentNames массива.

Следующее выражение будет извлекать следующую лексему, в данном случае, Meadow, а затем назначить элементу с индексом i,1.

Таким образом, в результате, имя и фамилия, будут храниться в разных местах в массиве.

Число студентов увеличивается на единицу каждый раз в цикле.

Перед выходом из метода, важно, чтобы закрыть входной файл.

После того как имена и фамилии были помещены в массив, для того чтобы сделать ссылку на них позже, данные должны быть выведены в файл.

Мы уже обсудили ранее использование PrintWriter для вывода данных в файл.

```
public void outputNameArrayToFile( ) throws IOException {
    // 1.1 Create a File and PrintWriter objects
    File outputFile = new File("output2.txt");
    PrintWriter writer = new PrintWriter(outputFile);
    // 2. read the content and then store in an array using a loop
    for (int i=0;  i < nStudents; i++) {
        // The last name is output before the first name
        writer.println(studentNames[i][1] + ", " + studentNames[i][0]);
    }
     // 3. close the file and print the result
    writer.close();
  }
}
```

Этот метод outputNameArrayToFile выведет массив имен, который мы только что создали, с помощью метода readStudentNamesToArray, в выходной файл.

Два выражения здесь создают объект PrintWriter, который будет связан с файлом с именем output2.txt.

Цикл здесь обеспечивает доступ к 2D массиву по одной строке за один раз, и использует метод println для объекта PrintWriter для вывода имен в файл.

Обратите внимание, что аргумент для println представляет собой объединение трех символьных строк, первой из них является 2-й столбец элемента массива с индексом i,1, и это даст фамилию, так что вместо вывода первым имени, как в исходном файле, первой используется фамилия.

Есть много приложений, которые рассматривают фамилию в качестве первичного ключа, например, при сортировке, имена обычно сначала сортируются по фамилиям.

Второй компонент строки аргумента, это запятая после фамилии, и третий компонент является элементом массива в первом столбце и индексируется i,0.

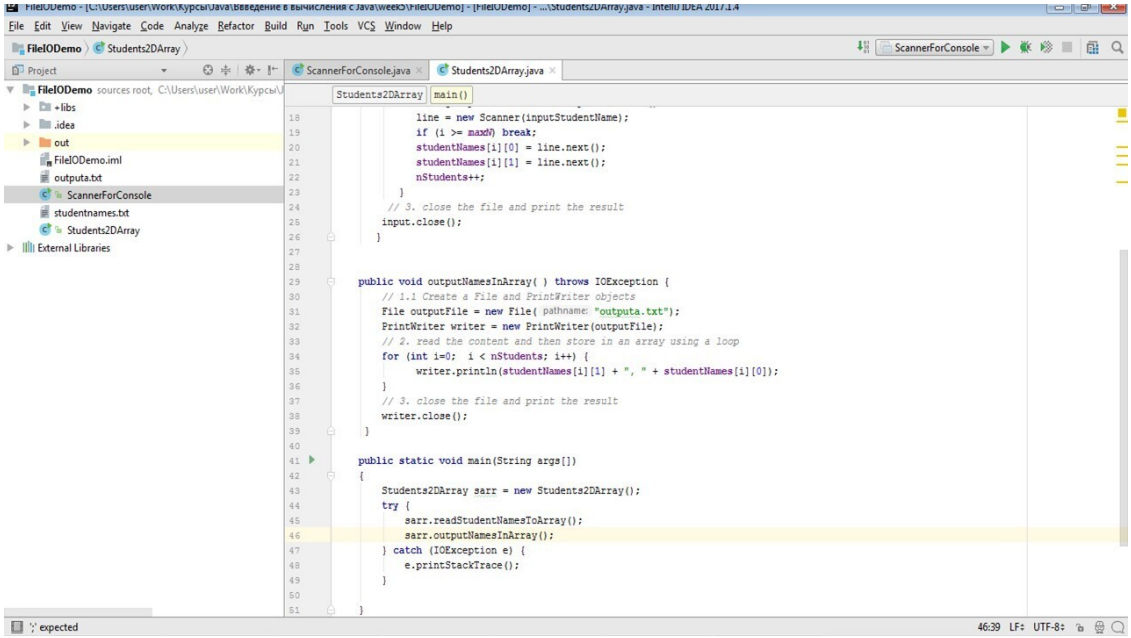
Это даст имя.

Таким образом, строка, которая должна быть выведена в файл, представляет собой строку символов, которая начинается с фамилии, через запятую, а затем идет имя.

Далее println будет двигаться к следующей позиции вывода на новую строку в выходном файле.

Опять же, важно, чтобы закрыть файл перед выходом из метода.

Давайте помотрим на быструю демонстрацию программы.



Это класс Students2DArray.

Вы можете видеть, что это то же самый класс, как тот, который мы только что обсуждали.

Здесь есть переменные экземпляра, метод readStudentNamesToArray и outputNamesInArray.

Обратите внимание, что оба метода имеют throw IOException.

Скомпилируем программу и создадим экземпляр класса.

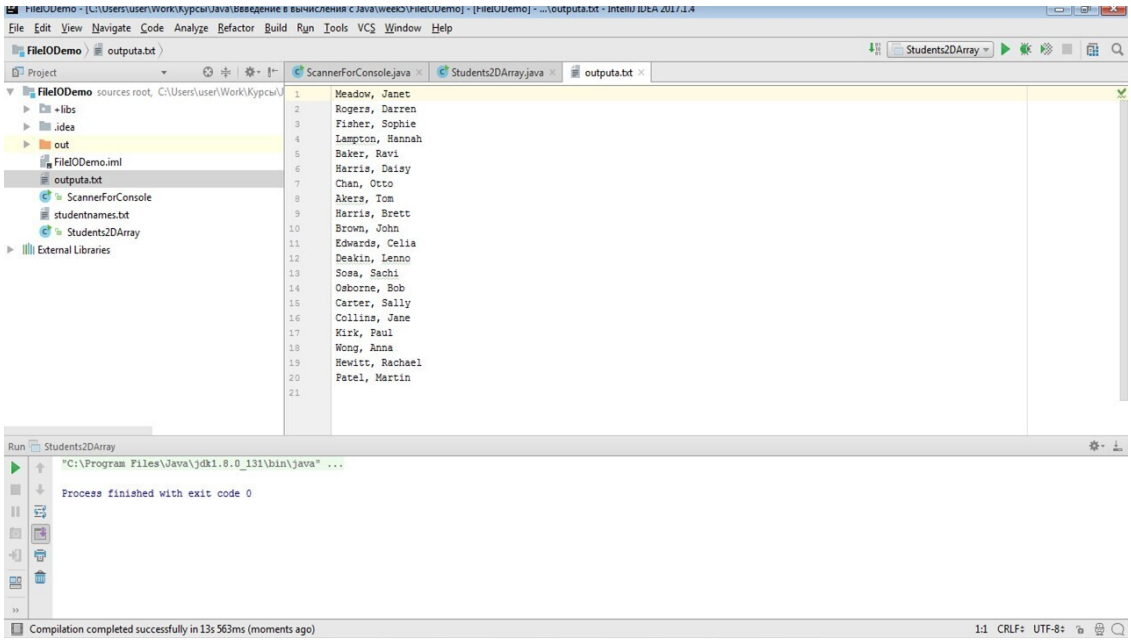
Нам нужно будет сначала вызвать метод readStudentNamesToArray(), чтобы прочитать имена в массив.

Мы не видим здесь никакого результата, потому что имена сохранены в массиве.

Затем мы вызываем метод outputNamesInArray(), чтобы вывести имена в выходной файл.

Здесь вы можете увидеть, что имя выходного файла определено как "outputa.txt".

Так что, если вы откроете файл output.txt, вы сможете увидеть результат.

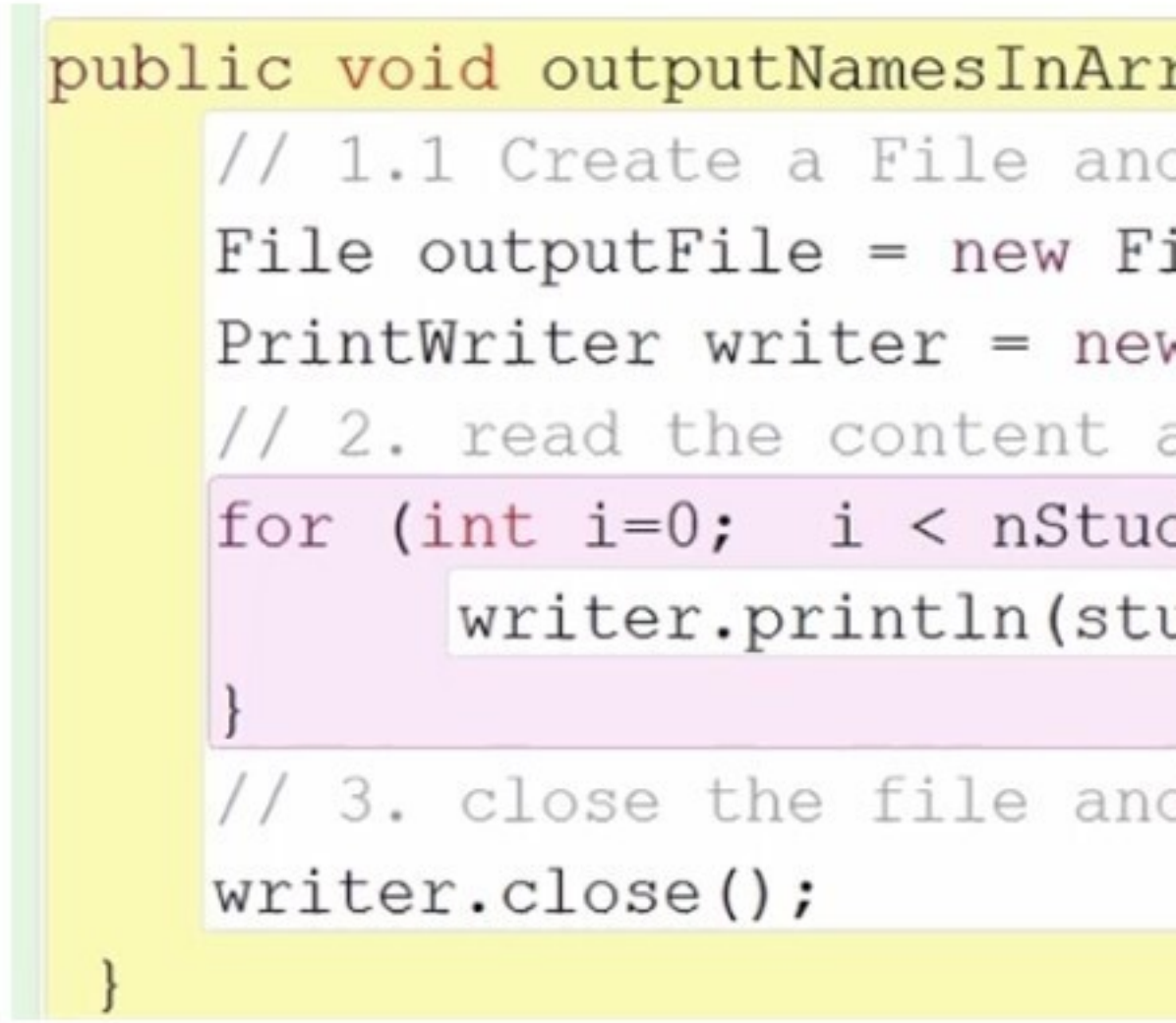


Если вы сравните это с исходным входным файлом, вы можете увидеть, что они в основном одинаковые, за исключением того, что в выходном файле фамилия отображается первой, а потом фамилия и имя разделяются запятой.

Давайте немного изменить программу.

В некоторых приложениях, вы можете отобразить фамилию заглавными буквами.

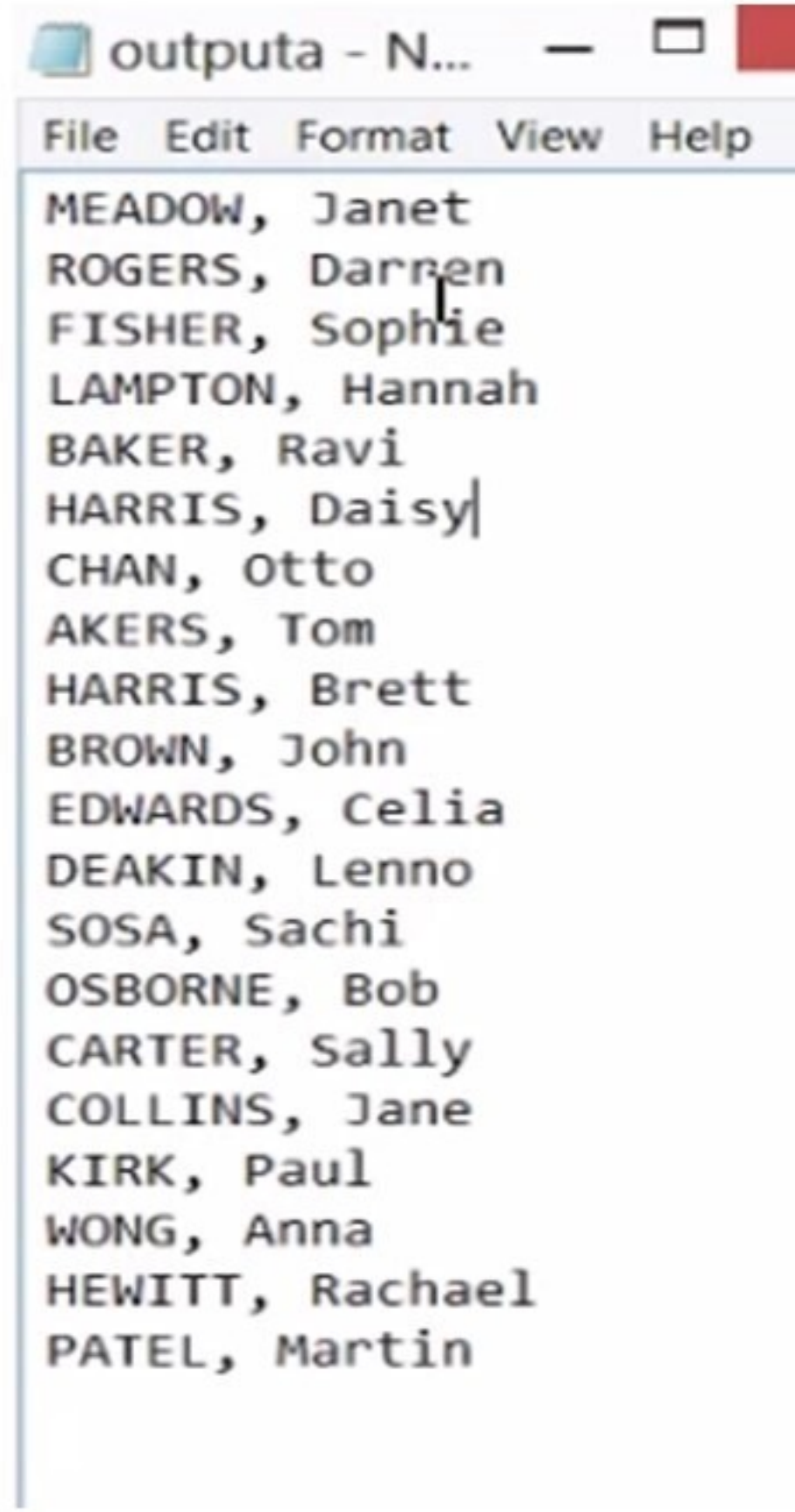
Для того чтобы сделать это, вы можете просто поставить метод toUpperCase() после элемента массива для фамилии.


public void outputNamesInArr
// 1.1 Create a File and
File outputFile = new Fi
PrintWriter writer = new
// 2. read the content a
for (int i=0; i < nStuc
writer.println(stu
}
// 3. close the file and
writer.close();
}

Помните, что это метод, так что пара скобок здесь необходима.

Если вы скомпилируете программу и запустите программу снова, во-первых, прочитаете имена в массив, а затем выходной мас-

сив в выходной файл, outputa.txt, и вы увидите этот результат.



Вопросы

Задача

Рассмотрим следующее выражение. Какой ответ из приведенных будет верным результатом?

int lenOfStr = "Is it possible?".length();

Варианты ответа:

1. Ошибка компиляции генерируется.
2. Ошибка выполнения генерируется.
3. Переменная, lenOfStr, устанавливается в 0.
4. Переменная, lenOfStr, установлена в 15

Ответ: 4.

Объяснение

"Is it possible?" является объектом String. Метод length() экземпляра класса String возвращает количество символов объекта String, включая пробелы. Значение 15 возвращается при вызове метода length() объекта String "Is it possible?". Таким образом, переменная lenOfStr устанавливается в 15.

Задача

Учитывая строковые объекты S1 и S2 ниже:

String s1 = "zzz";

String s2 = "ZZZ";

Каким будет возвращенное значение следующих сравнений?

i) int v1 = s1.compareTo("AAA");

ii) int v2 = s2.compareTo("aaa");

Варианты ответа:

1. i) v1 <0 ii) v2> 0
2. i) v1> 0 ii) v2 <0
3. i) v1> 0 ii) v2> 0
4. Ошибка компиляции.

Ответ: 2.

Объяснение

compareTo() метод экземпляра класса String сравнивает лексикографически последовательность символов, представленную текущей строкой, с символьной последовательностью, представленной другим объектом String.

Отрицательное целое число возвращается, если текущая строка лексикографически меньше другой строки. Положительное целое число возвращается, если текущая строка лексикографически больше другой строки. 0 возвращается, если две строки равны.

В первом сравнении, символ 'z' лексикографически больше символа 'A'. Таким образом, положительное значение присваивается переменной v1. Во втором сравнении, символ 'Z' лексикографически меньше символа 'a'. Таким образом, отрицательное значение присваивается переменной v2.

Задача

Завершите реализацию метода, isPalindrome(String), который берет строку в качестве параметра и возвращает true, если строка является палиндромом. В противном случае, возвращается false. Ваша реализация метода isPalindrome() не должна быть чувствительна к регистру.

Вы можете предположить, что метод reverseString(String) обеспечивается и имеет следующую реализацию.

public static String reverseString(String inputStr){

```
String revStr= "";
for(int i=0; i < inputStr.length( ); i++) {
revStr= inputStr.charAt(i) + revStr;
}
return revStr;
}
```

Для проверки метода, щелкните правой кнопкой мыши на классе PalindromeChecker и нажмите на boolean isPalindrome(String str) методе. Введите строку в двойных кавычках и нажмите Ok.

Ответ:

```
return str.equalsIgnoreCase(reverseString(str));
```

или

```
return str.toLowerCase().equals(reverseString(str).toLowerCase());
```

или

```
return str.toUpperCase().equals(reverseString(str).toUpperCase());
```

Задача

Завершите реализацию метода, searchFromRecord(String, String). Метод принимает имя файла, как первый параметр, и слово, в качестве второго параметра. Метод открывает файл записи под заданным именем и возвращает истину, если слово существует в файле записи. В противном случае, возвращается ложь. Ваша реализация метода должна учитывать разницу в регистре, то есть, сравнение слов должны быть чувствительно к регистру.

Вы можете сделать следующие предположения:

1. Запись файла всегда существует.
2. Слова в записи хранятся в отдельной строке.

Для проверки метода, нужно создать экземпляр SearchRecordDemo и запустить void searchWord() метод экземпляра. Вы можете затем вводить слово в окне терминала. Результаты поиска будут отображаться после ввода слова.

В дополнение к вводу слово, набором, мы можем также вводить слово из изображения. Метод inputTextImage() класса IO считывает изображение от пользователя, и выполняет оптическое распознавание символов (OCR) для распознавания строки в изображении.

Этот метод поддерживается только на платформе Windows, с установленным Visual C++ Redistributable для Visual Studio 2012. Вы можете скачать его здесь http://www.microsoft.com/en-us/download/details.aspx?id=30679. Пожалуйста, выберите VSU_4\vcredist_x86.exe установку, если вы используете 32-разрядную систему и выберите VSU_4\vcredist_x64.exe если вы используете 64-разрядную систему. Перезагрузите компьютер после завершения установки.

Чтобы включить OCR в программе, закомментируйте следующую строку в методе searchWord():

```
String word = IO.inputString();
```

Раскомментируйте следующую строку в блоке объявлений класса SearchRecordDemo:

```
private static Loader loader = new Loader();
```

После этого, раскомментировать следующие строки в методе searchWord():

```
String word = IO.inputTextImage();
word = removeExtraSpace(word);
IO.outputln(word);
```

Ответ:

```
File record = new File(recordName);
Scanner scanner = new Scanner(record);
String line;
while (scanner.hasNextLine()) {
```

```
line = scanner.nextLine();
if (line.equals(word)) {
scanner.close();
return true;
}
}
scanner.close();
return false;
```

Event Driven программирование

Далее я представлю вам краткое введение в управляемое событиями программирование и графический интерфейс пользователя или GUI.

Мы рассмотрим эти темы поверхностно.

Введение обеспечит вам достаточные знания, чтобы вы смогли освоить более сложные понятия.

В общем и целом, те программы, которые мы писали до сих пор, выполняют действия, следующие определенному процессуальному порядку.

То есть, выполнение осуществляется либо шаг за шагом, или управление передается другой процедуре или методу, который опять будет выполнять действия шаг за шагом.

Даже если мы здесь используем объектно-ориентированный подход, объекты, созданные с помощью программы, будут общаться друг с другом через вызовы методов, но действия будут по-прежнему следовать последовательно более или менее заранее определенными шагами.

Давайте, используем пример для дополнительной иллюстрации, что это значит.

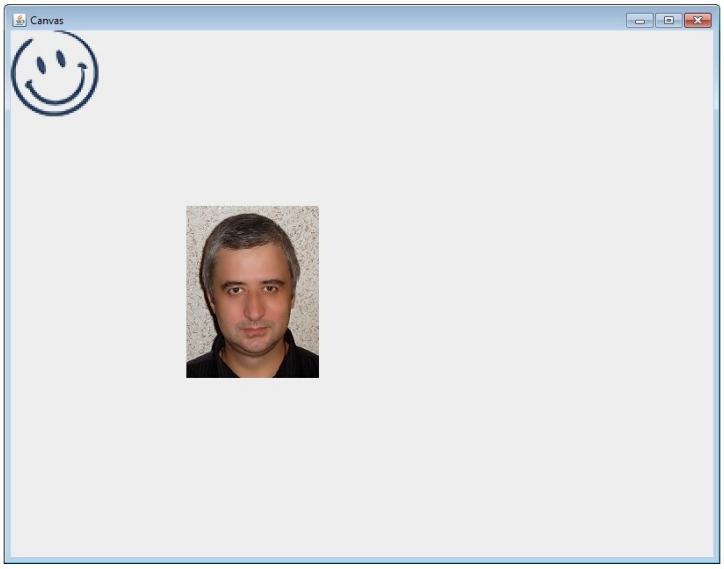
```
// A simple demo on Procedural vs Event Driven Programming
public class EventDrivenDemo
{
   private Canvas canvas = new Canvas();
   ColorImage image1 = new ColorImage("happyFace.png");
   ColorImage image2 = new ColorImage("tc.jpg");

   public EventDrivenDemo() {
      canvas.add(image2, 200, 200);
      canvas.add(image1);
   }
   public void moveHappyFace(int x, int y){
      image1.setX(x);
      image1.setY(y);
   }
}
```

В этой программе, мы используем знакомые нам классы Canvas и ColorImage.

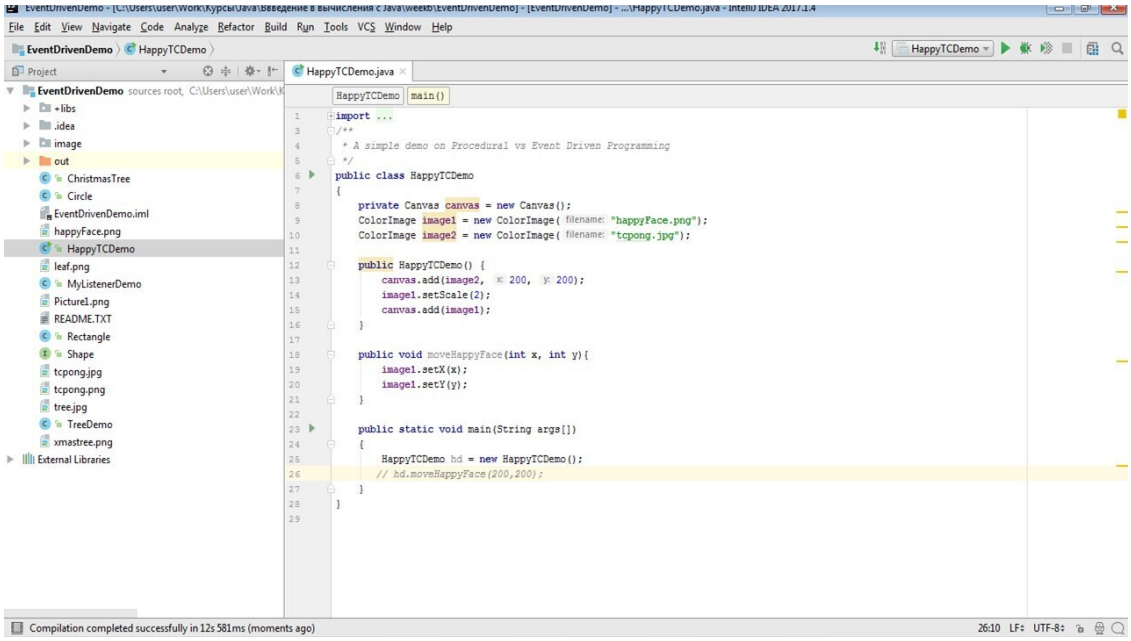
Программа просто пытается отобразить изображения happyface и tcpong на холсте.

И есть еще один метод, называемый moveHappyFace, который перемещает happyface изображение на новое место x, y.

Когда создается экземпляр класса EventDrivenDemo, изображение tcpong будет добавлено на холст с расположением 200, 200, и изображение HappyFace на позицию 0,0 по умолчанию, как показано здесь.



Откроем IDEA, чтобы увидеть, как эта программа действительно работает.



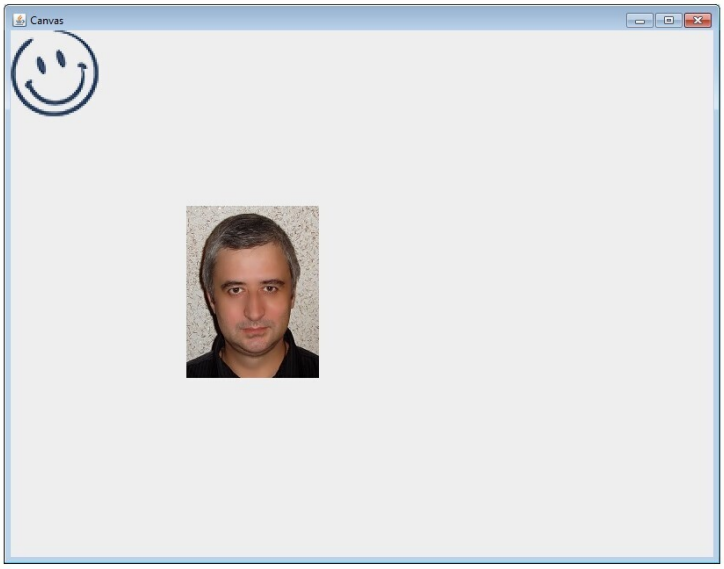
Давайте скомпилируем программу и создадим экземпляр объекта.

Два изображения отображаются на холсте, как и ожидалось.

Скажем, что если мы хотим переместить happyface изображение на верхнюю часть фотографии так, чтобы она смотрелась еще счастливее, и для этого мы можем использовать метод moveHappyFace.

Но мы не знаем точно, куда двигаться.

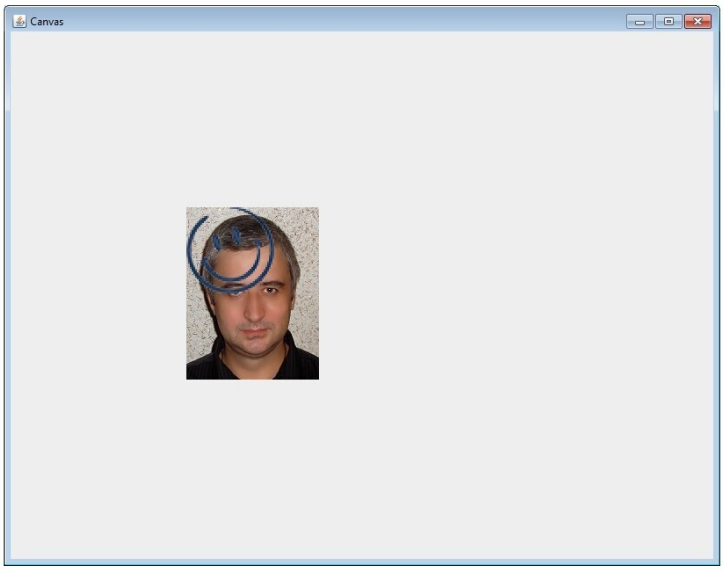
Давайте попробуем 200x200, потому что это место фотографии.

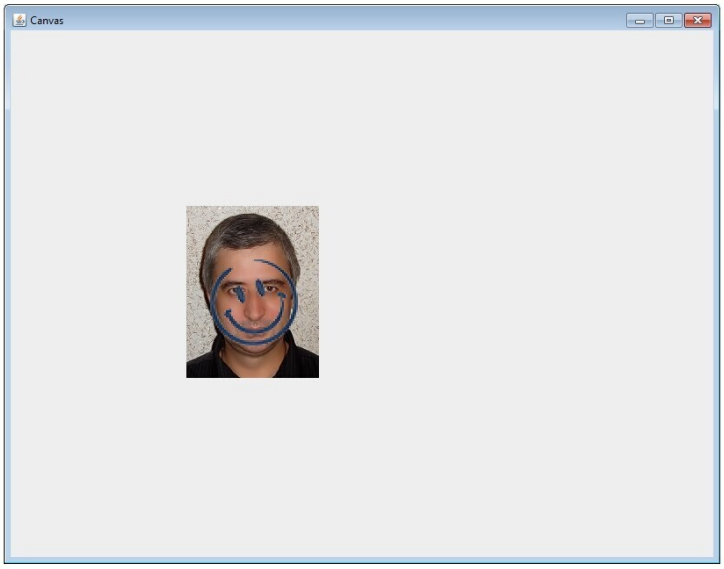


Ну, это не совсем то, что я хочу, потому что это на самом деле изображение не накладывается на лицо.

Но тогда, если вы помните, мы можем на самом деле использовать мышь, чтобы перетащить изображение.

Так что, мы можем перетащить счастливое лицо в нужное место фотографии.





В общем и целом, событийное программирование – это парадигма программирования, в которой поток программы определяется событиями, такими как клик мыши, нажатие мыши, сообщениями из другой программы, или событиями сенсорного экрана.

Событийное программирование особенно полезно в графических интерфейсах пользователя.

В примере, который я только что продемонстрировал, мы можем использовать мышь, чтобы перетащить изображение в другое место.

Перетаскивание мышью, в этом случае, является событием, созданным пользователем.

Хотя, вы можете не знать об этом, программа на самом деле работает за сценой, отслеживая движение мыши при обновлении положения happyface изображения.

Этот вид графических интерфейсов позволяет выполнять такие задачи просто.

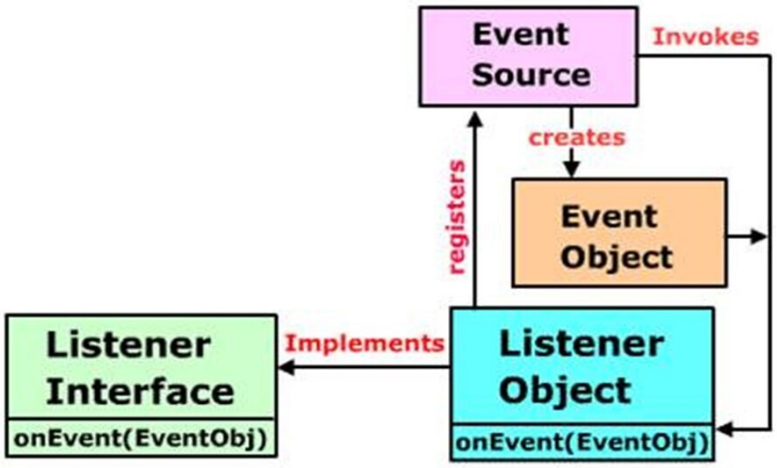
В управляемом событиями программировании, программный код выполняется при активации событий.

Событие может служить сигналом для программы, что что-то случилось.

Событие может быть сгенерировано внешними действиями пользователя, например, движением мыши, кликом мыши, и нажатием клавиши, событиями со стороны операционной системы, такими как таймер или исключение.

В нашем обсуждении, мы сосредоточим внимание на событиях, генерируемых в графическом интерфейсе пользователя.

В Java, для обработки событий используется модель делегации событий.



В модели делегации событий, событие генерируется, когда пользователь взаимодействует с графическим компонентом, например, кликая мышью по кнопке.

Графический компонент называется источником события.

После того, как событие генерируется, событие передается или делегируется другому объекту, который может обработать событие.

Эти объекты называются слушателями событий или обработчиками событий.

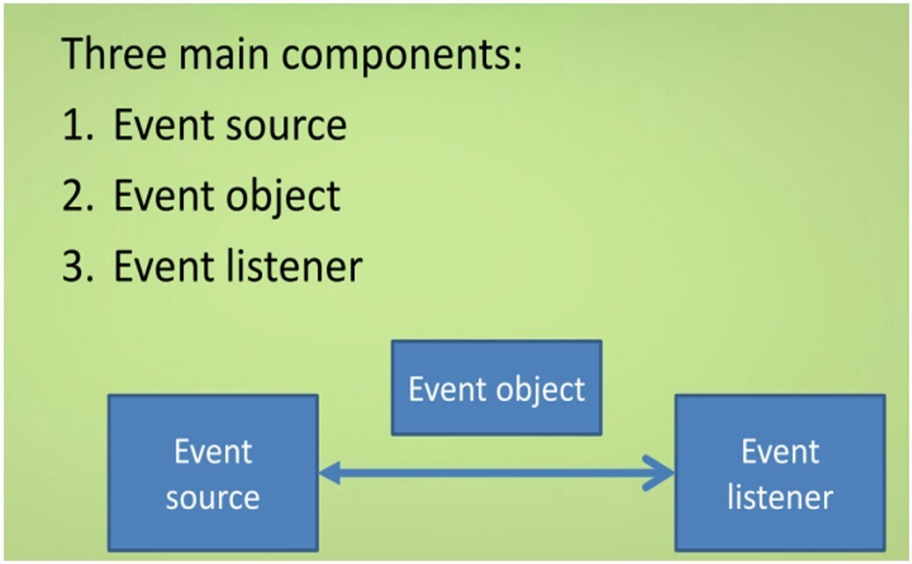
Для того чтобы получить уведомление о событии, слушатели должны быть зарегистрированы в исходном объекте.

Это эффективный способ обработки событий, потому что уведомления о событиях отправляются только тем слушателям, которые зарегистрированы.

Преимущество этого подхода заключается в том, что логика пользовательского интерфейса отделяется от логики, которая генерирует события.

Модель делегации событий состоит из трех основных компонентов, а именно источника событий, слушателя событий и объекта событий.

Диаграмма здесь показывает три компонента.



И я буду обсуждать каждый из них далее более подробно.

Источник события – это источник, из которого происходит событие.

Источник события отвечает за предоставление информации о произошедшем событии для своих обработчиков событий или слушателей.

Например, мы можем создать холст, как источник события нажатия мыши.

Объект события содержит всю необходимую информацию, в том числе об источнике события и другие данные, описывающие событие.

Например, событие нажатия мыши может включать X, Y координаты позиции мыши на холсте.

Приемник события, также известный как обработчик события, отвечает за генерацию ответов на событие.

Приемник событий ждет, пока не получает событие.

После того, как событие будет получено, слушатель запускает логику, определяющую как событие должно быть обработано.

Например, слушатель нажатия мыши может позиционировать цветное изображение в положение x, y, заданное событием щелчка мыши.

Интерфейсы

Сейчас я немного отклонюсь в сторону, потому что слушатель событий является Java интерфейсом.

Рассмотрение интерфейсов выходит за рамки этого курса, так что я обсужу эту тему очень кратко.

В основном и целом, интерфейс является группой связанных методов с пустым содержимым.

И эти методы должны быть определены любым классом, который реализует этот интерфейс.

Объявление интерфейса производится с помощью ключевого слова interface. Это похоже на объявление класса, за исключением того, что методы объявлены без тел методов.

```
public interface ActionListener {
    public void actionPerformed (ActionEvent e);
}
```

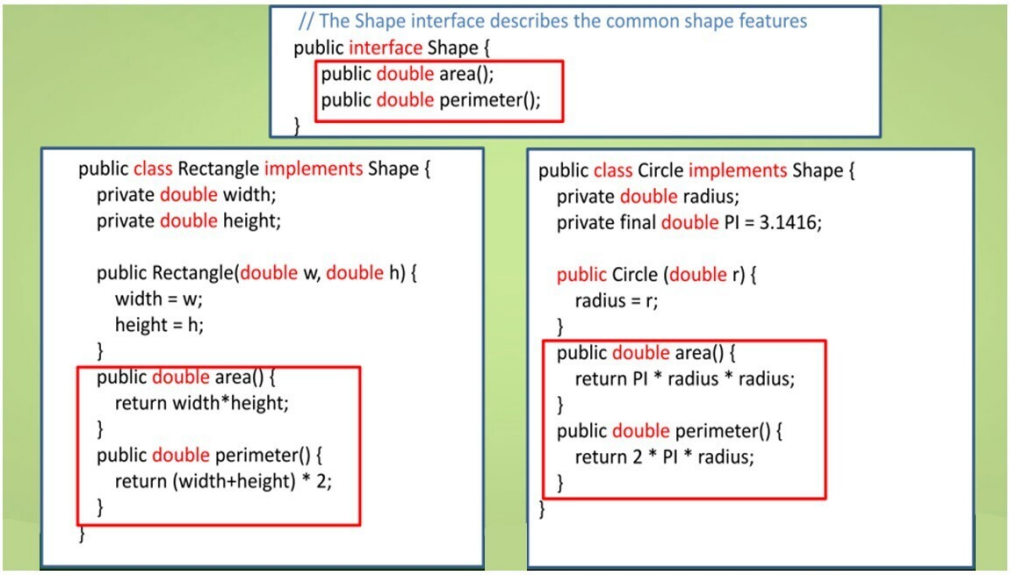
Он также не содержит никаких переменных экземпляра или статических переменных, за исключением того, что допускаются константы static final.

Здесь приведен пример, показывающий объявление для интерфейса ActionListener.

Как вы можете видеть, метод actionPerfomed объявлен с одним параметром типа ActionEvent, но тело метода отсутствует.

Обратите внимание, что даже нет пары фигурных скобок после метода.

Давайте рассмотрим другой пример интерфейса.



В этом примере, интерфейс Shape описывает функции, которые являются общими для всех форм.

Интерфейс определяет два абстрактных методов, а именно методы площади и периметра.

Эти методы должны вычислить свойства, общие для всех форм.

Но тело методов отсутствует, потому что реализация конкретного расчета может быть различной для разных форм.

Здесь реализовано вычисление для прямоугольника.

Как вы можете видеть, что тело методов площади и периметра обеспечивает конкретный расчет.

В случае площади, вы просто умножаете ширину на высоту.

Когда вы вычисляете периметр, вы складываете ширину и высоту вместе, а затем умножаете на 2.

Аналогично, реализация для круга также здесь представлена.

В этом случае площадь круга вычисляется как пи умножить на квадрат радиуса.

Аналогично также может быть создана реализация для других форм.

Модель делегации событий

Давайте вернемся к модели делегации событий и обсудим более детально, как работает этот механизм.

Предположим, у нас есть событие под названием Abc.


```
– public void AbcPerformed (AbcEvent e) {
        // handling the logic
  }
```

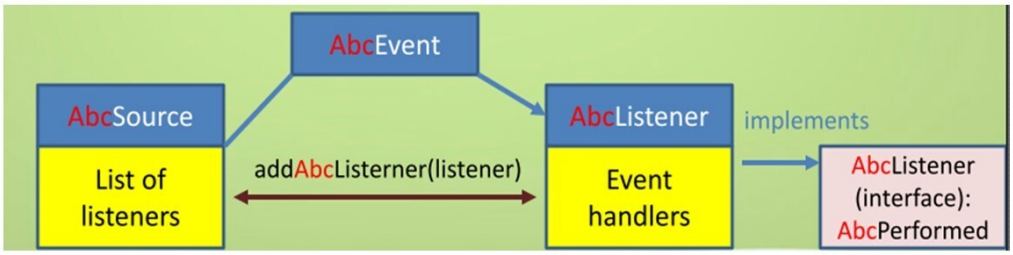
Тогда класс AbcSource будет иметь метод с именем addAbcListener с параметром типа AbcListener для объекта-источника, чтобы регистрировать своих слушателей таким образом, что может быть создан список слушателей.

Вызов этого метода позволяет классу источника знать, каких слушателей он должен уведомить, когда происходит событие.

Класс AbcListener будет иметь один или несколько методов с именем типа AbcPerformed, формируемым из типа события и вида выполняемого действия, которое принимает AbcEvent объект в качестве параметра.

Метод должен реализовать логику, каким образом это событие должно быть обработано.

Я вернусь к реализации этих методов позже.



Когда происходит событие Abc, такое как клик мыши или нажатие мыши, метод AbcPerformed будет вызываться классом AbcSource и посылать ему объект AbcEvent, который инкапсулирует всю необходимую информацию об активации.

Например, (x, y) положение курсора мыши, введенный текст, и так далее.

Все слушатели, заинтересованные в событии, должны реализовать слушатель событий.

Класс Event Listener представляет собой интерфейс Java, который содержит набор методов, которые будут реализованы.

```
class MyListener implements AbcListener {
    public void AbcPerformed(AbcEvent e) {
        // my handling logic
    }
}
```

Слушатели должны иметь свои собственные реализации для всех абстрактных методов, объявленных в интерфейсе слушателя событий.

Таким образом, слушатели смогут реагировать на эти события должным образом.

В этом примере, слушатель события или обработчик называется MyListener и заинтересован в событии Abc. Он должен реализовать интерфейс AbcListener, который содержит метод, называемый AbcPerformed.

Класс MyListener должен обеспечить реализацию внутри определения класса для метода AbcPerformed.

Давайте посмотрим на простой пример, чтобы проиллюстрировать, как работает этот механизм.

```
import comp1022p.Canvas;
import comp1022p.ColorImage;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;

public class MyListener implements MouseListener {
  private Canvas canvas;

  public MyListener ( ) {
    canvas = new Canvas();
    canvas.addMouseListener(this);
  }

  public void mouseClicked(MouseEvent e) {
    ColorImage image = new ColorImage("happyFace.png");
    int x = e.getX() - image.getWidth()/2;
    int y = e.getY() - image.getHeight()/2;
    canvas.add(image, x, y);
  }

  public void mousePressed(MouseEvent e) { }
  public void mouseReleased(MouseEvent e) { }
  public void mouseEntered(MouseEvent e) { }
  public void mouseExited(MouseEvent e) { }
}
```

В этом примере, имя слушателя событий, который мы собираемся использовать, это MyListener.

И мы заинтересованы в событии MouseEvent, в частности, в mouseClicked. Поэтому интерфейс, который мы собираемся реализовать это MouseListener.

Обратите внимание, что мы должны импортировать MouseListener и класс MouseEvent из пакета java.awt.event.

Остальные два выражения импорта для классов Canvas и ColorImage, которые мы использовали много-много раз.

В конструкторе, в дополнение к созданию холста, мы также устанавливаем холст в качестве источника событий, добавив MouseListener, потому что мы заинтересованы в событиях, генерируемых мышью.

В частности, мы заинтересованы в щелчке мыши и здесь есть метод в интерфейсе MouseListener называемый mouseClicked.

Этот метод будет вызываться, когда пользователь нажимает на кнопку мыши и информация о событии щелчка мыши инкапсулируется в параметре, который является объектом MouseEvent.

Как упоминалось ранее, методы в интерфейсе не реализованы в объявлении интерфейса.

И реализация того, как это событие mouseClicked предполагается быть обработанным, определена в этой программе.

То, что мы пытаемся сделать здесь, это просто попробовать отобразить happyFace изображение на холсте в месте, где произошел щелчок мыши.

Таким образом, мы сначала открываем happyFace изображение.

Расположение щелчка мыши можно получить из объекта события с помощью e.getX и e.getY.

Остальная часть выражения image.getWidth/2 и image.getHeight/2 только пытается произвести регулировку положения, так что изображение отображается в центре, где мышь щелкнула, а не в углу.

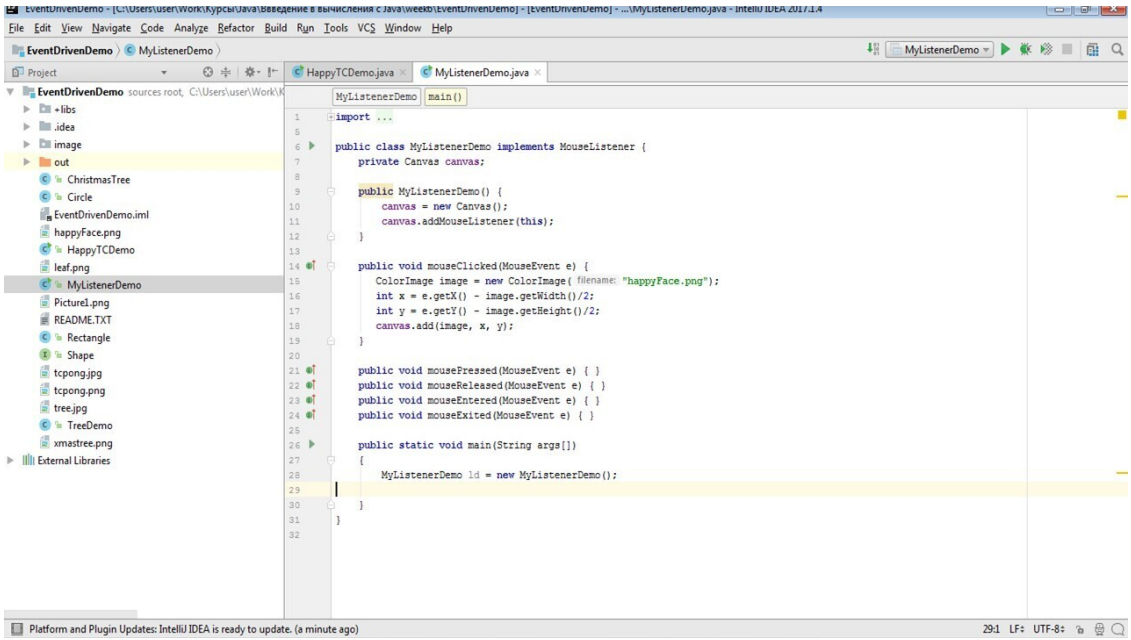
После того как нужное место получено, изображение отображается на холсте.

Кажется, что мы сделали все, что мы должны были сделать, кроме того, что, как я уже говорил, когда мы обсуждали интерфейс, все методы в интерфейсе должны быть определены в классе, который реализует интерфейс, даже если программа не собирается использовать какой-любо из них.

В случае интерфейса MouseListener, существует четыре другие методы, в дополнение к mouseClicked, а именно mousePressed и mouseReleased, фактически комбинированное действие mousePressed и mouseReleased приведет к щелчку мыши, другие методы mouseEntered и mouseExited указывают, находится ли мышь в пределах или за пределами области источника событий, в данном случае, на холсте.

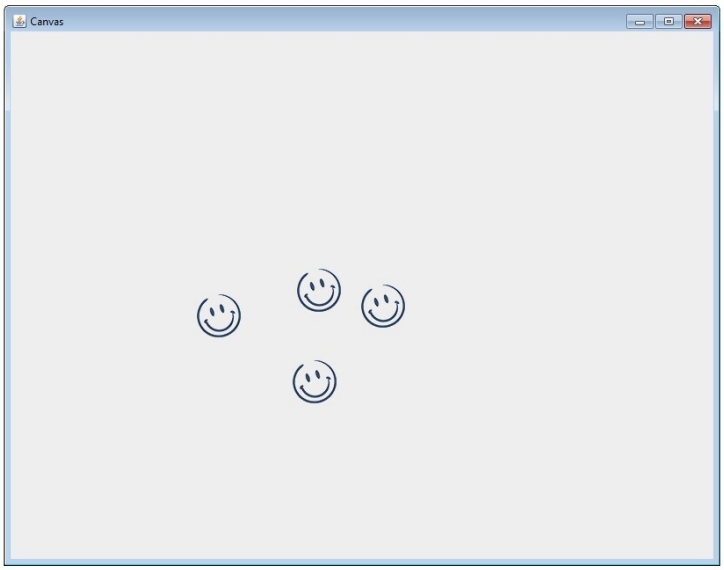
Демонстрация Event Driven программирования

Давайте теперь откроем среду разработки, чтобы посмотреть как работает эта программа. Мы откроем программу MyListenerDemo.



Это в основном то же, что мы видели ранее.

И если мы создадим экземпляр MyListenerDemo, мы можем увидеть, что на экране отображается холст.



И если вы будете просто нажимать мышью на холсте, то вы можете увидеть, что изображение happyface отображается с расположением в центре положения мыши.

Вы можете перемещать мышь в любое место, но без нажатия на кнопку мыши, ничего не будет отображаться на холсте.

Изображение happyface будет отображаться, только если щелкнуть мышью.

Я хотел бы показать вам еще один пример.

Я сначала покажу вам то, что, как предполагается, эта программа должна делать, прежде чем я покажу вам исходный код.

Мы снова создадим экземпляр программы, и эта программа называется TreeDemo.

При запуске программы, вы сможете увидеть, что отображается дерево.

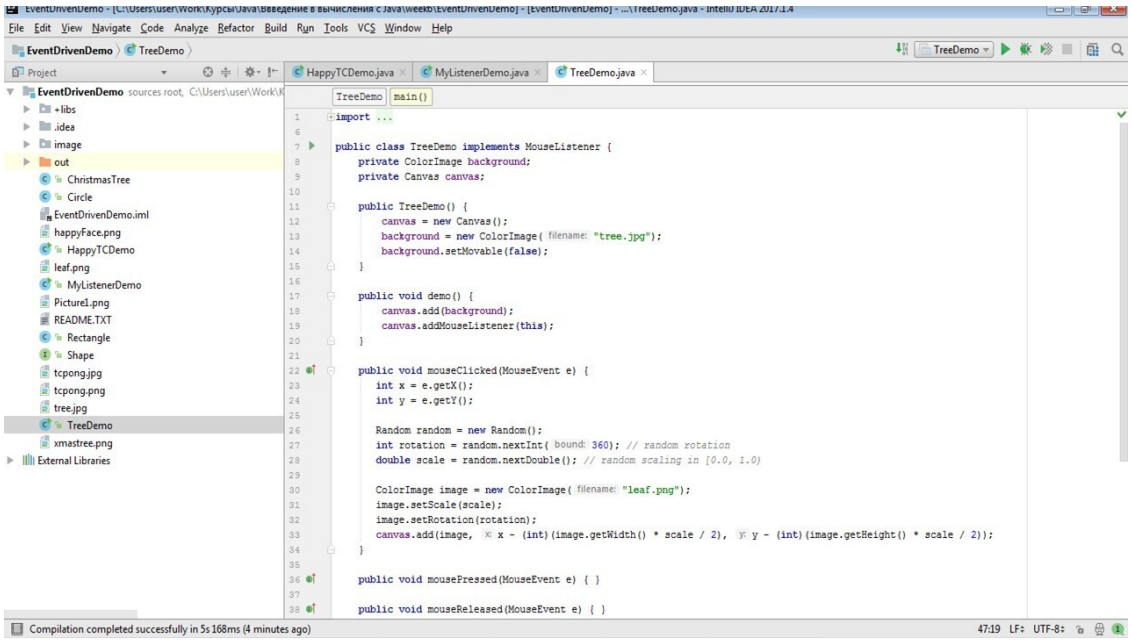
Опять же, когда вы нажимаете на холсте, вы обнаружите, что отображаются кленовые листы, в том месте, где вы кликнули мышью.

Если вы кликаете на другом месте, вы видите, что там отображается другой лист, и в этом случае лист ориентирован по-другому, а также его размер отличается от того, который был показан ранее.

И вы можете повторить этот процесс, нажимая на различных местах на холсте.

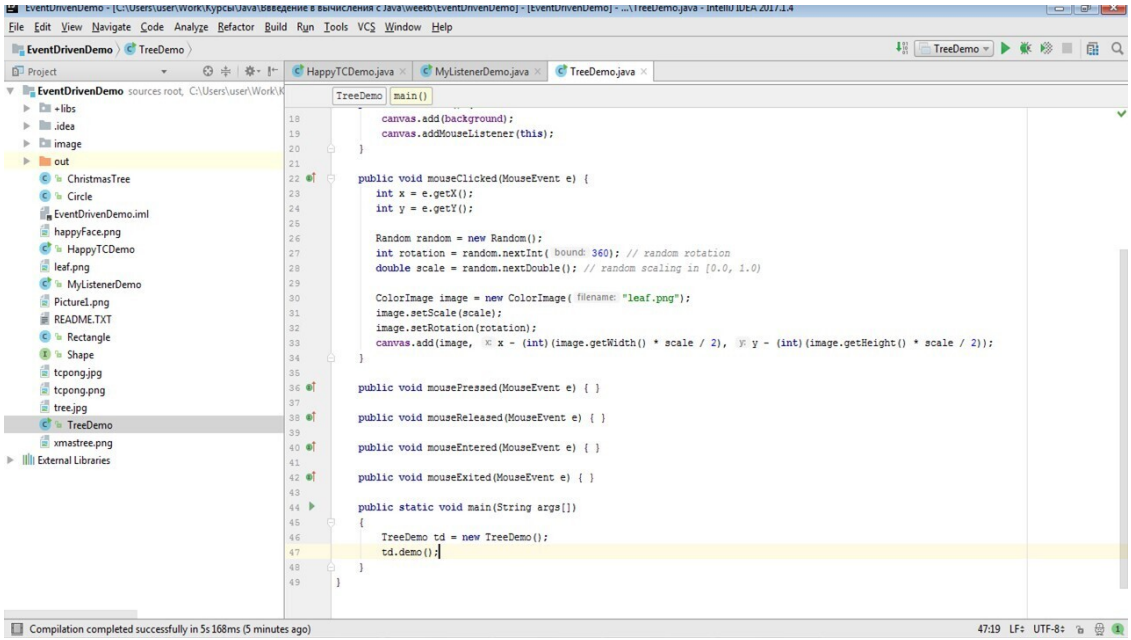
Вы можете видеть, что на холсте отображаются различные размеры кленовых листьев, с различной ориентацией.

Теперь мы посмотрели, что эта программа должна делать. И давайте посмотрим, как это реализовано в программе.



Здесь мы должны включить класс, который вы уже использовали и который называется Random, потому что, как вы могли видеть, листья ориентируются с помощью генератора случайных чисел. Размер листьев также рандомизирован.

И вы можете видеть, что мы открываем файл tree.jpg, и мы также открываем другой файл изображения под названием leaf.png в рамках метода mouseClicked.



Таким образом, вы можете видеть, что мы также добавили MouseListener к холсту, потому что холст, опять же, считается источником события.

Как я уже говорил, мы используем генератор случайных чисел для случайного расположения, а также масштаба листа, который будет отображаться на холсте.

После того, как мы сформировали местоположение и масштаб, мы просто устанавливаем масштаб, и мы устанавливаем местоположение для изображения кленового листа.

Метод mouseClicked отображает изображение в соответствии с генерируемым расположением.

Вы можете видеть, что мы также осуществляем корректировки в расположении, так что кленовые листья отображаются в центре расположения клика мыши.

Мы также определили методы mousePressed, mouseReleased, mouseEntered, и mouseExited. Хотя мы на самом деле не должны использовать их в программе. В этом случае мы можем просто объявить пустое тело для этих методов.

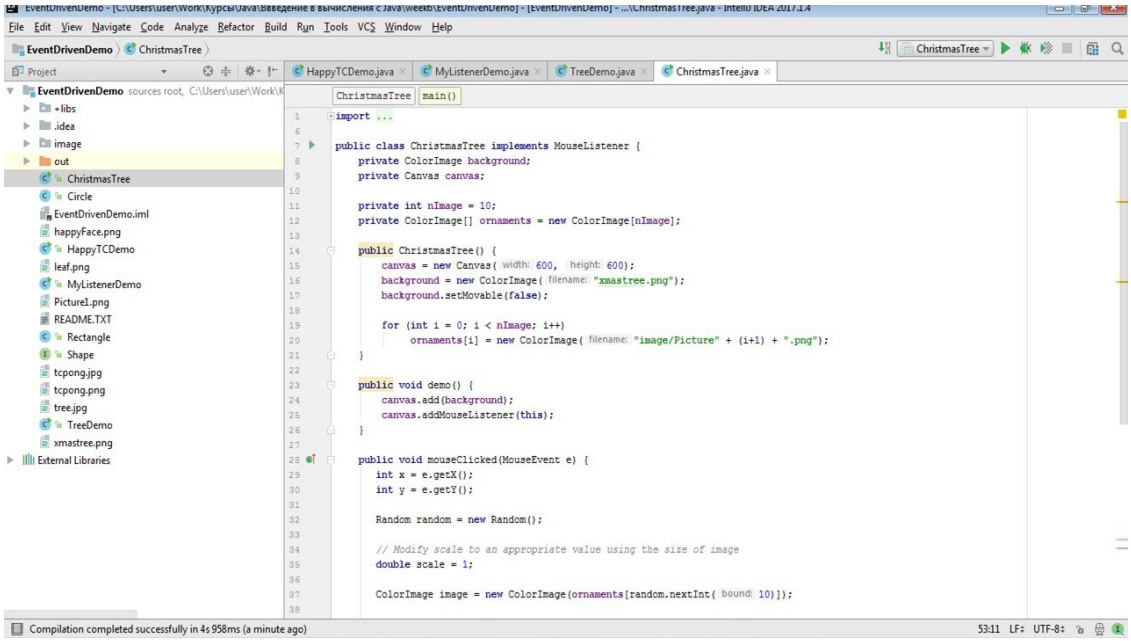
Основная работа ведется в методе mouseClicked, который определяется в интерфейсе MouseListener.

Таким образом, эта программа позволяет расти деревьям в любое время в течение года.

Можно также легко модифицировать эту программу так, что различные объекты могут быть добавлены к фоновому изображению.

Хотя сейчас не Новый Год, все же, будет приятно создать свою собственную елку.

Давайте изменим эту программу, чтобы создать класс ChristmasTree.

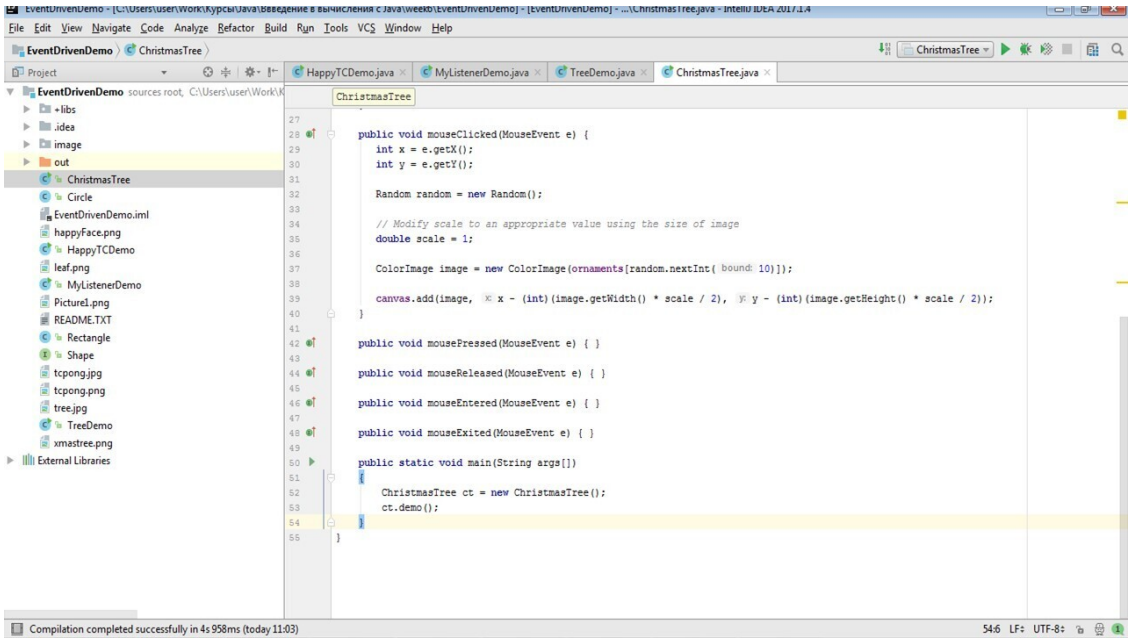


Здесь нас есть набор елочных украшений.

И мы можем создать массив для хранения этих украшений, предполагая, что у нас есть десять елочных украшений.

А потом мы можем инициализировать этот массив для хранения этих украшений.

Мы также заменим наше предыдущее изображение дерева на изображение рождественской елки под названием xmastree.png.



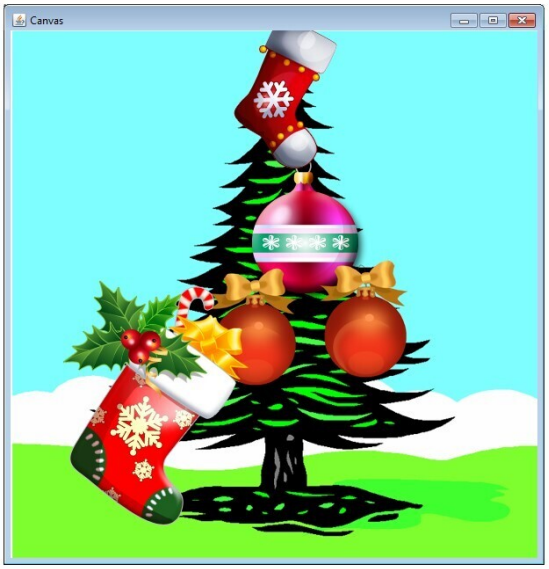
В этой реализации mouseClicked, для каждого щелчка мышью, выбирается случайное украшение из массива, так что нам все еще нужен здесь Random класс.

И здесь нет необходимости изменять ориентацию.

Но мы не знаем, что делать с размером, поэтому просто установим пока масштабирование в 1, и увидим, что произойдет.

Давайте скомпилируем программу и запустим программу путем создания экземпляра, и вызовем метод demo.

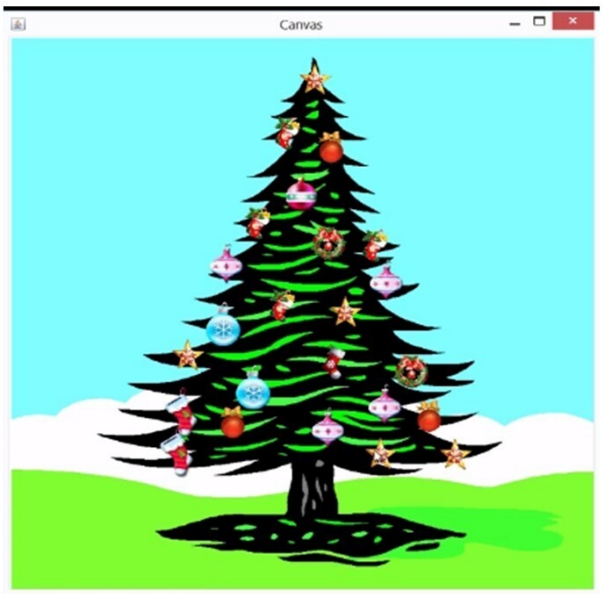
Вы видите, что отображается Рождественская елка и давайте попробуем добавить украшения для елки.



Мы видим, что с масштабированием что-то не так.

Так что подумайте о том, что вы могли бы сделать с масштабированием, чтобы все украшения были уменьшены до некоторого соответствующего размера перед добавлением в холст.

Здесь я изменил программу, так, как это должно примерно выглядеть.



Графический интерфейс пользователя

Трудно понимать событийное программирование без некоторого понимания как работает графический интерфейс пользователя, или GUI.

Так что сейчас я дам вам краткое введение в GUI.

Мы все должны быть знакомы с графическим пользовательским интерфейсом, современный графический интерфейс управляется событиями в том смысле, что он реагирует на действия пользователя через графические компоненты.

Например, пользователь может нажать на кнопку, сказать да или нет, и программа будет затем выполнять некоторые соответствующие действия.

Мы можем позиционировать различные графические объекты, скажем изображения, перетаскивая их с помощью мыши.

Когда мы заполняем форму, текст вводится в текстовое поле, и программа будет сохранять информацию в соответствующем месте в документе.

Вы можете делать выбор в выпадающем меню, чтобы нам не нужно было запоминать все доступные опции.

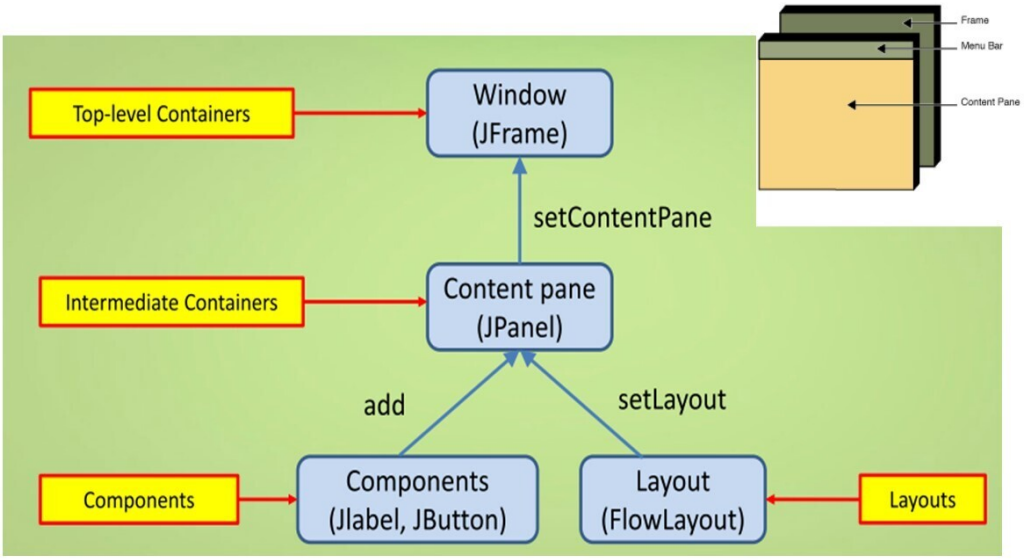
Когда мы работаем с несколькими окнами, мы часто должны минимизировать их или изменять их размер так, чтобы мы могли сосредоточиться на окне, с которым мы работаем, и закрывать их, если они вам больше не нужны.

Кроме того, с мобильными устройствами, мы можем сделать еще больше, например, изменить размер окна с помощью жестов на мультисенсорном экране, и др.

Существует много чего, что можно сделать с помощью графического интерфейса.

Что я буду делать здесь, так это просто дам вам общее понимание GUI для того, чтобы вы лучше понимали событийное программирование.

Эта диаграмма дает обзор общей организации структуры GUI.



Контейнерами верхнего уровня графического интерфейса являются окна, например, JFrame.

Каждое окно имеет два промежуточных контейнера, которые обычно используются, это меню бар, содержащее меню, и панель содержания, которая содержит остальные компоненты.

Промежуточный контейнер, например, JPanel, может содержать компоненты графического интерфейса пользователя, такие как текстовое поле и кнопки, которые будут помещены в окне.

Эти компоненты являются компонентами управления пользовательского интерфейса, такими как кнопки, JButton, метки, JLabel, и текстовые поля, JTextField.

Их добавляют в контейнер методом add().

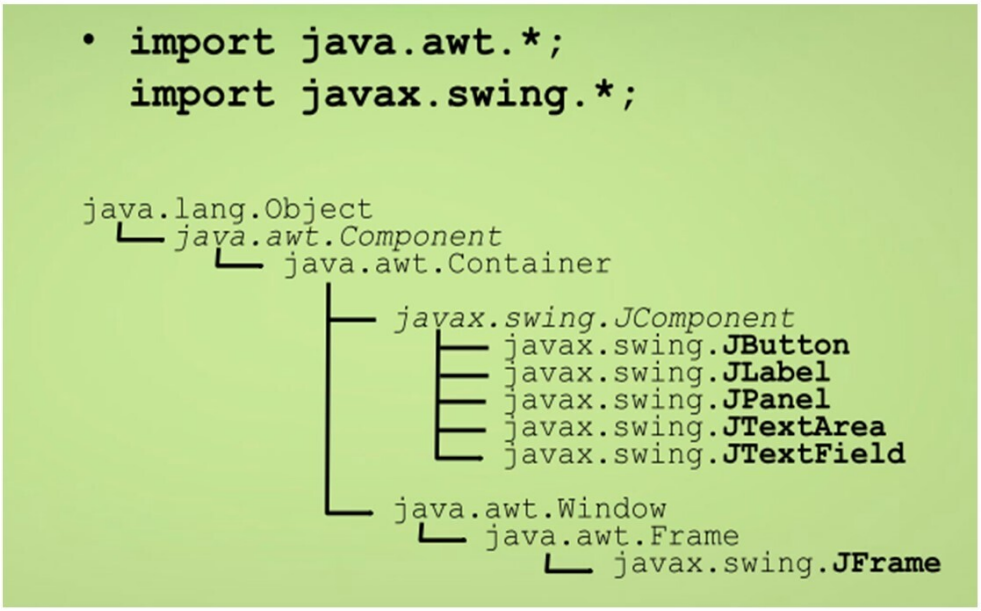
Макеты, иногда называемые менеджерами компоновки, определяют, как разместить и установить размер компонентов в JPanel или других промежуточных контейнеров.

Каждый контейнер JPanel начинается с менеджера компоновки по умолчанию, но лучше установить его явно.

FlowLayout является одним из наиболее часто используемых менеджеров компоновки.

Как вы можете видеть, эти графические компоненты организованы в иерархическую структуру.

В Java, классы, определяющие эти компоненты также организованы в некоторую иерархическую структуру, и они содержатся в двух пакетах, а именно java.awt и javax.swing.



Обратите внимание, что пакет swing идет под javax с «x» в конце, а не java, как в awt.

Библиотека AWT или abstract windowing toolkit является первоначальным набором классов и методов, который был разработан SUN microsystem для мультиплатформенного GUI.

Библиотека AWT была самой первой графической библиотекой платформы Java.

Далее библиотека AWT была дополнена библиотекой Java 2D двухмерной графики и изображений.

Библиотека AWT обеспечивает для разработчика приложений возможность использования таких основных компонентов GUI-интерфейса как кнопка, флажок, список выбора, прокручивающийся список, метка, диалоговое окно, окно выбора файла, меню, панель с прокруткой, текстовая область и текстовое поле, использования функции Drag-and-Drop, возможность обработки событий UI-компонентов, компоновки элементов GUI в рабочей области, работы с цветом, шрифтом, графикой, рисования и печати.

Библиотека AWT считается тяжеловесной, так как она содержит нативную библиотеку java.awt.peer, через которую взаимодействует с операционной системой компьютера таким

образом, что AWT-компоненты имеют своих двойников, реализованных для конкретной операционной системы, с которыми они связаны интерфейсами пакета java.awt.peer.

Поэтому отображение AWT GUI-интерфейса зависит от операционной системы, в которой приложение развернуто.

Для расширения набора GUI-компонентов библиотеки AWT, устранения ее тяжеловесности и возможности выбора внешнего вида и поведения (Look and Feel) GUI-компонентов была создана графическая библиотека Swing.

Библиотека Swing основывается на библиотеке AWT и напрямую не связана, как библиотека AWT, с операционной системой, в которой она работает.

Поэтому библиотека Swing является уже легковесной.

Кроме того, библиотека Swing дополняет библиотеку AWT такими компонентами GUI-интерфейса как панель выбора цвета, индикатор состояния, переключатель radio button, слайдер и спиннер, панель с закладками, таблицы и деревья, расширенными возможностями компоновки GUI-компонентов, таймером, возможностью отображения HTML-контента.

С помощью библиотеки Swing стало возможным создать набор отображений Look and Feel, которые разработчик может выбирать, не оглядываясь на операционную систему, и изменять внешний вид GUI-интерфейса.

Также библиотека Swing реализует архитектуру MVC (Model-View-Controller) и потоковую модель Event Dispatch Thread (EDT).

Список здесь показывает, как эти классы организованы в иерархическую структуру, с помощью подклассов и отношений суперкласса.

Например, JFrame является подклассом класса Frame, и Frame это подкласс класса Window.

И JButton, JLabel, JPanel и JTextField все подклассы класса JPanel.

Если вы откроете Javadoc этих классов, вы сможете получить более подробную информацию об отношениях подкласс-суперкласс этих графических компонентов.

Например, вы можете увидеть, что JFrame это подкласс Frame, который является подклассом класса Window.

Класс JPanel это подкласс JComponent, который является подклассом Container и далее до корневого класса Object.

Подкласс – это класс, который является производным от другого класса, который называется суперкласс.

Ключевое слово "extends" используется при объявлении подкласса.

Мы использовали примеры BankAccount, SavingsAccount и CheckingAccount, чтобы проиллюстрировать идею отношений подкласс-суперкласс.

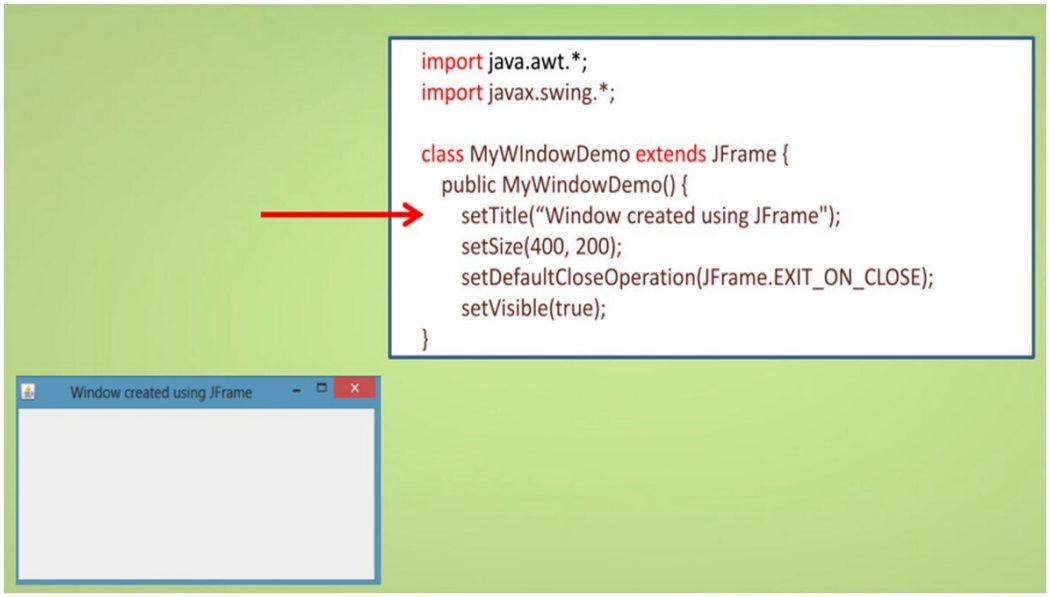
Здесь оба класса SavingsAccount и CheckingAccount являются подклассами класса BankAccount.

Наследование является важным понятием в объектно-ориентированном программировании, и это позволяет подклассам наследовать все поля и методы от своего суперкласса без необходимости переписывать их снова.

Пример GUI

Я буду использовать здесь несколько примеров для иллюстрации того, как GUI можно определить, а затем использовать Event Driven программирование в графическом интерфейсе.

MyWindowDemo это простая программа, которая определяется как подкласс класса JFrame.



Единственное, что можно найти в теле MyWindowDemo, так это объявление конструктора.

Я покажу вам, что конструктор должен создать, прежде чем углубляться в детали.

Как вы можете увидеть, это окно, похожее на те, которые вы часто видите при запуске окна на компьютере.

Вы видите синюю рамку вокруг окна.

В верхней части окна находится панель заголовка вместе со знакомыми кнопками.

В частности, кнопками, чтобы максимизировать или минимизировать окно.

А также красная кнопка с крестом внутри, что позволит вам закрыть окно.

Итак, используя JFrame, все это может быть создано для вас автоматически, без необходимости писать собственную программу.

Давайте теперь вернемся к конструктору.

В конструкторе метод setTitle обеспечит окну название, которое будет отображаться в панели заголовка окна, как вы можете здесь увидеть.

Метод setSize установит размер окна, в этом случае, это 400 пикселей в ширину и 200 пикселей в высоту.

Таким образом, ширина этого окна в два раза больше его высоты.

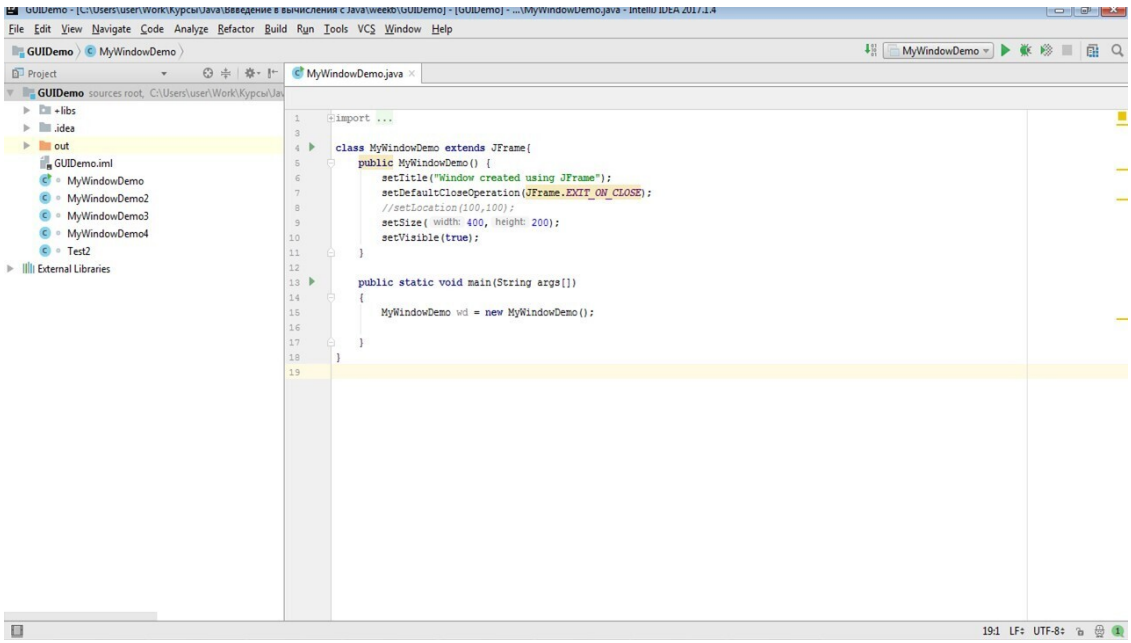
Метод setDefaultCloseOperation использует параметр JFrame.EXIT_ON_CLOSE.

Это выглядит немного сложно, но все, что он делает, это просто останавливает программу, когда окно закрывается, когда нажимается крест в верхнем правом углу кнопки.

Метод setVisible сделает окно видимым.

Все эти методы доступны из JFrame, так как MyWindowDemo это подкласс JFrame, и мы можем просто использовать эти методы при создании конкретного объекта.

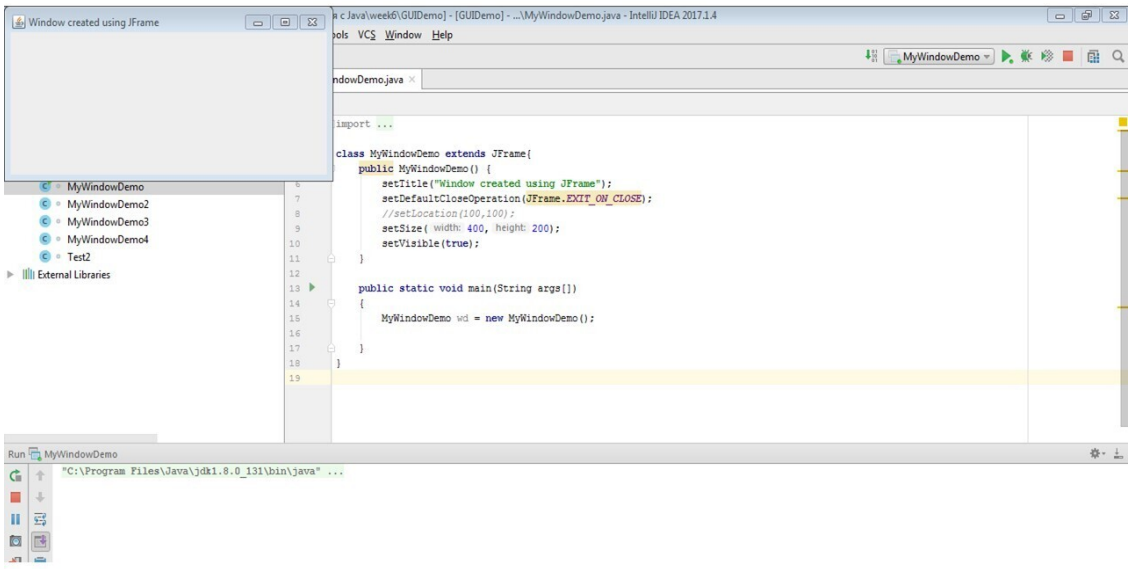
Откроем сейчас IDEA, чтобы продемонстрировать, как работает эта программа.
Откроем класс MyWindowDemo.



Как вы можете видеть, это программа, которую мы только что описали.

Скомпилируем и запустим программу путем создания экземпляра, и вы сможете увидеть, что окно создается в верхнем левом углу.

Вы можете использовать другие методы, например, setLocation, если вы не хотите поместить окно в углу, но в определенном месте в середине, скажем в позиции 200, 200.



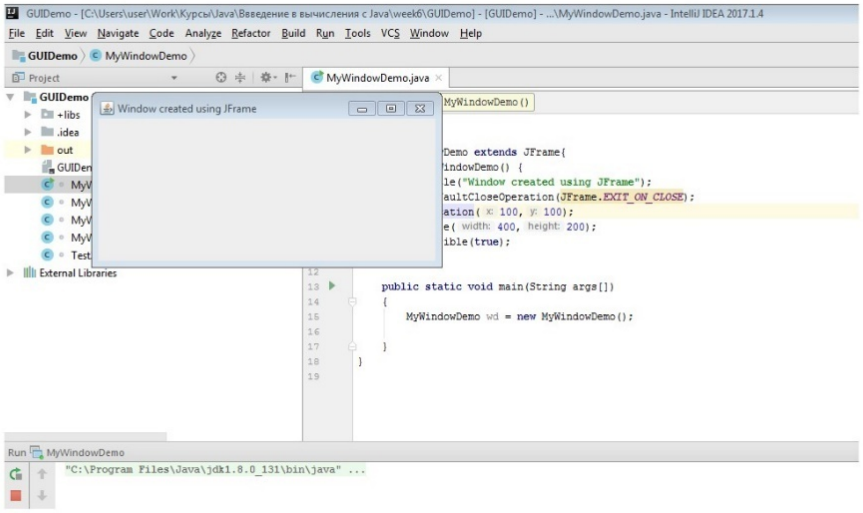
Если вы скомпилируете и запустите программу путем создания экземпляра, теперь вы сможете увидеть, что окно вместо позиции в углу, находится сейчас на позиции 200, 200.

Когда вы закроете окно, как я уже говорил, потому что мы используем EXIT_ON_CLOSE, вы сможете увидеть, что программа также остановилась.

Предыдущий пример просто создает пустое окно.

Чтобы сделать его более интересным, давайте добавим несколько элементов в окно.

Сначала мы создадим JPanel с именем content, чтобы мы могли поместить некоторые графические компоненты в панель.



Когда мы добавляем компоненты на панель, так или иначе мы должны сказать компьютеру, где поместить эти компоненты.

Было бы довольно утомительно определять позиционирование компонентов, один за другим.

К счастью, платформа Java предоставила некоторые макеты по умолчанию так, чтобы компоненты могли быть размещены упорядоченным образом.

Мы можем использовать метод setLayout, чтобы сообщить программе, какой макет используется.

В этом случае, менеджер компоновки будет использовать макет FlowLayout, который просто расставляет компоненты в ряд, масштабируя в их размер по умолчанию.

Если недостаточно свободного места, чтобы разместить все компоненты в текущей строке, менеджер компоновки будет создавать другую строку.

То есть, макет будет размещать компоненты слева направо, а затем сверху вниз.

Есть и другие стратегии компоновки, и вы можете найти их легко в Интернете.

Первый компонент, который мы здесь создаем, это некоторый текст, используя класс JLabel и переменную label.

Здесь мы используем строку “My Panel” для маркировки панели.

Метка затем может быть добавлена к содержанию панели с помощью метода add.

Вторым компонентом является кнопка.

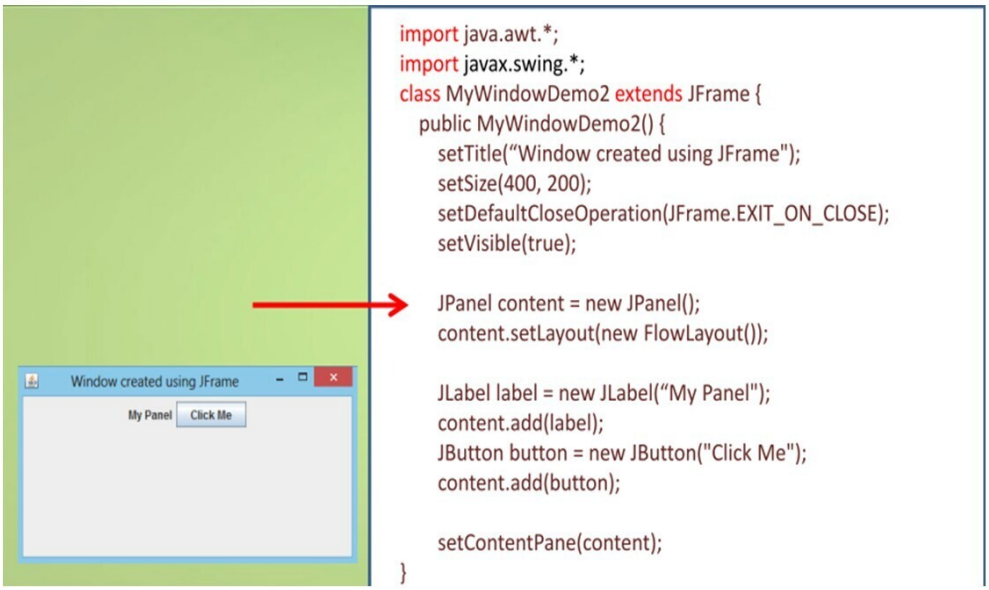
Кнопка создается как тип JButton и кнопке дается метка "Click Me".

Опять же, кнопка добавляется к панели content.

Панель содержимого JFrame, в этом случае установлена как панель JPanel, которую мы только что создали, с помощью метода setContentPane.

Это результирующее окно, которое мы увидим, если создадим экземпляр окна.

Откроем IDEA, чтобы посмотреть, как работает программа.



Здесь вы можете видеть, что это то же самое, что мы только что описали.

Скомпилируем и запустим программу.

Создадим экземпляр класса MyWindowDemo2,

И вы можете увидеть, что это окно в основном такое же, как и раньше, за исключением того, что сейчас у нас есть текст My Panel, добавленный к окну, и есть также кнопка, которая называется Click Me.

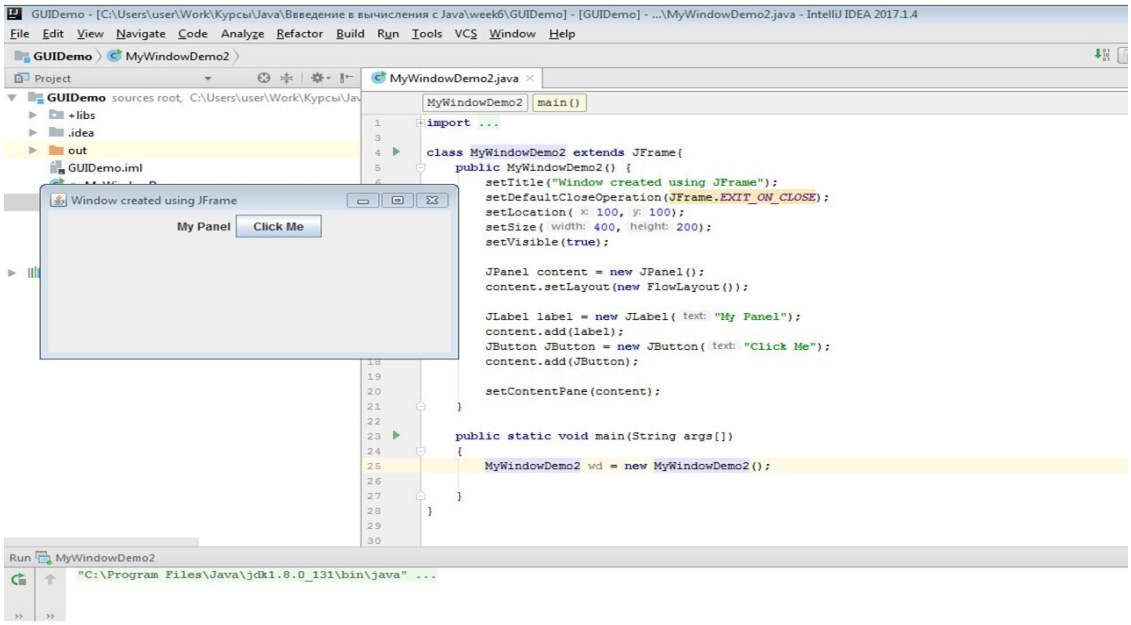
Но теперь, если вы попытаетесь нажать на эту кнопку, ничего не случится.

Потому что мы не добавили к ней слушателей, как мы обсуждали в предыдущих примерах событийного программирования.

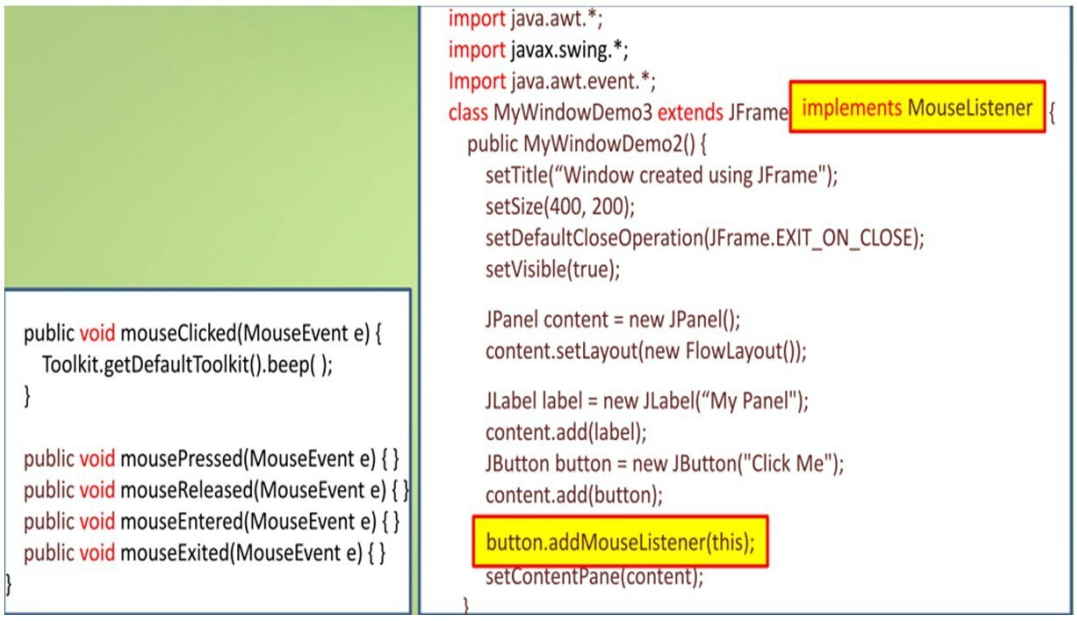
До сих пор, кнопка или что-то еще в окне не взаимодействовали с пользователем, потому что не было слушателя, присоединенного к чему-либо в окне.

Скажем, если мы хотим, чтобы кнопка что-то сделала, когда пользователь нажимает на нее.

При подключении кнопки к событию MouseClick, кнопка будет рассматриваться в качестве источника события.



Мы используем метод addMouseListener, чтобы зарегистрировать этот объект для источника событий кнопки.



Так что можно было бы получать уведомления, когда событие щелчка мыши происходит в кнопке.

Для того чтобы определить, что делать, когда мышь нажата, мы должны реализовать интерфейс MouseListener.

Следует помнить, что интерфейс состоит из набора абстрактных методов, которые еще не были реализованы.

Все методы в интерфейсе должны быть реализованы в классе.

В этом примере, мы сделаем что-то простое, просто проигрывание звукового сигнала, когда мышь нажимает на кнопку.

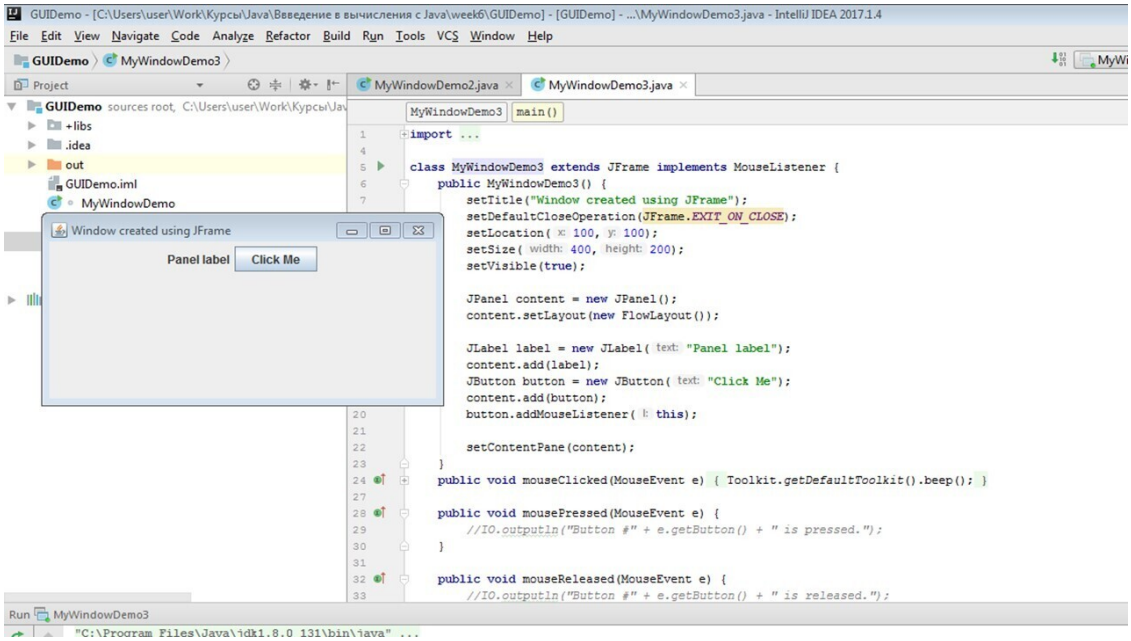
Таким образом, в реализации mouseClicked, метод beep(), который доступен в классе Toolkit, обеспечивает звуковые сигналы.

И мы ничего не делаем в других методах, включая mousePress, mouseRelease, MouseEnter и mouseExited.

Но они должны быть реализованы, даже если они ничего не делают, в противном случае программа не будет скомпилирована.

Конечно, вы можете выполнить другие какие-либо действия, если вы хотите, в любом из этих методов.

Давайте опробуем эту программу в IDEA.



Создадим экземпляр класса MyWindowDemo3.

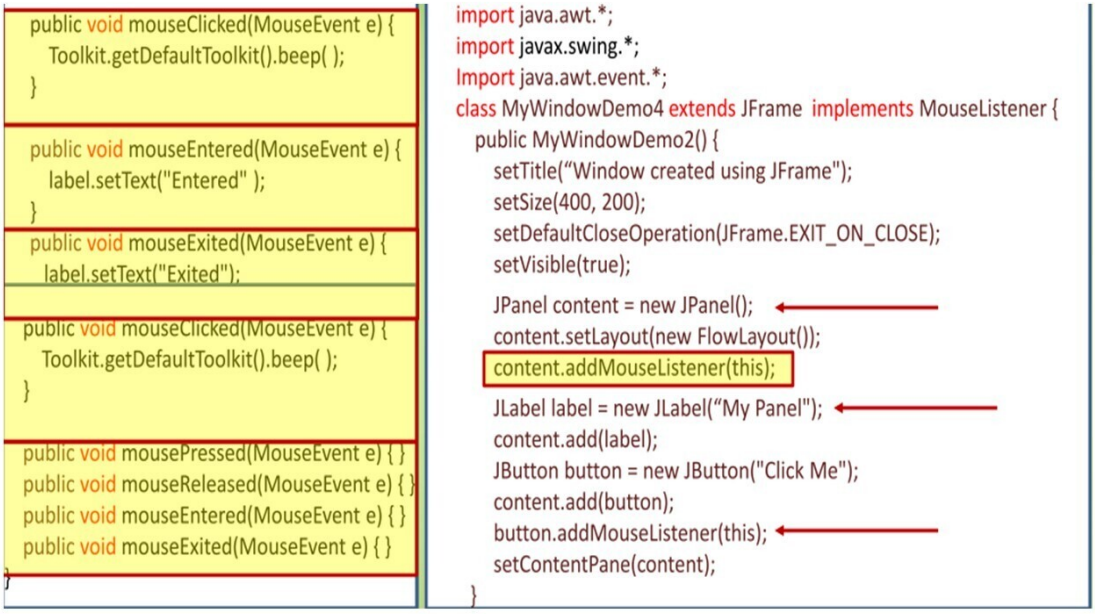
И здесь то же самое окно, которое мы уже видели, и вы можете видеть, что здесь реализован метод mouseClicked.

Для других методов, ничто не было реализовано.

Когда вы создаете объект этого класса, как вы можете видеть, это окно выглядит точно так же, как и раньше, но теперь, если вы нажмете на эту кнопку Click Me, вы можете услышать звук.

Но если вы щелкните в любом месте за пределами кнопки, вы не слышите звуковой сигнал.

В программе, которую мы только что видели, это был только один источник событий, то есть, кнопка "Click me".



```
public void mouseClicked(MouseEvent e) {
   Toolkit.getDefaultToolkit().beep( );
}

public void mouseEntered(MouseEvent e) {
   label.setText("Entered" );
}
public void mouseExited(MouseEvent e) {
  label.setText("Exited");

public void mouseClicked(MouseEvent e) {
   Toolkit.getDefaultToolkit().beep( );
}

public void mousePressed(MouseEvent e) { }
public void mouseReleased(MouseEvent e) { }
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }
}
```

```
import java.awt.*;
import javax.swing.*;
Import java.awt.event.*;
class MyWindowDemo4 extends JFrame  implements MouseListener {
  public MyWindowDemo2() {
    setTitle("Window created using JFrame");
    setSize(400, 200);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
    JPanel content = new JPanel();
    content.setLayout(new FlowLayout());
    content.addMouseListener(this);
    JLabel label = new JLabel("My Panel");
    content.add(label);
    JButton button = new JButton("Click Me");
    content.add(button);
    button.addMouseListener(this);
    setContentPane(content);
  }
```

Этот новый класс MyWindowDemo4 похож на тот, который вы только что видели, но вместо создания кнопки "Click me" в качестве единственного источника событий, давайте еще больше расширим сферу применения программы, таким образом, чтобы JPanel, которую мы создали, с именем content, также использовалась в качестве источника событий.

Поэтому здесь, при добавлении content.addMouseListerner, мы регистрируем content как другой источник событий для событий мыши.

В предыдущем примере, хотя все методы в MouseListener были реализованы, только метод mouseClicked на самом деле делал какую-то работу, то есть, выдавал звуковой сигнал.

Попробуем также реализовать другие методы, чтобы сделать что-то простое.

Метод mouseClicked останется таким же, чтобы издавать звуковой сигнал.

Метод mouseEntered вызывается, когда мышь входит в источник события, и здесь мы вызовем метод setText для объекта label.

Объект label объявляется как JLabel, который может быть использован для отображения какого-либо текста, и здесь мы инициализируем его символьной строкой "My Panel".

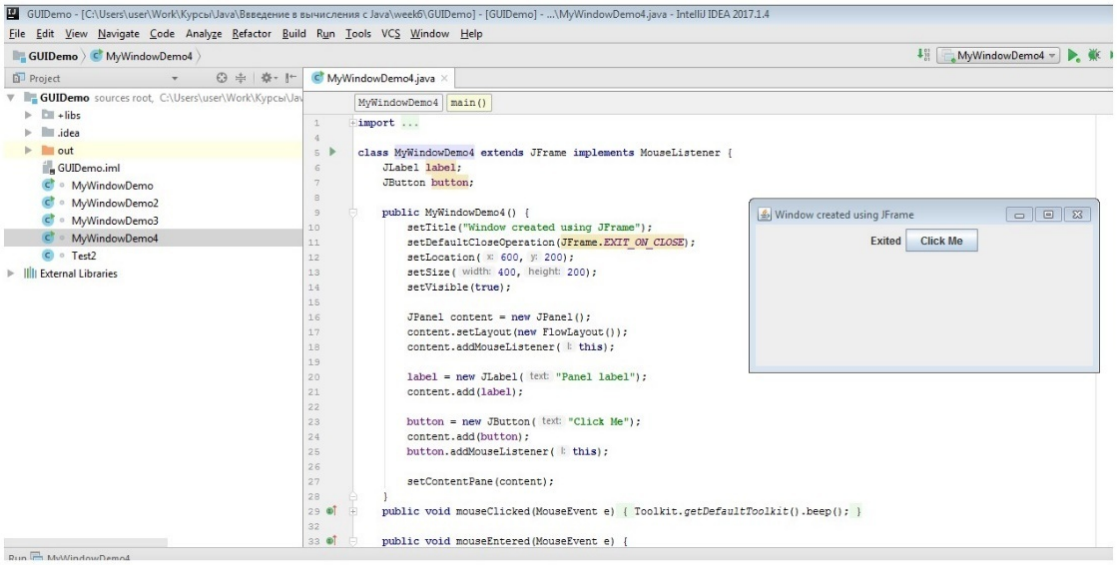
В методе mouseEntered setText изменяет метку на новую строку "Entered".

Точно так же, реализуем метод mouseExited, который вызывается, когда мышь выходит из источника, так чтобы изменить метку на строку "Exited".

Для метода mousePressed, который запускается при нажатии на кнопку мыши, кроме указания, что мышь нажата, реализация будет также определять место, где была нажата мышь, с помощью методов getX и getY для объекта события мыши, который является параметром метода, а затем отобразим информацию в метке.

Метод mouseReleased, который срабатывает, когда пользователь отпускает кнопку мыши, реализован таким же образом, за исключением того, что сообщение указывает, что кнопка мыши была отпущена.

Таким образом с помощью этой программы вы можете наглядно посмотреть различия между вызовами методов mouseClicked, mousePressed и mouseReleased.



Это класс MyWIndowDemo4 что мы только что описали.

Как вы можете видеть, панель контента и кнопка нажми меня, настроены как источники событий.

Пять методов реализованы, как это описано выше, а именно, метод mouseClicked, метод mouseEntered и метод mouseExited, а также mousePressed и mouseReleased.

Создадим экземпляр класса MyWindowDemo4.

Окно, которое здесь отображается, выглядит точно так же, как и раньше.

Сначала метка "Panel label" отображается рядом с кнопкой Click me, когда мышь вне окна.

Помните, что у нас сейчас есть панель как источник событий, которая является белой областью в окне, и кнопка "Click me" как источник событий.

Теперь, когда мышь перемещается внутри окна, метка меняется на "Entered".

Когда мышь перемещается из окна, метка теперь отображается как "Exited".

Давайте перейдем в окно снова, и нажмем кнопку мыши, и отображается сообщение "Pressed at x y".

Отпустим кнопку, и вы сможете увидеть сообщение "Released at".

Пока я испытал четыре метода на панели, а именно, mouseEntered, mouseExited, mousePressed и mouseReleased.

Давайте также протестируем оставшийся метод mouseClicked.

Если я нажимаю на кнопку мыши, можно увидеть, что она нажата, например, в позиции 166 80, и, если вы отпустите ее в той же самой позиции, вы должны также слышать звуковой сигнал.

Так происходит, если мышь нажата, а затем отпущена в том же месте на экране.

В предыдущем демо, мышь генерировала звуковой сигнал только при щелчке мышью на кнопке Click me, а теперь будет реакция на щелчок мыши в любом месте в панели, так как панель также настроена в качестве источника событий.

Давайте переместим мышь в кнопку, и вы можете увидеть, что метка будет изменена на "Entered".

Это потому, что кнопка также настроена в качестве источника событий для одного и того же слушателя мыши.

Но когда мышь перемещается за пределы кнопки, метка отображается по-прежнему, как "Entered" вместо "Exited", потому что мышь вернулась в панель, которая также источник событий.

Давайте переместимся в кнопку еще раз, и ничего не изменилось, потому что вы вошли в область кнопки.

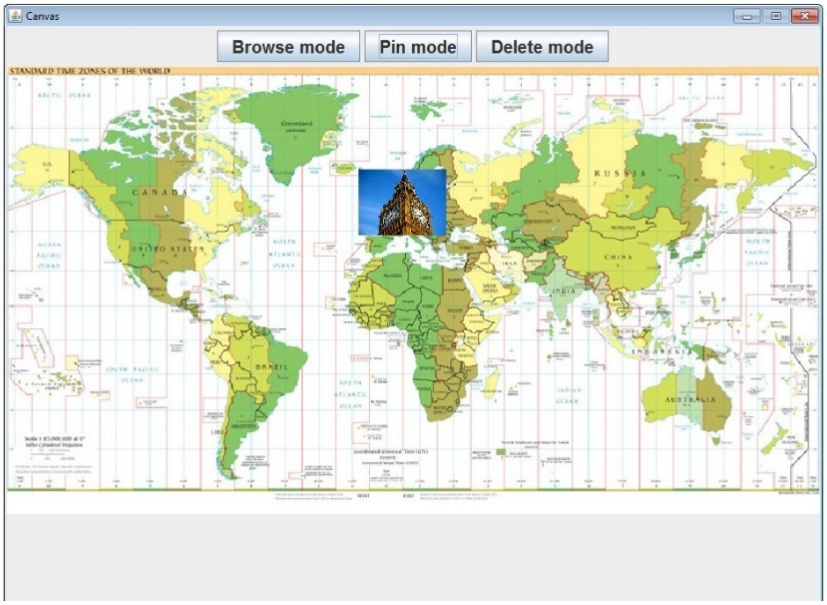
Теперь, если вы щелкните внутри кнопки, не перемещая мышь, вы все еще можете услышать звуковые сигналы, но интересно то, что место, где была нажата кнопка мыши, отличается от того места, где мышь была отпущена.

Несмотря на то, что мышь фактически не перемещается.

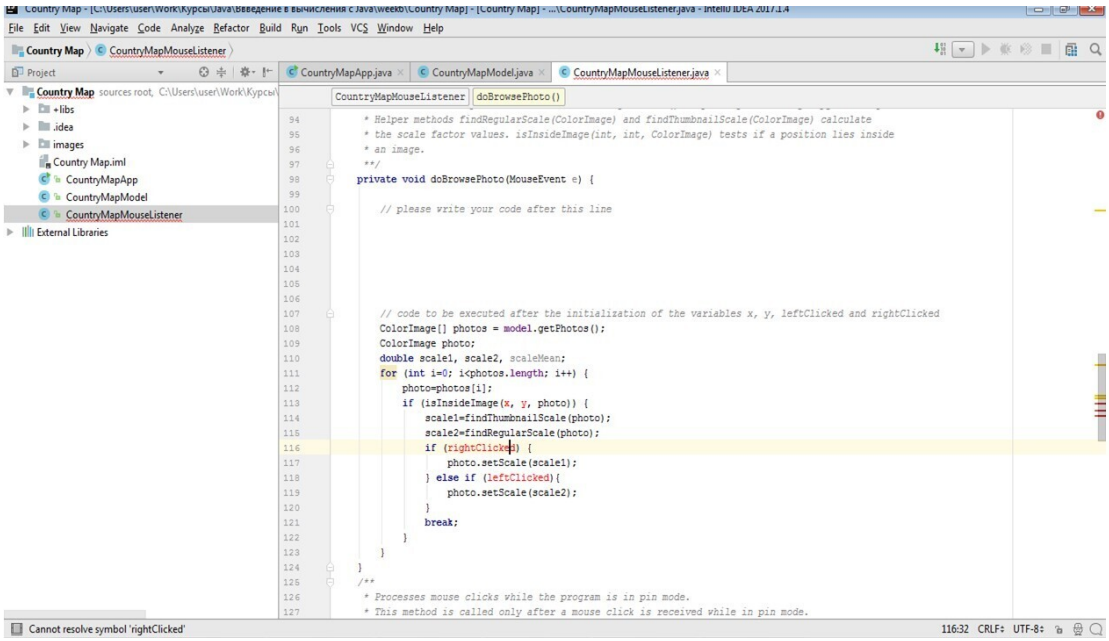
Это потому, что расположение кнопки Click me было перенесено, когда отображаемая строка символов имеет разную длину.

Упражнение

Сейчас вы увидите программу Java, которая была разработана с помощью Event Driven программирования как простое приложение, которое позволит вам закреплять и просматривать фотографии на карте.



После того как вы скачали шаблон проекта, вы можете открыть проект CountryMap.



Здесь есть три класса, и вы должны завершить класс CountryMapMouseListener.

Вы увидите, что в дополнение к использованию MouseListener в классе CountryMapMouseListener, интерфейсы ActionListener и PropertyChangeListener используются в классе CountryMapApp.

Мы не рассматривали их в этой лекции.

Но если вы заинтересованы в получении дополнительной информации, вы должны быть в состоянии самостоятельно изучить их на основе знаний, которые вы уже получили.

Если вы посмотрите класс CountryMapMouseListener, вы увидите, что здесь есть реализации методов mouseClicked, mouseEntered, mouseExited, mousePressed и mouseReleased для интерфейса MouseListener.

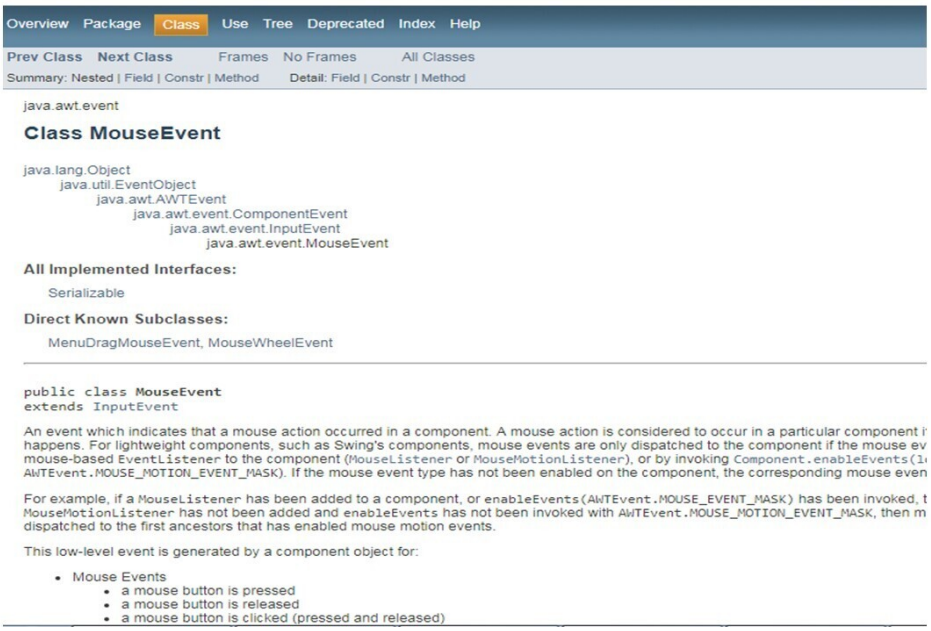
То, что вы должны сделать, это завершить реализацию метода doBrowsePhoto в классе CountryMapMouseListener.

Это упражнение научит вас на практике извлекать информацию из объекта события, который инкапсулирует информацию о событии, в этом случае, событии мыши.

Мы говорили о getX и getY методах класса MouseEvent, но вы могли бы получить больше информации из объекта MouseEvent.

Чтобы завершить реализацию метода doBrowsePhoto, в дополнение к получению позиции, где произошло событие мыши, вы должны также выяснить, действительно ли было выполнено нажатие мыши правой или левой кнопкой.

Подробнее информацию о классе MouseEvent можно получить здесь.



Overview Package Class Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.awt.event

Class MouseEvent

java.lang.Object
  java.util.EventObject
    java.awt.AWTEvent
      java.awt.event.ComponentEvent
        java.awt.event.InputEvent
          java.awt.event.MouseEvent

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

MenuDragMouseEvent, MouseWheelEvent

```
public class MouseEvent
extends InputEvent
```

An event which indicates that a mouse action occurred in a component. A mouse action is considered to occur in a particular component i
happens. For lightweight components, such as Swing's components, mouse events are only dispatched to the component if the mouse ev
mouse-based EventListener to the component (MouseListener or MouseMotionListener), or by invoking Component.enableEvents(1
AWTEvent.MOUSE_MOTION_EVENT_MASK). If the mouse event type has not been enabled on the component, the corresponding mouse even

For example, if a MouseListener has been added to a component, or enableEvents(AWTEvent.MOUSE_EVENT_MASK) has been invoked, t
MouseMotionListener has not been added and enableEvents has not been invoked with AWTEvent.MOUSE_MOTION_EVENT_MASK, then m
dispatched to the first ancestors that has enabled mouse motion events.

This low-level event is generated by a component object for:

- Mouse Events
  - a mouse button is pressed
  - a mouse button is released
  - a mouse button is clicked (pressed and released)

Это веб-страница документации для класса MouseEvent.

Большинство мышек, имеют по крайней мере две кнопки, правую кнопку или левую кнопку, но некоторые мыши имеют третью кнопку или колесо.

Вы сможете увидеть из Javadoc документации, что они представлены как константы полей BUTTON1, BUTTON2 и BUTTON3, где BUTTON1 является левой кнопкой и BUTTON3 правой кнопкой.

В Javadoc документации, вы также сможете найти метод getButton, который позволит определить, у какой кнопки мыши изменилось состояние.

Вопросы

Задача

Рассмотрим модель делегации событий для события щелчка мыши. Метод mouseClicked(MouseEvent e) является методом, определенным в …

Варианты ответа:

1. класс источника события
2. класс объекта события
3. класс слушателя событий
4. как класс источника событий, так и класс слушателя события
5. как класс объекта события, так и класс слушателя события

Ответ: 3.

Объяснение.

Методы обработки события определяются в классе слушателя событий. Для события щелчка мыши, метод mouseClicked() определен в классе слушателя, который реализует интерфейс MouseListener.

Задача

Учитывая, что слушатель мыши зарегистрирован в объекте источника, который из следующих методов зарегистрированного слушателя мыши вызывается после щелчка кнопки мыши на объекте источника?

1. public void mousePressed(MouseEvent e)
2. public void mouseReleased(MouseEvent e)
3. public void mouseEntered(MouseEvent e)
4. public void mouseExited(MouseEvent e)
5. public void mouseClicked(MouseEvent e)

Варианты ответа:

1. 5
2. 1 and 2
3. 1, 2 и 5
4. Все

Ответ: 3.

Объяснение

Событие щелчка мыши включает в себя два действия, нажатие кнопки мыши и отпускание. Поэтому, когда мы нажимаем на исходном объекте, метод mousePressed() вызывается первым, а затем mouseReleased() метод. Наконец, метод mouseClicked() вызывается.

Задача

В этом упражнении вы должны завершить реализацию метода doBrowsePhoto() в классе CountryMapMouseListener. По окончании, прикрепленное фото можно масштабировать левой кнопкой мыши и масштабировать обратно до исходного размера, щелкнув правой кнопкой мыши.

Инструкции:

1. Метод doBrowsePhoto() принимает объект MouseEvent в качестве параметра. Используйте методы экземпляра класса MouseEvent для инициализации четырех различных переменных в соответствии с приведенными ниже шагами, используя Javadoc класса MouseEvent.

2. Создать int локальную переменную x, и инициализировать ее в x положение события мыши.

3. Создать int локальную переменную y, и инициализировать ее в y позицию события мыши.

4. Создать логическую локальную переменную leftClicked, и инициализировать ее в true значение, если щелкается левая кнопка мыши во время события. В противном случае, инициализировать ее false. Советы: левая кнопки мыши щелкается, если метод getButton() экземпляра возвращает значение равное статической постоянной BUTTON1 в классе MouseEvent.

5. Создать логическую локальную переменную rightClicked, и инициализировать ее в true значение, если правая кнопка мыши щелкается во время события. В противном случае, инициализировать ее false. Советы: правая кнопка мыши щелкается, если метод getButton() экземпляра возвращается значение равное статической постоянной BUTTON3 в классе MouseEvent.

Инструкции запуска программы:

1. Чтобы запустить программу, создайте экземпляр класса CountryMapApp и вызовите метод main.

2. Программа работает в трех режимах, Browse mode, Pin mode и Delete mode. Программа в режиме Browse mode по умолчанию.

3. Чтобы добавить фотографию, нажмите кнопку режима Pin mode и нажмите на местоположение на карте. Изображение, выбранное в выпадающем диалоге, будет добавлено в местоположение на карте.

4. Чтобы изменить масштаб фото, нажать кнопку Browse mode. Щелкните левой кнопкой мыши на фото для его масштабирования и щелкните правой кнопкой мыши на фотографии для масштабирования до исходного размера.

5. Чтобы удалить фотографию, нажмите кнопку Delete mode. Щелкните левой кнопкой мыши на фото, чтобы удалить его.

Рекурсия

Теперь мы обсудим новую тему, которая называется рекурсия.

Когда мы решаем задачи, мы часто пытаемся разложить задачу на подзадачи.

В объектно-ориентированном программировании, мы стараемся реализовать эти подзадачи, как методы в классах.

Такой подход часто называют разделяй и властвуй.

Такого рода стратегии решения проблем довольно широко используются в решении повседневных жизненных задач.

Это обычная практика, когда мы не можем решить проблему сами, мы пробуем просить других о помощи.

Спрашивая помощь у других, вы можете сделать решение задачи легче.

В программировании, такой подход, как вызов другой функции или метода следует подобной идеи.

То есть, метод может вызывать другие методы, чтобы сделать часть работы в решении большой задачи.

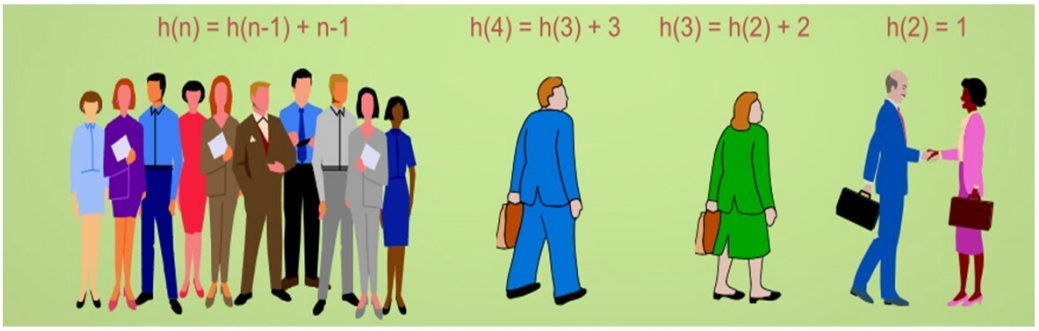
Тем не менее, как правило, когда вы просите о помощи, вы просите помочь других.

Очень редко, вы просите себя, потому что вы, как правило, обращаетесь за помощью только тогда, когда вы не можете решить проблему сами.

Но в программировании, метод может вызывать себя самого, а в некоторых случаях, это может быть естественным способом что-то сделать.

Метод, вызывающий сам себя, реализует основную идею рекурсии.

Чтобы проиллюстрировать идею рекурсии, давайте посмотрим на пример.



Это называется задачей рукопожатия.

Вы все должны знать, что рукопожатие является распространенной формой приветствия среди людей, хотя это может быть не так, во всех культурах.

Скажем, есть какое-то количество людей в комнате, и предполагается, что их число n.

Если каждый человек пожмет руку один раз и только один раз каждому другому человеку в комнате, вопрос в том, какое будет общее число рукопожатий, которые будут сделаны, когда есть n человек?

Давайте использовать обозначение h(n) для представления общего количества рукопожатий для любого числа n.

Подумайте об этом, как бы вы решили эту задачу.

Когда есть только один человек, в этом случае не будет никакого рукопожатия.

В простейшем случае, когда у вас есть только два человека в комнате, число рукопожатий будет h(n) с n, равным 2.

Понятно, что там может быть только одно рукопожатие, когда есть только два человека, таким образом, h(2) будет равно 1.

Что делать, если есть еще один человек, и входя в комнату, человек будет пожимать руку двум лицам, которые уже находятся в комнате, и так как эти два человека уже пожимали руку друг другу, никакого дополнительного рукопожатия не будет для двух из них.

Таким образом, h(3) будет h(2) плюс 2.

То есть, предыдущее число рукопожатий плюс два новых рукопожатия для человека, который только что вошел в комнату.

Когда четвертый человек войдет в комнату, нужно использовать тот же алгоритм.

Этот четвертый человек будет пожимать руку трем людям, которые уже находятся в комнате и для трех человек, которые уже в комнате, не будет никаких дополнительных рукопожатий между собой.

Таким образом, общее число рукопожатий для 4-х человек, или h(4) будет h(3) плюс 3, это новые рукопожатия для четвертого человека.

Надеюсь, вы уже видите здесь некий шаблон.

В общем, когда имеется n лиц, общее количество рукопожатий будет h(n-1), то есть, количество рукопожатий для n-1 человек, которые уже в комнате, плюс n-1 новых рукопожатий для n-го человека, входящего в комнату.

Так что, общий вид решения является $h(n) = h(n-1) + n - 1$.

Функция факториала

Для некоторых задач может быть естественным определить задачу с точки зрения самой задачи.

Рекурсия особенно полезна для задач, которые могут быть представлены более простой версией той же задачи, как пример с рукопожатием, что мы только что обсуждали.

В таких задачах, мы можем сформулировать решение задачи, просто следуя путем, как задача определена.

Давайте рассмотрим другой пример, который мы видели раньше.

Мы ввели функцию факториала, когда мы обсуждали циклы.

Напомним, что факториал целого числа n представляет собой последовательное умножение чисел от n до 1.

Например, при вычислении 6!, результат будет умножение 6, 5, 4, 3, 2, 1, который даст 720.

Также легко видеть, что 6! можно переписать в виде 6 умножить на 5!, где 5! равен оставшейся части умножений, следующих за 6. Так что мы выражаем 6! как 5!, который является упрощенной задачей, по сравнению с 6!.

$$n! = n * (n-1)!$$

Таким образом, в целом, мы можем выразить функцию факториала как n! = n * (n-1)!.

Здесь n! может быть получен путем вычисления n-1 факториал.

Вы решили полностью задачу? Почти.

У нас еще есть вычисление n-1!. Как мы вычисляем n-1!?

Мы можем получить n-1!, вычисляя n-2!.

Но это, кажется никогда не заканчивающейся последовательностью, если нет какого-то завершающего состояния.

В случае факториала, мы знаем, что он определяется только для неотрицательных чисел.

И мы также знаем, что 1! или 0! дает значение 1.

Давайте игнорировать 0! в нашем обсуждении, и n-1! продолжает уменьшаться до тех пор, пока n-1 не станет равен 1, и мы можем остановить повторный вызов функции факториала.

Это часто называют базовым состоянием.

Чтобы написать это в явном виде, когда n равно 1, мы должны получить n! равен 1, в противном случае, n! равно n * n-1!.

Важно помнить, что, когда мы вызываем рекурсивную функцию, там должен быть способ, чтобы закончить рекурсивный вызов не рекурсивным образом.

В Java можно определить рекурсивную функцию.

Когда у нас есть функция или метод, который вызывает сам себя, мы обращаемся к такой функции, как рекурсивной функции или методу.

На самом деле, рекурсия – это не просто задача вычислительной науки, это также представляет интерес для математиков, потому что есть много функций, которые могут быть определены рекурсивно.

В 70-х годах, термин фрактал был введен для визуализации самоподобных шаблонов, используя компьютерное моделирование.

Мы обсудим некоторые из них позже.

Давайте теперь постараемся реализовать вычисление n! рекурсивным образом.

```
public static int fact(int n){
    if(n<=1)
        return 1;
    else
        return n * fact(n-1);
}
```

Здесь мы объявляем метод с именем fact, который возвращает целое значение.

Мы используем статический метод, так что мы не должны создавать объект, прежде чем использовать этот метод.

Метод возьмет целый параметр n, для которого мы вычислим факториал.

Остальная часть определения будет просто следовать рекурсивному определению, как мы только что описали.

То есть, n! равен 1, если n меньше или равно 1, при условии, что n является неотрицательным.

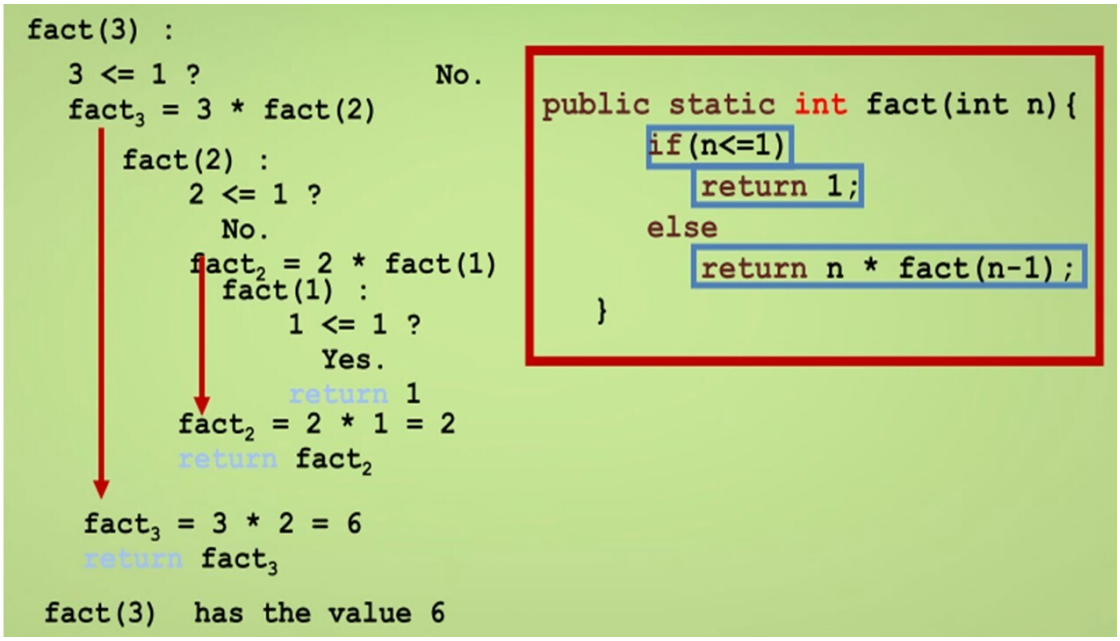
В противном случае, то есть, когда n больше, чем 1, n! будет n * n-1!.

Обратите внимание, что мы вызываем метод внутри определения метода, и это то, что мы раньше не видели.

Рекурсивный вызов методов

Давайте более внимательно посмотрим на то, как рекурсивный вызов метода фактически работает в Java.

Возьмем простой случай, когда n равно 3.



Это функция факториала, которую мы только что определили.

Когда метод вызывается в первый раз с параметром, равным 3, внутри метода, первое, что нужно проверить, является ли 3 меньше или равно 1, что явно ложно, так что будет выполнен код else.

То есть, будет вычисляться fact(3), который равен 3 умножить на fact (n-1), с n равным 3, то есть, метод будет вызываться с параметром 2 или 3-1.

В этом месте мы не знаем пока результат для fact(2).

Поэтому, прежде чем мы можем перемножить fact(2) на 3, мы должны сначала выяснить значение fact(2).

Далее, мы должны сделать рекурсивный вызов метода с 2 в качестве параметра.

Обратите внимание, что параметр n в каждом рекурсивном вызове является локальной переменной.

Каждый раз, когда n упоминается внутри метода, используется локальная копия n, которая замещает предыдущую копию параметра, которая несет такое же имя n.

Так что n в fact(2) будет иметь значение 2 и условие n меньше чем или равно 1 снова дает ложь. Поэтому будет выполнен код else.

То есть, два умножить на fact(n-1) или 1 при n, равным 2.

В этот момент, значение для fact(1) еще не определено.

Далее, производится рекурсивный вызов метода с 1 в качестве параметра.

В этом случае, с n, равным 1, if условие становится верным, и 1 будет возвращена в качестве результата для fact(1).

После того, как значение для fact(1) определяется, мы можем вернуться к вычислению fact(2) путем умножения 2 на fact(1), который только что получили как возвращенное значение 1.

Таким образом fact(2) может быть вычислен путем умножения 2 на 1 и получим 2 как возвращенный результат для fact(2).

Используя значение для fact(2), мы можем снова вернуться к вычислению fact(3) путем умножения 3 на fact(2) или 2 и получить значение 6.

Таким образом, мы имеем возвращаемое значение для fact(3), который является конечным результатом, который мы хотим получить для 3!.

Мы видим в этом примере, что метод вызывается несколько раз.

Опять же, отметим, что параметр n является локальной переменной и новая копия n создается в области памяти, которая называется программным стеком, каждый раз, когда метод вызывается.

Подробности того, как программный стек реализуется, выходит за рамки данного курса.

Помните, что, когда мы обсуждали структуру цикла, используя факториал как пример, мы реализовали факториал с использованием различных видов циклов, а именно, while, do-while и for цикла.

На самом деле, все задачи, которые можно решить с помощью рекурсивных вызовов, также могут быть решены с использованием итеративного решения.

Для некоторых задач, рекурсивное решение может привести к более короткой и элегантной реализации.

Тем не менее, часто бывает, что рекурсивное решение является более дорогим с точки зрения времени вычислений и памяти из-за необходимости поддерживать программный стек и память для локальных переменных.

В случае факториала, вы можете увидеть, что большого отличия нет между итерацией или рекурсивным решением.

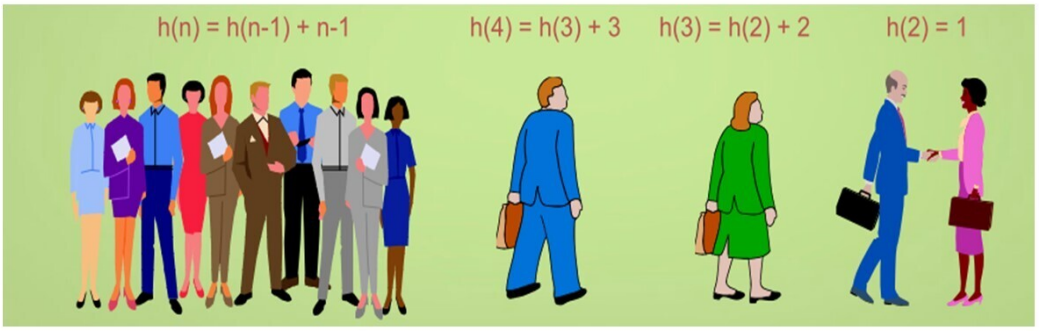
```
public static int fact(int n){
   int t = 1;
   int counter = 1;

   while (counter <= n) {
      t = t * counter;
      counter = counter + 1;
   }
   return t;
}
```

```
public static int fact(int n){
      if(n<=1)
         return 1;
      else
         return n * fact(n-1);
   }
```

На левой стороне, это реализация с использованием while цикла, и метод на правой стороне является рекурсивным решением, которое мы только что видели.

Вы можете посмотреть на инициализацию t как базовое решение, в то время как цикл будет соответствовать рекурсивному вызову метода, если сравнить итерационное решение с рекурсивной реализацией.

Давайте вернемся к задаче рукопожатия.



Напомним, что общее количество рукопожатий для n человек, это h(n) = h(n-1) + (n-1).

То есть, h(n) вычисляется с помощью h(n-1).

Поскольку h определен в терминах h меньшего n, это может быть реализовано с использованием рекурсивного метода.

В этой задаче, базовый случай, это n равно 2, это минимальное количество рукопожатий, когда есть два человека.

```
public static int handShake(int n){
    if(n <= 2)
        return n - 1;
    else
        return handShake(n-1) + (n-1);
}
```

Предполагая n положительным, здесь мы также заботимся о том случае, когда n равно 1, и будет n-1 или ноль рукопожатий.

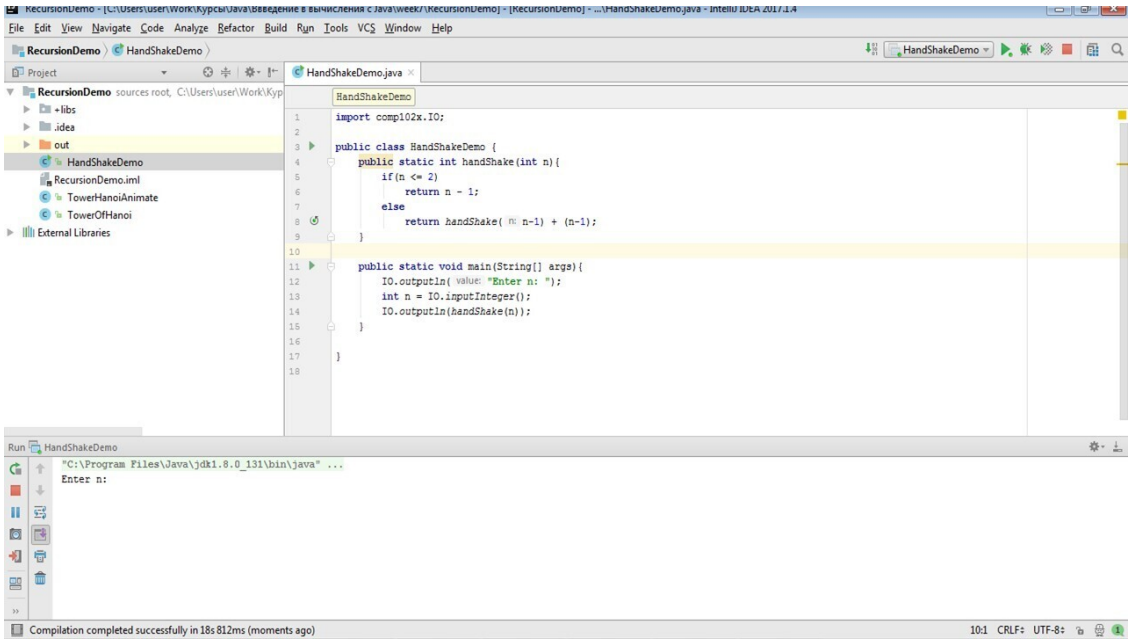
Код else рекурсивного вызова обеспечивает случаи для n больше, чем 2.

Как альтернатива, это также может быть реализовано с помощью цикла.

На самом деле, здесь мы можем переформулировать решение как сумму целых чисел от 1 до n-1 = n(n-1) / 2.

Но, беря эту формулу, может быть трудно объяснить реализацию общего количества рукопожатий для n человек.

Откроем IDEA и посмотрим на работающую задачу рукопожатий.

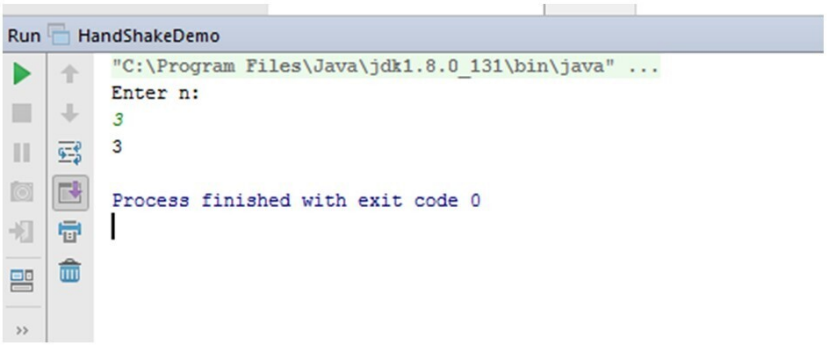


Здесь мы реализовали программу рукопожатий, как обсуждали перед этим.

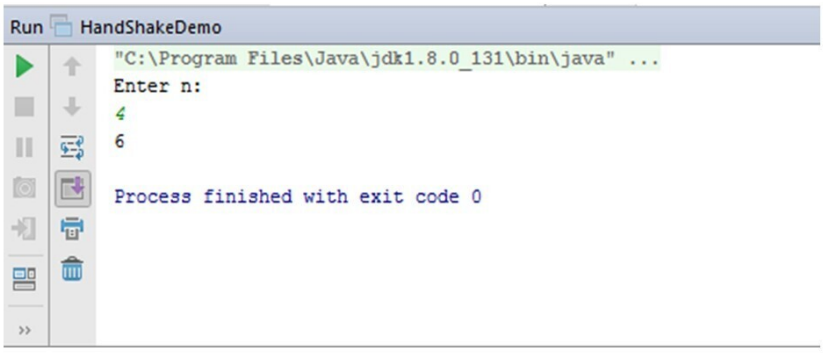
Скомпилируем и запустим эту программу.

Попытаемся запустить программу, вводя различные значения n.

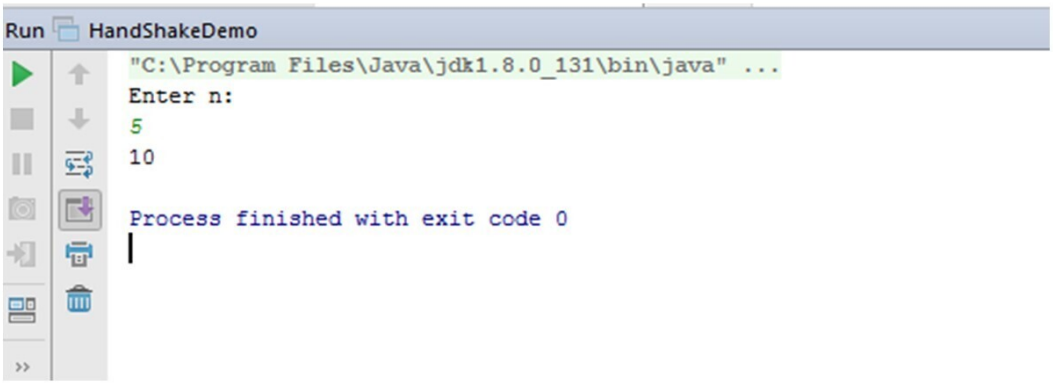
Если мы введем 3 в качестве значения n, вы увидите, что 3 является общим количеством рукопожатий.



Давайте снова запустим программу, введя 4 в качестве значения n, и тогда вы обнаружите, что 6 это возвращаемый результат.



Давайте сделаем еще один запуск, введя 5.
Теперь у вас есть 5 человек в комнате.



И вы можете видеть, что возвращаемый результат 10 здесь равен 6 плюс 5 минус 1, что есть 4.

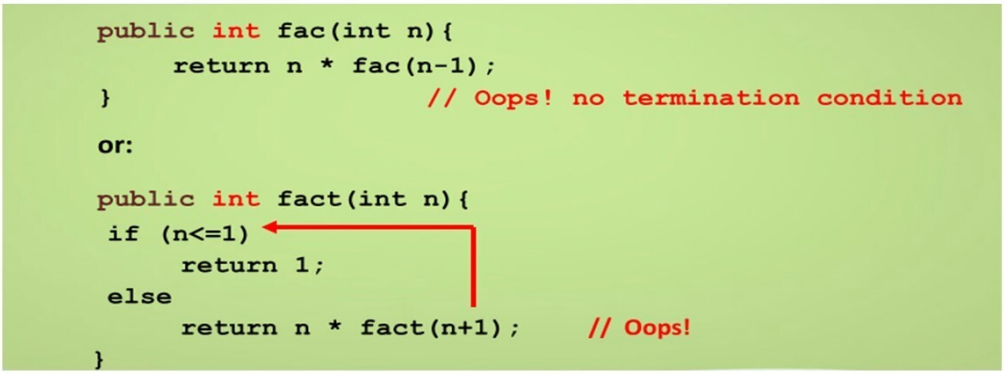
Так что у вас есть 6 плюс 4 равно 10.

Числа Фибоначчи

При работе с циклами итерации, нужно быть очень осторожным, чтобы не попасть в бесконечный цикл.

Точно так же для рекурсии, мы должны быть осторожны, чтобы не создавать бесконечную цепочку рекурсивных вызовов метода.

Вот несколько примеров, где вы можете попасть в неприятности.



Посмотрите на этот первый пример и подумайте о том, где тут может быть проблема.

Как уже упоминалось ранее, должен быть завершающий базовый случай, когда выполняется не рекурсивный вызов.

В противном случае, рекурсивный вызов никогда не прекращается.

Что об этом втором примере?

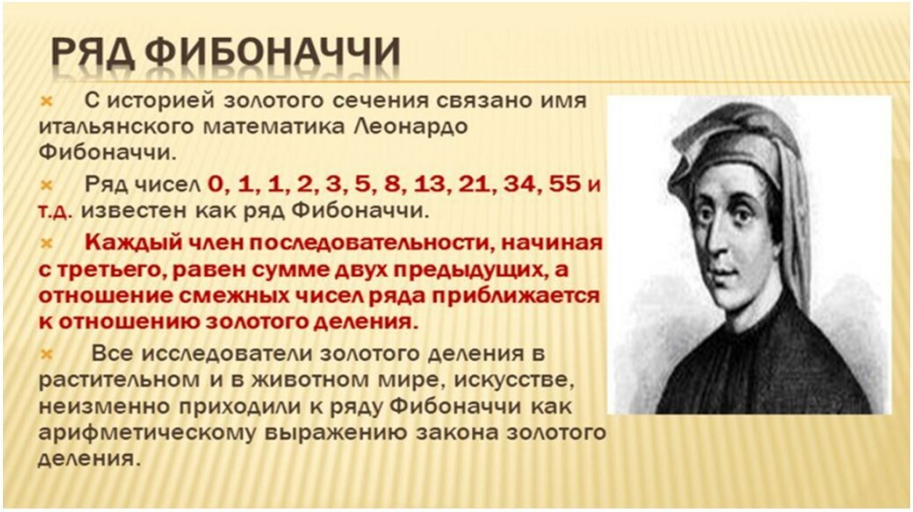
Это выглядит очень похоже на то, что мы видели в вычислении n факториала.

Проблема в том, что n+ 1 используется здесь по ошибке.

Таким образом, вместо уменьшения n каждый, n растет и никогда не достигнет прекращения состояния n меньше или равное 1, таким образом, будет бесконечная цепочка рекурсивных вызовов.

Давайте рассмотрим другой, более интересный пример, это знаменитая последовательность Фибоначчи.

Последовательность Фибоначчи впервые появилась в 13 веке.



Когда наблюдался рост популяции кроликов, было установлено, что рост следует следующему шаблону:

– Каждая пара зрелых кроликов, один самец и одна самка, производит новую пару крольчат каждый месяц;

– Каждая новая пара становится половозрелой через один месяц после рождения, и самка будет производить еще одну пару кроликов через месяц;

– И далее, предположим, что ни один из кроликов не умрет в этом году.

Вопрос в том, сколько пар кроликов будет через год.

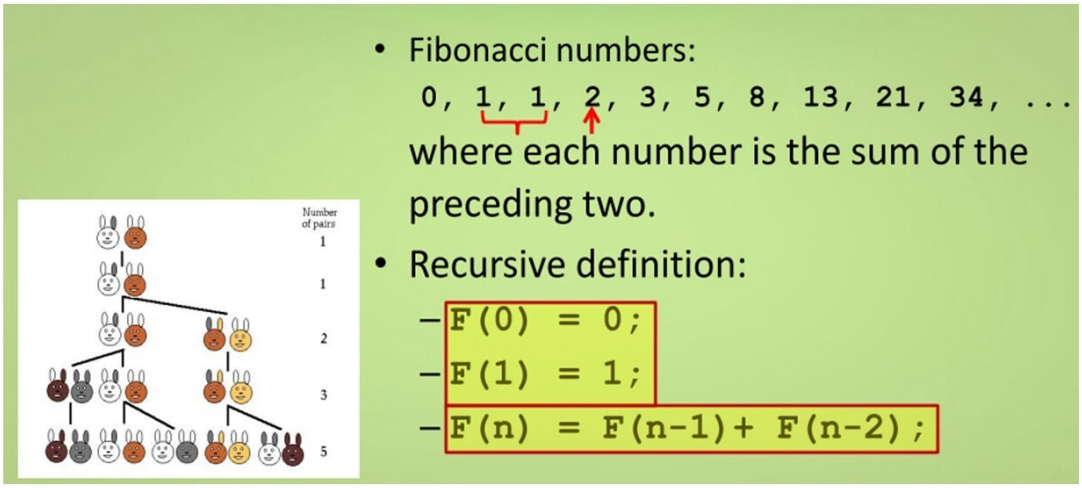
Чтобы проиллюстрировать это с помощью примера, начиная с пары кроликов, один самец и одна самка, если они оба взрослые, после одного месяца, будет 2 пары кроликов, исходная пара и пара новорожденных.

В ходе 2-го месяца, из двух пар кроликов, зрелая пара будет продолжать производить потомство в 2-м месяце, в то время как пара новорожденных должна просто повзрослеть, но еще не иметь детей.

Так что будет 3 пары кроликов в конце 2-го месяца.

В конце 3-го месяца, две пары взрослых кроликов каждая родит пару новорожденных, то есть будет 2 дополнительные пары или 5 пар вместе.

И пара, родившаяся во 2-й месяц, теперь стала половозрелой.



Такой рост популяции может быть показан на этом рисунке здесь, начиная от пары кроль-чат.

Числа Фибоначчи, как полагают, моделируют природу определенным образом, и есть другие природные феномены, которые напоминают последовательность Фибоначчи, например, наблюдение Кеплером листьев и цветов в 17 веке в его исследовании симметрии.

Вот некоторые из первых чисел Фибоначчи.

Если вы посмотрите на эту последовательность чисел более тщательно, вы можете найти, что каждое из этих чисел может быть получено как сумма двух чисел, предшествующих им.

Например, 2 получается путем сложения 1 и 1, 3 получают путем сложения 1 и 2, 5 полу-чают путем сложения 2 и 3, 8, добавляя 3 к 5, и так далее.

В общем, энное число Фибоначчи можно определить рекурсивно, путем суммирования (n -1) и (n – 2) чисел Фибоначчи.

И базовый случай определен для n = 0 и n = 1 со значениями, равными 0 и 1 соответ-ственно.

Вот реализация Java для числа Фибоначчи.

```
//Calculate Fibonacci numbers using recursive method
public class Fibonacci
{
    static int fib(int n){
        if (n == 0) return 0;
        if (n == 1) return 1;
        return (fib(n-1) + fib(n-2));
    }

    public static void main(String[] args) {
        IO.output("Enter the value n: ");
        int n = IO.inputInteger();
        int fibN = fib(n);
        IO.outputln("Fib(" + n + ") = " + fibN) ;
    }
}
```

Фактический метод вычисления числа Фибоначчи является методом, который принимает один параметр n и возвращает целое число, представляющее число Фибоначчи.

Метод main реализован ниже, просто, чтобы позволить ввод различных значений n для проверки метода.

Обратите внимание, что метод объявлен статическим, так что нам не нужно создавать объект, прежде чем использовать этот метод.

И это очень простая реализация, просто следуя рекурсивному определению, как мы уже говорили ранее.

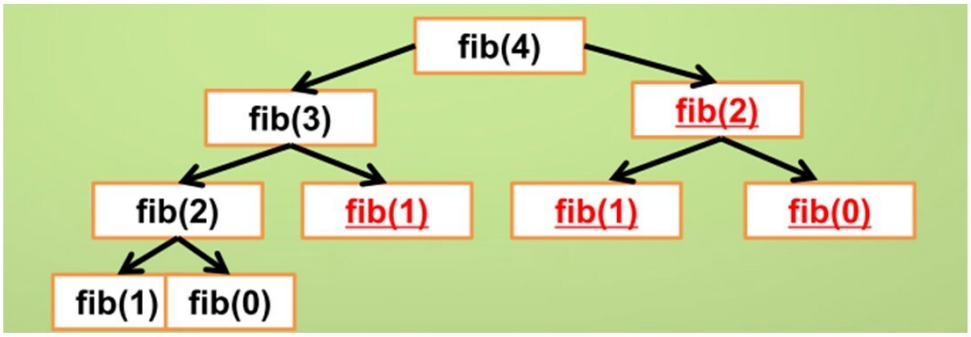


График здесь показывает последовательность шагов в рекурсивных вызовах метода.

Хотя FIB(4) будет вызывать как FIB(3) так и FIB(2), так как каждый из них должен быть выполнен последовательно, FIB (3) должен быть вычислен в первую очередь.

FIB(2) будет затем вызвать FIB(1) и FIB(0), и так как оба представляют собой базовые случаи, будут возвращены целочисленные результаты.

Затем вычисление возвратится к FIB(3), который требует вычисления FIB(1), и так как это снова базовый вариант, значение для FIB(3) может быть определено.

И далее будет попытка завершить расчет для FIB(4) путем вычисления FIB(2).

Процесс будет продолжаться до тех пор, пока значения для FIB(3) и FIB(2) не будут получены для определения конечного значения для FIB(4).

Как вы можете видеть на графике, здесь много промежуточных шагов пересчитываются для расчета метода для различных значений n.

Например, FIB(1) повторно вычисляется 3 раза, FIB(2) повторно вычисляется 2 раза и так далее.

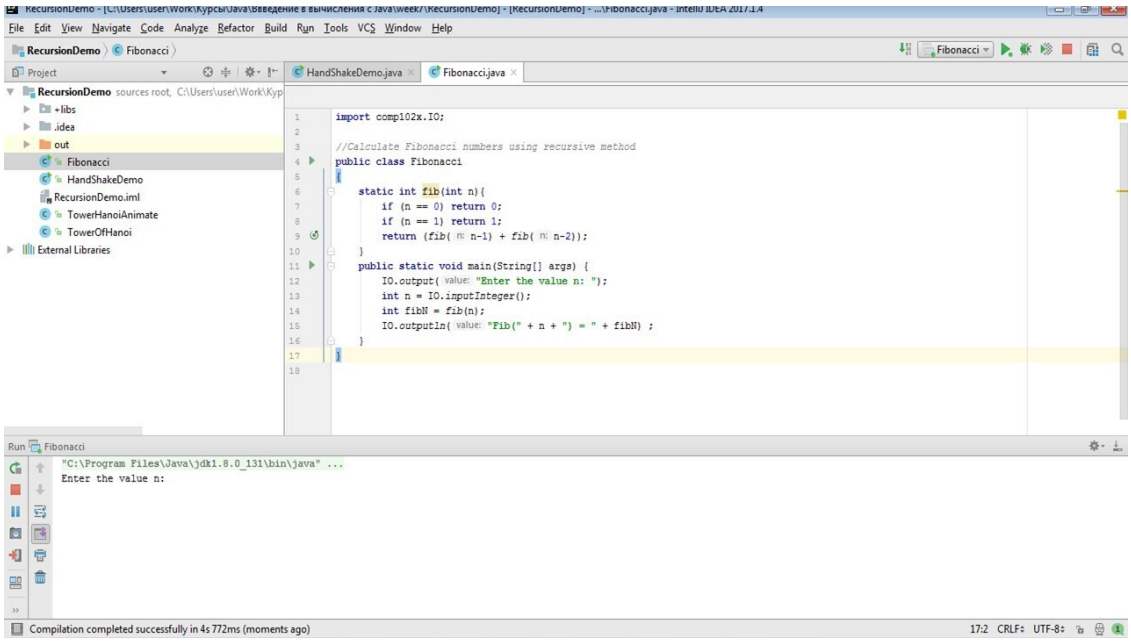
Подобно задаче рукопожатия, есть также другая форма решения для вычисления чисел Фибоначчи.

Хотя это намного сложнее.

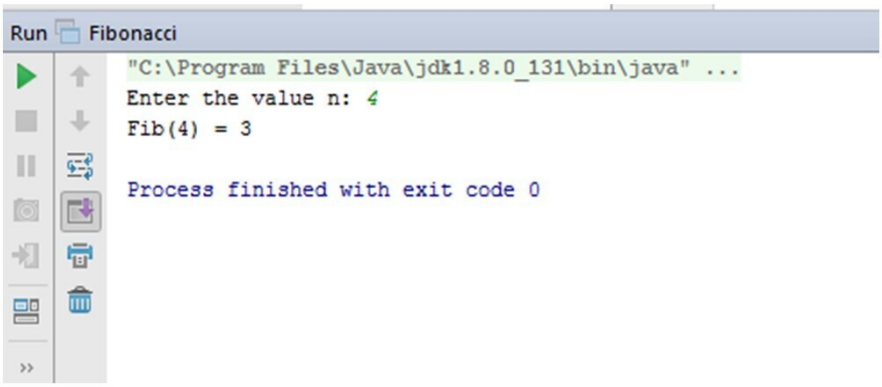
$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Мы можем, конечно, реализовать ряд Фибоначчи, используя итеративный подход, но реализация не будет такая простая, как рекурсивное решение.

Давайте теперь откроем IDEA чтобы посмотреть на фактическое выполнение программы.



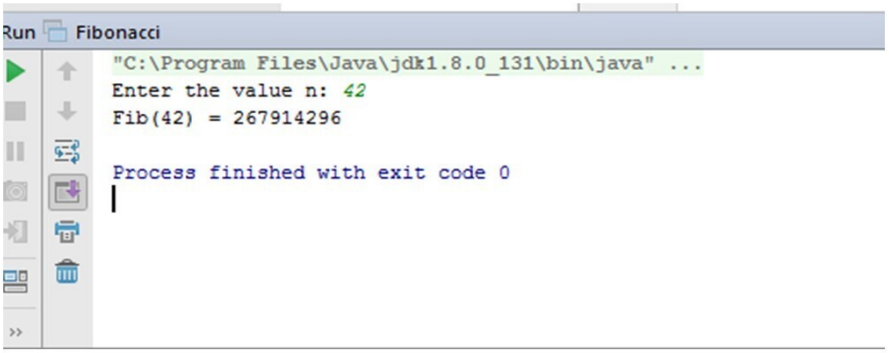
Так что здесь показана реализация ряда Фибоначчи, которую мы только что видели. Давайте скомпилируем и запустим программу.



При попытке запустить программу, используя небольшое значение n = 4, вы сможете обнаружить, что четвертый номер Фибоначчи это 3.

Когда мы пытаемся ввести большее число, скажем, 42, и вы можете увидеть, что нужно немного подождать, прежде чем результат возвратится.

Вы должны ждать пару секунд, прежде чем возвратится результат.



Если вы посмотрите на результаты, можно увидеть, что число Фибоначчи для 42 это сумма числа Фибоначчи для 40, что составляет около 100 млн., и число Фибоначчи для 41, что будет примерно 165 млн.

Если сложить их, вы получите около 267 млн.

Когда вы захотите вычислить число Фибоначчи для n, равным 45, вы найдете, что это займет много времени, прежде чем вы увидите результат.

Двоичный поиск

Давайте посмотрим на другой пример рекурсии, который имеет широкий спектр применений.

Когда мы обсуждали массивы, мы кратко рассмотрели задачи сортировки и поиска.

Сортировка и поиск являются двумя из наиболее часто используемых операций в компьютерных приложениях, и они часто используются вместе, потому что, если данные отсортированы, операция поиска может быть выполнена гораздо более эффективно.

Если вы до сих пор помните те времена, когда вы смотрели на значение слова из бумажного словаря или некоторые имена из телефонной книги, вместо того чтобы всегда искать информацию с начала словаря или каталога, вы начнете где-то посередине, потому что вы знаете, что слова или имена уже перечислены в алфавитном порядке.

Идея бинарного поиска следует подобной идеи.

Обратите внимание, что бинарный поиск будет работать только для отсортированных массивов.

Вместо того, чтобы начать поиск с начала массива, идея бинарного поиска в том, чтобы начать с середины.

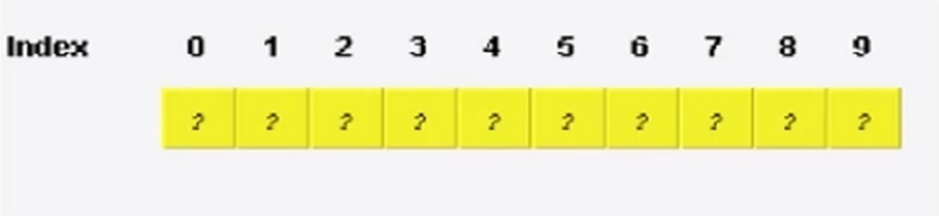
При этом сначала сравнивается искомый элемент со средним элементом массива.

Если соответствие нашли, тогда поиск будет успешным.

В противном случае, применяется бинарный поиск к той половине массива, где искомый элемент, вероятно, может быть найден.

Обратите внимание, что поскольку массив отсортирован, поиск элемента может быть сделан только в одной из двух половинок матрицы, если он есть вообще.

Рассмотрим пример.



Вы можете видеть, что здесь есть массив из неизвестных элементов, и мы пытаемся найти значение 14 в этом массиве.

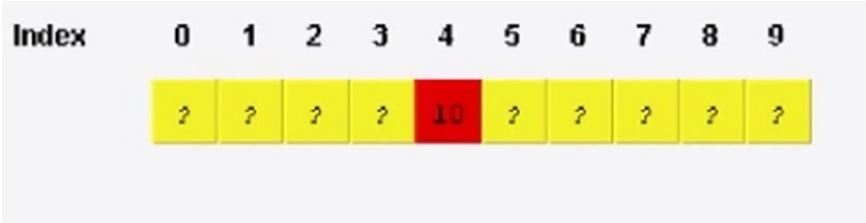
Разделим массив на две части.

В начале, весь массив неизвестен, но отсортирован.

Теперь, мы будем искать значение 14 в этом массиве.

Разделим массив на две части, и изучим среднюю величину, является ли оно равным значению нашего поиска.

Индекс среднего значения вычисляется по формуле (start_index + end_index) / 2. В этом случае это индекс 4.



Значение 10 не наша искомая величина, так что мы продолжаем поиск.

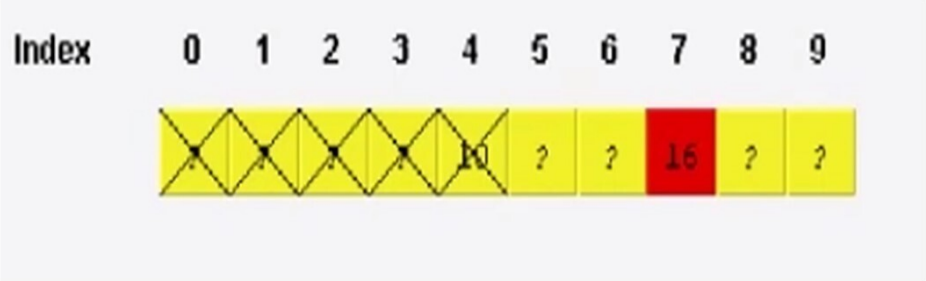
Так как массив отсортирован, часть перед текущим значением должна быть меньше, чем 10.

И теперь мы знаем, что все значения в этой части также меньше, чем искомое значение 14.

Таким образом, мы можем оставить левую часть и сделать проверку только в правой части.

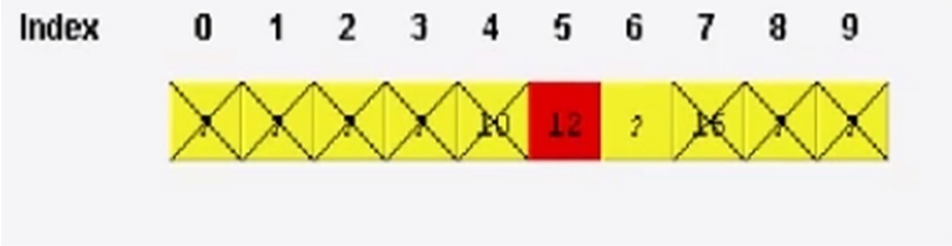
Теперь, продолжим нашу проверку в этой оставшейся части.

Разделим эту часть на две части и изучим среднюю величину, то есть величину с индексом 7.



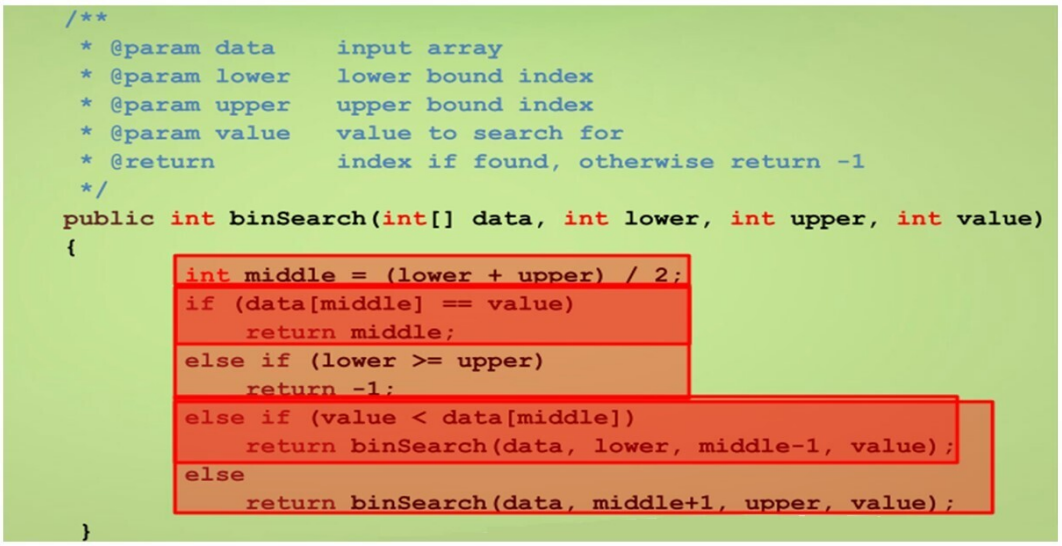
Мы можем сделать вывод, что искомое значение теперь не в правой части.

Повторите этот процесс на оставшейся части путем определения, является ли среднее значение искомым значением.



Мы нашли наше искомое значение 14, которое здесь имеет индекс 6.

Двоичный поиск может быть осуществлен путем прямой передачи этой идеи в Java коде с помощью рекурсии.



```
/**
 * @param data    input array
 * @param lower   lower bound index
 * @param upper   upper bound index
 * @param value   value to search for
 * @return        index if found, otherwise return -1
 */
public int binSearch(int[] data, int lower, int upper, int value)
{
        int middle = (lower + upper) / 2;
        if (data[middle] == value)
            return middle;
        else if (lower >= upper)
            return -1;
        else if (value < data[middle])
            return binSearch(data, lower, middle-1, value);
        else
            return binSearch(data, middle+1, upper, value);
 }
```

Здесь показана реализация двоичного поиска данных, хранимых в целочисленном массиве.

Переменные lower и upper являются двумя ограничивающими индексами для части массива, где программа в настоящее время ведет поиск.

В начале, будет lower = 0 и upper будет размером массива, если вы хотите искать для всего массива.

Переменная value представляет данные, которые вы ищете.

Программа начинается с определения среднего индекса в массиве, который является просто (lower + upper) / 2.

Затем программа проверяет, равен ли средний элемент значению, которое вы ищете, и если это правда, это значение находится и индекс возвращается.

Если переменная lower больше или равна переменной upper, это означает, что программа выполнила поиск в массиве полностью и элемент нельзя найти, поэтому возвращается значение -1.

Далее идет рекурсивная часть, где, если искомое значение меньше среднего элемента, то так как массив отсортирован, значение может быть только в начале половины массива, если оно вообще там есть.

И вызов бинарного поиска делается путем изменения верхнего индекса на middle-1, таким образом, что только первая половина массива ищется на следующем этапе.

Если код else выполняется, то есть, когда искомое значение больше среднего элемента, искомый элемент может быть только во второй половине массива, если он есть вообще, поэтому поиск выполняется только в части с индексом middle+1 до первоначального верхнего индекса.

Обратите внимание, что рекурсивный вызов binSearch производится в двух местах, но с разными ограничивающими индексами.

Поскольку диапазон индексов продолжит уменьшаться в рекурсивном вызове, для массива фиксированного размера, он в конечном итоге достигнет базового случая, как указано здесь.

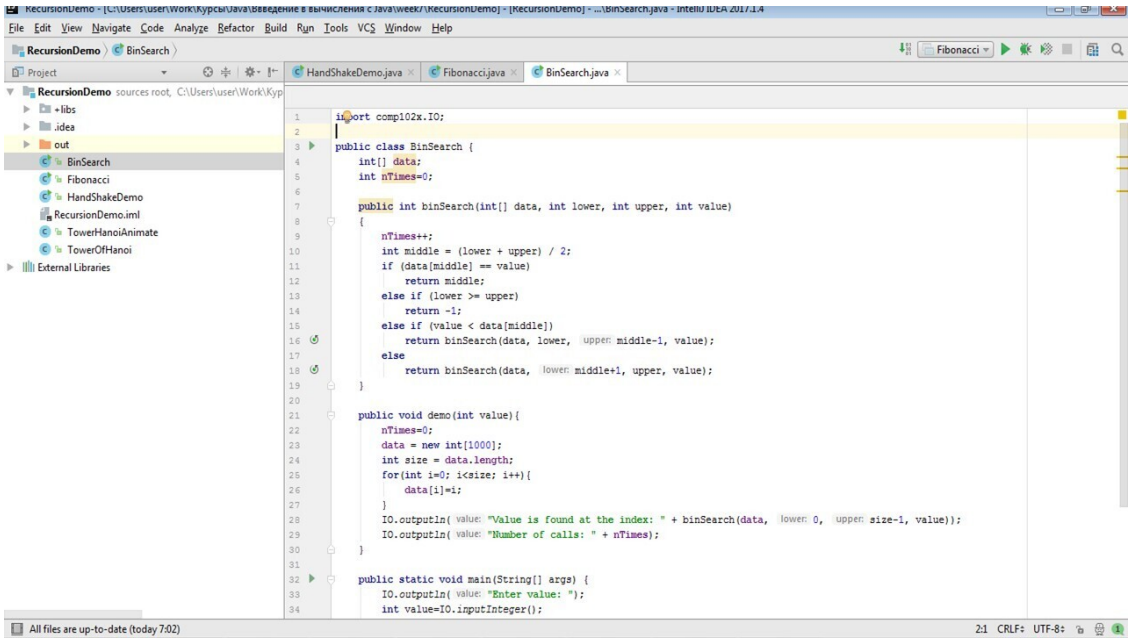
Вы можете реализовать бинарный поиск, используя итеративный подход.

Перевод бинарного поиска в итерационный метод должен быть довольно простым.

Вместо того чтобы использовать рекурсивный вызов, вы можете использовать while цикл.

Было бы хорошим упражнением для вас попробовать реализовать бинарный поиск, используя итеративный подход.

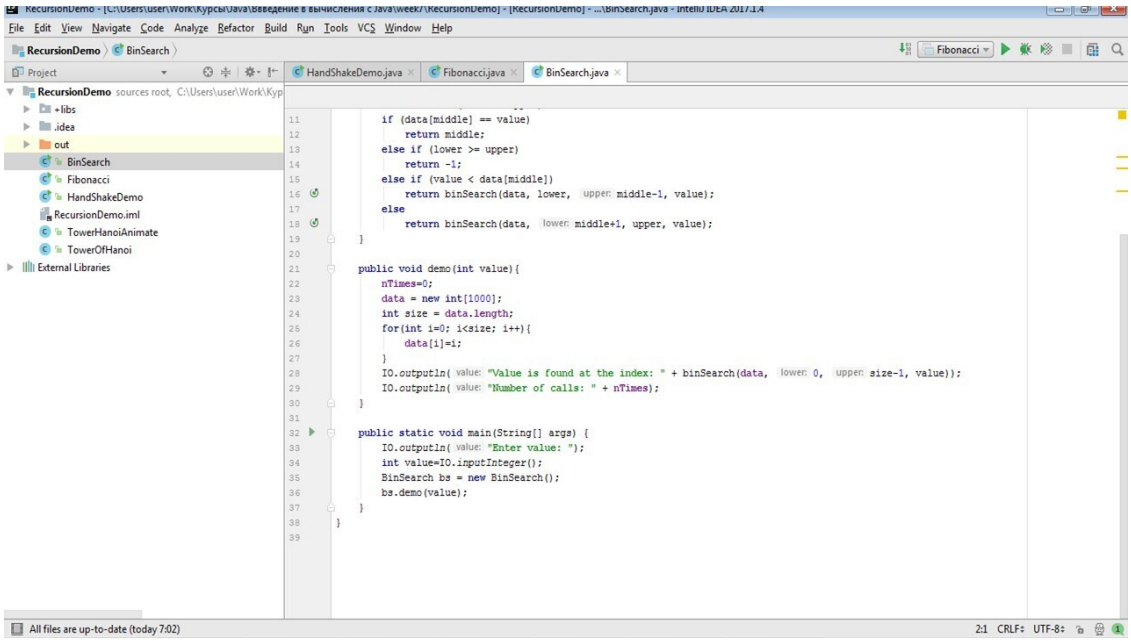
Давайте посмотрим на демонстрацию программы бинарного поиска.



Здесь мы используем целый массив data для хранения контента для поиска, который будет проводиться.

Переменная nTimes используется для отслеживания количества раз, сколько было сделано рекурсивных вызовов, потому что я хочу сказать немного о эффективности алгоритма двоичного поиска.

Метод binSearch здесь такой же, как тот, который мы только что описали, кроме того, что nTimes увеличивается на 1 каждый раз, когда binSearch вызывается.



Мы также создали метод demo с одним параметром, который дает текущее значение для поиска.

Здесь мы создаем большой массив с 1000 элементами, потому что мы хотим получить представление о эффективности алгоритма двоичного поиска.

Каждый элемент массива данных просто инициализируется в то же значение, что и его индекс.

Затем вызывается метод binSearch с 0 в качестве нижней границы индекса, и size-1 в качестве верхней границы индекса таким образом, что поиск будет охватывать весь массив.

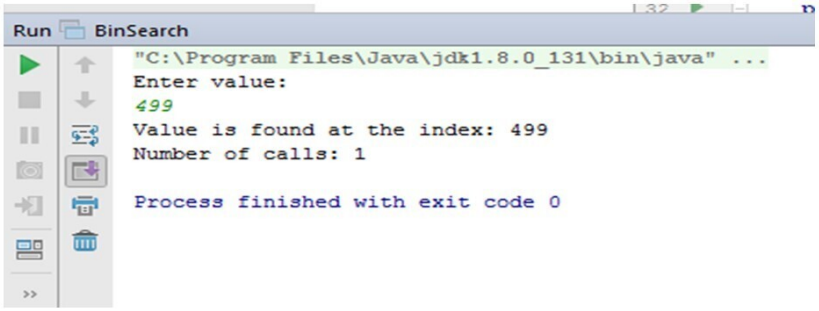
Обратите внимание, что верхняя граница size-1, потому что это максимальный индекс массива.

Перед завершением метода demo, выводится nTimes.

Давайте скомпилируем программу, создадим экземпляр и запустим метод demo.

Значение для поиска вводится в качестве параметра.

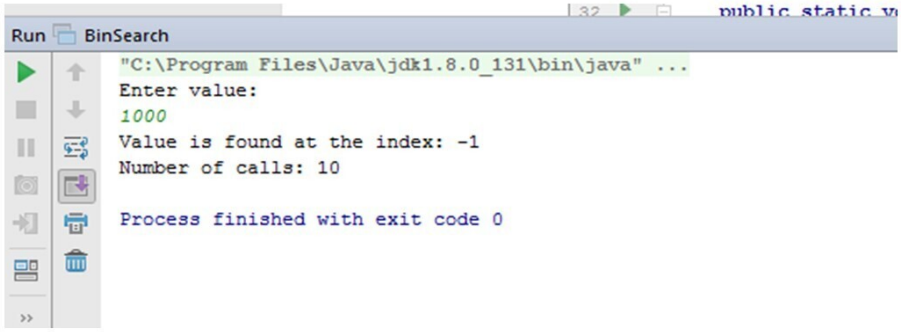
Давайте попробуем 499.



Программа успешно находит индекс 499, и был сделан только один вызов binSearch.

Это происходит потому, что 499 находится в середине массива, то есть, когда вы добавляете 0 к 999, а затем делите на 2, используя целое деление; средний индекс получает значение 499.

Давайте снова запустим программу с 1000 в качестве входного значения.



Известно, что 1000 нет в массиве, поскольку максимальное значение 999.

Так что правильный результат возвращается -1.

Что интересно, это то, что количество вызовов binSearch здесь всего 10 раз.

Подумайте, сколько вызовов вы должны сделать, если вы ищете в массиве от начала до конца последовательно.

Для массива с 1000 элементов, вы должны пройти через 1000 сравнений прежде, чем может быть подтверждено, что искомого значения там не было.

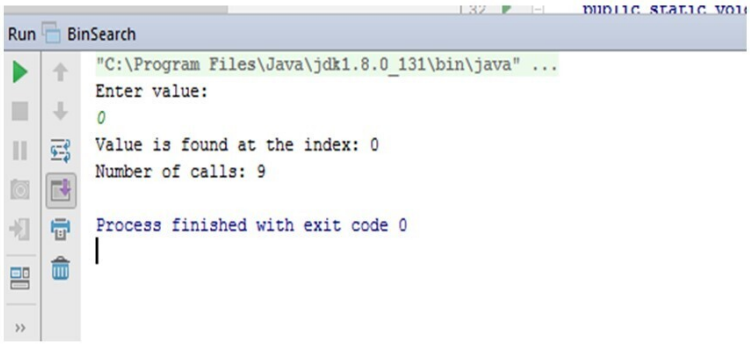
Но с бинарным поиском только 10 сравнений необходимы.

В общем, вы можете увидеть, что для любого числа n, логарифм n с базой 2 это максимальное количество операций, которые необходимы для поиска.

Например, двоичный логарифм 1000 равен 9.99, так что бинарный поиск считается очень эффективным алгоритмом для больших n.

Но помните, что бинарный поиск будет работать, только если данные отсортированы.

Давайте попробуем еще раз запустить программу, на этот раз мы хотим найти значение 0.



Значение находится по индексу 0.

Но заметьте, что будут необходимы 9 вызовов для получения результата. Это происходит потому, что поиск начинается с середины.

Вы можете сказать, что было бы лучше, если бы мы начинали поиск с начала.

Это верно в этом случае, но в среднем, бинарный поиск, является гораздо более эффективным алгоритмом.

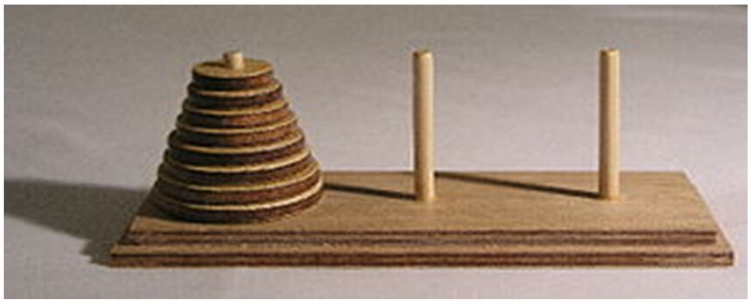
Эффективность алгоритма является важной темой в области компьютерных наук, хотя это выходит за рамки данного курса.

Пример задачи

Давайте рассмотрим другой пример, он основан на головоломке, разработанной более века назад.

По преданию, монахи в отдаленном монастыре могли предсказать, когда наступит конец света.

У них был набор из 3 алмазных игл.



Вот это современный игровой набор башен Ханоя, и здесь у нас есть 3 деревянные иглы вместо алмазных игл.

На первой алмазной игле было нанизано 64 дисков с уменьшением размера.

Здесь, у нас есть только 8 дисков.

Задача состоит в том, чтобы переместить все диски с одной иглы на другую, следуя определенным правилам.

По преданию, конец света наступит, когда они закончат решение этой задачи!

Монахи могли переносить только один диск на другую иглу каждый день, может быть, это были очень тяжелые диски.

Вот правила, которым они должны следовать, когда они перемещают диски.

Только один диск может быть перемещен за один раз.

Это было бы просто, если бы они могли переместить всю стопку из 64 дисков только в одном направлении.

Но это не было разрешено.

Больший диск никогда не должен быть уложен выше меньшего.

В любом случае, они могут перемещать только один диск за другим от большего к меньшему.

И, наконец, только одна дополнительная игла может быть использована для промежуточного размещения дисков.

Следуя этим правилам, это, казалось бы, простая задача потребует 2 в 64-й степени минус один перемещений!

Но как долго это будет до конца мира?

Вы не можете в это поверить, но 2 в степени 64 является огромным числом.

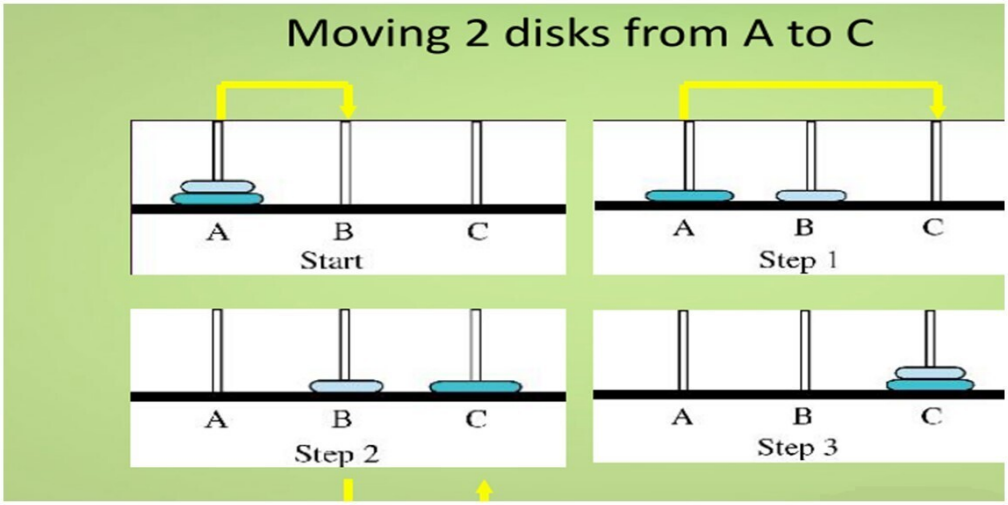
Даже если мы ускорим задачу путем перемещения каждого диска один раз в секунду вместо каждого дня, это потребует 580 миллионов лет, для выполнения поставленной задачи.

Возможно, в легенде есть доля правды, никто не будет знать, как будет выглядеть мир через 500 миллионов лет, но это, конечно, будет другой мир.

В общем, для n дисков, количество необходимых шагов составляет 2 в n степени минус один.

Давайте посмотрим, как эта проблема может быть решена с помощью программы.

Давайте сначала рассмотрим несколько простых примеров.



Давайте посмотрим на более простой случай с участием всего двух дисков.

Здесь показано начальное состояние для n, равного 2, и цель состоит в перемещении двух дисков из A в C.

Вы не можете просто переместить меньший диск из A в C, а затем больший диск из A в C, потому что тогда больший диск будет в верхней части над меньшим диском.

Поэтому меньший диск первым переносится в B.

Это будет промежуточный результат.

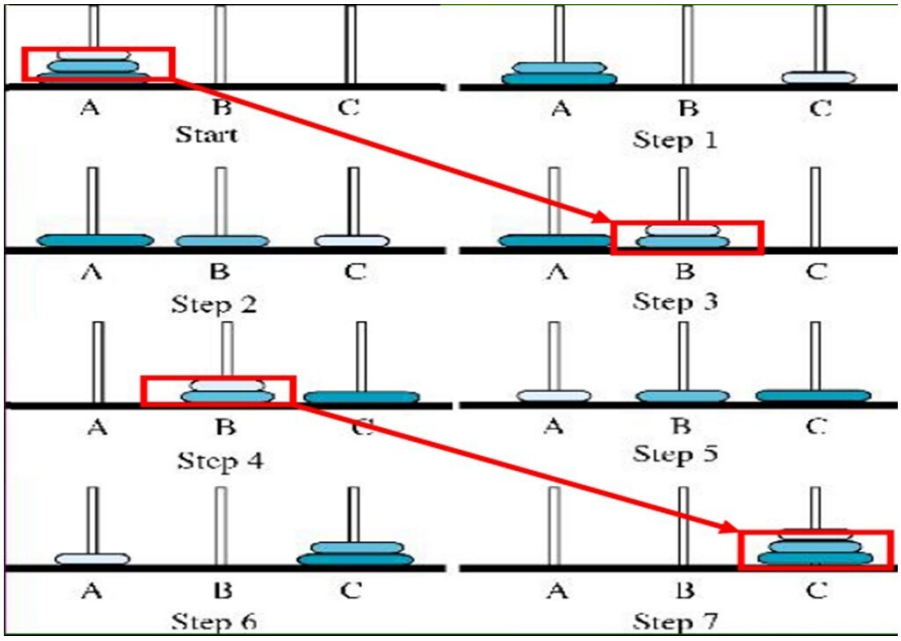
Теперь, больший диск освобождается и может быть перемещен из A в C, и далее меньший диск может быть перемещен из B в C.

И мы достигли нашей цели в 3 этапа.

Как я упоминал ранее, необходимые шаги это 2 в n-й степени минус один.

В этом случае, n = 2, и 2 в квадрате это 4, 4 минус 1 дает 3, такое же, как количество шагов, которые мы здесь нашли.

Давайте вернемся к задаче 3 дисков. Вот начальное состояние.



Мы только что решили задачу 2 дисков, и теперь мы знаем, как бороться с n равным 2. И мы знаем, что для того, чтобы переместить больший диск или нижний диск из A в C, мы должны сначала освободить его путем перемещения диска выше него куда-нибудь в другое место.

Давайте предположим, что 2 меньших дисков были перемещены в B, пропуская промежуточные шаги, потому что мы уже знаем, что это может быть сделано в 3 этапа.

После того, как самый большой диск освобождается, он может быть перемещен из A в C, теперь его конечный пункт назначения.

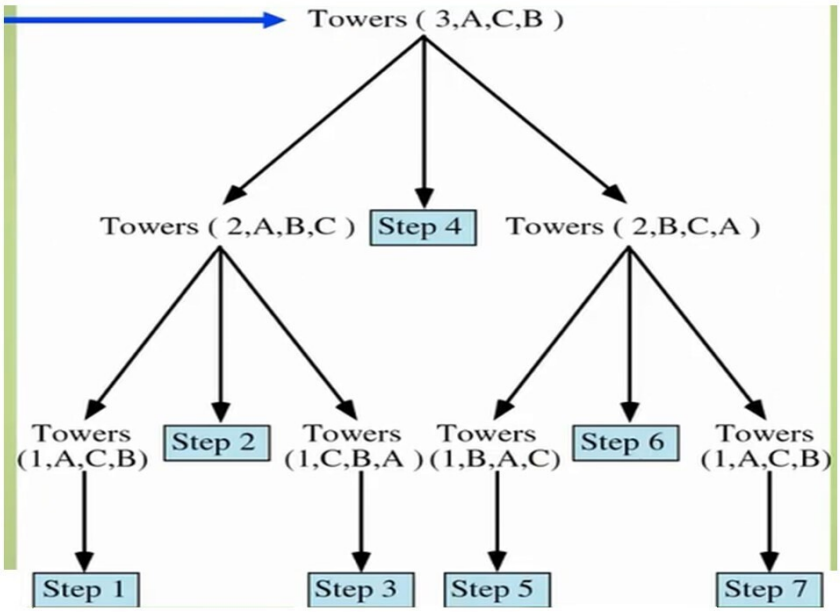
Остальную работу предстоит сделать, чтобы переместить два диска из B в C.

Опять мы знаем, как решить эту проблему 2 дисков в 3 этапа, так что мы можем пропустить детали.

Обратите внимание на то, что здесь одно отличие состоит в том, что B становится основной иглой и A используется в качестве вспомогательного средства, в то время как C, это по-прежнему игла назначения.

Как показано на схеме, решение может быть достигнуто в 7 шагов, снова, для n, равным 3, 2 в 3-й степени это 8, 8 минус 1 дает 7.

Вот схема, которая иллюстрирует идею алгоритма для решения задачи, используя метод, называемый Towers с 4 параметрами.



Мы начинаем решать задачу, указав размер задачи, в этом случае, 3, а затем A является источником, C является местом назначения и B является вспомогательным местом.

Чтобы решить эту задачу для n, равным 3, метод будет вызывать Towers для меньшей задачи с n, равным 2.

В этом случае, A является источником, B назначением и C вспомогательным местом.

Вот шаги для решения этого вызова Towers.

Это один простой шаг, чтобы переместить большой диск из A в C.

И здесь есть вызов Towers, чтобы завершить поставленную задачу, перемещая 2 диска из B в C с помощью A как вспомогательной иглы.

Обратите внимание, что два вызова Towers здесь являются рекурсивными вызовами, также как другие вызовы Towers ниже.

И шаг 4 это не рекурсивный шаг, также, как и шаги 1, 2, 3, 5, 6 и 7.

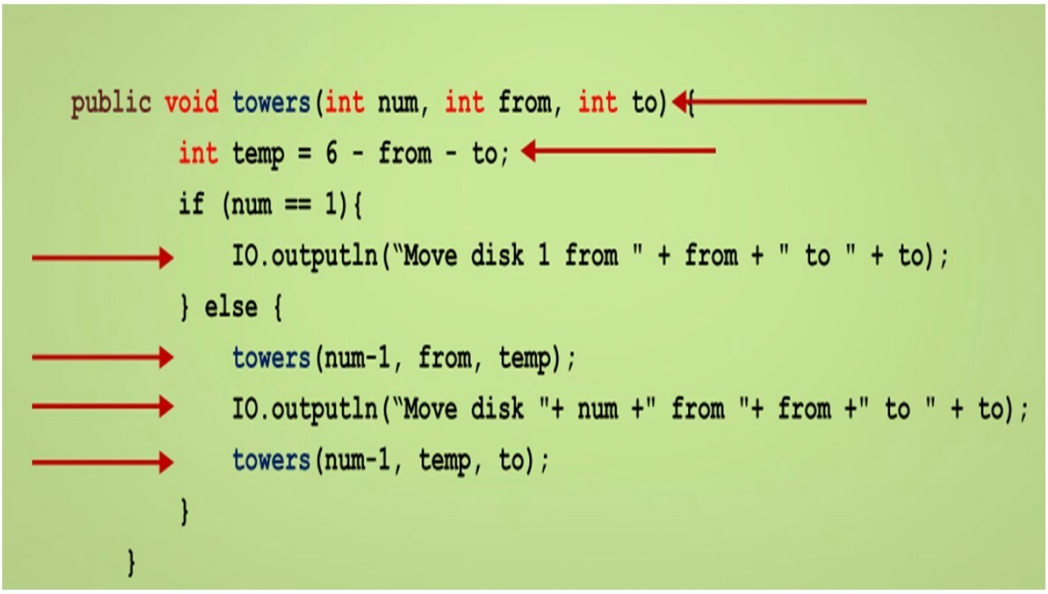
Надеюсь, что вы уже можете здесь увидеть шаблон.

Чтобы решить проблему Башня Ханоя для любого заданного n, мы можем сначала переместить n-1 дисков с вершины самого большого диска на вспомогательную иглу, следуя последовательным шагам, используя рекурсивный вызов, в котором A является источником, B назначением и C, как вспомогательная игла, затем переместить большой диск до места назначения с помощью нерекурсивного шага, и наконец, переместить n-1 дисков из вспомогательной иглы в конечный пункт назначения, используя другой рекурсивный вызов.

Важно заметить, что в этом вызове, идет перемещение n-1 дисков из B, в качестве источника, в C, в качестве места назначения, используя A в качестве вспомогательной иглы.

После того, как мы отработали стратегию, реализация рекурсивного метода решения проблемы Башен Ханоя является довольно простой.

Вот определение рекурсивного метода.



В этом варианте реализации, вместо того чтобы использовать 4 параметра, необходимы всего три параметра, num это количество дисков, from является источником и to является пунктом назначения.

Мы не нужен 4-й параметр, потому что это может быть определено внутри метода.

Если предположить, что иглы обозначены как 1, 2, и 3, это выражение здесь, где определяется расположение вспомогательной иглы.

На самом деле, как это делается, довольно умно.

Вы можете подумать о том, как это работает. Я вернусь к этому позже.

Эта часть является не рекурсивным вызовом, когда num равен 1, или есть только один диск.

Внутри кода else, три выражения представляют три ветви, которые мы видели при описании решения.

Три красных стрелки соответствуют трем выражениям, которые находятся в теле else, в том числе двум рекурсивным вызовам и не рекурсивному шагу.

Первый шаг, это рекурсивный вызов для перемещения num-1 дисков из from места в место temp.

Второе выражение должно фактически перемещать диск из верхней части from иглы в иглу назначения.

3-й выражение, это рекурсивный вызов перемещения num-1 дисков из temp места в место to.

Вы выяснили, как вспомогательная игла может быть определена, используя это выражение для temp.

Если предположить, что иглы обозначены как 1, 2 и 3, при этом 1, 2 и 3 будет составлять 6.

После того, как два из трех чисел фиксированы, мы можем определить третье путем вычитания двух известных значений, в этом случае, from и to, из 6.

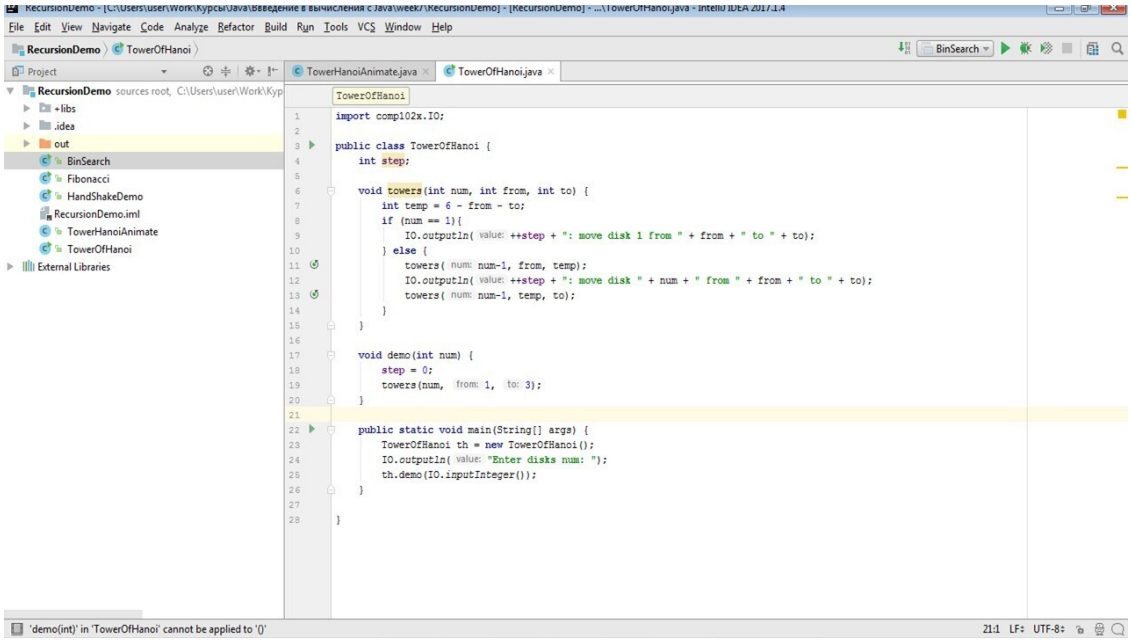
Например, если from имеет значение 1 и to имеет значение 3, вычитая 1 и 3 из 6 будет давать 2, то есть, третью иглу.

Так что это вся реализация решения Башен Ханоя, хотя это может занять 580 миллионов лет, чтобы решить проблему, сама программа очень проста.

Давайте взглянем на эту программу в IDEA.

Демонстрация задачи

Вот метод towers, который вы только что видели.

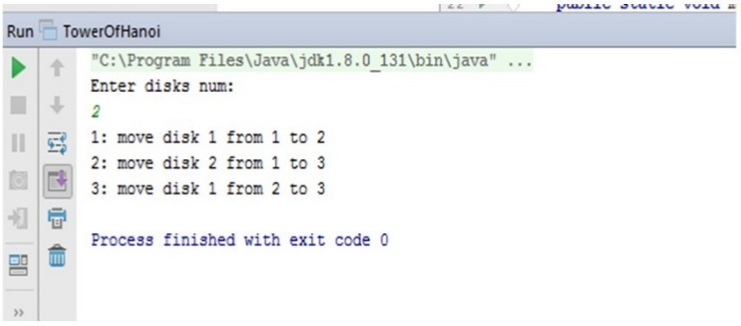


Я добавил здесь переменную step, чтобы подсчитать количество шагов для решения задачи. Так, что здесь отображаются шаги.

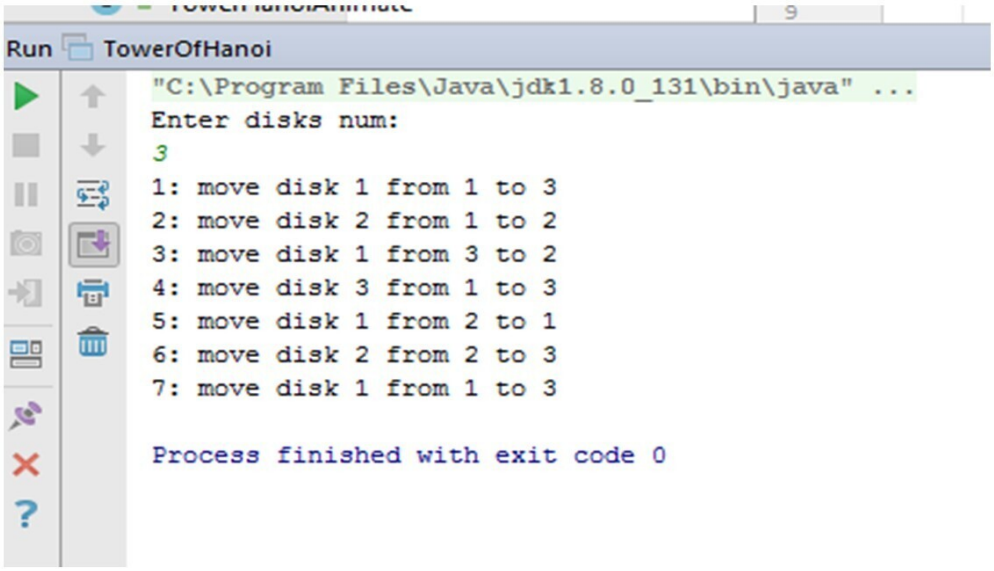
Там также есть метод demo с параметром num для установки различного количества дисков.

Скомпилируем и запустим программу.

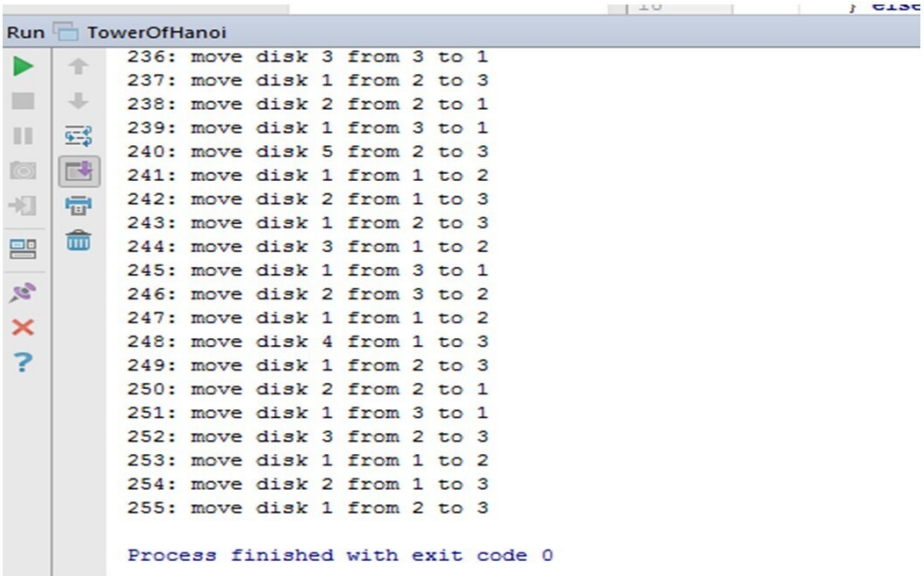
Вызовем метод demo, и давайте сначала попробуем с 2 дисками, и задача выполнена в 3 хода.



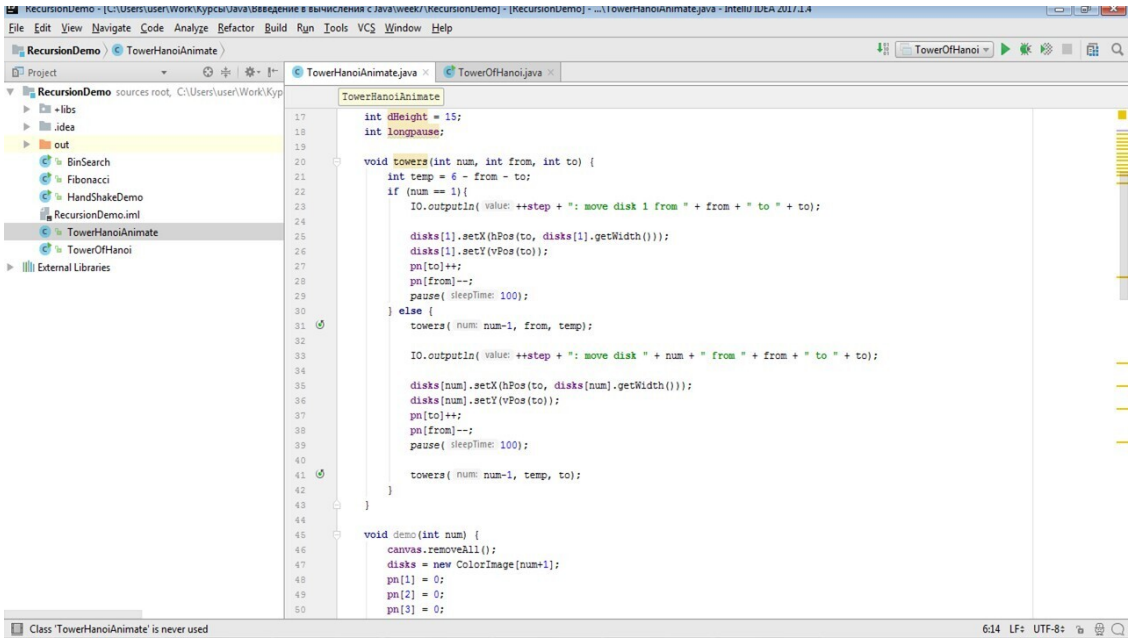
Вы можете убедиться, что эти шаги действительно правильные.
Давайте попробуем еще раз, используя 3 диска.



И решение выполняется в 7 шагов.
Давайте попробуем большее число, скажем, 8 дисков.



Были сделаны 255 или 2 в 8-й степени -1 ходов.
Но здесь трудно проверить, что эти шаги действительно правильные.
Здесь также есть анимированная версия программы Башни Ханоя, так что вы можете на самом деле визуализировать все движение.



Программа в основном такая же, как и раньше, и как вы можете видеть, здесь есть вывод для 1 диска.

И здесь есть первый рекурсивный вызов для перемещения num-1 дисков.

Не рекурсивный шаг для перемещения одного диска. И второй рекурсивный вызов для перемещения num-1 дисков снова.

Остальной код для обработки некоторого наглядного представления дисков и для создания дисков разных цветов.

Скомпилируем программу и давайте запустим программу с n равным 6.

Здесь вы можете увидеть, что программа, во-первых, переместит 5 маленьких диска в середину, и теперь, большой диск может перемещаться к месту назначения и так далее, пока задача не будет завершена.

Не пытайтесь запускать программу с n больше, чем 15, это будет длиться вечно, пока она не завершится.

Фракталы

Я хотел бы кратко рассказать о фракталах. Некоторые из вас, возможно, видели фрактальные изображения.

Фракталом является математическое множество, которое отображает самоподобные узоры, то есть, фракталы отображаются одинаково или почти одинаково, в различных масштабах или ориентациях.

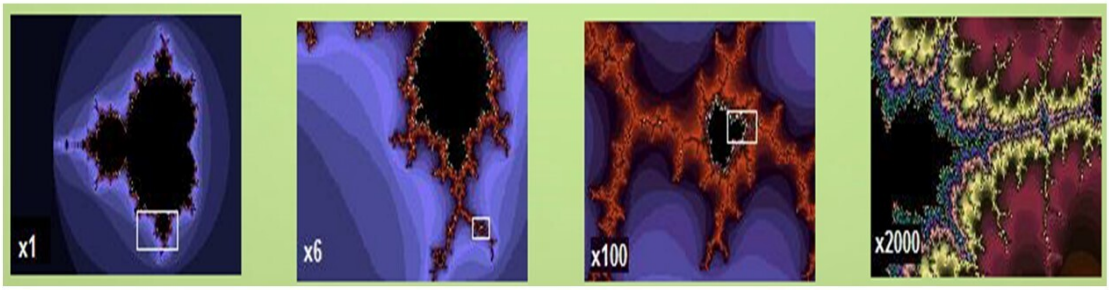
Рекурсивные функции также описывают процессы, которые самоподобны.

Вы увидите, что многие фрактальные узоры могут быть получены с использованием рекурсивных функций.

Термин «фрактал» был впервые введен Мандельбротом, математиком в 1975 году.

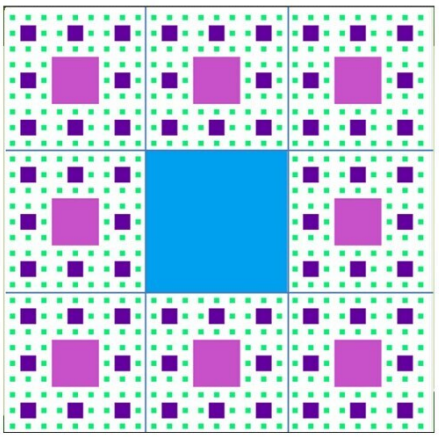
Множество комплексных чисел, которое называется множеством Мандельброта, можно визуализировать с помощью набора фрактальных изображений.

Построив эти числа на комплексной плоскости, используя различные цвета, в зависимости от того принадлежат ли они множеству, и как далеко они находятся от множества, красивые фрактальные изображении могут быть получены в разных масштабах.



Вы можете увидеть в этих изображениях, что аналогичные шаблоны встречаются в разных масштабах и ориентациях.

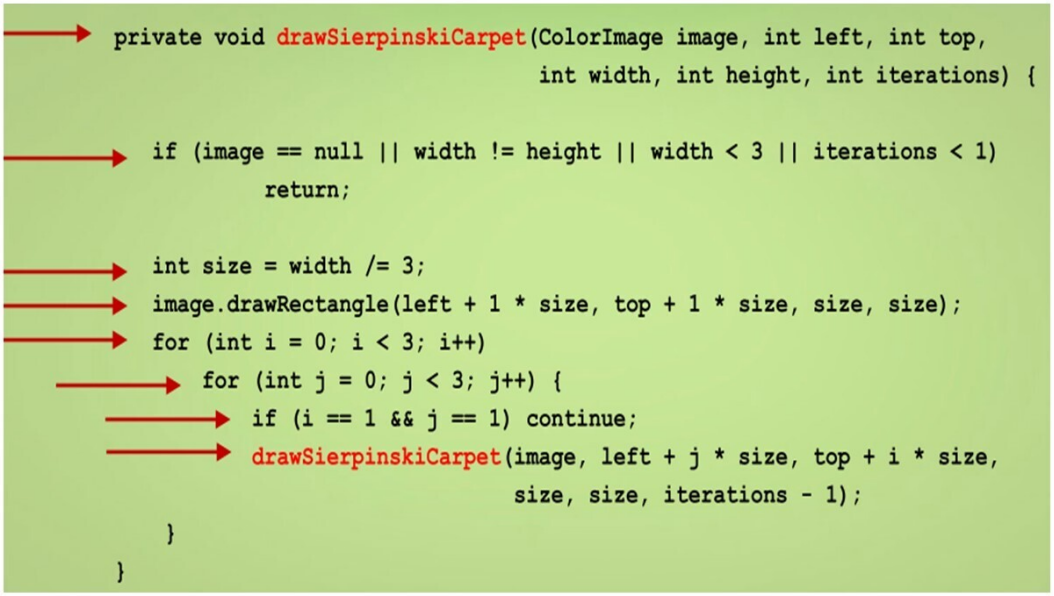
То, что я хочу показать, это простой фрактал, который называется ковром Серпинского.



Ковер Серпинского создается путем создания квадрата, а затем делением его на девять меньших квадратов одинакового размера, то есть с помощью матрицы 3x3, удалив центральный квадрат, поочередно, вы можете покрасить его другим цветом.

Процесс затем будет повторяться для восьми окружающих квадратов.

Последний шаг здесь будет, это сделать каждый из меньших квадратов мини-версией большего ковра Серпинского, то есть, ковер Серпинского определяется рекурсивно в терминах 8 уменьшенных вариантов самого себя.



```
private void drawSierpinskiCarpet(ColorImage image, int left, int top,
                                  int width, int height, int iterations) {

    if (image == null || width != height || width < 3 || iterations < 1)
            return;

    int size = width /= 3;
    image.drawRectangle(left + 1 * size, top + 1 * size, size, size);
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++) {
            if (i == 1 && j == 1) continue;
            drawSierpinskiCarpet(image, left + j * size, top + i * size,
                                 size, size, iterations - 1);
        }
    }
```

Вот реализация рекурсивного метода генерации ковра Серпинского.

Метод называется drawSierpinskiCarpet с 6 параметрами.

Во-первых, это изображение на котором ковер Серпинского должен быть изображен.

Параметры left и top дают расположение квадрата, с которым в настоящее время идет работа.

Параметры width и height дают размер текущего квадрата, и iterations дает глубину рекурсивных вызовов, которые должны быть сделаны.

Выражение if для проверки того, что изображение на самом деле существует, ширина не равна высоте означает, что квадрат не является квадратом, ширина <3 означает, что квадрат не может быть разделен на 3x3, и итерации меньше 1 означает, что все рекурсивные вызовы были завершены.

Ничего не будет сделано, если какое-либо из этих условий верно.

Размер меньших квадратов получается путем деления ширины на 3, то есть генерацией матрицы 3x3.

Метод drawRectangle для класса ColorImage рисует прямоугольник на изображении, в месте, указанном первыми двумя параметрами.

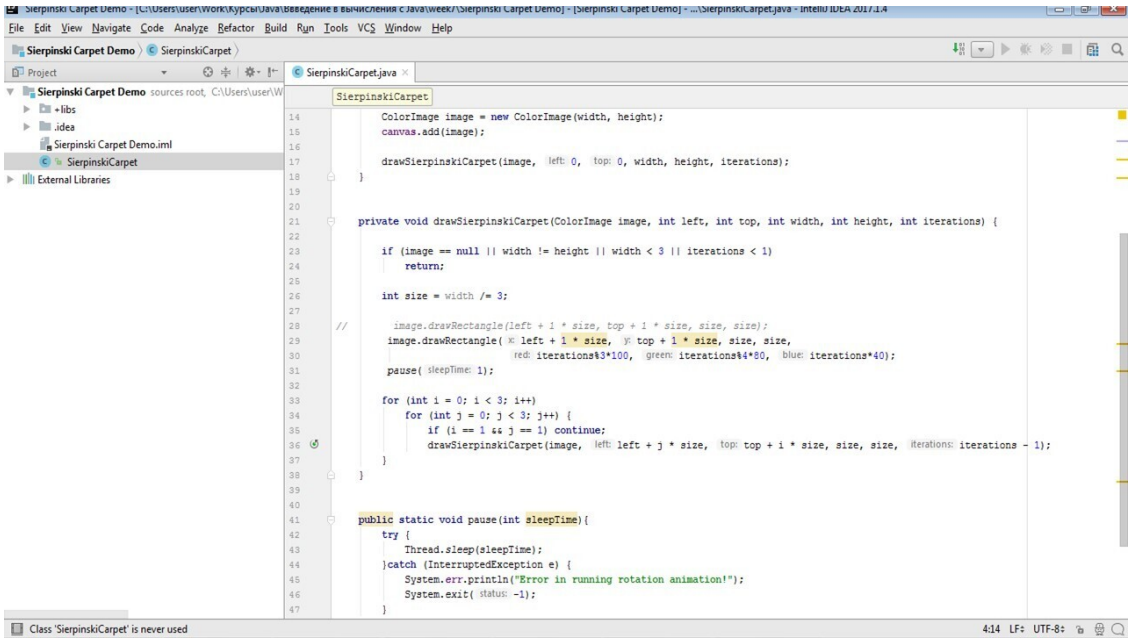
И третий, и четвёртый параметры определяют размер прямоугольника, который рисуется.

В этом случае, это квадрат с размером в количество пикселей.

Вложенные циклы здесь для рисования 8 мини ковров Серпинского, делая рекурсивные вызовы метода drawSierpinskiCarpet.

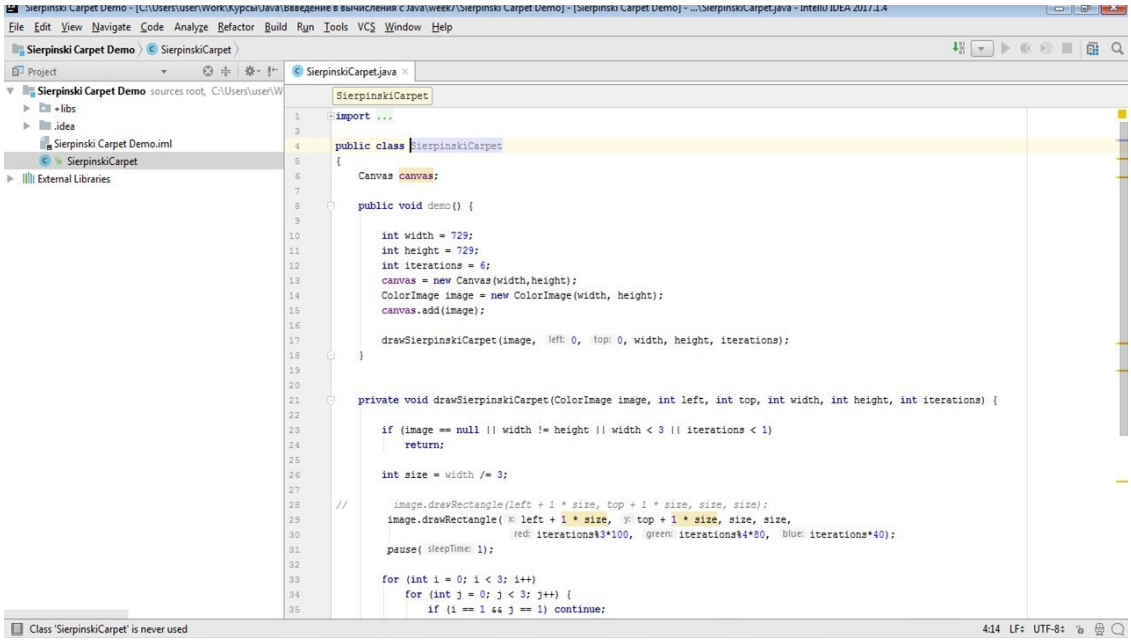
Обратите внимание, что if выражение, чтобы пропустить средний квадрат, который индексируется как 1,1.

Откроем IDEA, чтобы посмотреть демонстрацию программы.



Это тот же метод drawSierpinskiCarpet, который мы только что видели.

Я добавил метод demo здесь, чтобы настроить холст, ColorImage и параметры для метода drawSierpinskiCarpet.

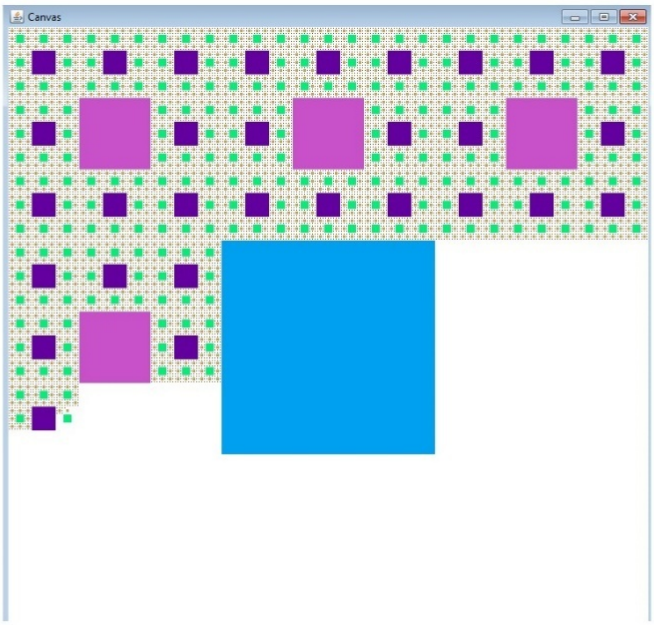


Ширина и высота устанавливаются в 729, что является 3 в степени 6.

Число итераций установлено на 6, так что в 6 итераций, размер площади будет снижен до единицы.

Давайте скомпилируем программу и создадим экземпляр SierpinskiCarpet, и запустим метод demo.

Здесь создается ковер Серпинского. Это выглядит захватывающим.



Здесь метод drawRectangle для ColorImage вызывается таким образом, что квадраты рисуются различными цветами.

```
private void drawSierpinskiCarpet(ColorImage image, int left, int top, int width, int height, int iterations) {

    if (image == null || width != height || width < 3 || iterations < 1)
        return;

    int size = width /= 3;

      image.drawRectangle(left + 1 * size, top + 1 * size, size, size);
     image.drawRectangle( x: left + 1 * size,  y: top + 1 * size, size, size,
                          red: iterations%3*100,  green: iterations%4*80,  blue: iterations*40);
     pause( sleepTime: 1);

    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++) {
            if (i == 1 && j == 1) continue;
            drawSierpinskiCarpet(image,  left: left + j * size,  top: top + i * size, size, size,  iterations: iterations - 1);
    }
}
```

Три дополнительные параметры здесь указывают цвет прямоугольника, который рисуется.

Они будут меняться для различных итераций. Вы можете попробовать различные цвета, если вы захотите.

Здесь также замедляется выполнение программы, используя паузу, так что вы можете увидеть программу в действии.

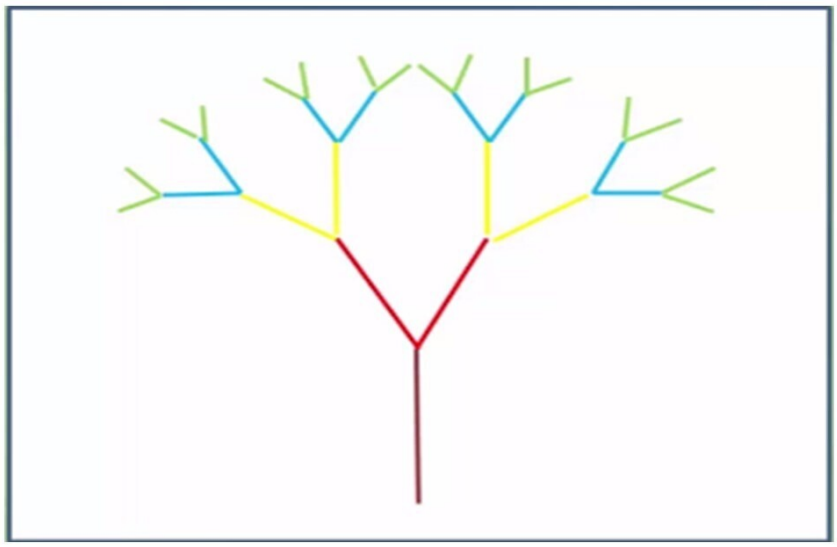
Таким образом, здесь вы можете увидеть рекурсивные вызовы в действии.

Фрактальное дерево

Я хочу показать вам еще один пример использования рекурсии для генерации фрактальных изображений.

Существует множество фракталов, называемых фрактальными деревьями.

Проиллюстрируем идею фрактального дерева с помощью следующей диаграммы.



Изображение начинается со ствола дерева, к нему могут быть добавлены ветви.

Ветви добавляются симметрично на определенный угол, равностоящий от текущей ветви.

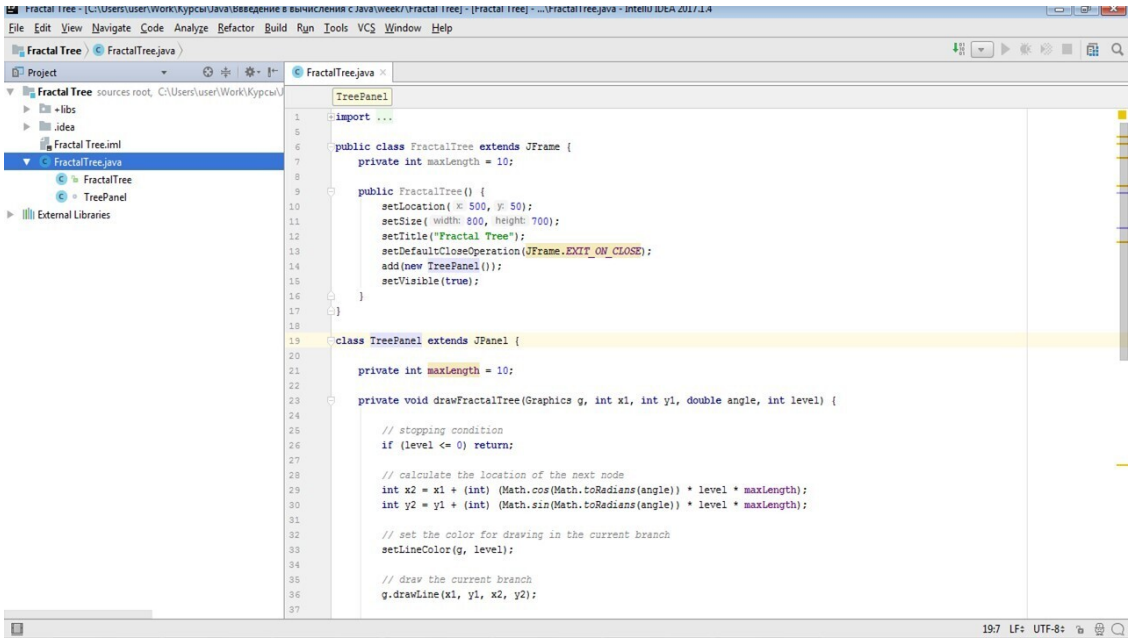
Новые ветви добавляются сокращенной длины.

Ветви дерева могут иметь такой же цвет или различные цвета.

Таким образом, любая часть дерева, начинающаяся от ветви, похожа на другие ветви.

Такое самоподобие дает фрактальное дерево.

Давайте посмотрим программу, реализующую фрактальное дерево.



```
import ...

public class FractalTree extends JFrame {
    private int maxLength = 10;

    public FractalTree() {
        setLocation( x: 500, y: 50);
        setSize( width: 800, height: 700);
        setTitle("Fractal Tree");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        add(new TreePanel());
        setVisible(true);
    }
}

class TreePanel extends JPanel {

    private int maxLength = 10;

    private void drawFractalTree(Graphics g, int x1, int y1, double angle, int level) {

        // stopping condition
        if (level <= 0) return;

        // calculate the location of the next node
        int x2 = x1 + (int) (Math.cos(Math.toRadians(angle)) * level * maxLength);
        int y2 = y1 + (int) (Math.sin(Math.toRadians(angle)) * level * maxLength);

        // set the color for drawing in the current branch
        setLineColor(g, level);

        // draw the current branch
        g.drawLine(x1, y1, x2, y2);
```

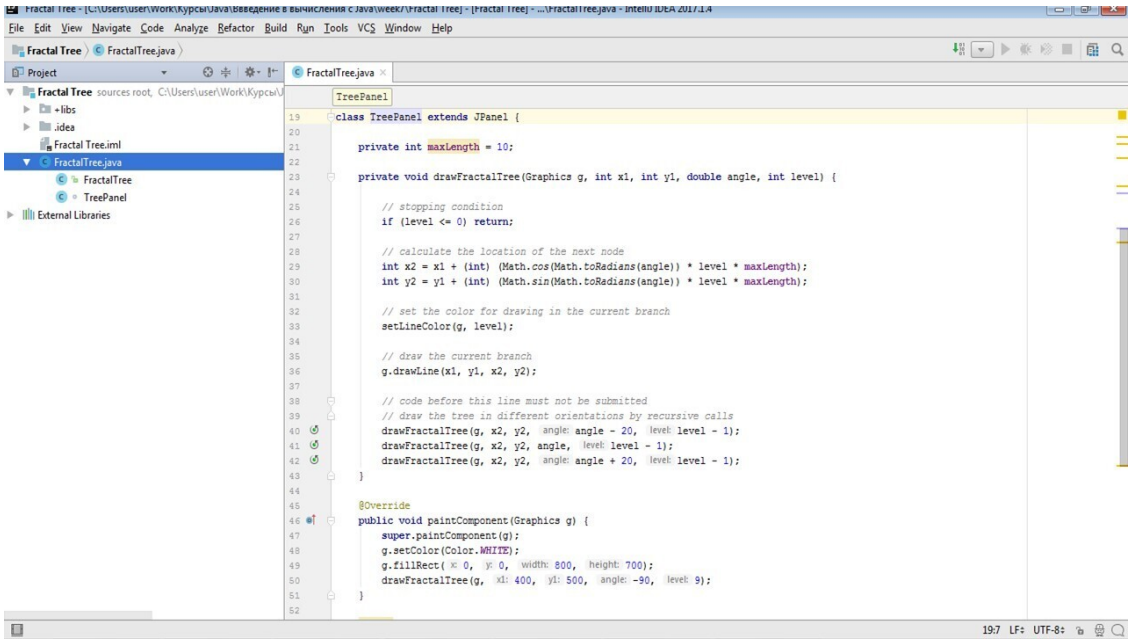
Класс называется FractalTree который расширяет JFrame.

Конструктор здесь настраивает JFrame.

Основная задача программы делается этим методом drawFractalTree.

Первый параметр является объектом класса Graphics.

Graphics является абстрактным классом для всех графических объектов, отображаемых графическими компонентами, x1 и y1 дают начальное положение ветви, которая рисуется, angle это угол отклонения от текущей ветви, level это текущий уровень рекурсии.

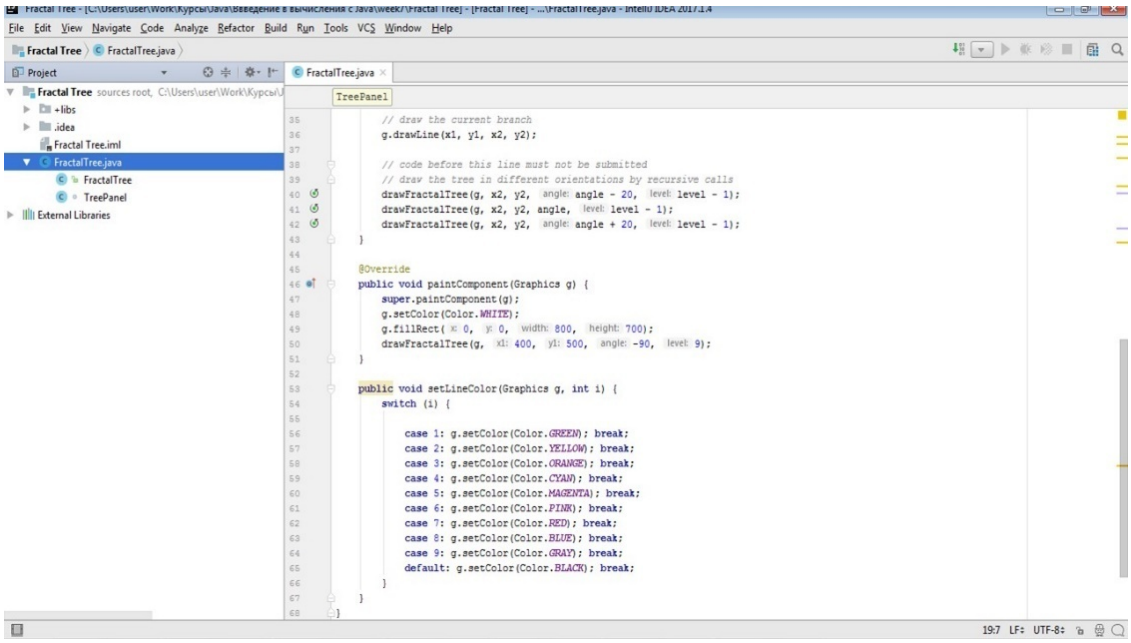


```
class TreePanel extends JPanel {

    private int maxLength = 10;

    private void drawFractalTree(Graphics g, int x1, int y1, double angle, int level) {

        // stopping condition
        if (level <= 0) return;

        // calculate the location of the next node
        int x2 = x1 + (int) (Math.cos(Math.toRadians(angle)) * level * maxLength);
        int y2 = y1 + (int) (Math.sin(Math.toRadians(angle)) * level * maxLength);

        // set the color for drawing in the current branch
        setLineColor(g, level);

        // draw the current branch
        g.drawLine(x1, y1, x2, y2);

        // code before this line must not be submitted
        // draw the tree in different orientations by recursive calls
        drawFractalTree(g, x2, y2,  angle: angle - 20,  level: level - 1);
        drawFractalTree(g, x2, y2, angle,  level: level - 1);
        drawFractalTree(g, x2, y2,  angle: angle + 20,  level: level - 1);
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.WHITE);
        g.fillRect( x: 0,  y: 0,  width: 800,  height: 700);
        drawFractalTree(g,  x1: 400,  y1: 500,  angle: -90,  level: 9);
    }
```

Здесь есть не рекурсивный шаг, который ничего не делает, когда уровень меньше или равен 0.

Два выражения здесь определяют конечную точку для новой ветви, которая составляет угол от текущей ветви.

Метод setLineColor задает цвет на разных уровнях.



```
        // draw the current branch
        g.drawLine(x1, y1, x2, y2);

        // code before this line must not be submitted
        // draw the tree in different orientations by recursive calls
        drawFractalTree(g, x2, y2, angle: angle - 20, level: level - 1);
        drawFractalTree(g, x2, y2, angle, level: level - 1);
        drawFractalTree(g, x2, y2, angle: angle + 20, level: level - 1);
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.WHITE);
        g.fillRect( x: 0, y: 0, width: 800, height: 700);
        drawFractalTree(g, x1: 400, y1: 500, angle: -90, level: 9);
    }

    public void setLineColor(Graphics g, int i) {
        switch (i) {

            case 1: g.setColor(Color.GREEN); break;
            case 2: g.setColor(Color.YELLOW); break;
            case 3: g.setColor(Color.ORANGE); break;
            case 4: g.setColor(Color.CYAN); break;
            case 5: g.setColor(Color.MAGENTA); break;
            case 6: g.setColor(Color.PINK); break;
            case 7: g.setColor(Color.RED); break;
            case 8: g.setColor(Color.BLUE); break;
            case 9: g.setColor(Color.GRAY); break;
            default: g.setColor(Color.BLACK); break;
        }
    }
}
```

Метод drawLine будет рисовать линию в JFrame.

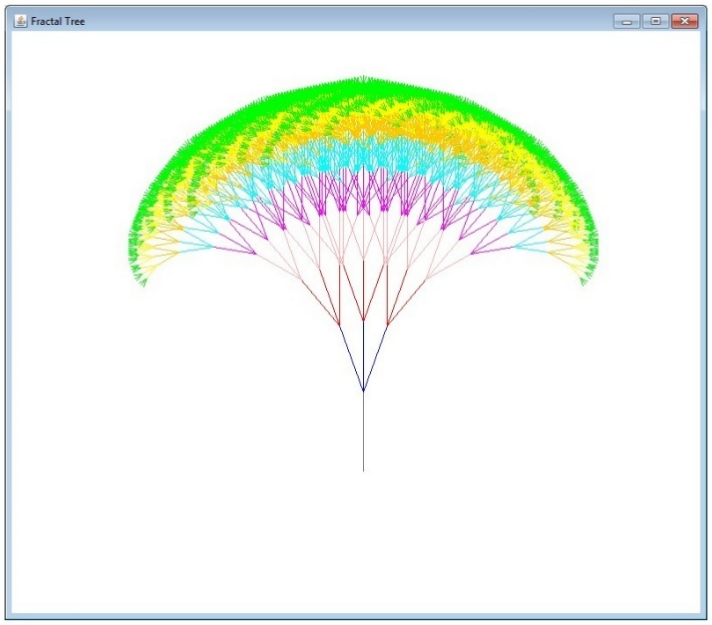
Ниже главная часть программы, которая делает рекурсивные вызовы drawFractalTree.

Два вызова будут сделаны с углами, равностоящими от текущей ветви.

Уровень уменьшается на единицу, так что не рекурсивный шаг будет выполнен, когда уровень в конце концов уменьшится до нуля.

Метод drawFractalTree вызывается здесь с начальной позицией, угол -90 создаст ствол, который растет прямо вверх, максимальный уровень установлен на 9.

Давайте посмотрим на выполнение программы по созданию экземпляра фрактального дерева.



Вы можете видеть, что это дерево, как структура с ветвями, симметричными от текущей ветви.

Вы на самом деле можете создать больше ветвей, сделав дополнительные рекурсивные вызовы метода drawFractalTree.

Например, здесь добавлена дополнительная ветвь, которая растет прямо из текущей ветви.

```
private void drawFractalTree(Graphics g, int x1, int y1, d
    if (level <= 0) return;

    int x2 = x1 + (int) (Math.cos(Math.toRadians(angle)) *
    int y2 = y1 + (int) (Math.sin(Math.toRadians(angle)) *
    setLineColor(g, level);
    g.drawLine(x1, y1, x2, y2);

    drawFractalTree(g, x2, y2, angle - 20, level - 1);
    drawFractalTree(g, x2, y2, angle, level - 1);
    drawFractalTree(g, x2, y2, angle + 20, level - 1);

}
```

Программа запускается, путем создания нового экземпляра FractalTree.

```
FractalTree  main()
7       private int maxLength = 10;
8
9       public FractalTree() {
10          setLocation( x: 500, y: 50);
11          setSize( width: 800, height: 700);
12          setTitle("Fractal Tree");
13          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14          add(new TreePanel());
15          setVisible(true);
16      }
17
18      public static void main(String[] args) {
19          FractalTree ft = new FractalTree();
20      }
21  }
22
23  class TreePanel extends JPanel {
24
25      private int maxLength = 10;
26
27      private void drawFractalTree(Graphics g, int x1, int y1, double angle, int level) {
```

Далее в задаче, вы должны изменить программу, так что четыре ветви будут генерироваться из каждой ветви.

Вопросы

Задача

Учитывая метод f1() ниже, что возвращает метод f1(4)?

```
public static int f1(int n) {
if (n == 1)
return 1;
return n + f1(n – 1);
}
```

Варианты ответа:

1. 13
2. 1
3. 10
4. 7

Ответ: 3.

Объяснение.

Рекурсивный метод f1() вычисляет сумму от 1 до входного параметра n, для любого натурального n. Таким образом, f1(4) дает результат 1 + 2 + 3 + 4 = 10.

Задача

Учитывая метод f2() ниже, что возвращает метод f2(3, 2)?

```
public static int f2(int a, int b) {
if (b >= 1)
return f2(a + 1, b – 1);
else
return a;
}
```

Варианты ответа:

1. 3
2. 5
3. Ошибка – бесконечный цикл
4. 2

Ответ: 2.

Объяснение.

Рекурсивный метод f2() принимает два целочисленных параметра a и b. В каждом рекурсивном вызове, если b больше 0, a увеличивается на 1, пока b уменьшается на 1. Рекурсивный вызов продолжается до тех пор пока b не равен 0. Таким образом, величина b добавляется к a этим рекурсивным методом. Величина a возвращается методом в конце. Таким образом, f2(3, 2) возвращает значение 3 + 2 = 5.

Задача

Учитывая метод f3() ниже, что возвращает метод f3(10200)?

```
public static int f3(int n) {
if (n == 0)
return 1;
else if (n < 10 && n > -10)
return 0;
else
return f3(n / 10) + f3(n % 10);
}
```

Варианты ответа:
1. 10200
2. 3
3. 12
4. 0
Ответ: 2.
Объяснение.

Рекурсивный метод f3() определяет количество нулей в заданном целом. Если входной параметр равен 0, то возвращается 1. Если входной параметр превышает -10 и меньше 10, то метод возвращает 0. В противном случае, он возвращает сумму результатов рекурсивного вызова для числа n с последней удаленной цифрой и рекурсивным вызовом для последней цифры. Таким образом, f3(10200) возвращает значение 3.

Задача

Теперь, когда вы узнали о рекурсии, каким будет вывод для каждого из рекурсивного метода ниже с данным входом?

Рекурсивный метод 1

```
int f1(int n) {
if (n > 1)
return f1(n – 1) + f1(n – 2);
else
return 1;
}
```

f1(3) = ?

Рекурсивный метод 2

```
int f2(int a, int b) {

if (b == 0)
return 1;
else
return a * f2(a, b – 1);
}
```

f2(2, 3) = ?

Рекурсивный метод 3

```
String f3(String s, int length) {
if (length == 0)
return "";
return s.charAt(length – 1) + f3(s, length – 1);
}
```

f3("tsukh", 5) = ?

Рекурсивный метод 4

```
String f4(int n) {
if (n == 0) return "0";
if (n == 1) return "1";
return f4(n / 2) + (n % 2);
}
```

f4(13) = ?

Рекурсивный метод 5

```
String f5(int n) {
if (n == 0) return "0";
```

```
return n + f5(n – 1) + n;
}
```

f5(3) = ?

Ответ:

Q1	int f1(int n) {
	
	if (n > 1)
	return f1(n - 1) + f1(n - 2);
	else
	return 1;
	}

Q2	int f2(int a, int b) {
	if (b == 0)
	return 1;
	else
	return a * f2(a, b - 1);
	}

Q3	String f3(String s, int length) {
	
	if (length == 0)
	return "";
	
	return s.charAt(length - 1) + f3(s, leng
	}

Задача

В этом упражнении вы должны изменить рекурсивный метод drawFractalTree() в классе TreePanel, так что фрактальное дерево с четырьмя ветвями в каждом узле будет изображено.

Инструкции:

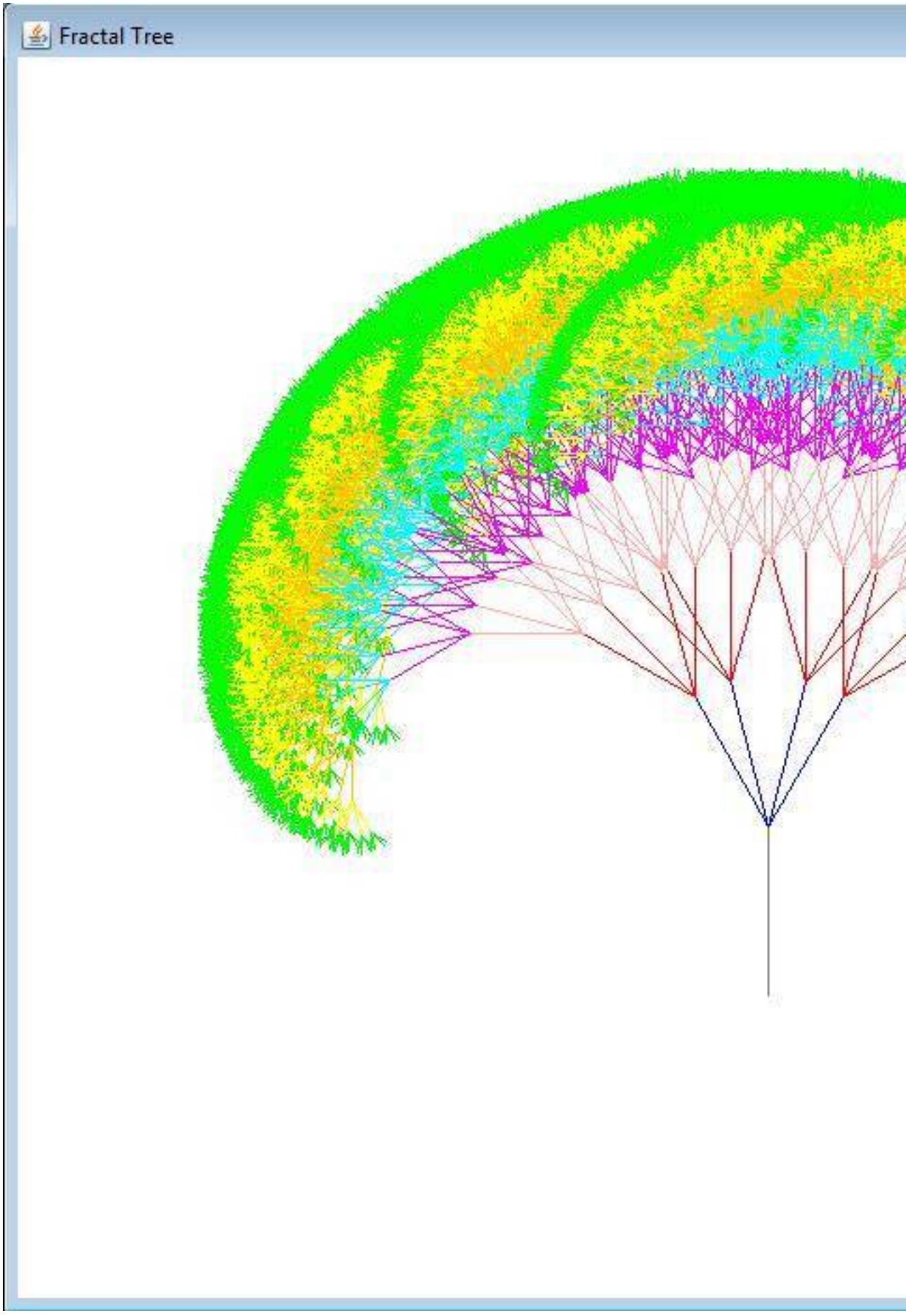
1. Запустить программу, создать новый экземпляр FractalTree.

2. Текущая реализация метода drawFractalTree() рисует дерево с тремя ветвями в каждом узле, ориентированными на 20 градусов, 0 градусов и -20 градусов по отношению к узлу, используя три рекурсивных вызова.

3. Изменить реализацию метода таким образом, что он рисует дерево с четырьмя ветвями в каждом узле, ориентированными под углом 30 градусов, 15 градусов, -15 градусов и -30 градусов по отношению к узлу, используя четыре рекурсивных вызова.

4. Вам нужно будет изменить четвертый параметр для каждого из четырех рекурсивных вызовов метода drawFractalTree(), чтобы выполнить эту задачу.

Пример вывода:


Fractal Tree

Ответ:

```
drawFractalTree(g, x2, y2, angle – 15, level – 1);
drawFractalTree(g, x2, y2, angle + 15, level – 1);
drawFractalTree(g, x2, y2, angle – 30, level – 1);
drawFractalTree(g, x2, y2, angle + 30, level – 1);
```

Абстрактные типы данных

Теперь, давайте рассмотрим такую тему как абстрактный тип данных или ADT.

Вы сейчас должны быть хорошо знакомы с различными видами типов данных, в том числе примитивными типами данных, такими как int для целых чисел и double для чисел с плавающей точкой.

Мы также обсудили массив, как простую структуру данных, которая облегчает хранение и извлечение данных одного и того же типа.

Когда мы используем тип данных, важно думать не только о данных, которые этот тип представляет, важно также учитывать операции, которые связаны с этими данными.

Например, когда мы используем тип данных int, сами данные были бы не слишком полезны без арифметических операций, таких как сложение, вычитание, умножение и деление.

Когда мы используем тип String, методы, такие как charAt, compareTo и substring часто используются для манипулирования строкой символов, которую этот тип представляет.

В общем, абстрактный тип данных (ADT) представляет собой структуру данных, которая определяет характеристики набора данных и операции, которые можно выполнять с набором данных.

Абстрактные типы данных

Абстрактный тип данных (Abstract Data Type) – это тип данных, который предоставляет набор функций для работы с его элементами.

Клиентские программы не имеют доступа к внутренней реализации типа данных (реализация сокрыта – инкапсулирована).

Тип данных переменной (Data Type) – множество значений, которые она может принимать (`int`, `char`, `double`, `float`).

Но подробности о том, как данные представлены, и реализация скрыты от пользователей.

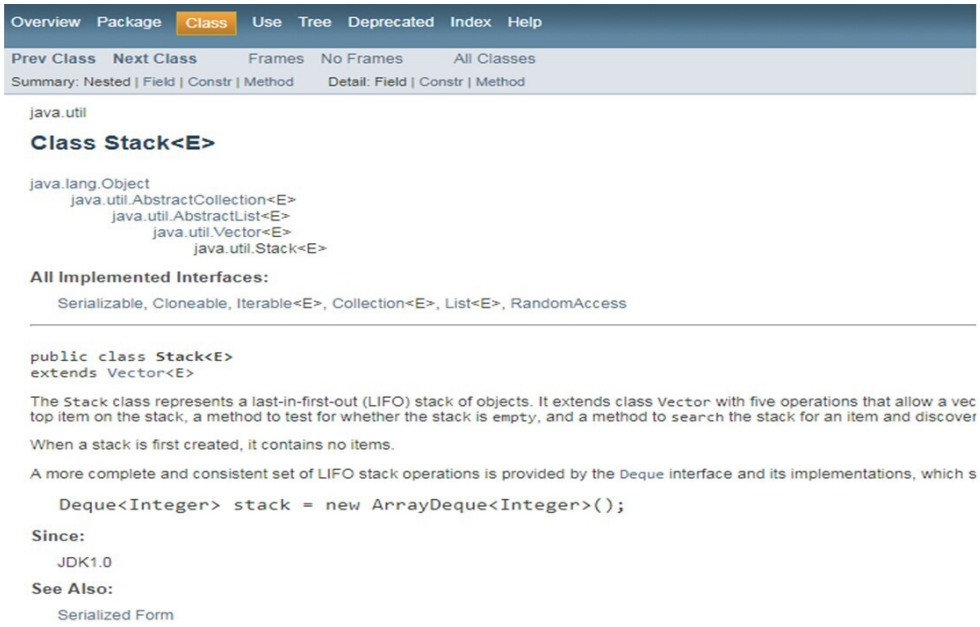
Например, когда мы используем строку символов, мы не должны знать, как строка символов представлена внутренне и как метод charAt реализуется, когда он возвращает char значение выбранной позиции строки, как указано в спецификации метода charAt.

Таким образом, ключевой концепцией ADT является то, что он скрывает детали реализации от пользователей.

Ключевыми преимуществами в использовании ADT является то, что он делает проще программирование и нам не нужно повторно реализовать тип данных каждый раз, когда он нам нужен.

Что еще более важно, любые изменения в конкретной реализации ADT не влияют на использование типа данных, поэтому важно, что, когда используется ADT, мы не должны рассматривать какие-либо специальные функции реализации.

Я буду использовать ADT, известный как стек, чтобы проиллюстрировать полезность ADT.



Overview Package Class Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Class Stack<E>

java.lang.Object
java.util.AbstractCollection<E>
java.util.AbstractList<E>
java.util.Vector<E>
java.util.Stack<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

```
public class Stack<E>
extends Vector<E>
```

The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class Vector with five operations that allow a vec
top item on the stack, a method to test for whether the stack is empty, and a method to search the stack for an item and discover

When a stack is first created, it contains no items.

A more complete and consistent set of LIFO stack operations is provided by the Deque interface and its implementations, which s

```
Deque<Integer> stack = new ArrayDeque<Integer>();
```

Since:

JDK1.0

See Also:

Serialized Form

Можно придумать множество примеров из реальной жизни, которые используют ситуации со стеком.

Например, стопка монет может храниться в денежном разменнике. Или мы можем увидеть штабеля книг в библиотеке или книжных магазинах.

Когда мы идем в столовую, то вынимаем пищевые лотки из стопки. И мы вынимаем корзину для покупок в супермаркете.

Вы можете не знать об этом, но стеки часто используются в компьютерных операциях, когда мы отменяем операции, которые мы сделали в программном приложении, стек используется для хранения последовательности операций, и последняя выполненная операция будут находиться на вершине стека, так что она будет первой для отмены.

Когда мы используем веб-браузер, список сайтов, которые мы посетили, может храниться в стеке, так что, когда нажимаешь на кнопку "назад", можно вернуться к последнему сайту, на котором побывали.

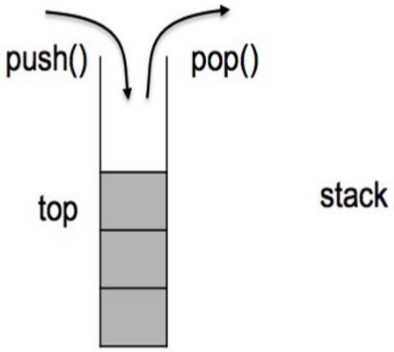
Мы также упоминали, что вызовы методов хранятся в стеке программы таким образом, что могут сохранятся локальные переменные.

Для всех этих примеров можно увидеть, что одна общая характеристика стека – это то, что добавление и удаление записей может осуществляться только в верхней части стека.

Так как добавление и удаление записей может быть сделано только на вершине стека, последняя запись, которая добавляется в стек, всегда будет первой записью, которая должны быть удалена из стека.

Стек часто называют last-in-first out структурой данных (LIFO).

LIFO Last In, First Out – последним пришёл — первым ушёл



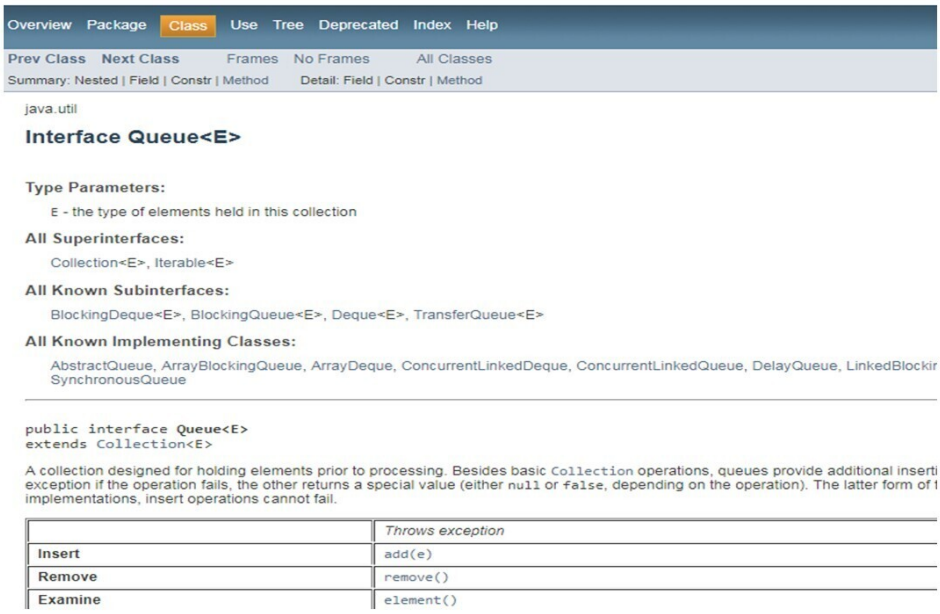
Две важные операции, которые поддерживаются ADT стеком, это операции push и pop.

Операция push добавляет запись на вершину стека и pop является операцией по удалению записи из верхней части стека.

Обратите внимание, что, если задача требует вставки записи в середину набора или удаления записи из нижней части набора, стек не будет соответствующим типом данных для данной задачи.

Другие ADT также часто доступны в большинстве объектно-ориентированных языков программирования.

Другой ADT, похожий на стек, называется очередью.



Overview Package Class Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.util

Interface Queue<E>

Type Parameters:

E - the type of elements held in this collection

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Subinterfaces:

BlockingDeque<E>, BlockingQueue<E>, Deque<E>, TransferQueue<E>

All Known Implementing Classes:

AbstractQueue, ArrayBlockingQueue, ArrayDeque, ConcurrentLinkedDeque, ConcurrentLinkedQueue, DelayQueue, LinkedBlockir
SynchronousQueue

```
public interface Queue<E>
extends Collection<E>
```

A collection designed for holding elements prior to processing. Besides basic `Collection` operations, queues provide additional inserti
exception if the operation fails, the other returns a special value (either `null` or `false`, depending on the operation). The latter form of t
implementations, insert operations cannot fail.

	Throws exception
Insert	`add(e)`
Remove	`remove()`
Examine	`element()`

В отличие от стека, добавление записей в очереди, можно проводить только в хвосте или в конце очереди.

Это похоже на то, когда вы выстраиваетесь купить билет или ждать своей очереди в супермаркете,

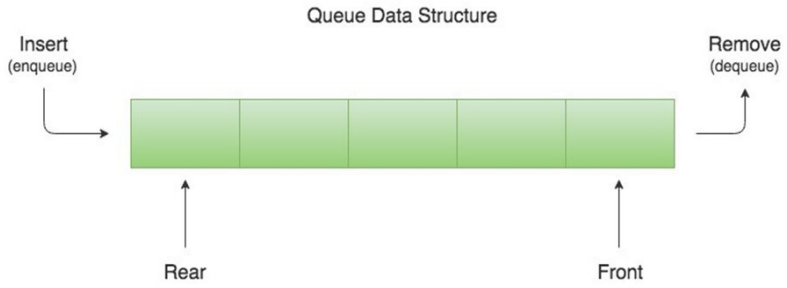
Вы должны встать в конец очереди.

Если же вы встанете в начало или в середину очереди, я уверен, что на вас будут кричать все другие, которые уже стоят в очереди.

Удаление же записей, подобно стеку, может быть осуществлено только в начале очереди.

Очередь часто называют структурой данных FIFO, так как элементы, которые должны быть удалены из очереди, будут основываться на их порядке в очереди.

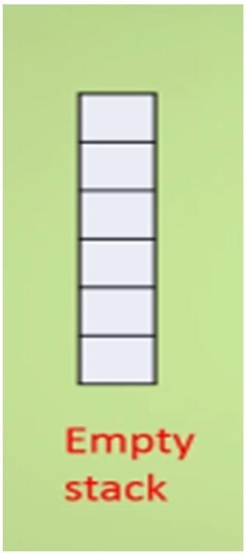
FIFO first in, first out — «первым пришёл — первым ушёл»



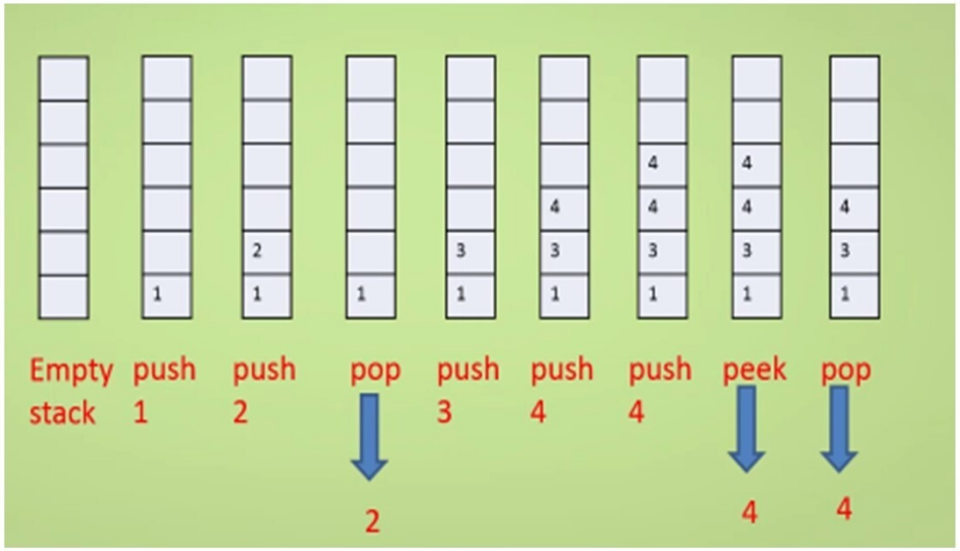
Здесь две широко используемые операции, это add для добавления объекта в конец очереди и remove для удаления объекта из начала очереди.

Пример стека

Рассмотрим некоторые примеры, иллюстрирующие использование push и pop для стека. Предположим, что стек целых чисел уже создан.



Изображение здесь, кажется, указывает, что стек фиксированного размера. На самом деле стек фактически реализован с использованием динамического размера.



Операция push 1 разместит значение 1 в пустой стек, push 2 поместит значение 2 сверху 1.

Операция pop, которая не принимает никаких аргументов, вытолкнет верхний элемент из стека, который несет значение 2, push 3 добавит 3 на вершину стека, push 4 добавит еще один элемент 4 на вершину стека.

Мы можем добавить еще один элемент с тем же самым значением, что и предыдущие элементы в стеке, в этом случае, push 4 поставит 4 поверх другого 4.

Операция peek, которая не принимает никаких аргументов, будет возвращать значение на вершине стека, то есть 4, не выталкивая верхний элемент из стека, pop удаляет, а также возвращает значение на вершину стека, в данном случае, со значением 4.

Давайте рассмотрим простой пример с использованием стека для преобразования десятичного числа в двоичное число.

Первоначальный подход для проведения такого преобразования может быть описан следующим образом.

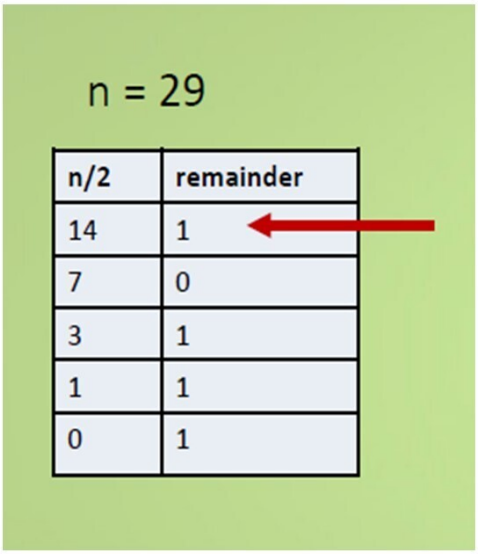
Для данного числа n повторяется следующий процесс:

– Находится остаток от деления на 2.

– Остаток выводится.

– n обновляется до n/2.

Мы хотим сначала найти остаток от деления числа на два.



Например, дается число 29, и первый остаток будет 1.

n затем обновляется на n/2 с использованием целочисленного деления, в данном случае, 29/2 приведет к 14.

Продолжая процесс, следующий остаток 0, и далее следуют три 1.

Результат, полученный данным процессом, будет 10111, если остатки выводятся в порядке их создания.

Но правильный ответ должен быть 11101.

Так или иначе, мы должны хранить промежуточные результаты перед их выводом.

После того как мы закончили конверсию, то результаты будут выводиться в обратном порядке.

То есть, последний остаток будет выводиться первым, что предполагает, что стек будет соответствующим представлением для этой задачи.

Предложенный подход будет применить push к остаткам для занесения в стек, а затем выводить результат, удаляя запись по одной из вершины стека.

```
import java.util.Stack;
public class ToBinary {

public static Stack<Integer> s = new Stack<Integer>();

   public void outputBinary(int n) {
   while (n > 0) {
      int bit = n%2;
      s.push(Integer.valueOf(bit));
      n = n/2;
   }
   while (!s.empty()) {
        int bit = s.pop().intValue();
        System.out.print(bit);
   }
   System.out.println("");
   }
}
```

Вот реализация этой идеи, использования стека для вывода десятичного числа в двоичном виде.

Изначально настроен пустой стек целочисленного типа.

Двоичное число n, которое должно быть преобразовано, вводится в метод outputBinary в качестве параметра.

В этом while цикле, остатки n вычисляются и помещаются в переменную bit.

Остаток вносится в целочисленный стек S.

n затем заменяется на n/2 с использованием целочисленного деления.

Обратите внимание, что мы явно обеспечиваем объект Integer как оболочку целого типа с помощью метода valueOf для класса Integer.

Когда while цикл завершается, стек должен содержать список остатков в обратном порядке, как они были получены.

Второй while цикл будет удалять остатки по одному, сверху с помощью метода pop для стека S и присваивать bit.

Каждый из остатков, удаленных из стека, будет выводиться с помощью System.out.print.

Опять же, мы преобразовываем объект оболочки целого типа в целый тип, используя метод intValue.

Вот еще один вариант предыдущего примера.

```
import java.util.Stack;
public class ToBinary {

public static Stack<Integer> s = new Stack<Integer>();

   public void outputInBinary(int n) {
   while (n > 0) {
      int bit = n%2;
      s.push(bit);  //autoboxing will convert this to s.push(Integer.valueOf(bit));
      n = n/2;
   }
   while (!s.empty()) {
        int bit = s.pop( );  //unboxing will convert this to int bit = s.pop().intValue();
        System.out.print(bit);
   }
   System.out.println("");
   }
}
```

Единственное различие заключается в том, что тип данных для внесения в стек или удаления из стека не был преобразован в объект Integer явно.

Здесь, мы можем просто использовать bit вместо целого объекта в качестве параметра для метода push, потому что механизм Java автоупаковки преобразует bit, параметр push, в "Integer.valueOf (bit)" во время компиляции.

Точно так же, для присвоения s.pop(), которое, как предполагается, возвращает объект Integer, автораспаковка будет конвертировать левую часть в s.pop(). intValue() во время компиляции.

Таким образом, обе программы будут эффективно вычислять то же самое.

Пример задачи

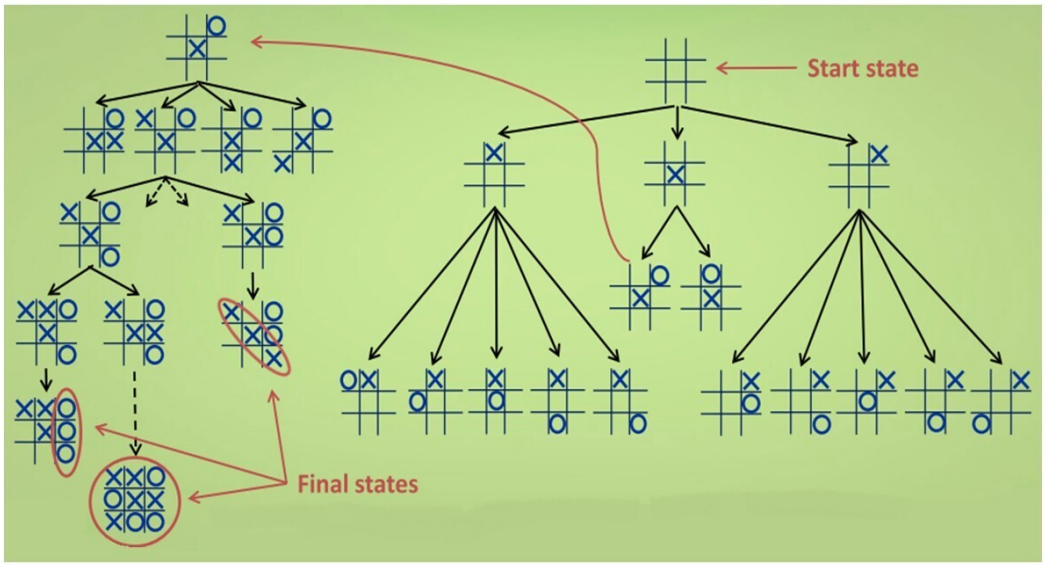
Если вы до сих пор помните, в нашей первой лекции мы говорили об использовании пространства состояний для решения задач.

В целом, в представлении пространства состояний, задача представляется в виде множества состояний.

В частности, есть множество начальных состояний, то есть, там, где начинается задача, и набор конечных состояний, в том числе всех возможных решений задачи.

Два состояния связаны, если существует операция, которая может превратить одно состояние в другое.

Один из примеров, которые мы использовали для иллюстрации представления пространства состояний, это игра в крестики-нолики.



Таким образом, игра начинается с начального состояния и может закончиться набором конечных состояний.

Два промежуточных состояний связаны, если существует операция между двумя состояниями.

Чтобы найти выигрышный путь (или, по крайней мере, ничью), нужно искать путь через пространство состояний с помощью представления дерева, как на графике, который вы видите здесь.

Широко используемая стратегия для поиска решения в пространстве состояний называется откатом.

Откат – это общая стратегия решения задачи для систематического поиска решения проблемы среди всех возможных вариантов.

И стеки часто используются в реализации алгоритмов отката.

Легче всего проиллюстрировать, как откат работает с использованием примеров.

Классическим примером использования отката в поиске решения является проблема n-ферзей.

Для тех, кто играл раньше в шахматы, вы знаете, что ферзь является самой сильной фигурой в шахматах.

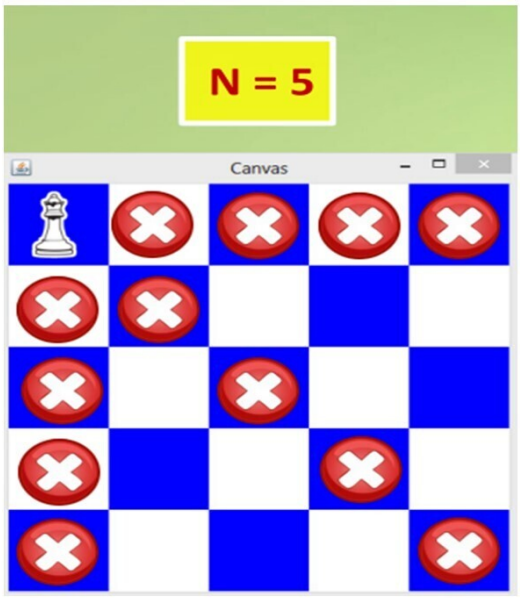
Он может перемещаться на любое количество шагов по вертикали, горизонтали и даже по диагонали.

В шахматной игре, каждый игрок имеет только одного ферзя, но в задаче n-ферзей, есть в общей сложности n ферзей, то есть, для обычной 8x8 шахматной доски, там будет 8 ферзей.

В общем, n может быть любым числом.

Цель задачи n-ферзей является размещение n ферзей на шахматной доске NxN так, чтобы никакие два ферзя не смогли напасть друг на друга.

Давайте использовать n равный 5 в качестве примера.



Вот шахматная доска 5x5.

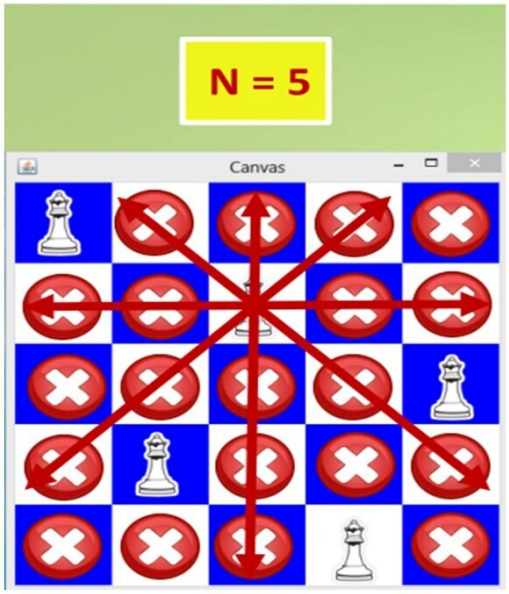
Поскольку ферзь может двигаться горизонтально, он может атаковать любых других ферзей, которые находятся на той же строке.

Так что, если ферзь находится на 0-й строке и в 0-в столбце, как показано здесь, ни один другой ферзь не может быть размещен на 0-й строке.

Точно так же, поскольку ферзь может двигаться по вертикали и по диагонали, ни один другой ферзь не может быть размещен в 0-м столбце и на основной диагонали.

Таким образом, после размещения первого ферзя на 0-й строке, все эти позиции с крестом были исключены.

На 2-й строке, первая доступная позиция в 3-м столбце или столбце с индексом 2.



Размещение этого второго ферзя устранит все остальные позиции в этой строке и все другие позиции в столбце с индексом 2.

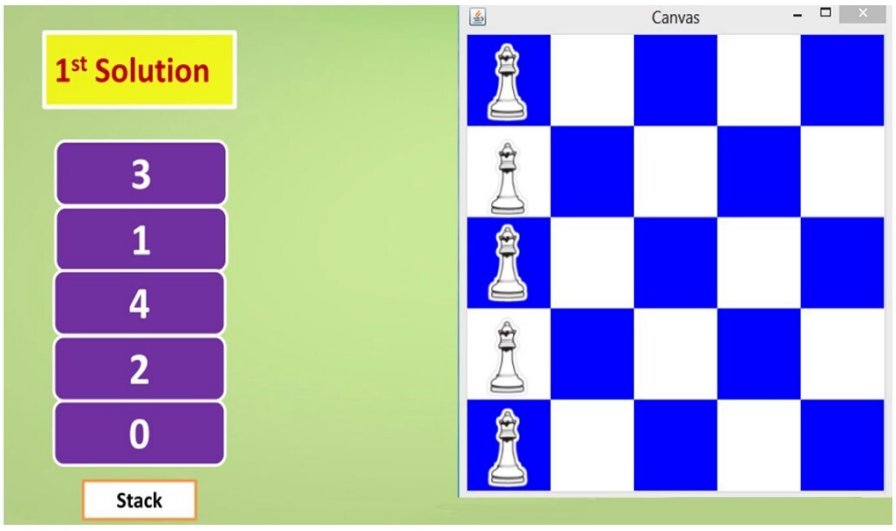
Что касается диагонали, здесь есть две диагонали, как здесь показано на схеме.

По аналогии разместим остальных ферзей, и у нас есть наше первое решение задачи 5-ферзей.

Вы можете проверить, чтобы убедиться, что в этом решении, никакие два ферзя не могут напасть друг на друга по горизонтали, вертикали или диагонали.

Прежде чем двигаться дальше, вы можете сначала подумать о том, существуют ли другие варианты решения задачи 5-ферзей.

Теперь, когда вы понимаете, что представляет собой задача n-ферзей, давайте подумаем о том, как мы можем решить эту задачу.



Я буду использовать здесь пример, чтобы проиллюстрировать стратегию, которую мы собираемся использовать, чтобы решить задачу n-ферзей.

Мы разместим ферзей с первой строки по последнюю строку, и будем двигаться от самого левого столбца до самого правого столбца в каждой строке.

Давайте поместим первого ферзя в 0,0 положение, и мы будем использовать стек для хранения промежуточных решений.

Значение, сохраненное в стеке, является меткой столбца, в данном случае, 0.

Нам не нужно хранить метку строки, потому что мы можем использовать позицию элемента стека для обозначения метки строки.

После того, как у нас есть ферзь на определенной строке, мы не должны пытаться поставить другого ферзя в той же строке, потому что они могут напасть друг на друга в горизонтальном направлении.

Теперь мы можем двигаться дальше, чтобы попытаться поместить второго ферзя на 2-й строке.

Мы начнем с 1-го столбца, и это не является правильным шагом.

Далее мы постараемся двигать ферзя вправо или на 2-й столбец.

Опять же, тут есть конфликт, потому что первый ферзь может атаковать по диагонали.

2-й ферзь перемещается в следующий столбец снова, и это будет признано безопасной позицией, и это положение, то есть столбец с индексом 2, как метка записывается в стек.

Теперь мы можем попытаться поместить третьего ферзя в 3-й строке, начиная с 1-го столбца.

Конфликт возникает с первым ферзем, так что он перемещается в следующий столбец.

Это также не хорошо, потому что конфликт возникает со 2-м ферзем.

Следующий столбец тоже не хорош, потому что это находится в противоречии как с 1-м, так и со 2-м ферзем.

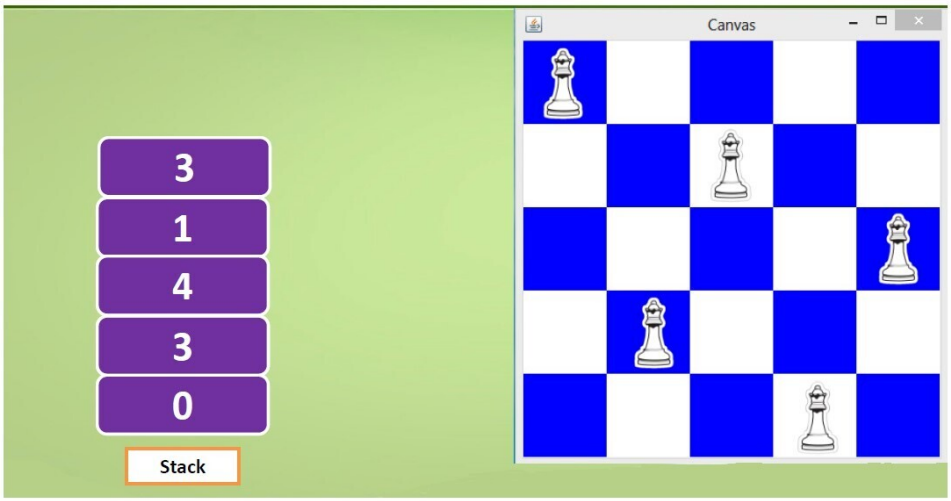
Точно так же, 4-й столбец также исключен из-за 2-го ферзя.

Так что мы остались в последнем столбце, и к счастью, нет конфликтов с остальными ферзями.

И позиция 4 теперь может быть записана в стек.

Размещение 4 ферзя может быть получена с использованием той же стратегии, и метка 1 записывается в стек.

Точно так же, правильное размещение получается и для последнего ферзя, и метка 3 помещается в стек.



Так что это первое решение, которое мы получили в задаче 5-ферзей.

Что делать, если мы хотим получить все остальные решения?

Где мы должны возвратиться к предыдущему шагу, и определить, где продолжать искать другое решение.

Поскольку нынешние позиции всех ферзей записываются в стек, мы можем удалить верхний элемент из стека, чтобы узнать местоположение предыдущего движения.

Установлено, что 5-й ферзь был помещен в столбец с меткой 3, так что вместо того, чтобы начинать с самого левого столбца, как то, что мы делали раньше, мы можем начать проверять следующий столбец с этой позиции.

Установлено, что эта позиция в конфликте с обеими, первым и третьим ферзем.

В этом случае, если мы будем двигаться дальше вправо, ферзь будет перемещен за пределы доски, что не позволительно.

Это означает, что не будет другого решения с текущим размещением предыдущих 4 ферзей.

Так что мы должны вернуться к предыдущей строке, и начать поиск решения.

Чтобы узнать текущее размещение ферзя на 4-й строке, верхний элемент в стеке удаляется и дает метку 1 или второй столбец.

Так он может двигаться вправо от этой позиции, а не из первого столбца.

К сожалению, все остальные позиции в строке находятся в противоречии с предыдущими ферзями.

Когда ферзь перемещается за пределы доски, мы знаем, что новое решение не может быть найдено с текущим размещением предыдущих 3 ферзей.

Таким образом, мы должны двигаться на одну строку вверх, вынимая верхний элемент из стека и пробуя еще раз.

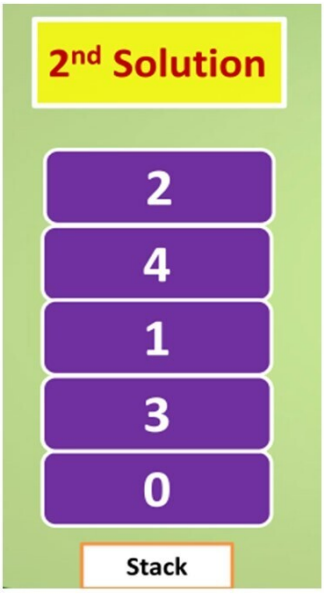
В случае, если ферзь уже в последнем столбце, следующая позиция будет двигать его за пределы доски.

Таким образом, мы должны двигаться на одну строку вверх снова.

Мы обнаружили, что второй ферзь в настоящее время в столбце с меткой 2.

Перемещение его в следующий столбец не найдет никакого конфликта с каким-либо другим ферзем, так он может стать частью нового решения, и его метка 3 помещается в стек.

Таким образом, второе решение будет следующим.

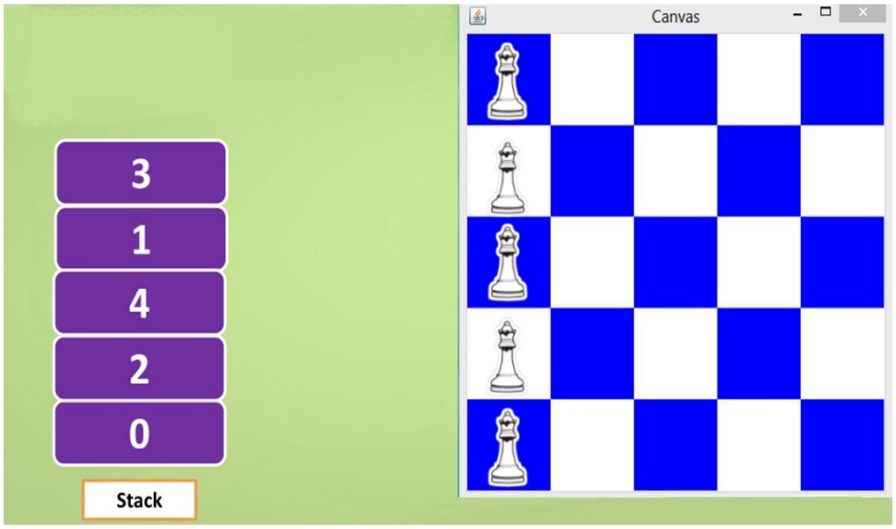

2nd Solution
2
4
1
3
0
Stack

Реализация задачи

Попробуем обобщить стратегию отката, которую я только что представил.

Алгоритм будет пытаться поместить ферзя на каждой строке сверху донизу.

Для каждой строки, ферзь будет перенесен слева направо, изначально начиная со столбца с индексом 0.



Если нет конфликта с текущим размещением, промежуточный результат помещается на вершине стека.

В противном случае, идет переход к следующему столбцу справа.

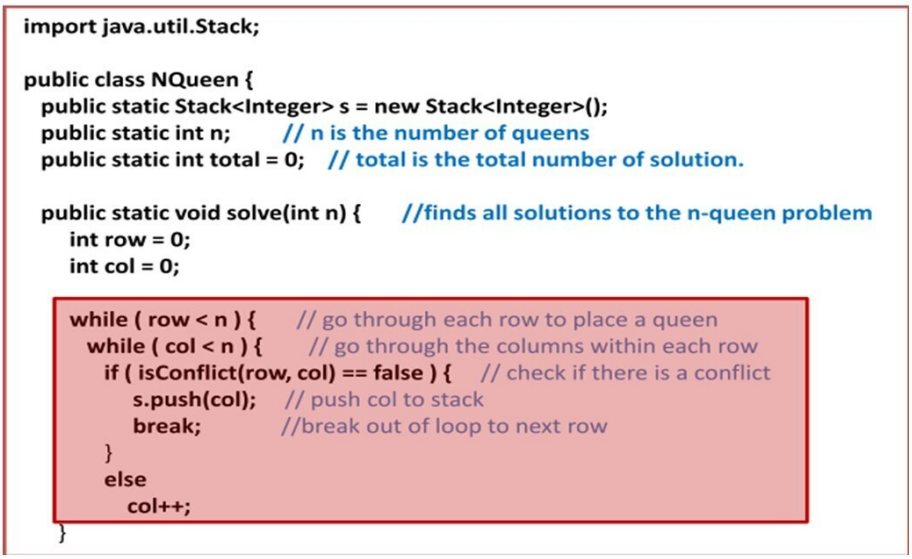
Каждый раз, когда новый шаг вносится в стек, вполне возможно, что решение будет найдено.

Если это так, то решение выводится, а затем идет возврат к предыдущему шагу, чтобы увидеть, может ли быть найдено другое решение.

Откат может быть достигнут путем опустошения стека.

Если это не является решением, и нет больше места, чтобы передвинуть ферзя на текущей строке, идет возврат к предыдущей строке, снова опустошая стек.

После отката, вместо того чтобы начинать с самого левого столбца, перемещать ферзя справа налево от столбца, как записано в стеке.



```
import java.util.Stack;

public class NQueen {
  public static Stack<Integer> s = new Stack<Integer>();
  public static int n;       // n is the number of queens
  public static int total = 0;   // total is the total number of solution.

  public static void solve(int n) {      //finds all solutions to the n-queen problem
     int row = 0;
     int col = 0;

     while ( row < n ) {      // go through each row to place a queen
       while ( col < n ) {       // go through the columns within each row
         if ( isConflict(row, col) == false ) {    // check if there is a conflict
            s.push(col);    // push col to stack
            break;          //break out of loop to next row
         }
         else
            col++;
       }
```

Вот часть Java программы, реализующей решение проблемы n-ферзей.

Программа начинается с импорта класса стека из пакета java.util.

Здесь устанавливается стек класса-оболочки целых чисел с именем S.

Помните, что, так как стек работает только для объектов, мы должны использовать класс-оболочку, а не примитивный тип int.

n это число ферзей, размещенных на NxN доске.

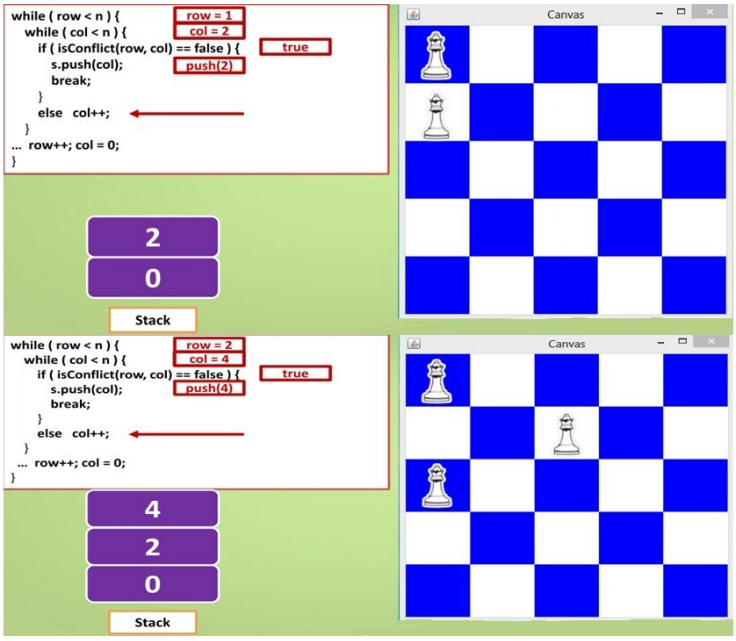
total это общее количество решений, которое установлено изначально в 0.

Часть программы для решения задачи – это метод solve, который берет n, число ферзей, в качестве параметра.

row и col это индексы строки и столбца шахматной доски, и индексы начинаются с 0, а не с 1.

Часть метода, которая выполняет большую часть работы, это два while цикла, один для индексов строк и другой для индексов столбцов.

Я буду использовать пример, чтобы проиллюстрировать, как это работает.



Два while цикла здесь дублируются, индексы строк и столбцов инициализируются 0.

Размещение 1-го ферзя находится в 0,0, что явно не вызывает конфликт, так как он является единственным ферзем и метод isConflict является логическим методом, который вернет false, если нет конфликтов ни с какими другими ферзями, которые были помещены на доску до сих пор.

Я вернусь к его реализации позже.

Когда тело if выражения выполняется, метод push вызывается для стека s, а текущая метка столбца 0 вносится в пустой стек.

Далее происходит выход из внутреннего while цикла.

Строка будет увеличена на 1 и получает значение 1, а столбец сбрасывается в 0.

То есть, делается попытка поместить второго ферзя на позицию 1,0.

В этом случае, isConflict возвращает true, потому что он находится в том же столбце, что и первый ферзь.

Часть else if выражения будет выполнена, что увеличит столбец на 1.

(1,1) позиция далее проверяется методом isConflict, который снова возвращает true, потому что это в той же диагонали, что и первый ферзь.

Часть else выполняется снова.

Позиция (1,2) теперь проверяется методом isConflict, который возвращает false, т.е. тут нет конфликта.

Таким образом, текущее значение col или 2 вносится на вершину стека.

Далее происходит выход из внутреннего while цикла, после того как 2 вносится в стек.

Строка увеличивается на 1 и получает значение 2, в то время как столбец сбрасывается в 0.

Позиции (2,0), (2,1), (2,2) и (2,3) будут проверяться последовательно для конфликта и для всех из них возвратится true, до тех пор, пока позиция (2,4) не будет достигнута и isConflict не возвратит false.

Таким образом, значение 4 будет внесено в стек.

Этот процесс будет продолжаться до тех пор, пока определенные условия не будут выполнены.

Эти условия проверяются в следующей части программы.

```
if (s.empty() == true) break; // either no solution or all solutions have been found

if (col >= n) {            // finished all possible placements in a row
   row--;
   col = s.pop() + 1;
}
else {
  row++;
  col = 0;
}
  if (s.size()==n){        // if stack size is n a solution is found
      total++;
      System.out.println(total + ": " + s);
      col = s.pop() + 1; // continue to find next solution
      row--;
    }
  }
}
```

Первое условие, которое может привести к прекращению программы, это когда стек пуст.

Это может быть проверено с помощью метода empty класса стека.

Пустой стек означает либо нет решения, или все решения были найдены в задаче n-ферзей на данном n.

2-й условие, которое мы должны проверить, перемещается ли ферзь за пределы доски или, что тоже самое, если индекс столбца больше или равен n.

Давайте посмотрим на этот случай более подробно.

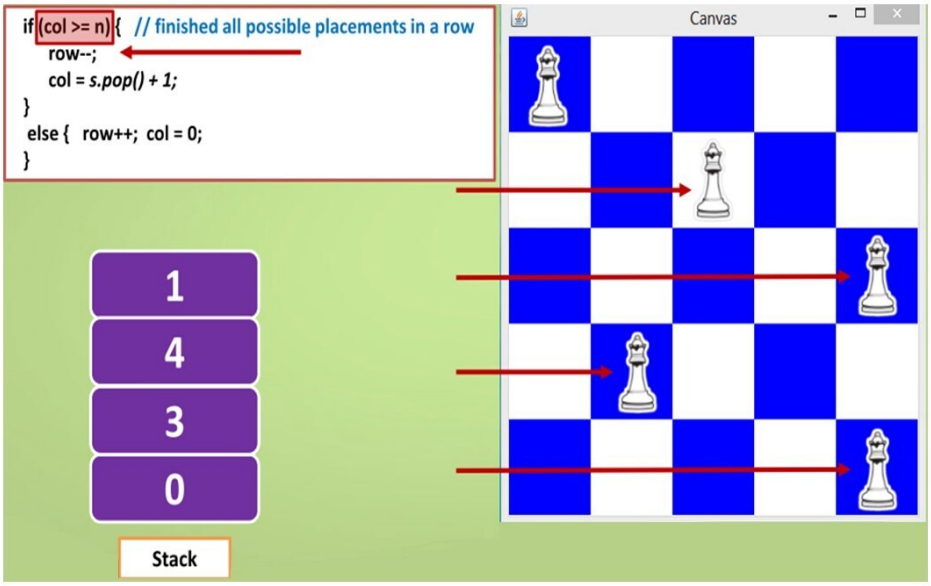
Будем считать, что это текущее состояние задачи, и мы находимся в процессе принятия решения следующего размещения ферзя в последней строке и последнем столбце.

Когда столбец снова вырос на 1, ферзь будет перемещен за пределы доски.

То есть, мы закончили исследовать все возможные размещения ферзя в текущей строке и условие col >=n становится истиной.

Когда тело if выражения выполняется, строка будет сокращена на 1, то есть, мы вернемся к предыдущей строке.

Стек извлекается таким образом, что мы можем определить текущее размещение ферзя в строке, в этом случае, со столбцом 1.



Чтобы проверить следующую позицию столбца, 1 добавляется к этой метке столбца, то есть столбец получит новое значение 2.

Эти новые позиции будут проверяться while циклом, как мы обсуждали ранее, и все они будут найдены в конфликте с другими ферзями.

Таким образом, ферзь перемещается за пределы доски или столбец больше или равен n.

Стек освобождается таким образом, что будет рассмотрена предыдущая строка, или 3-я строка.

Чтобы добавить 1 к метке 4 столбца, мы снова будем двигать ферзя за пределы доски.

Тело if выражения снова выполняется для изучения второй строки, которая имеет ферзя в столбце с меткой 2.

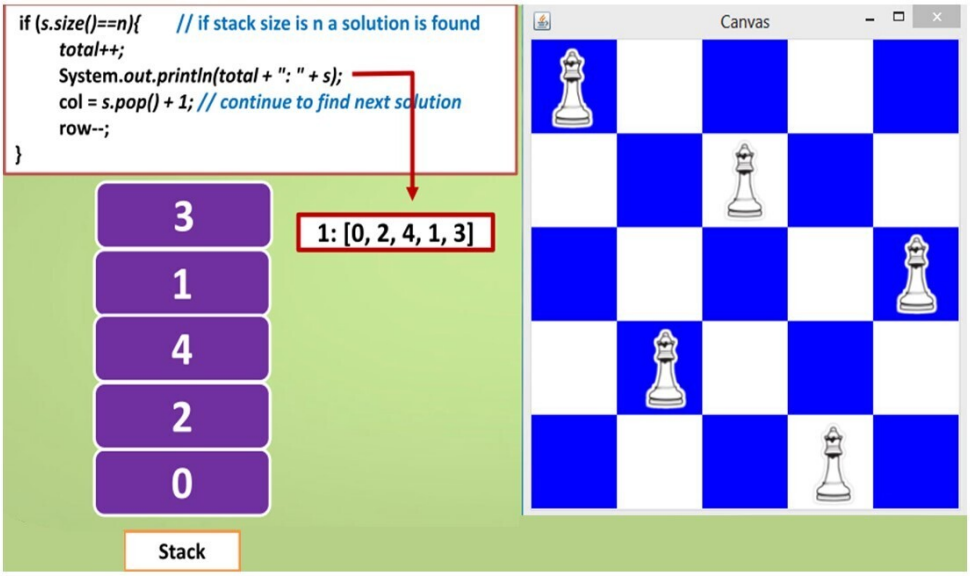
Единица добавляется к нему так, что рассматривается следующий столбец, и никакого конфликта не будет найдено, так что метка столбца 3 вносится в стек.

Последняя часть метода solve будет проверять условие, когда будет найдено решение.

```
        if (s.isEmpty() == true) break;  // either no solution or all solutions have been found

        if (col >= n) {            // finished all possible placements in a row
           col = s.pop() + 1;
           row--;
         }
        else {
          row++;
          col = 0;
        }
        if (s.size()==n){          // if stack size is n a solution is found
            total++;
            System.out.println(total + ": " + s);
            col = s.pop() + 1; // continue to find next solution
            row--;
         }
    }
}
```

Это может быть определено путем проверки, если размер стека, с использованием метода size для класса стека, равен значению n.

Если это правда, это означает, что все n ферзей были размещены на шахматной доске, и не конфликтуют друг с другом.



Когда решение найдено, total, представляющее общее число решений, будет увеличена на 1.

Метод System.out.println может быть использован для вывода результатов, сохраненных в стеке, указав S в качестве параметра.

Для того чтобы найти следующее решение, последнее размещение ферзя определяется, вынимая верхний элемент из стека, так что может быть рассмотрен следующий столбец.

Обратите внимание, что строка должна быть уменьшена на 1, потому что строка увеличивается на 1 каждый раз, когда внутренний while цикл заканчивается, то есть строке будет дано значение, равное n, когда решение найдено.

Важной частью программы, которая по-прежнему отсутствует, является реализация логического метода isConflict для определения находится ли размещение нового ферзя в конфликте с предыдущими размещениями ферзей.

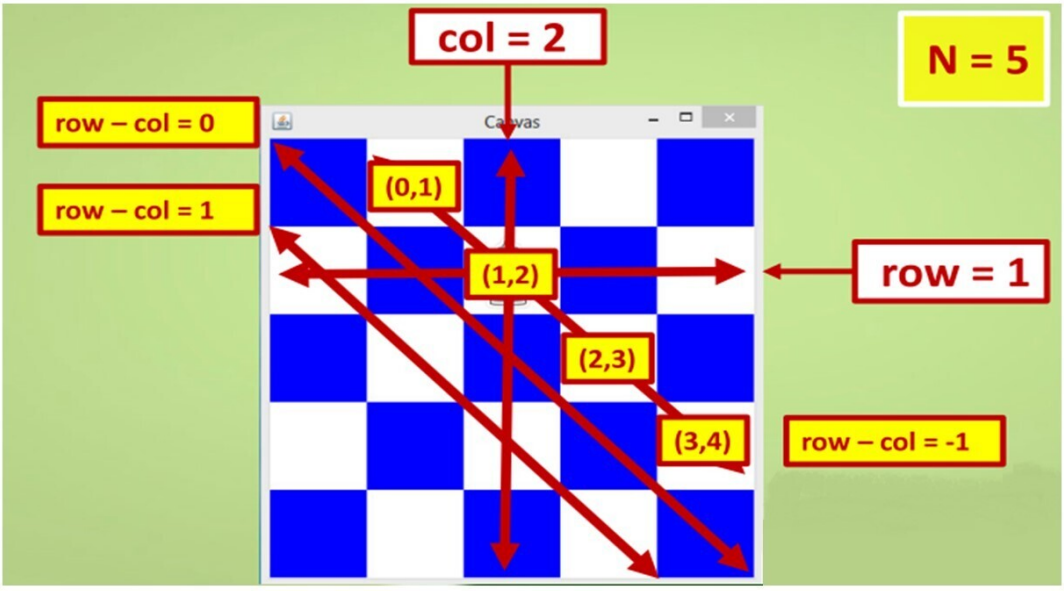
Вот реализация этого метода.

```
public static boolean isConflict(int row, int col) {
   int diff = row-col;
   int sum  = row+col;
   for (int i = 0; i < row; i++) {
      int t = s.get(i);
      if (t==col || i-t == diff || i+t == sum) return true;
   }
   return false;
}
```

Давайте, используем пример, чтобы объяснить, как это работает.

Рассмотрим пример, чтобы проиллюстрировать, как определить, есть ли конфликт между ферзями.

Если предположить, что ферзь находится в строке 1 и столбце 2, легко определить, есть ли конфликт вдоль одной и той же строки или в одном столбце, так как мы можем просто проверить индекс строки или индекс столбца, но как насчет конфликта по диагоналям.



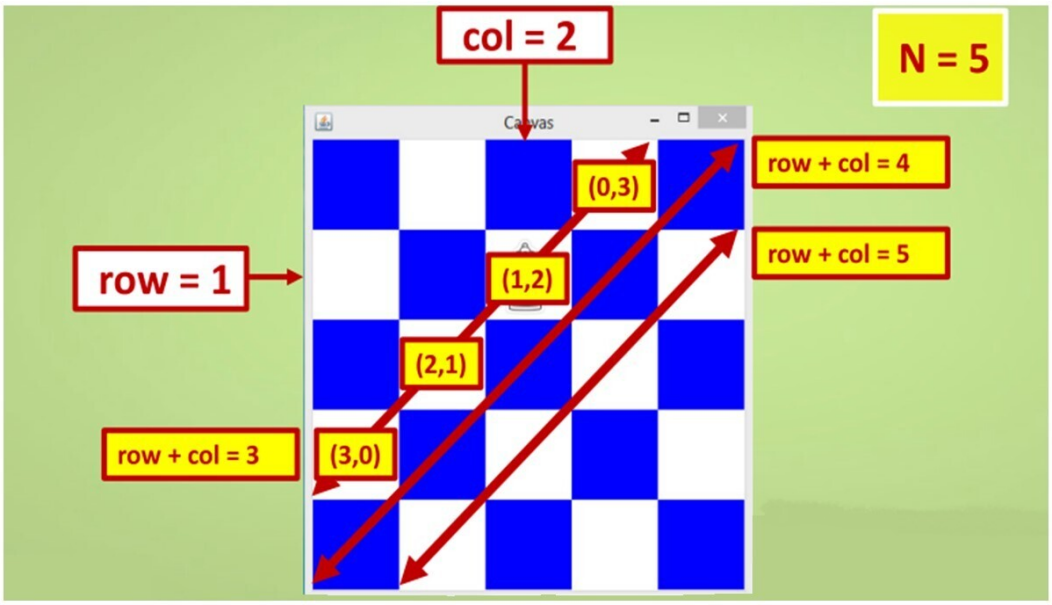
Вы можете сначала понаблюдать, какое свойство является общим для всех точек по диагонали.

Что вы найдете общего, когда вы посмотрите на индексы этих точек.

Когда разницы между индексами строк и индексами столбцов вычисляются для всех этих точек, все они приводят к одним и тем же значениям -1.

Является ли это просто совпадением?

Давайте посмотрим на другую диагональ в том же направлении.
Вы увидите, что их различия все приведены в значение 0.
Давайте посмотрим на еще одну диагональ.
В этом случае, различия дают 1 в качестве значения.
Как насчет диагонали в другом направлении.



Вот индексы всех точек вдоль этой диагонали, и их различия, безусловно, не те же самые, как и раньше.

Но что интересно, их суммы все дают одно и то же значение, в данном случае 3.

Давайте рассмотрим еще пару диагоналей в том же направлении, суммы для одной дают значение 4, и для другой дают значение 5.

Так что в целом, точки на одной и той же диагонали либо дают одинаковые суммы индексов или одинаковые разницы индексов.

В реализации метода isConflict, суммы и разницы показателей вычисляются и проверяются на равенство в дополнение к индексу столбца.

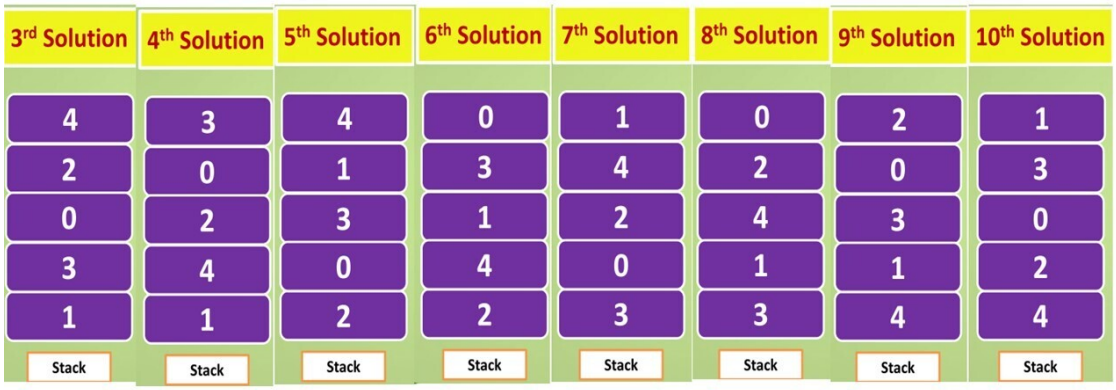
```
public static boolean isConflict(int row, int col) {
   int diff = row-col;
   int sum  = row+col;
   for (int i = 0; i < row; i++) {
      int t = s.get(i);
      if (t==col || i-t == diff || i+t == sum) return true;
   }
   return false;
}
```

Здесь нет необходимости проверять индекс строки, потому что только один ферзь может быть размещен на каждой строке.

Отметим также, что метод get наследуется от класса вектора, который является суперклассом класса Stack.

Можно установить другой массив, чтобы отслеживать промежуточные шаги, если не хотите использовать метод get.

Вот другие решения для этой проблемы 5-ферзей.



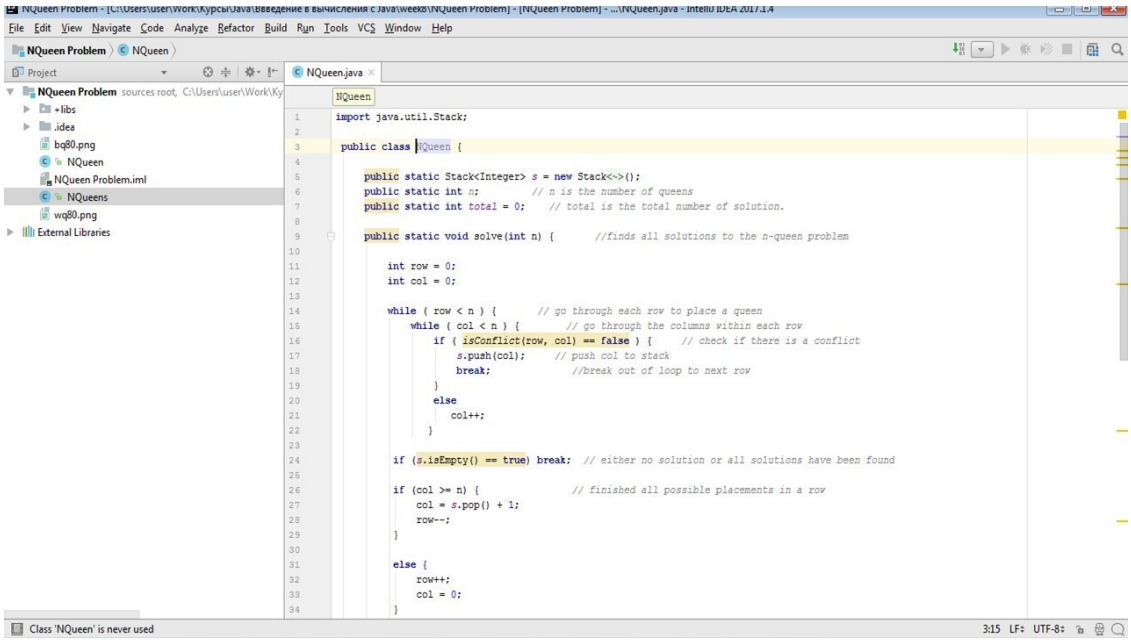
Существует в общей сложности 10 решений.

Откроем IDEA, чтобы посмотреть на фактическое выполнение программы.

Демонстрация задачи

Давайте посмотрим на выполнение задачи n-ферзей.

Это программа, которую мы обсуждали в лекции.



Как вы можете видеть, здесь есть два while цикла.

Один для управления строками, другой для управления столбцами.

Мы будем вносить метку столбца в стек, если не будет обнаружен конфликт для текущего размещения ферзей.

И здесь есть условие проверки, является ли стек пустым.

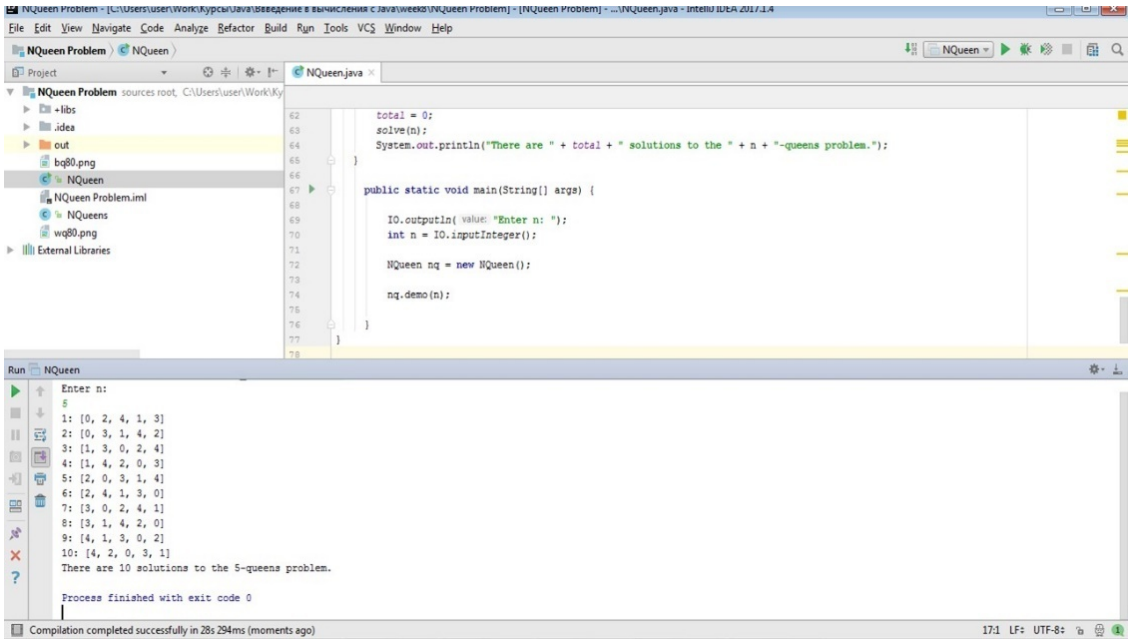
В этом случае, либо решение не может быть найдено, или все решение найдены.

В случае, когда вы имеете метку столбца больше или равно n, это означает, что ферзь вышел за границы доски.

Когда размер стека равен n, это означает что решение найдено, и мы будет выводить значения стека.

Мы опустошим стек, чтобы определить следующий шаг для поиска следующего решения.

Давайте скомпилируем и запустим программу, создавая экземпляр класса.

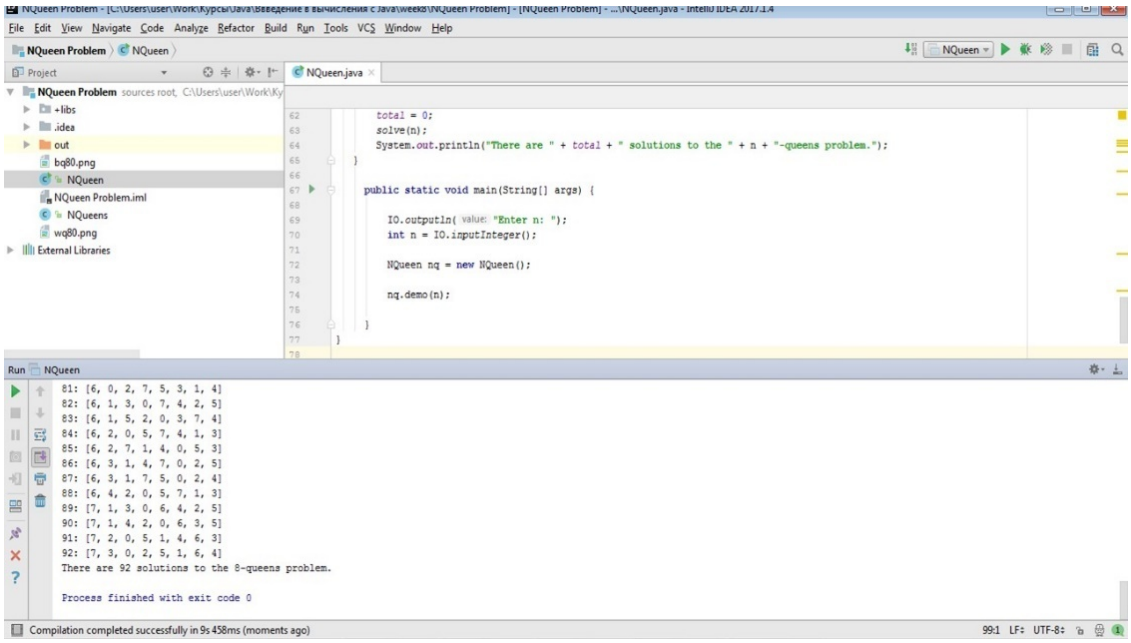


Метод demo запросит n, и введем n равным 5, и вы можете увидеть, что было найдено 10 решений.

Хотя трудно сказать, являются ли эти решения правильными.

Давайте попробуем найти решения для другого n.

Скажем, n, равным 8.



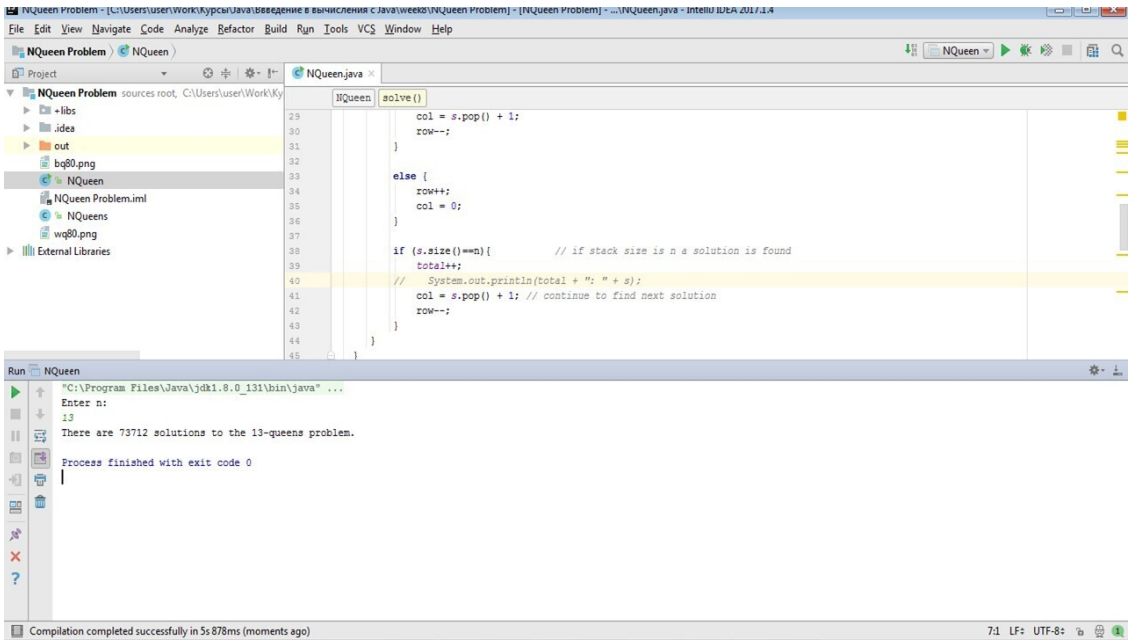
И у нас есть 92 решений.

Если мы попытаемся использовать еще больший n, тогда вы найдете, что вычисление задачи n-ферзей является довольно затратным.

Прежде, чем мы снова запустим программу, давайте уберем выражение вывода.

```
if (s.size()==n){          // if stack size is n a solution is found
        total++;
        //System.out.println(total + ": " + s);
        col = s.pop() + 1; // continue to find next solution
        row--;
    }
```

И снова скомпилируем, и запустим эту программу;
Давайте опять попробуем использовать большой n.
Скажем 13.

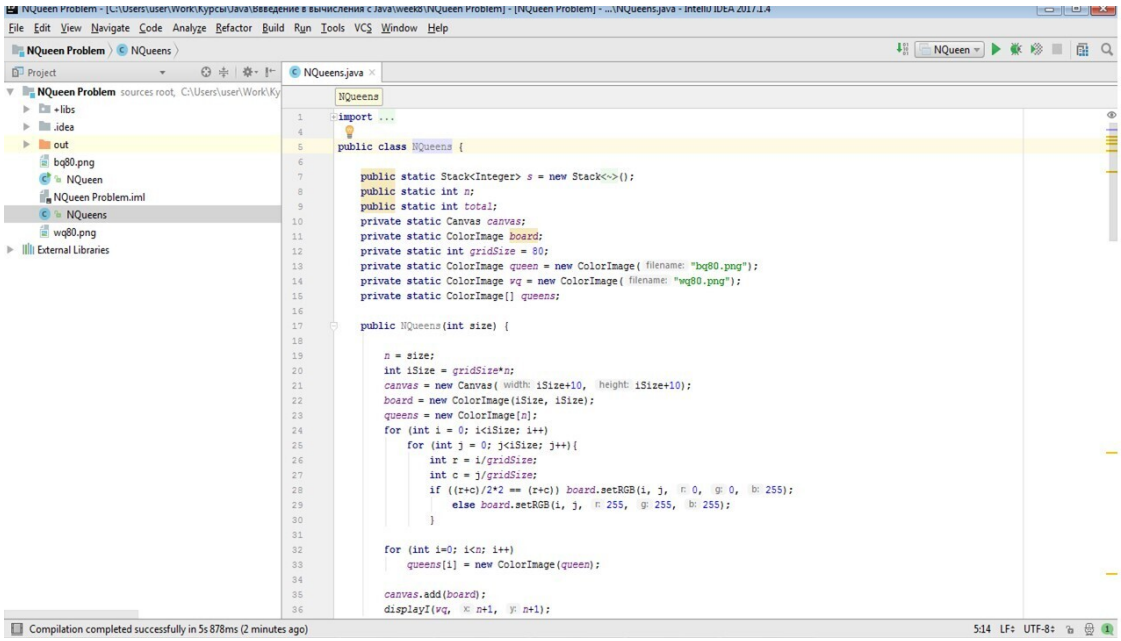


Вы сможете увидеть, что это займет некоторое время, перед тем как будет возвращен результат.

У вас есть здесь более 73 000 решений для проблемы 13-ферзей.

В предыдущей демонстрации проблемы n-ферзей, действительно трудно сказать, являются ли генерируемые решения действительно правильными.

Поэтому у нас здесь есть графическая версия программы NQueens.



```
import ...

public class NQueens {

    public static Stack<Integer> s = new Stack<>();
    public static int n;
    public static int total;
    private static Canvas canvas;
    private static ColorImage board;
    private static int gridSize = 80;
    private static ColorImage queen = new ColorImage( filename: "bq80.png");
    private static ColorImage wq = new ColorImage( filename: "wq80.png");
    private static ColorImage[] queens;

    public NQueens(int size) {

        n = size;
        int iSize = gridSize*n;
        canvas = new Canvas( width: iSize+10,  height: iSize+10);
        board = new ColorImage(iSize, iSize);
        queens = new ColorImage[n];
        for (int i = 0; i<iSize; i++)
            for (int j = 0; j<iSize; j++){
                int r = i/gridSize;
                int c = j/gridSize;
                if ((r+c)/2*2 == (r+c)) board.setRGB(i, j,  r: 0,  g: 0,  b: 255);
                    else board.setRGB(i, j,  r: 255,  g: 255,  b: 255);
                }

        for (int i=0; i<n; i++)
            queens[i] = new ColorImage(queen);

        canvas.add(board);
        displayI(wq,  x: n+1,  y: n+1);
```

В этой программе, как вы можете увидеть, в начале есть некоторые настройки для графических объектов, в том числе ферзей, которые будут отображаться на холсте.

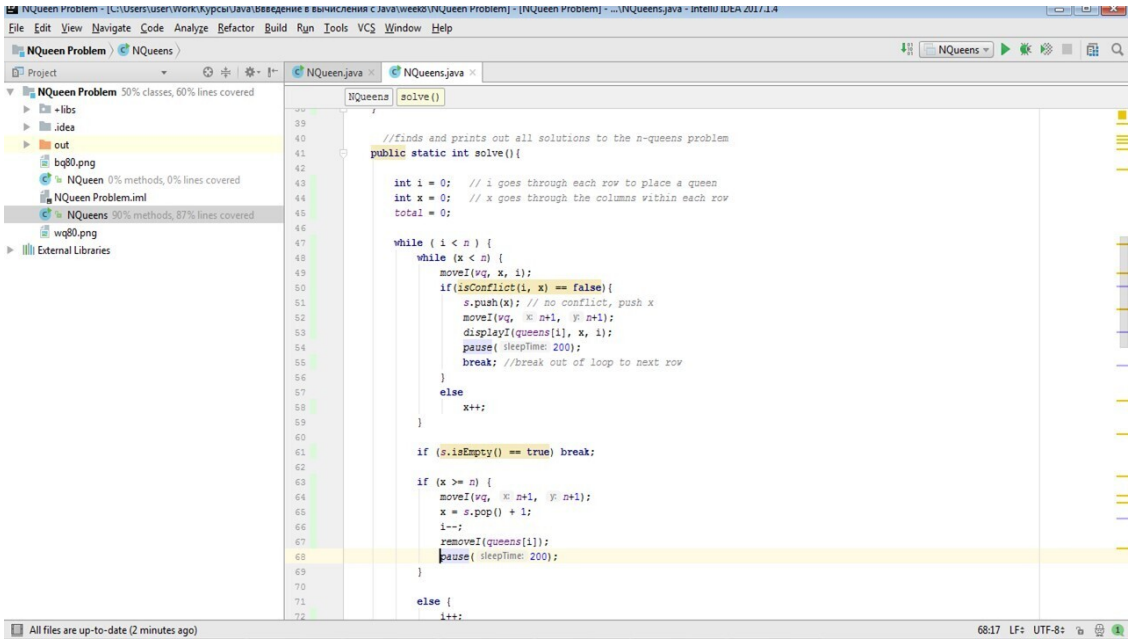
Основная программа в основном такая же, как и раньше, здесь есть два while цикла для прохождения через строки и столбцы.

Мы проверяем, является ли стек пустым.

Вышел ли ферзь за пределы шахматной доске.

И было ли найдено решение.

И метод isConflict такой же, как и раньше.

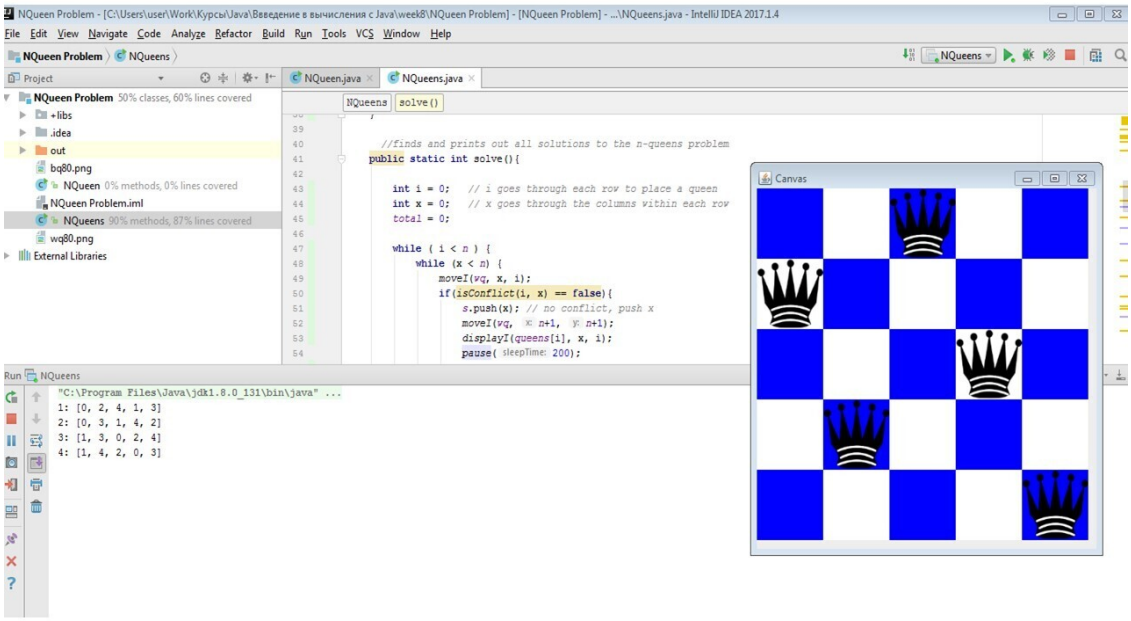


```
    //finds and prints out all solutions to the n-queens problem
    public static int solve(){

        int i = 0;   // i goes through each row to place a queen
        int x = 0;   // x goes through the columns within each row
        total = 0;

        while ( i < n ) {
            while (x < n) {
                moveI(wq, x, i);
                if(isConflict(i, x) == false){
                    s.push(x); // no conflict, push x
                    moveI(wq,  x: n+1,  y: n+1);
                    displayI(queens[i], x, i);
                    pause( sleepTime: 200);
                    break; //break out of loop to next row
                }
                else
                    x++;
            }

            if (s.isEmpty() == true) break;

            if (x >= n) {
                moveI(wq,  x: n+1,  y: n+1);
                x = s.pop() + 1;
                i--;
                removeI(queens[i]);
                pause( sleepTime: 200);
            }

            else {
                i++;
```

Давайте скомпилируем программу, и мы установим здесь паузу, чтобы мы могли представить себе выполнение программы на определенных этапах.

Давайте генерировать экземпляр с использованием n, равным 5.

Это означает, что мы будем решать задачу 5-ферзей.



Здесь отображается шахматная доска.

Давайте посмотрим выполнение программы шаг за шагом.

Вы можете заметить, что ферзи, которые вы видите, имеют два вида представления.

Белый ферзь означает, что он находится в перемещении или еще не определено, находится ли он в противоречии с любым другим ферзем.

Когда ферзь черный, это означает, что было подтверждено, что он не в конфликте с любым другим ферзем на шахматной доске.

В цикле вы можете видеть, что отображается белый ферзь, что означает, что мы все еще должны определить, является ли этот ферзь в конфликте с любым другим ферзем.

В этом случае, так как есть только один ферзь, вы можете видеть, что выполняется if выражение и позиция вносится в стек.

Представление будет изменено на черного ферзя, что означает, что это не противоречит какому-либо другому ферзю.

Если мы будем продолжать выполнение, "i" будет увеличен на 1 или вторую строку.

Далее мы пытаемся определить, находится ли этот белый ферзь в конфликте с любым другим ферзем, и очевидно, что это так.

Таким образом, if выражение не выполняется.

Вместо этого, метка столбца будет увеличиваться на 1.

И этот процесс будет продолжаться, пока ферзь не перейдет в такое положение, когда он не будет в конфликте с любым другим ферзем, и вы сможете увидеть, что ферзь будет изменен на черного ферзя.

И мы сможем продолжить процесс.

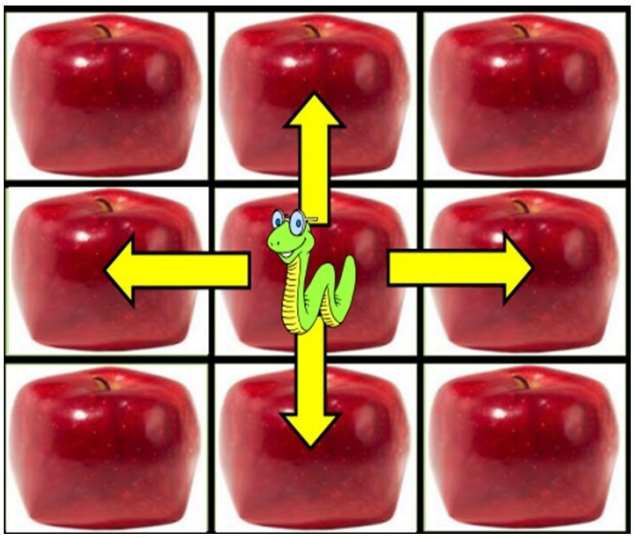
И этот процесс будет продолжаться, пока не будут найдены все решения.

Пример

Давайте посмотрим еще один пример, чтобы проиллюстрировать использование стека.

Мы уже использовали задачу квадратных яблок, чтобы проиллюстрировать важность нахождения правильного представления задачи для ее решения.

Если вы до сих пор помните, задача квадратных яблок в том, чтобы определить, может ли червь съесть все яблоки, которые размещаются в сетке NxN, следуя определенным правилам.



Во-первых, он может двигаться только в другую клетку, которая имеет общую сторону с текущей клеткой, и второе правило в том, что он не может повторно посещать любые клетки, которые он посетил ранее.

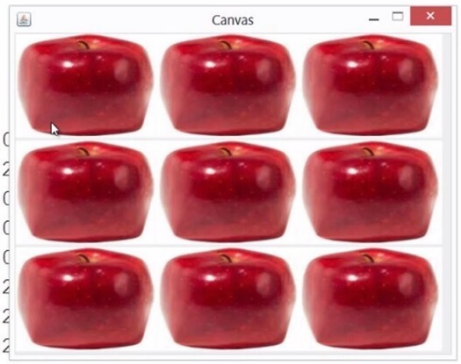
Легко увидеть, что для конфигурации сетки 3x3, решения существуют, когда червь начинает свое движение из средней ячейки.

Мы видели демонстрационную программу для иллюстрации процесса поиска решения.

И сейчас мы сможем увидеть, что здесь используется способ отката, и используется стек для отслеживания движения в данной реализации.

Посмотрим сначала выполнение программы на паре примеров.

```
a | b | c
---------
d | e | f
---------
g | h | i

Please choose a starting location (a - i):
1: (1,1),(2,1),(2,2),(1,2),(0,2),(0,1),(0,0
2: (1,1),(2,1),(2,0),(1,0),(0,0),(0,1),(0,2
3: (1,1),(1,2),(2,2),(2,1),(2,0),(1,0),(0,0
4: (1,1),(1,2),(0,2),(0,1),(0,0),(1,0),(2,0
5: (1,1),(0,1),(0,2),(1,2),(2,2),(2,1),(2,0
6: (1,1),(0,1),(0,0),(1,0),(2,0),(2,1),(2,2
7: (1,1),(1,0),(2,0),(2,1),(2,2),(1,2),(0,2
8: (1,1),(1,0),(0,0),(0,1),(0,2),(1,2),(2,2
There is(are) 8 solution(s).
```



Это то, что вы найдете, если червь начнет с середины.

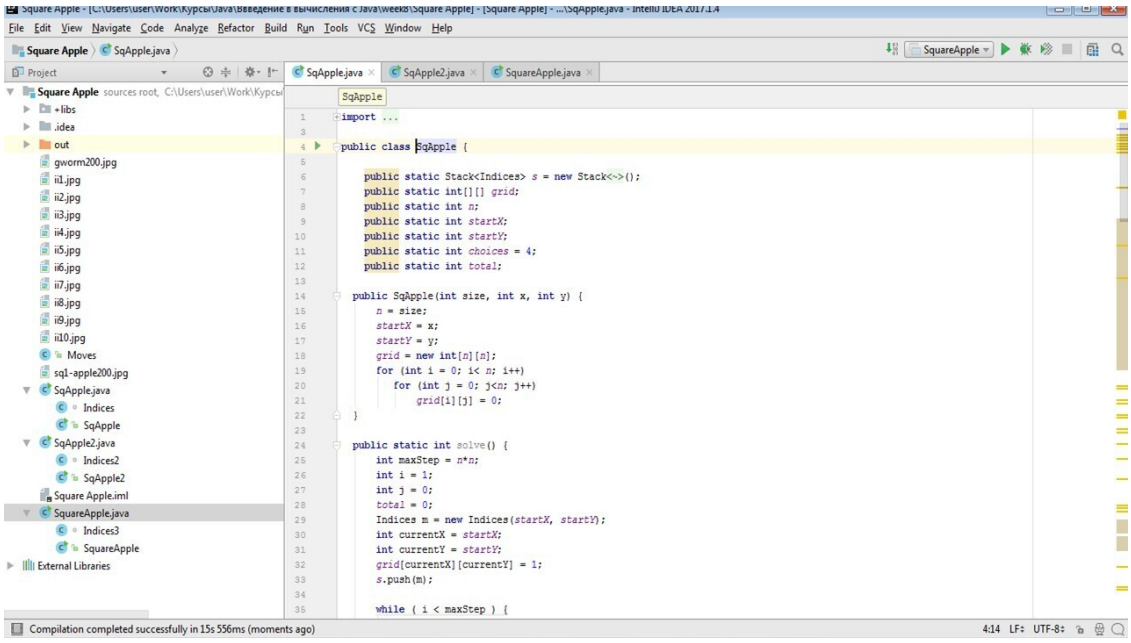
Как и ожидалось, были найдены 8 возможных решений.

Теперь, если червь начнет со стороны, скажем, b теперь, можно будет увидеть, что ни одно решение не будет найдено.

```
a | b | c
---------
d | e | f
---------
g | h | i

Please choose a starting location (a - i):
There is(are) 0 solution(s).
```

В случае, когда червь начинает с какой-либо стороны, мы обсуждали простой подход, чтобы определить, существует ли решение, и я вернусь к нему.

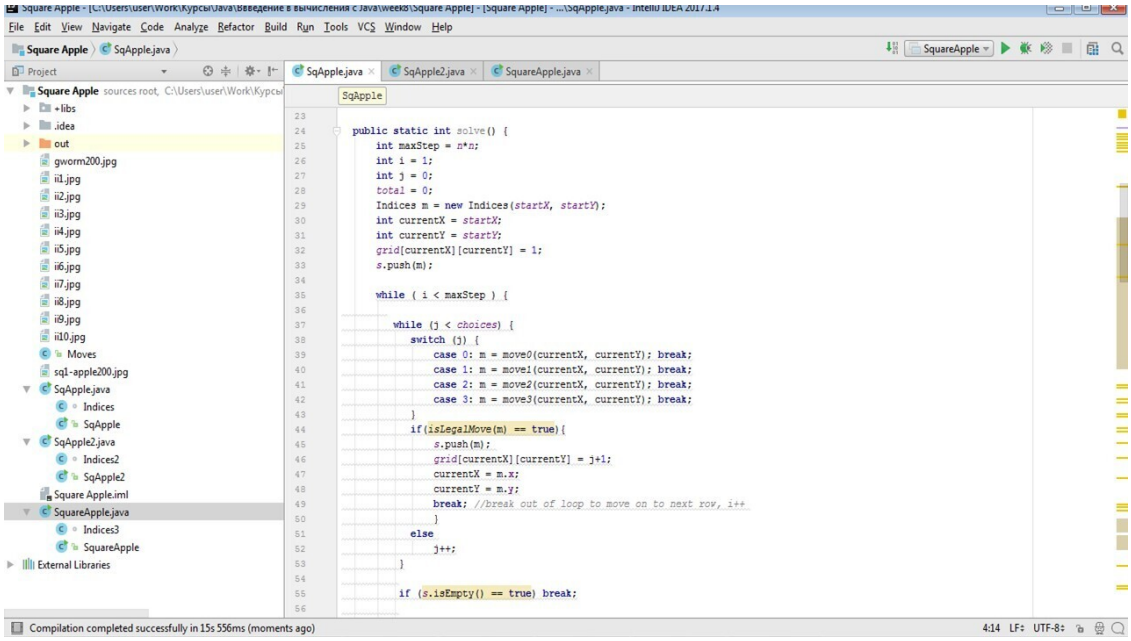
Вот реализация неграфической версии программы.



```
import ...

public class SqApple {

    public static Stack<Indices> s = new Stack<>();
    public static int[][] grid;
    public static int n;
    public static int startX;
    public static int startY;
    public static int choices = 4;
    public static int total;

public SqApple(int size, int x, int y) {
    n = size;
    startX = x;
    startY = y;
    grid = new int[n][n];
    for (int i = 0; i< n; i++)
       for (int j = 0; j<n; j++)
          grid[i][j] = 0;
}

public static int solve() {
    int maxStep = n*n;
    int i = 1;
    int j = 0;
    total = 0;
    Indices m = new Indices(startX, startY);
    int currentX = startX;
    int currentY = startY;
    grid[currentX][currentY] = 1;
    s.push(m);

    while ( i < maxStep ) {
```

Compilation completed successfully in 15s 556ms (moments ago)

Вы можете видеть, что структура программы очень похожа на программу n-ферзей.



```
public static int solve() {
    int maxStep = n*n;
    int i = 1;
    int j = 0;
    total = 0;
    Indices m = new Indices(startX, startY);
    int currentX = startX;
    int currentY = startY;
    grid[currentX][currentY] = 1;
    s.push(m);

    while ( i < maxStep ) {

        while (j < choices) {
            switch (j) {
                case 0: m = move0(currentX, currentY); break;
                case 1: m = move1(currentX, currentY); break;
                case 2: m = move2(currentX, currentY); break;
                case 3: m = move3(currentX, currentY); break;
            }
            if(isLegalMove(m) == true){
                s.push(m);
                grid[currentX][currentY] = j+1;
                currentX = m.x;
                currentY = m.y;
                break; //break out of loop to move on to next row, i++
                }
            else
                j++;
        }

        if (s.isEmpty() == true) break;
```

Compilation completed successfully in 15s 556ms (moments ago)

Для программы n-ферзей, был использован стек целочисленных объектов, потому что мы должны были только следить за меткой столбца, здесь же, используется стек индексов, чтобы отслеживать каждый ход.

```
class Indices {
    public int x, y;
    public Indices(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Как вы можете видеть здесь, в определении класса индексов, переменные экземпляра x и y представляют индексы строк и столбцов в nxn сетке.

```
while ( i < maxStep ) {

    while (j < choices) {
        switch (j) {
            case 0: m = move0(currentX, currentY); break;
            case 1: m = move1(currentX, currentY); break;
```

В то время как два цикла здесь аналогичны тем, которые в программе n-ферзей, вместо того чтобы идти циклом через индексы строк и столбцов, для задачи квадратных яблок, первый цикл предназначен для управления каждым ходом до maxStep, где maxStep установлен в n*n.

```
public static Indices move0(int x, int y) {
    Indices m0 = new Indices(x+1, y);
    return m0;
  }

public static Indices move1(int x, int y) {
    Indices m1 = new Indices(x, y+1);
    return m1;
  }

public static Indices move2(int x, int y) {
    Indices m2 = new Indices(x-1, y);
    return m2;
  }

public static Indices move3(int x, int y) {
```

В случае сетки 3x3, существует максимум 27 или 3*3 перемещения, и 2-й цикл для 4 возможных ходов, вправо, влево, вверх и вниз для каждого положения, и эти четыре движения соответствуют 4 случаям в switch выражении.

Методы move здесь определяют положение после каждого хода.

```
        case 1: m = move1(currentX, currentY); break;
        case 2: m = move2(currentX, currentY); break;
        case 3: m = move3(currentX, currentY); break;
    }
    if(isLegalMove(m) == true){
        s.push(m);
        grid[currentX][currentY] = j+1;
        currentX = m.x;
        currentY = m.y;
        break; //break out of loop to move on to next row, i+
        }
    else
        j++;
}

if (s.isEmpty() == true) break;
```

Логический метод isLegalMove аналогичен методу isConflict в программе n-ферзей.

```
if (s.isEmpty() == true) break;

if (j >= choices) {
    m = s.pop();
    grid[m.x][m.y] = 0;
    if (m.x == startX && m.y == startY) break;
    else
    {
        currentX = s.peek().x;
        currentY = s.peek().y;
    }
    j = grid[currentX][currentY];
    i--;
  }
else {
  i++;
  j = 0;
}

if (s.size()==maxStep){ // if stack size is n, a solution has been found
        total++;
        System.out.print(total + ": ");
        printSolution(s);

        m = s.pop();
        grid[m.x][m.y] = 0;
        currentX = s.peek().x;
        currentY = s.peek().y;
        j = grid[currentX][currentY];
        i--;
  }
```

Каждое правильное перемещение вносится в стек.

Три условия здесь также похожи на те, в задаче n-ферзей.

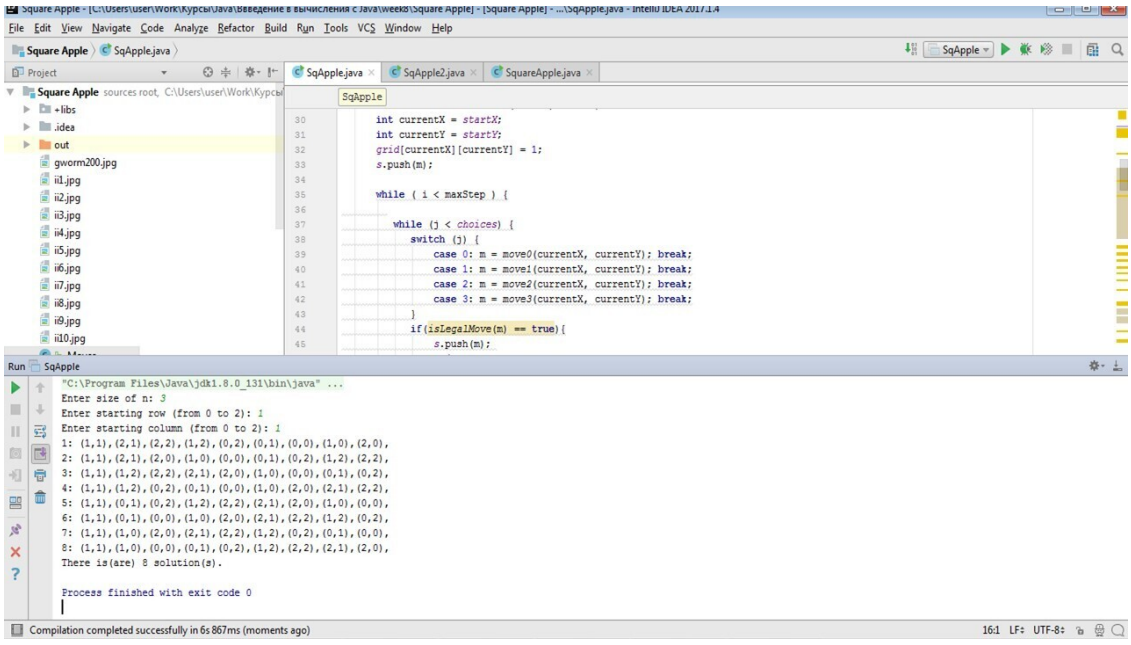
Вот условие для проверки пустого стека.

И здесь условие для проверки, все ли возможные ходы исчерпаны, и вытолкнуть предыдущий шаг из стека.

Когда есть maxStep элементов в стеке, решение найдено.

Последний шаг вынимается из стека в целях поиска нового решения.

Давайте скомпилируем и запустим программу для 3x3 и начнем с середины.



Как и ожидалось, есть 8 решений.

Тем не менее, если начальное положение находится на какой-либо стороне, в этом случае из строки 0 и столбца 1, то решение не может быть найдено.

Вы обнаружите, что сложность проблемы растет очень быстро.

При n = 5, если мы начнем с угла, существует более 700 решений.

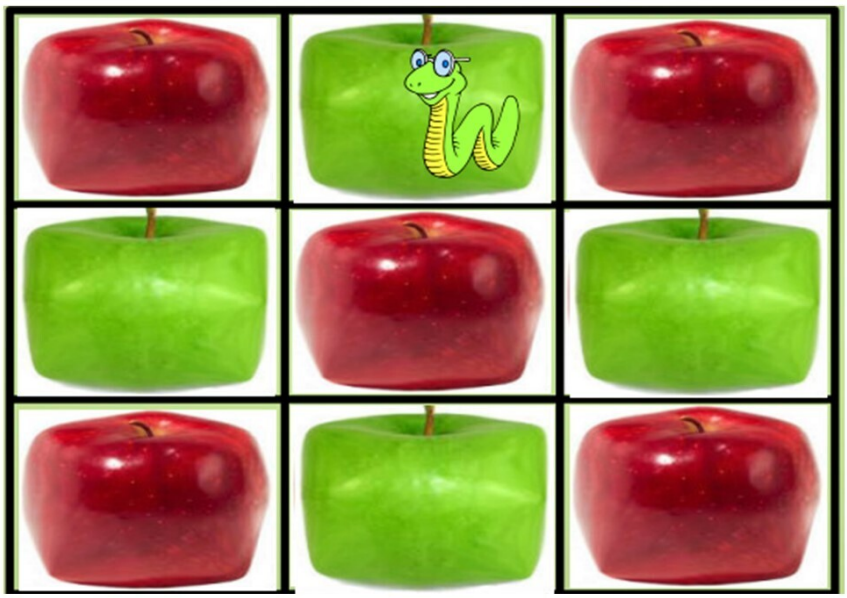
Вы найдете, что для n = 7, это займет более двух часов, чтобы найти более 1,5 млн решений для запуска в позициях, где существуют решения.

Но для положений, которые не приводят к решениям, программа все равно должна пройти через исчерпывающий поиск в течение более двух часов, прежде чем объявить, что решения не существует.

Вы найдете, что сложность программы составляет порядка 2 в степени n в квадрате, и для n, равного 7, это 2 в степени 49.

Так что сложность задачи здесь аналогична сложности задачи башня Ханоя.

Но если вы до сих пор помните, если проблема представлена немного по-другому, в частности, заполнение ячеек с альтернативными красными и зелеными яблоками, можно утверждать, что не будет никакого решения, если червь начинает с цвета с меньшим количеством яблок.

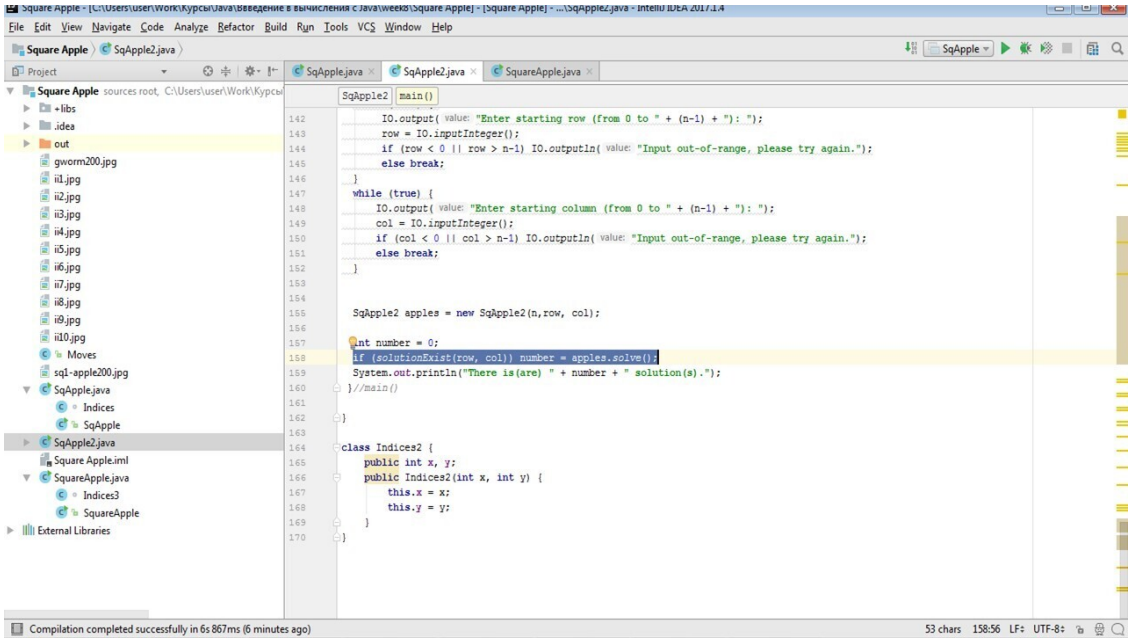


В этом примере, не существует решения, если червь начинает с зеленого яблока, потому что есть только 4 зеленых яблок, в то время как есть 5 яблок красного цвета.

Используя эту идею, мы можем легко добавить метод для выявления таких случаев и сэкономить время в поисках решения.

```
public static boolean solutionExist(int row, int col) {
    int sum = row + col;
    return (sum/2*2 == sum);
}
```

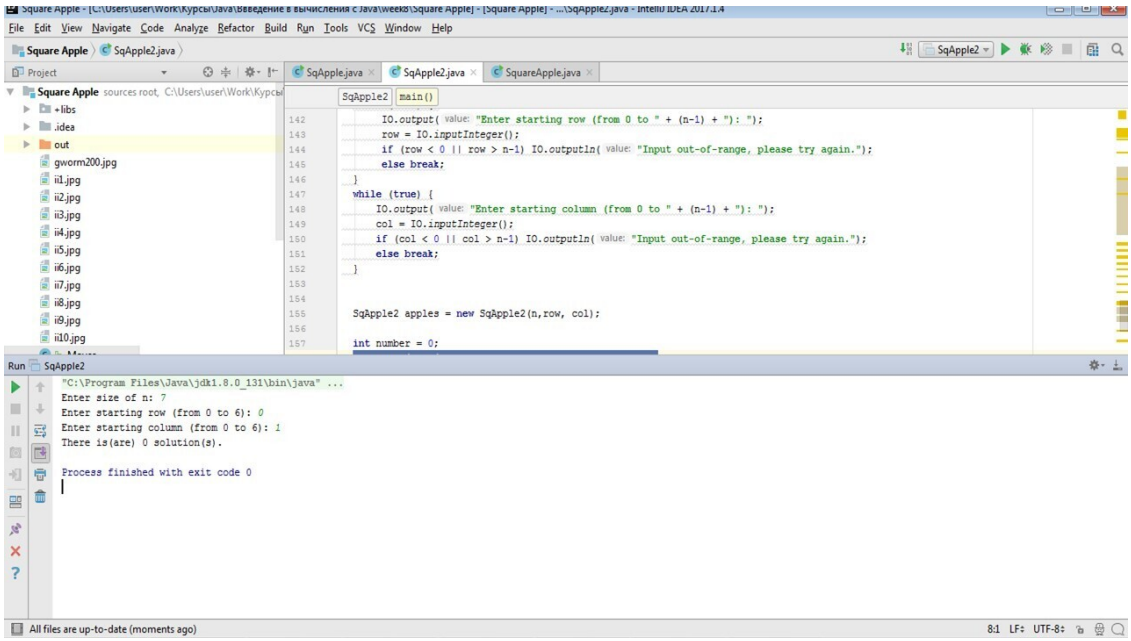
Таким образом, в этой модифицированной реализации программы SqApple, логический метод solutionExist будет выполнять такое тестирование.



И поиск будет выполняться только тогда, когда решение существует.

Давайте попробуем еще раз запустить программу в ее новой версии.

Скажем, если n равно 7, и если мы начнем со стороны, скажем, для индекса строки 0, индекса столбца 1, вы можете видеть, что решения никакого нет, потому что стартовая позиция на стороне.



Однако если вы запустите программу снова, для положения, где решение существует, поиск будет по-прежнему занимать много времени, чтобы найти решение для больших n.

Вопросы

Задача

Учитывая следующий метод:

```
public static int f(String input) {
Stack<Character> stack = new Stack<Character>();
int n = 0;
for (int i = 0; i < input.length(); i++) {
if (input.charAt(i) == 'd')
stack.push('d');
else if (input.charAt(i) == 'b') {
if (!stack.empty() && stack.pop() == 'd') n++;
}
}
return n;
}
```

Какими будут возвращаемые значения для указанного выше метода, если вход равен:

(i) "bdbdbd"

(ii) "dddbbb"

(iii) "buddy"

Варианты ответа:

1. i) 3 ii) 0 iii) 1
2. i) 3 ii) 3 iii) 1
3. i) 2 ii) 3 iii) 0
4. i) 2 ii) 1 iii) 0

Ответ: 3.

Объяснение.

Программа считывает символы из входной строки, по одному.

Всякий раз, когда программа читает символ 'd', она вносит символ 'd' в стек.

Всякий раз, когда программа читает символ 'b', она вынимает символ из стека и проверяет, является ли он это символом 'd'.

Если это действительно 'd', переменная счетчика n будет увеличена на 1.

Целью программы является соединить символ 'b' с ранее считанным символом 'd' и подсчитать количество уникальных пар.

Таким образом, для ввода "bdbdbd" мы можем найти 2 пары.

Для ввода "dddbbb", мы можем найти 3 пары.

Для ввода "buddy", мы не можем найти какие-либо пары.

Задача

Как много различных решений (зеркальные решения считаются различными) будут у задачи n-ферзей с:

n=3

n=4

n=5

n=6

n=7

Ответ:

0

2

10
4
40
Объяснение.
http://en.wikipedia.org/wiki/Eight_queens_puzzle#Counting_solutions
Задание
Завершить метод leaveQueue программы CarQueue, следуя инструкциям, указанным в коде.