```
        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

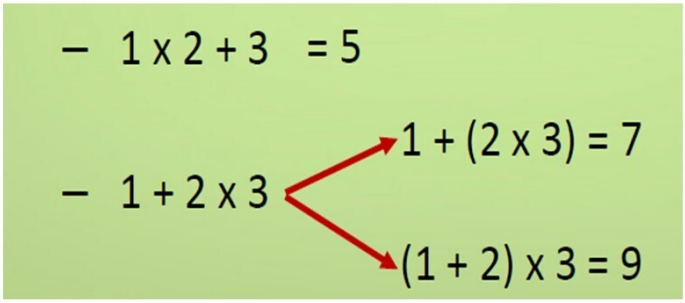
Все мы знаем, что основная цель использования компьютера, это помочь нам в решении задач. И мы используем компьютеры для решения задач, потому что они, как правило, более эффективны и надежны в решении тех или иных задач.

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.

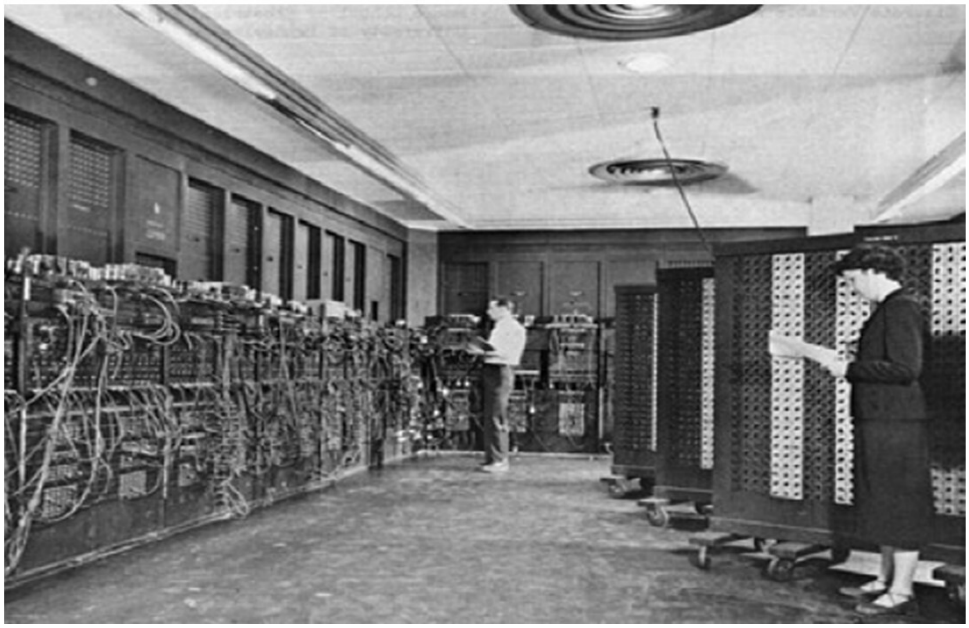
Аппаратные средства

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.

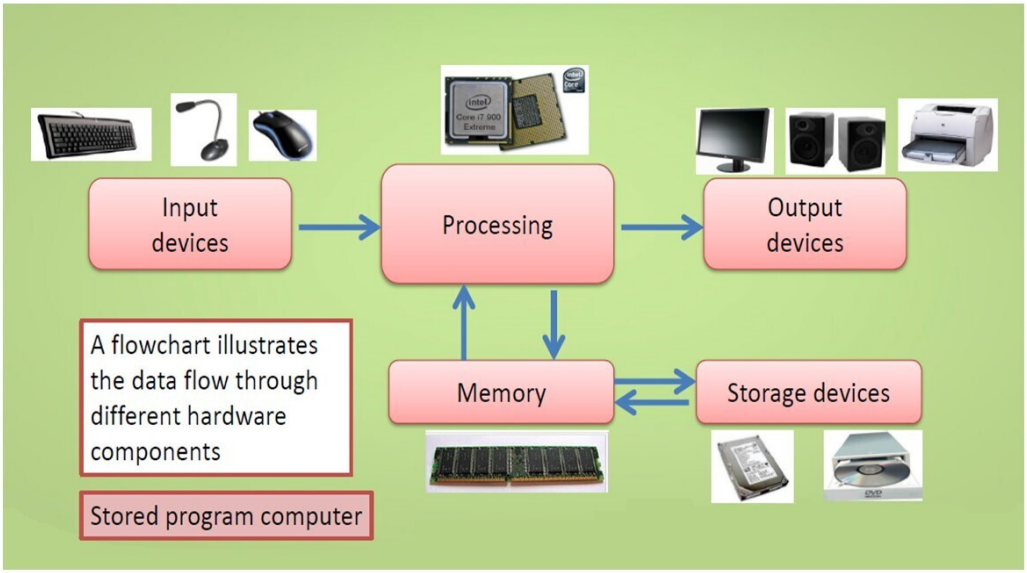


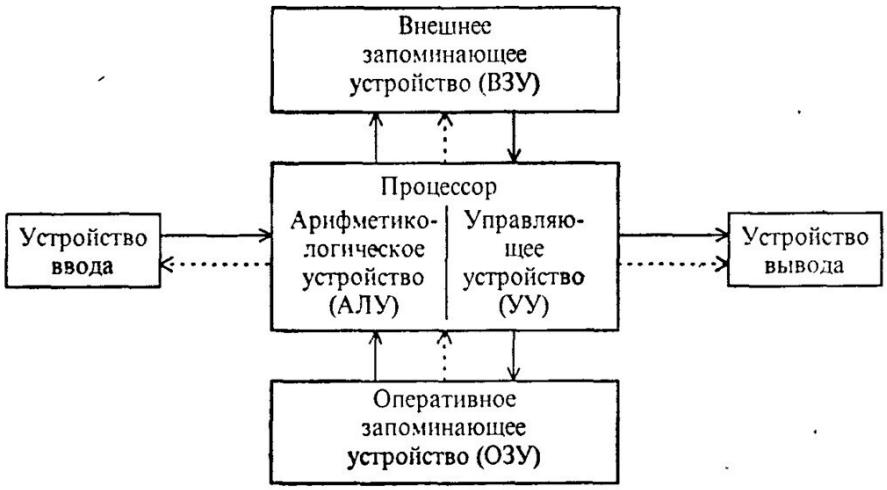
Диаграмма здесь иллюстрирует поток данных с помощью различных аппаратных компонентов.

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.

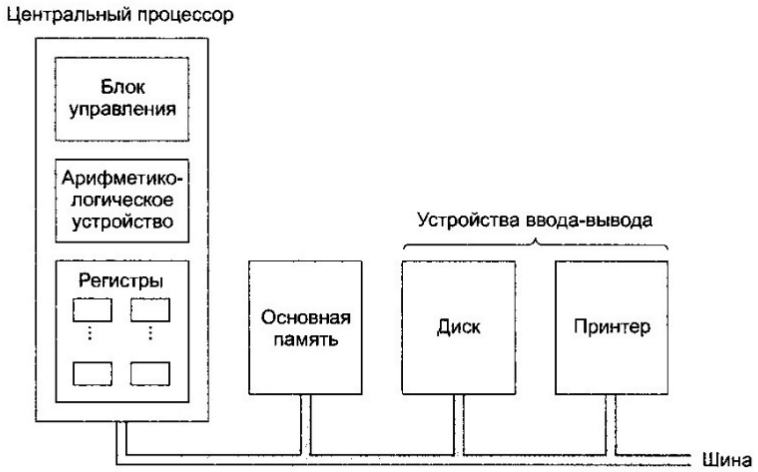
Управление последовательностью, в которой команды выполняются.

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

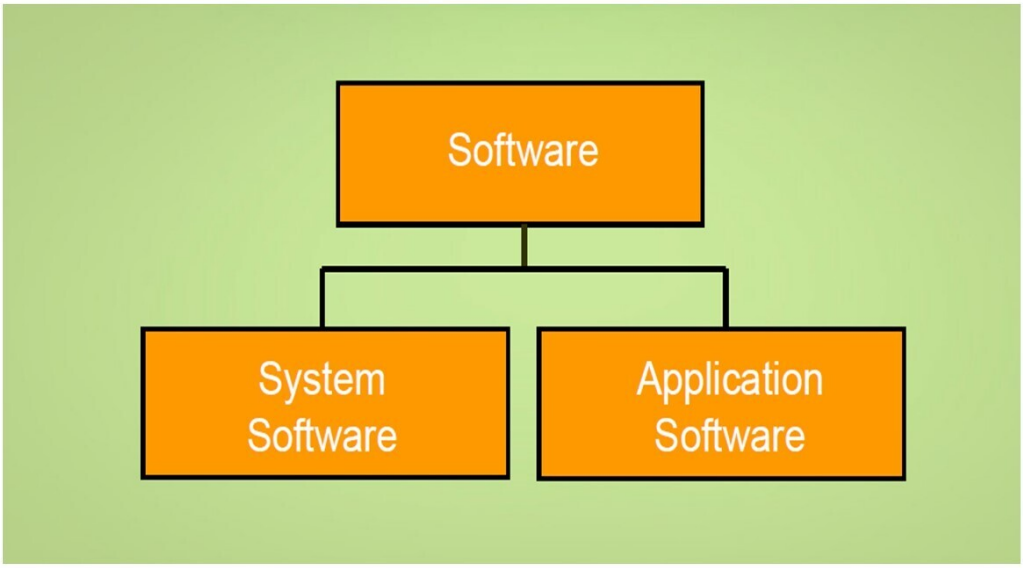
Давайте продолжим обсуждение основных компонентов компьютерных систем и рассмотрим программное обеспечение компьютера.

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.

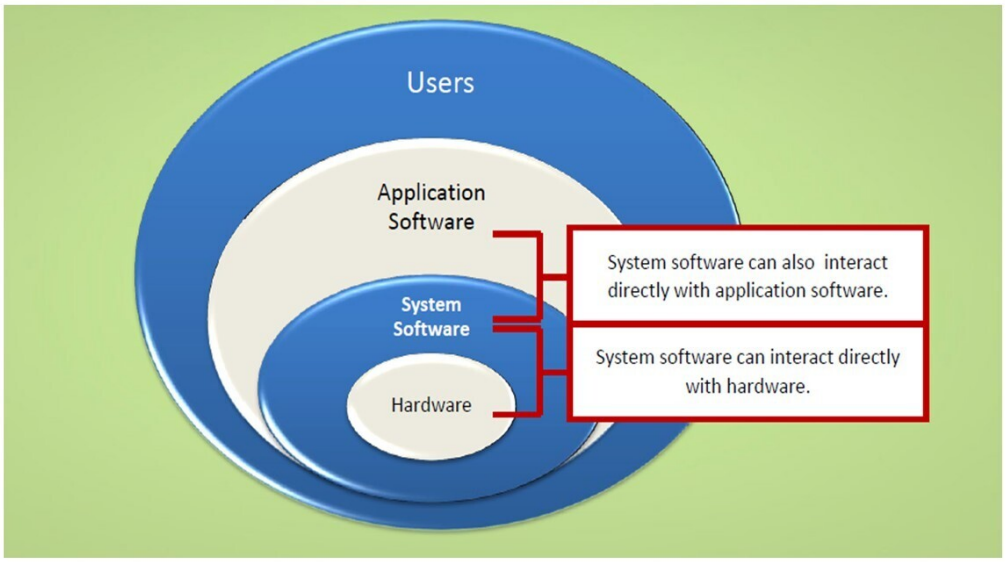
Системное программное обеспечение, это программы, которые поддерживают выполнение и разработку других программ.

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.

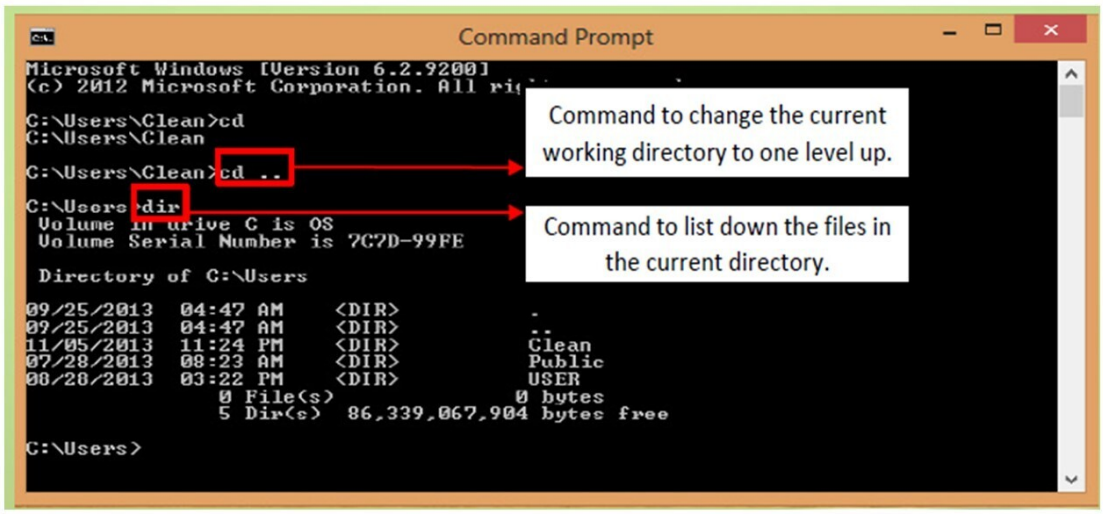
Важной частью ОС является обеспечение пользовательского интерфейса между операционной системой и пользователем.

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.

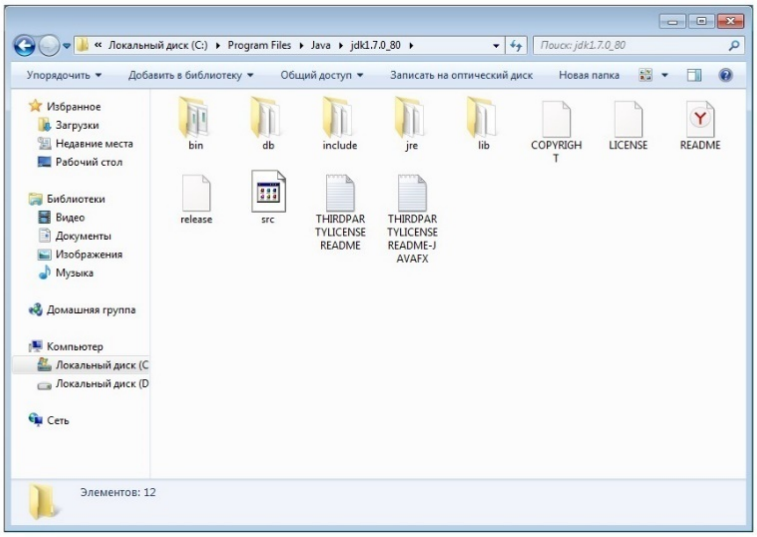
Из этого простого примера, можно понять, почему такие системы стали настолько непопулярны и редко используются, за исключением системных программистов или системных администраторов.

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.

Формулировка задачи: определение и анализ задачи.

Что такое вход и выход?

Какая другая информация необходима?

Найдите подходящее представление задачи.

Разработать алгоритм.

Каковы шаги для решения задачи?

Последовательность точных шагов для выполнения функции.

Реализация программы.

Компилирование, тестирование и отладка программы.

Документирование и поддержка программы.

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.

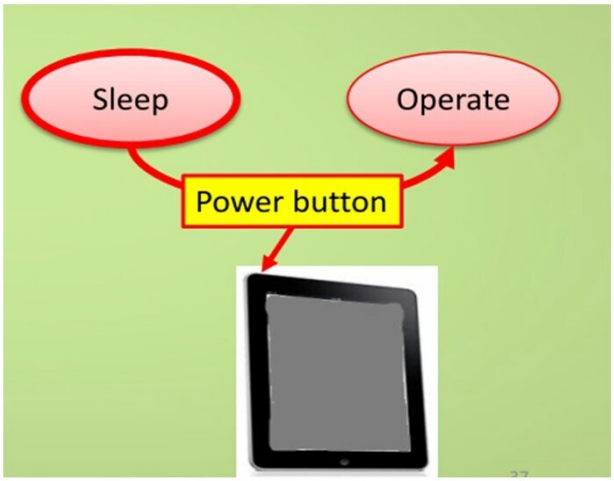
Пространство состояний, это множество всех возможных состояний, в которых задача может находиться

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".

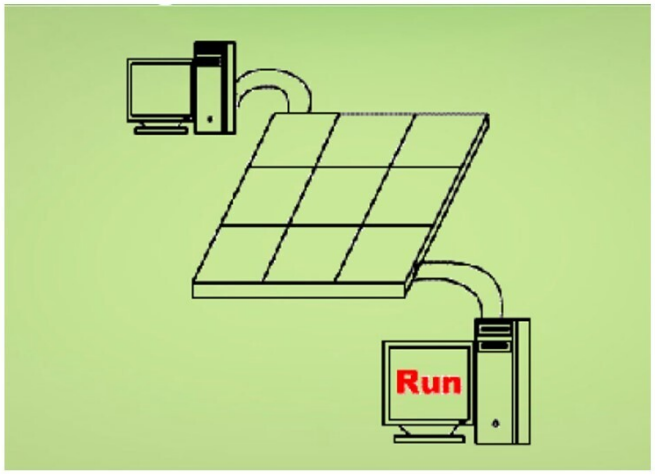
Пользователь может переключиться на «спящее» состояние смартфона снова, нажав на кнопку питания.

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.

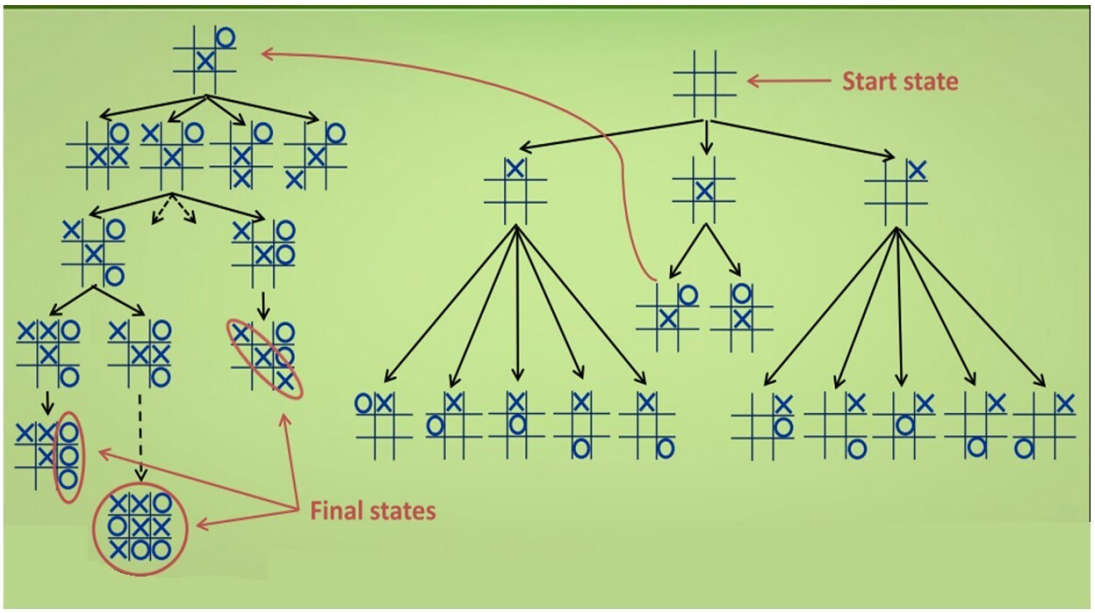
И тот, кто получает 3 крестика или нолика в ряд первым либо горизонтально, либо вертикально, либо по диагонали будет победителем.



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.

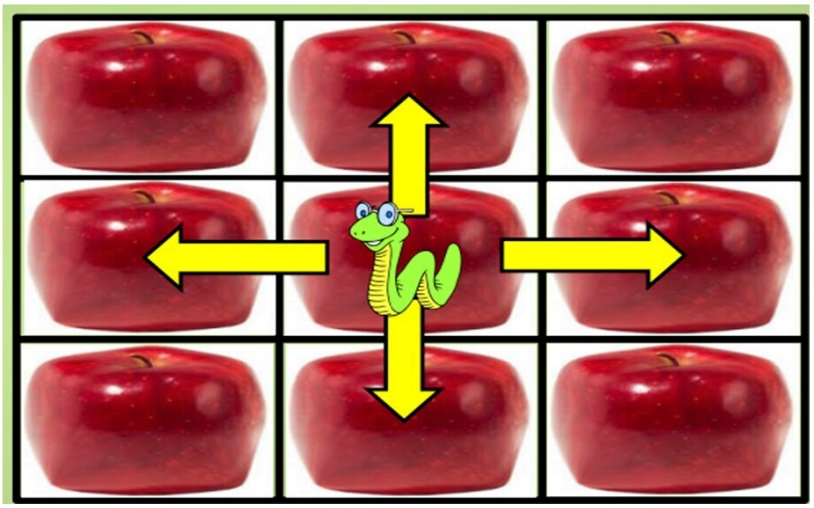
Есть много больше конечных состояний, которые не показаны здесь.

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.

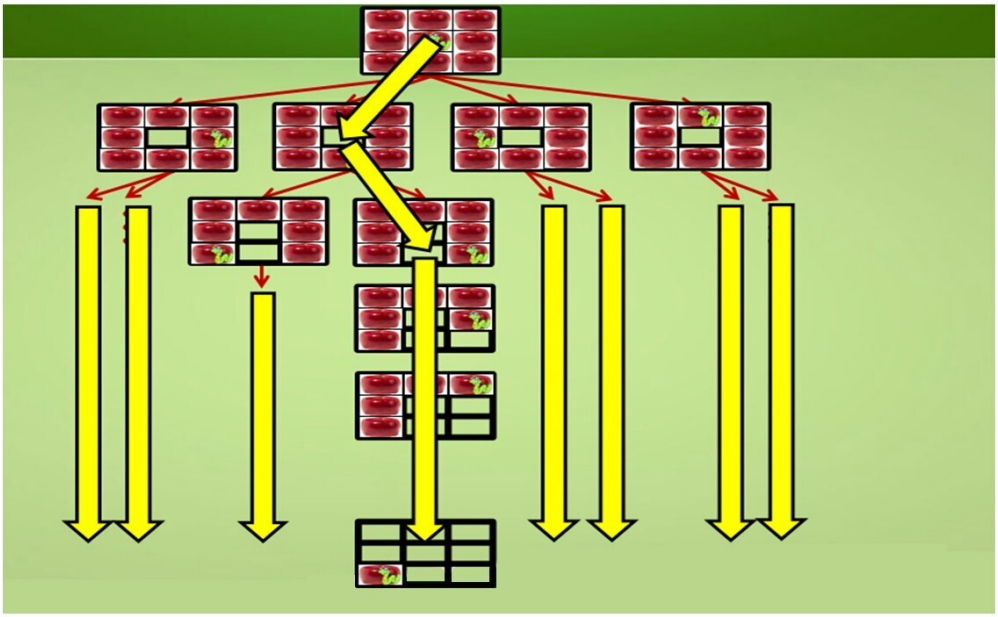
То есть, червь может двигаться только в ячейку, где есть еще несъеденное яблоко внутри.

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.

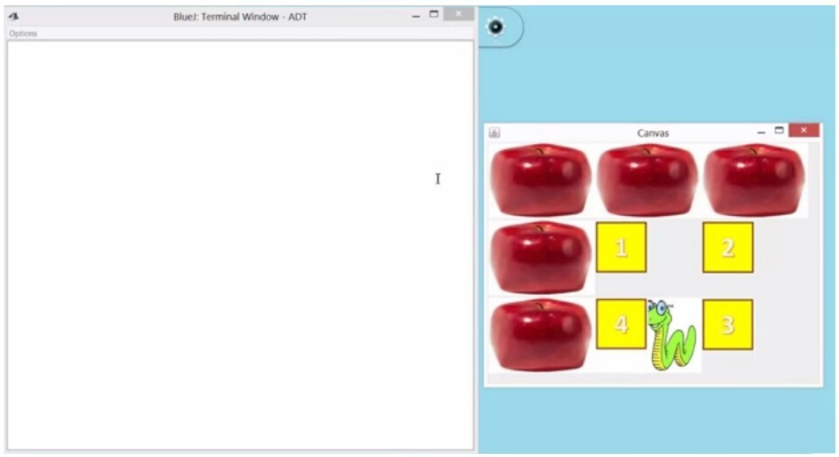
На самом деле, вы обнаружите, что есть 7 других пути для решения, пробуя другие ходы.

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.

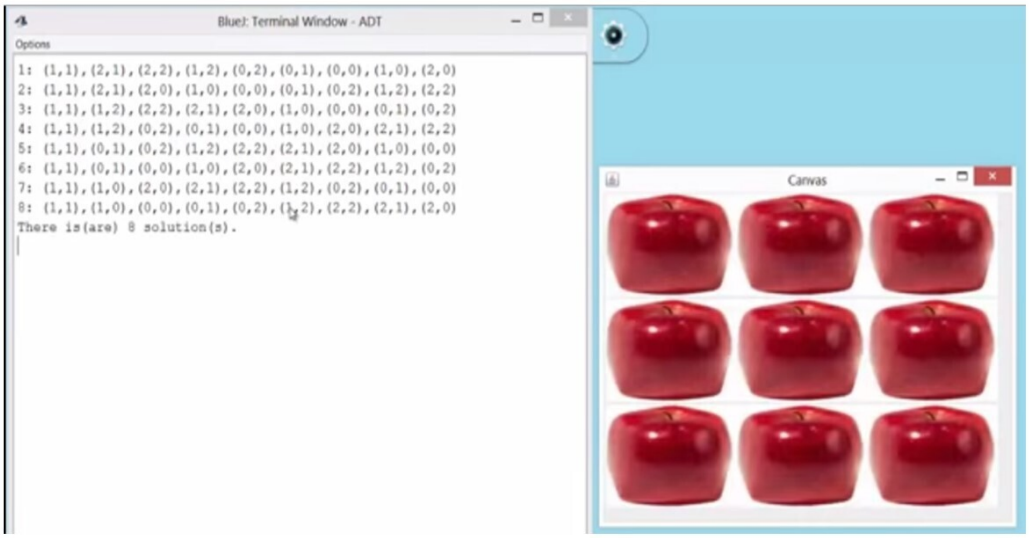
Исходный код к данной книги вы можете скачать по адресу https://github.com/novts/java-base.



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

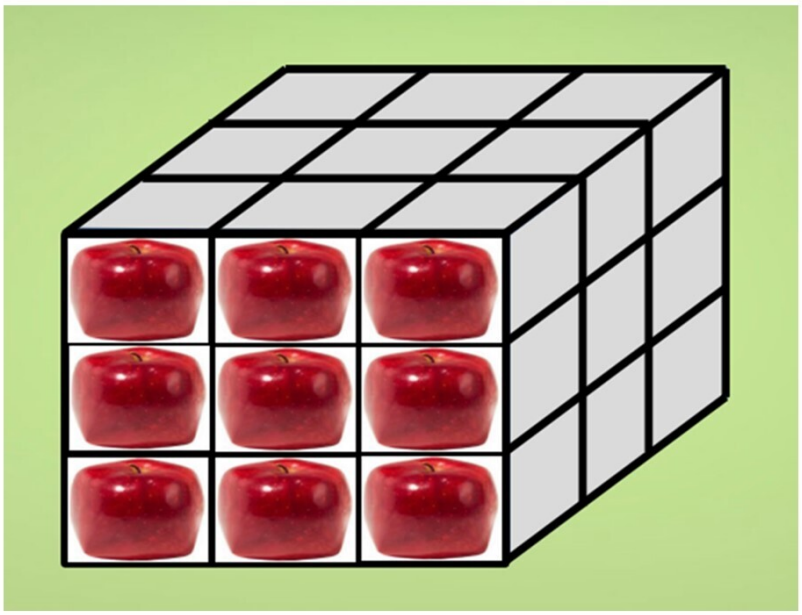
Отображая условие – не посещать клетки, которые были захвачены ранее, цифры показывают последовательность шагов.

Программа также отображает последовательность шагов в другом окне.

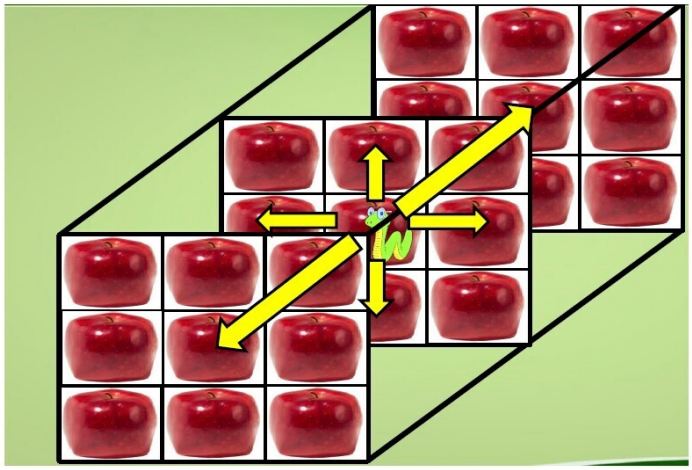


Таким образом, вы можете видеть, что есть в общей сложности 8 решений, как обсуждалось ранее.

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.



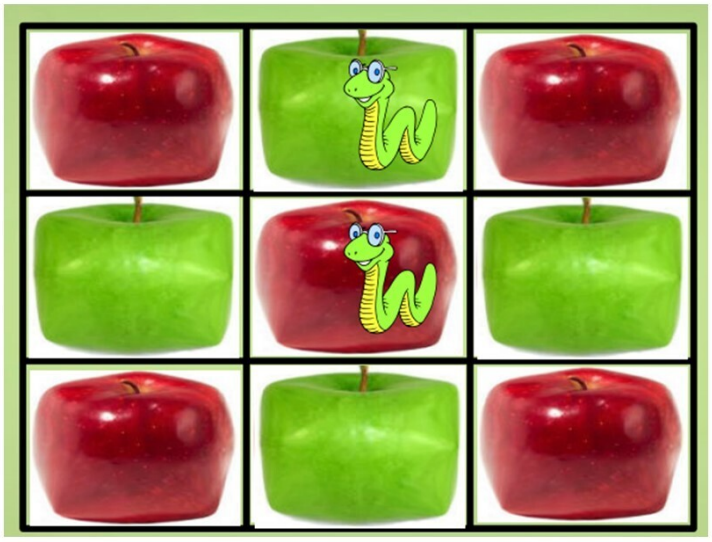
Подобно тому, что у нас было раньше, червь прячется в середине, и мы должны следовать тем же правилам.

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.

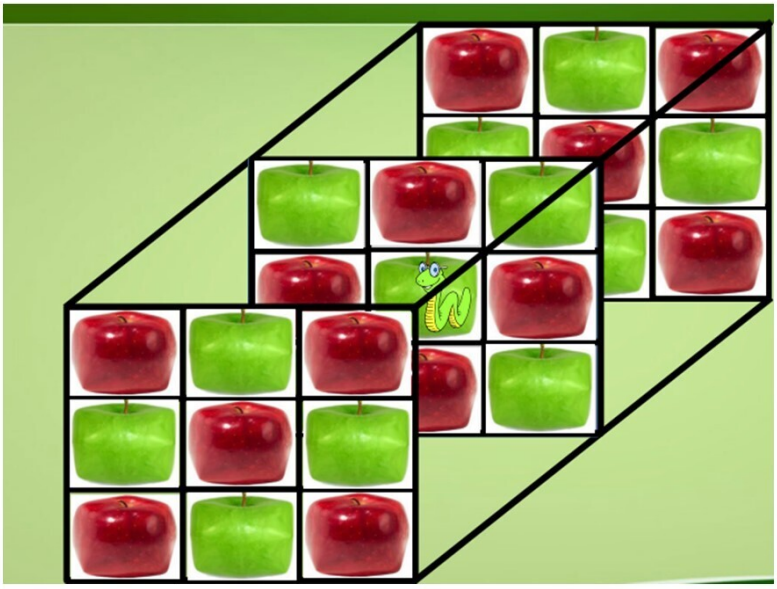


Если червь опять должен начать с середины, следуя тем же правилам, это та же задача, как и раньше, независимо от цвета яблок, и у нас есть те же 8 решений, как и в предыдущем случае 2D.

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.

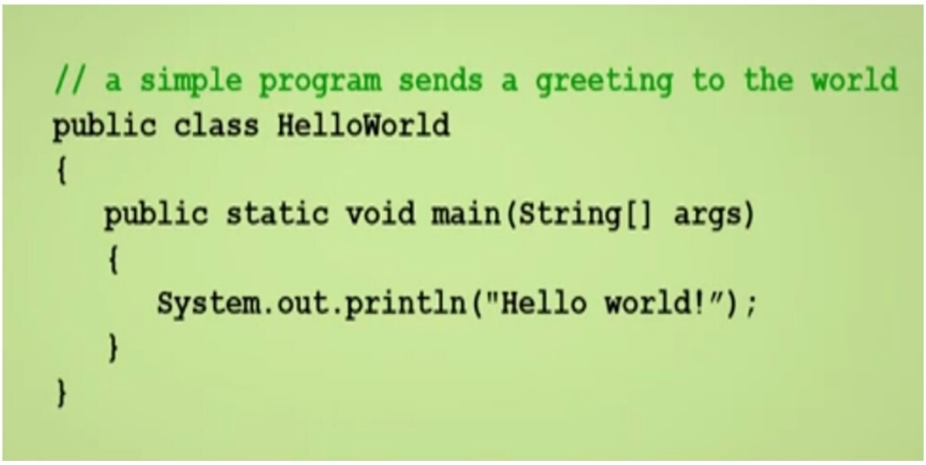
На самом деле, тот же аргумент может быть использован для более широкой задачи, скажем 5x5x5, 7x7x7, или даже больше.

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.

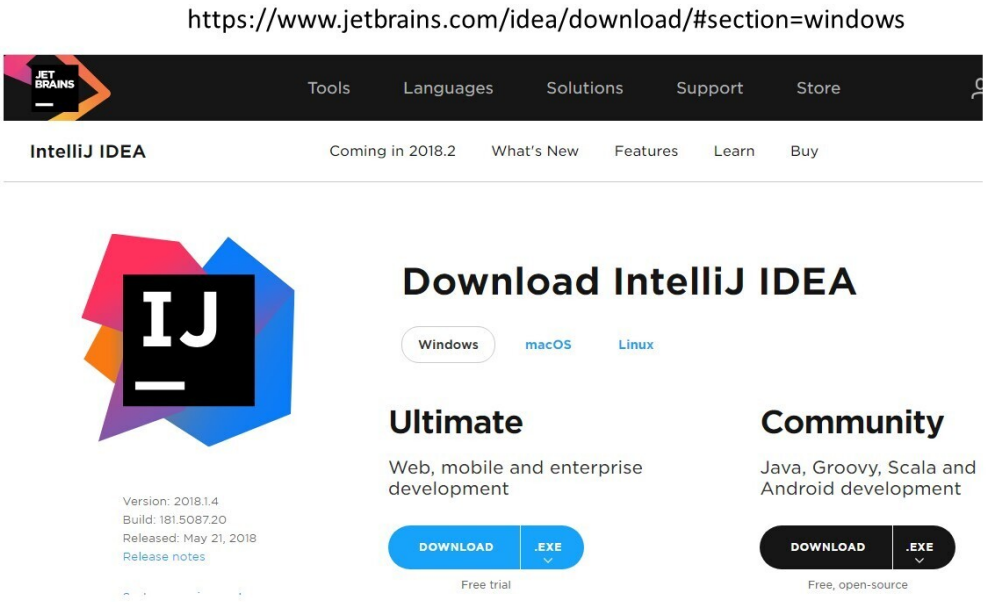
Вы можете смотреть на это как на какой-то формат или шаблон, которому вы должны следовать.

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".

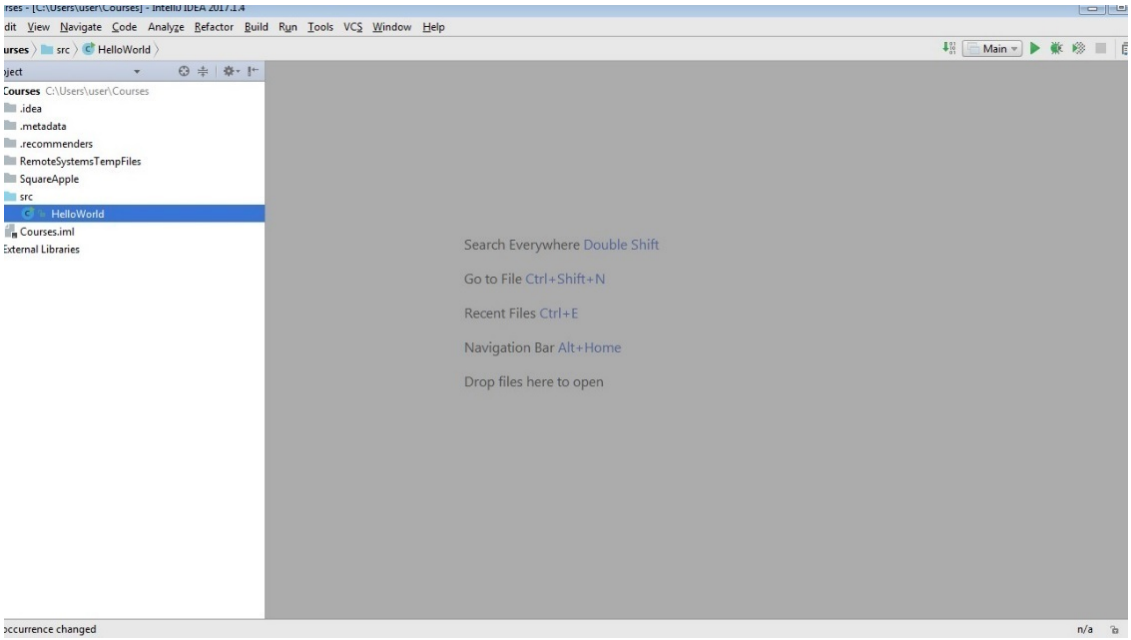
Перейдем теперь к среде разработке, чтобы посмотреть на программу более детально.



Скачайте и установите среду разработки IntelliJ IDEA версии Community.

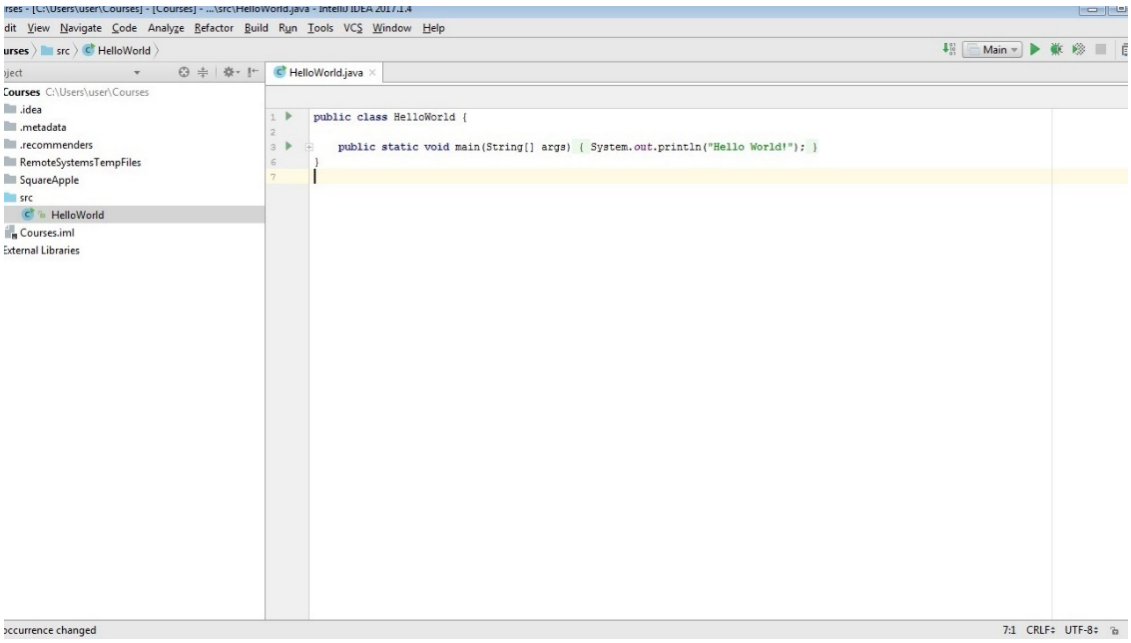
И скачайте проект приложения, который доступен в ресурсах к этой книги (https://github.com/novts/java-base).

Распакуйте этот проект и откройте его в среде разработки.

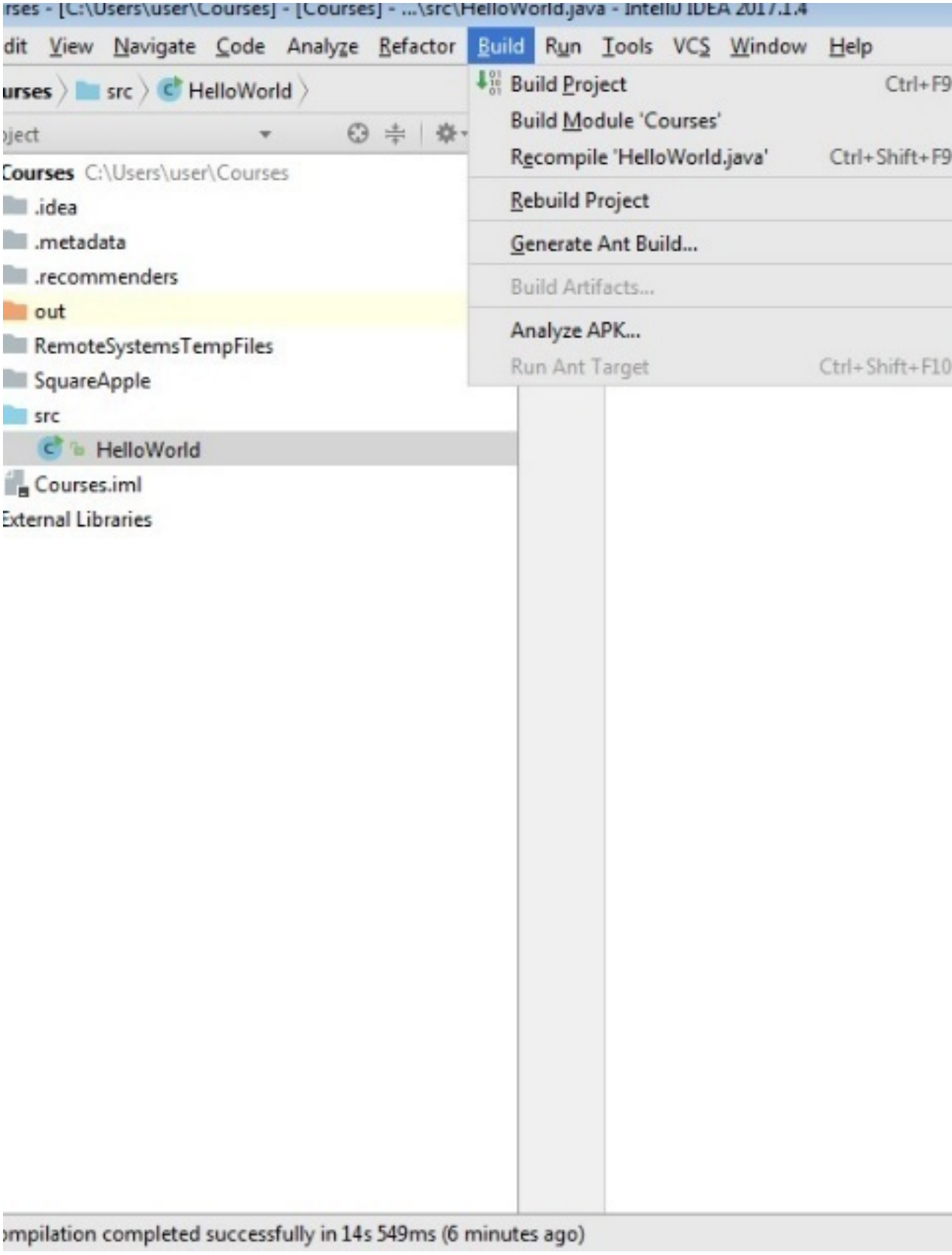


Вот этот файл, соответствующий классу приложения.

Вы можете дважды щелкните по нему, чтобы открыть программу.

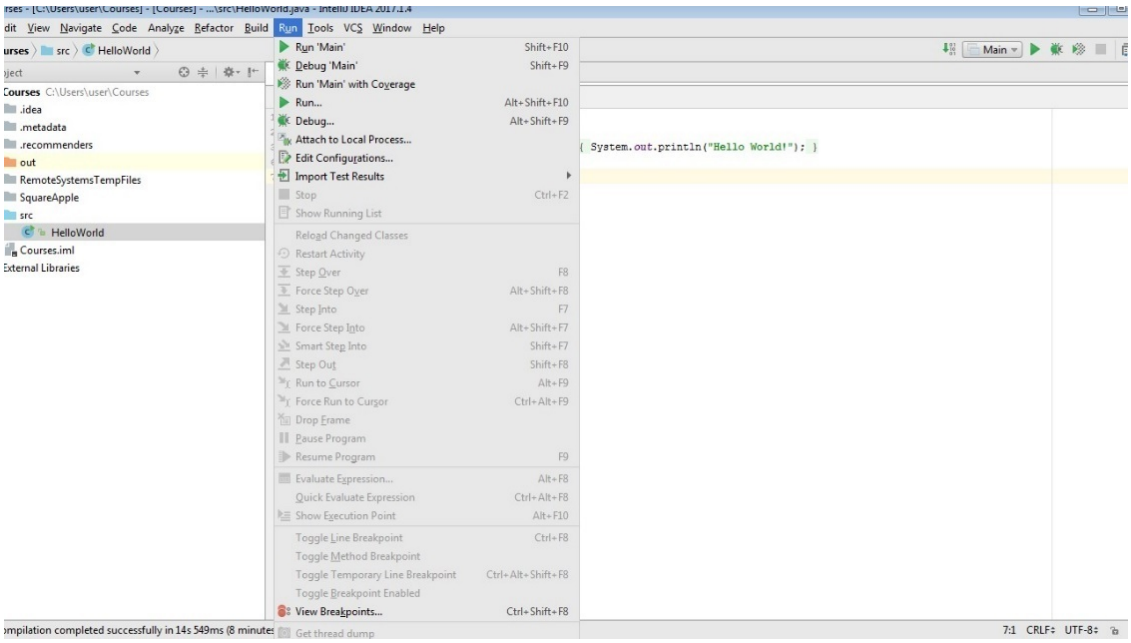


И вы можете скомпилировать программу, нажав на кнопку "Build Project".

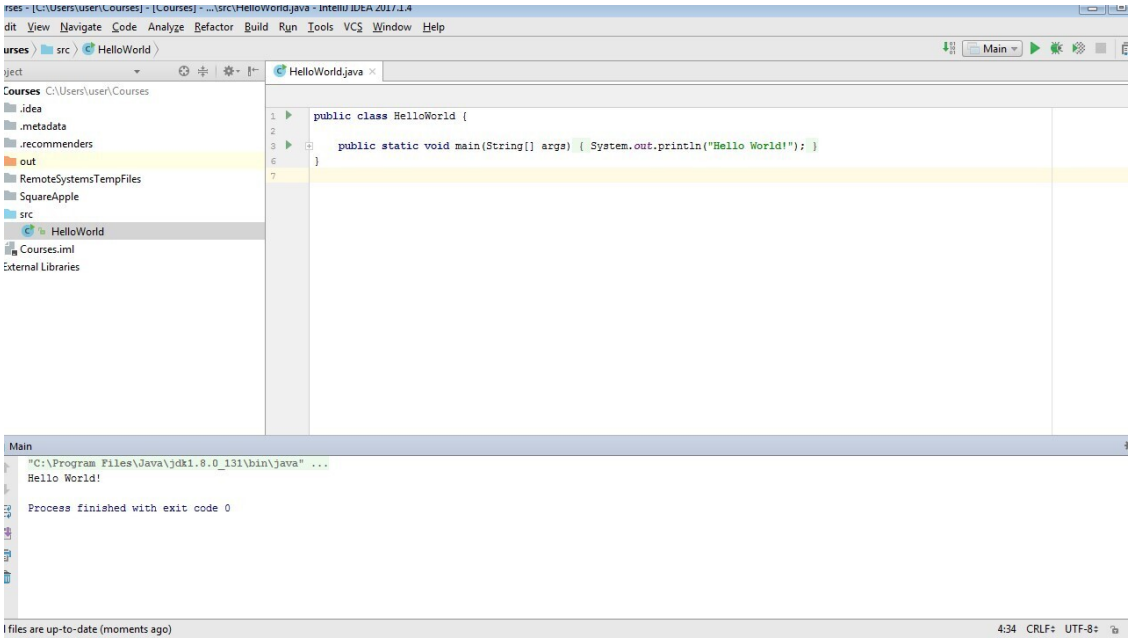


И вы можете увидеть, что программа составлена правильно, так как в сообщении будет указано, что здесь нет ошибок синтаксиса.

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



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

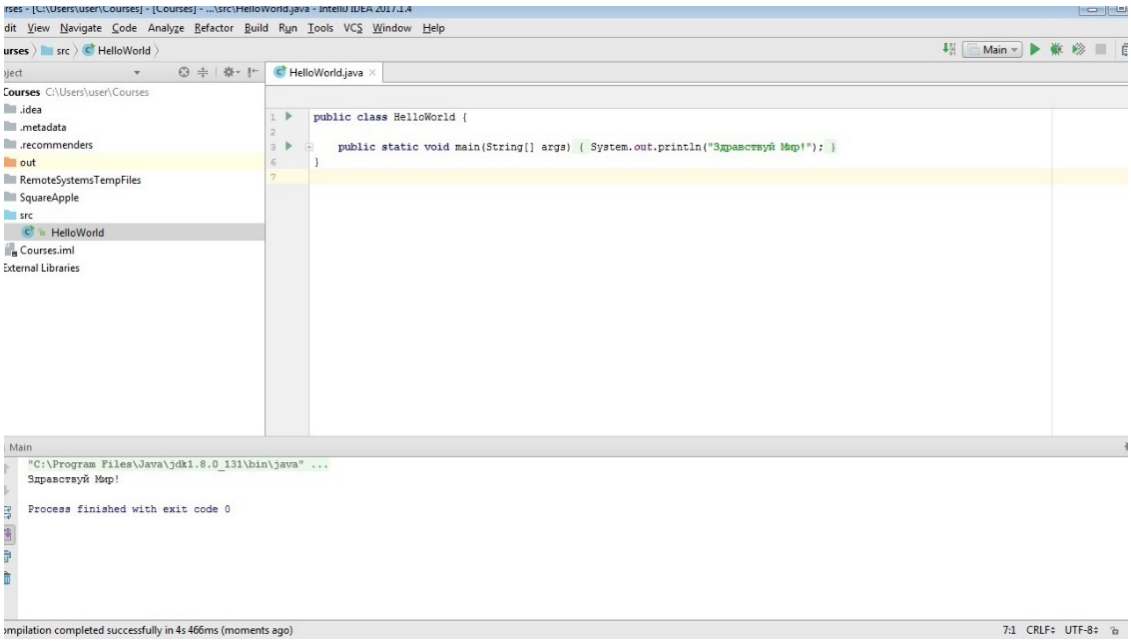


Поздравляем! Вы успешно написали свою первую программу Java.

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.

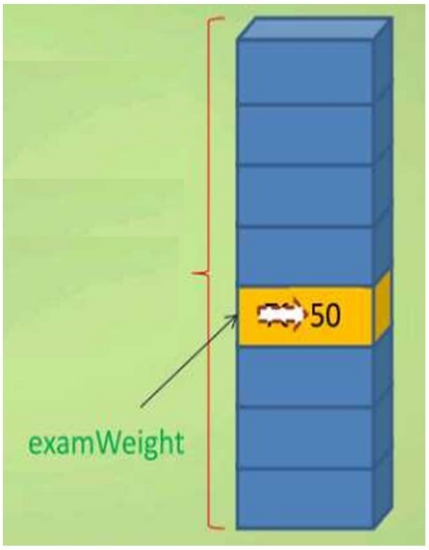
Знак равенства, который вы видите здесь это оператор присваивания, в данном случае, examweight = 70 является утверждением присваивания.

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.

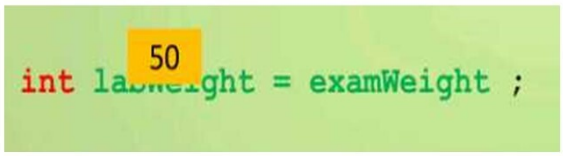
После объявления переменной и инициализации, ее значение может быть изменено с помощью оператора присваивания.

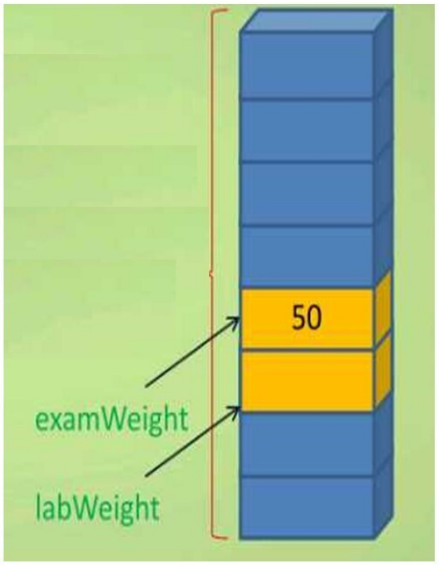
```
examWeight = 50;
```



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

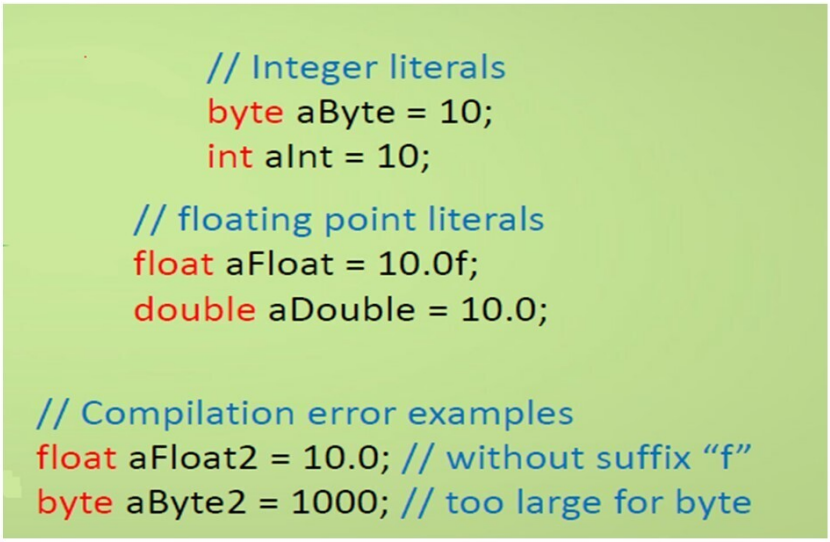
Цельсий, – это переменная, 9, 5 и 32 – литералы, и литералы соединены тремя операторами *, / и +.

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.

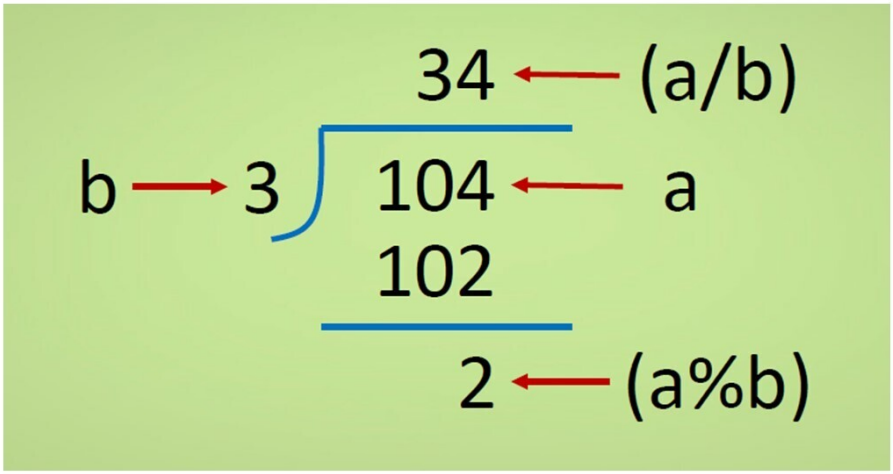
Например, если 2 разделить на 3, результат будет 0.

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.

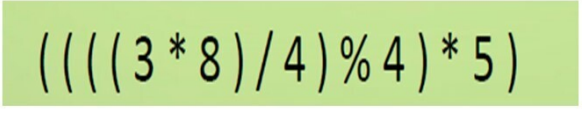
Приоритет операторов задает порядок, в котором различные операторы выражения вычисляются.

( )

* / %

\+ -

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.

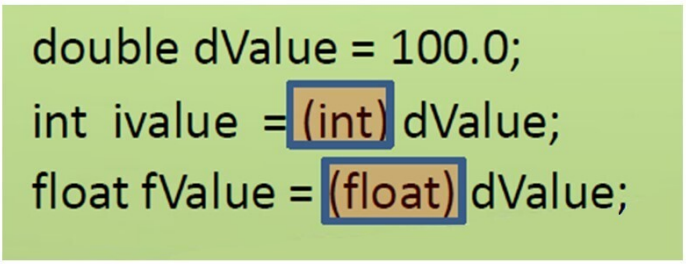
Например, целый тип int разрешено присваивать типу float, хотя при этом может быть потеряна точность.

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

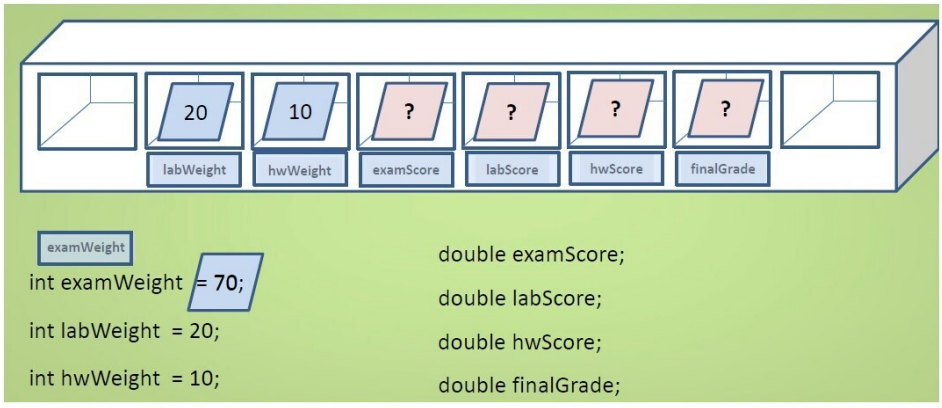
Давайте теперь используем пример CourseGrade для того, чтобы проиллюстрировать эффект объявления и выполнения программы.

```
/**
 * 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.

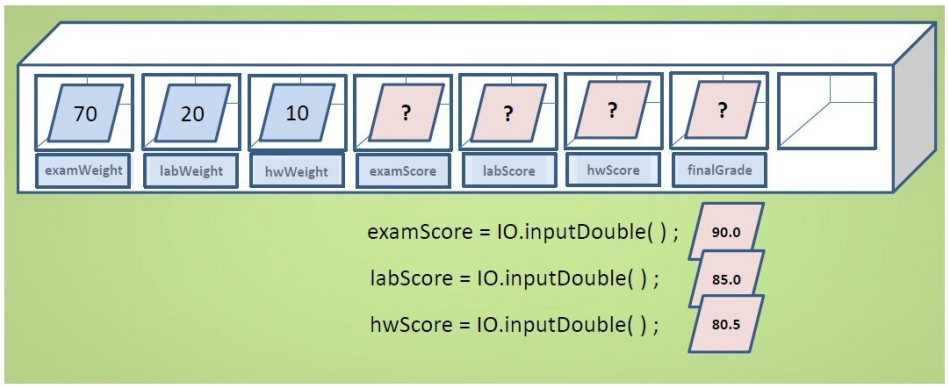
Подобное выделение памяти будет сделано для labScore, hwScore и finalGrade.

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.

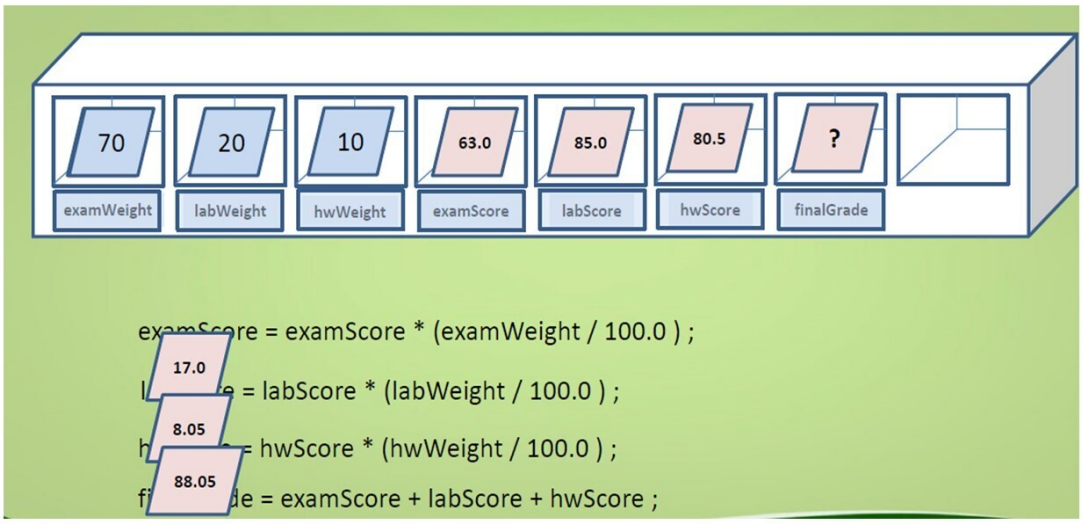
Следует отметить, что examWeight представляет собой целое число, и если оно делилось бы на другое целое число 100, то результатом был бы ноль.

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.

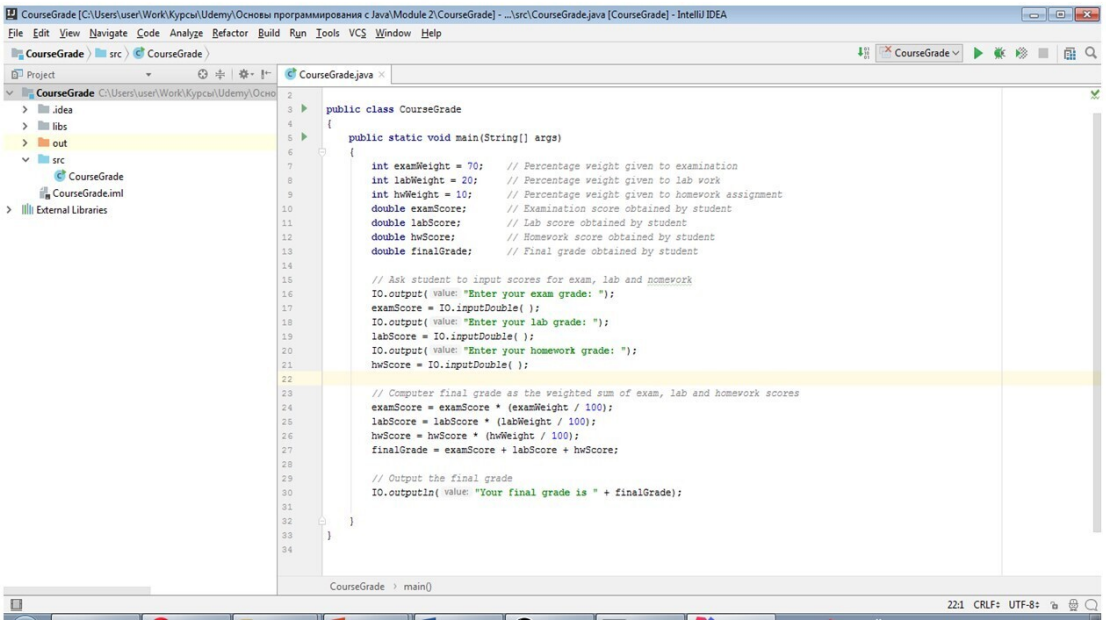
```
/**
 * CourseGrade determines the final grade which is computed as
 * the weighted sum of the grades obtained in exam, lab and homework
 */
public class CourseGrade
{
    public static void main(String[] args)
    {
        final int examWeight = 70;  // Percentage weight given to examination
        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.

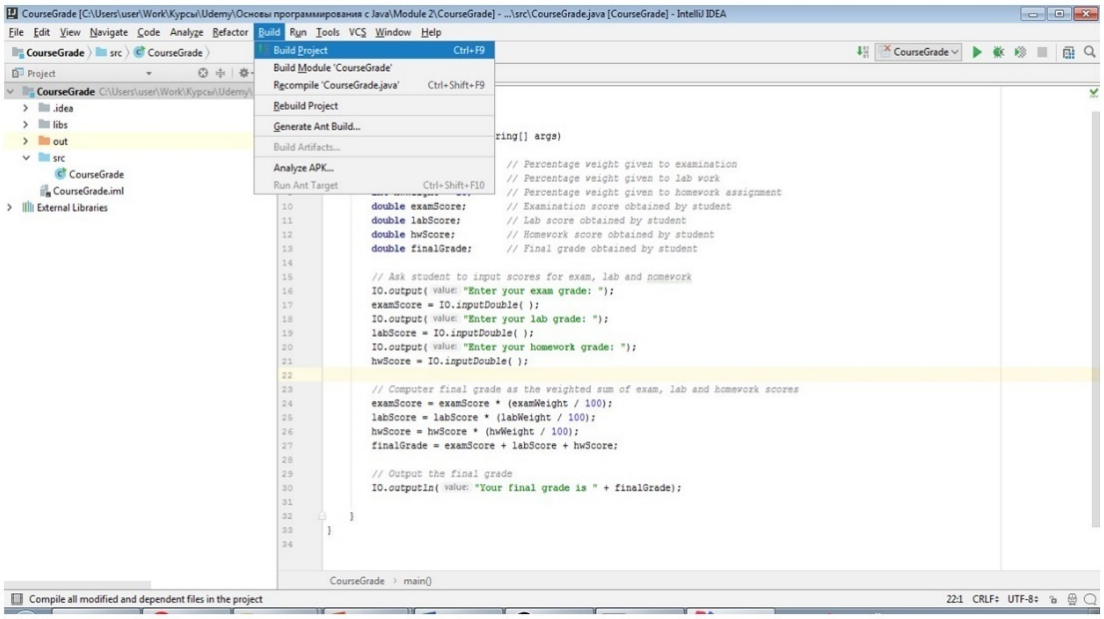
Можно сказать, что, если студент сдал плохо экзамены, но сделал хорошо лабораторные работы, попытка уменьшить вес для экзамена и увеличить вес для лабораторных будет блокирована.

Демонстрация примера

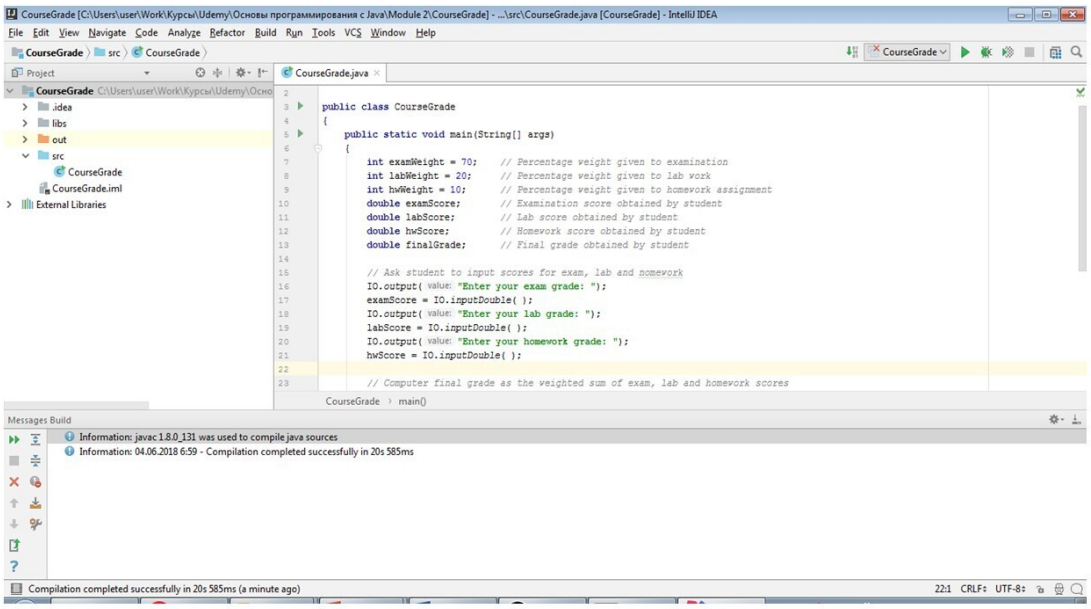
Давайте теперь посмотрим на программу в среде IntelliJ IDEA. Мы откроем проект под названием CourseGrade, который является программой, которую мы только что обсуждали.



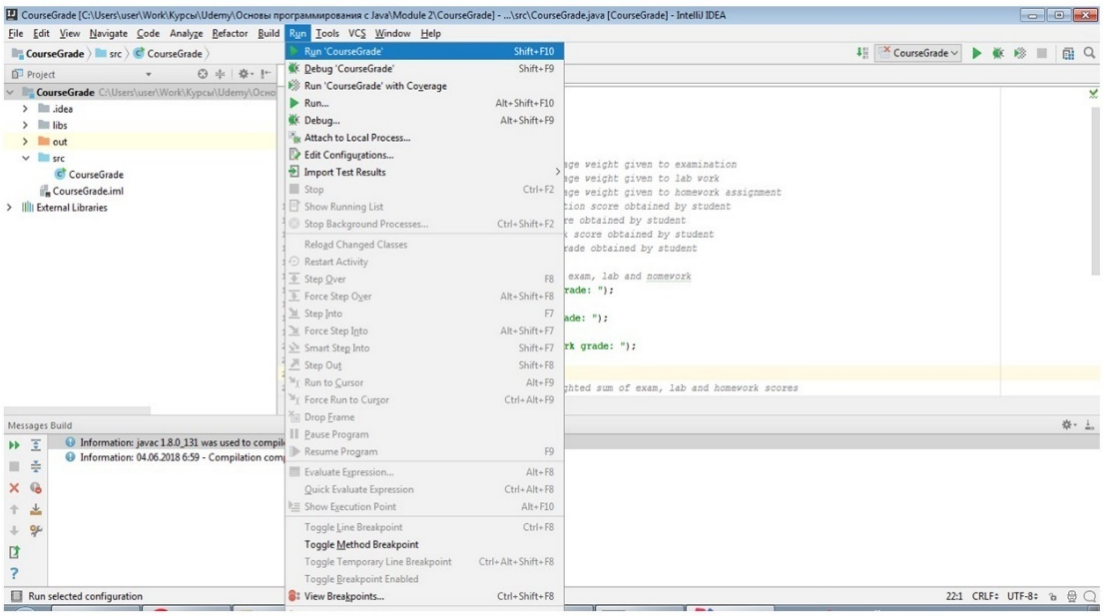
Откроем файл CourseGrade в редакторе исходного кода.
И вы можете видеть, что эта программа та, которую мы только что обсуждали.
И эта программа еще не скомпилирована.
Попробуем скомпилировать программу с помощью меню Build Project.



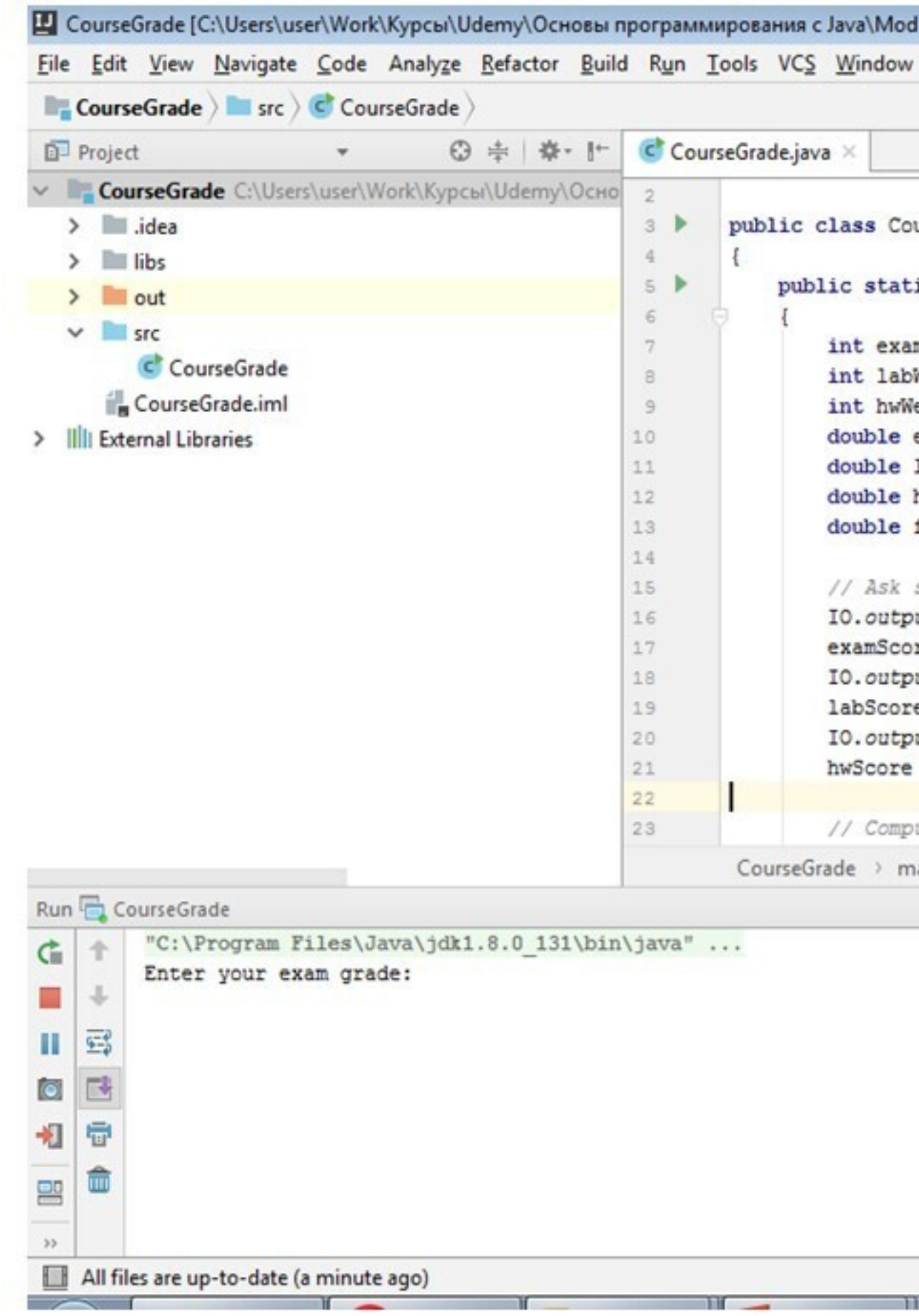
Вы можете видеть, что программа успешно компилируется без ошибок синтаксиса.



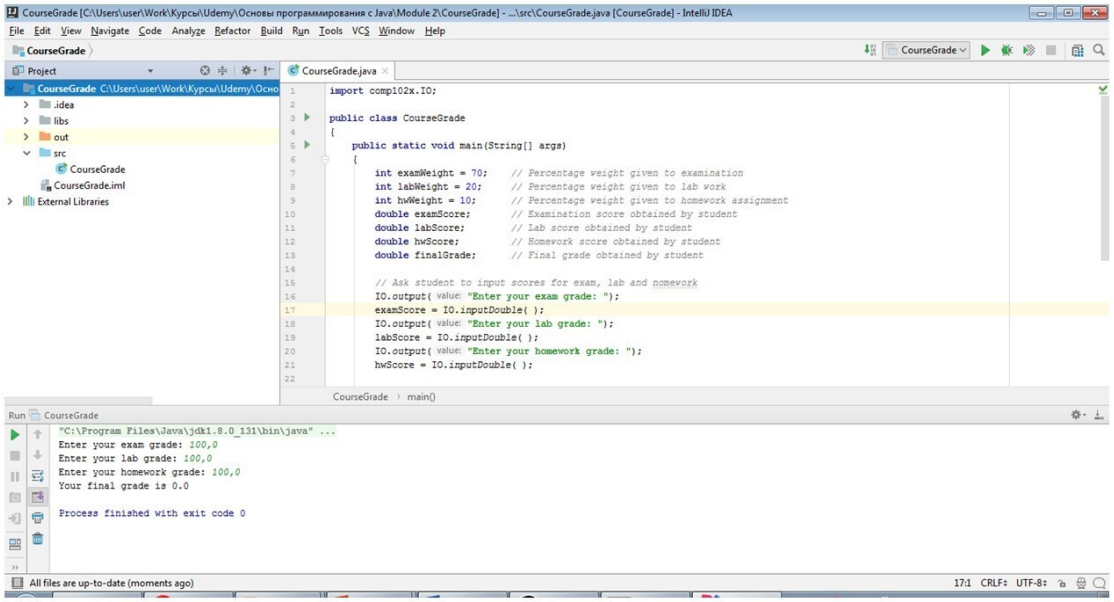
Давайте попробуем запустить программу с помощью кнопки Run.



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.

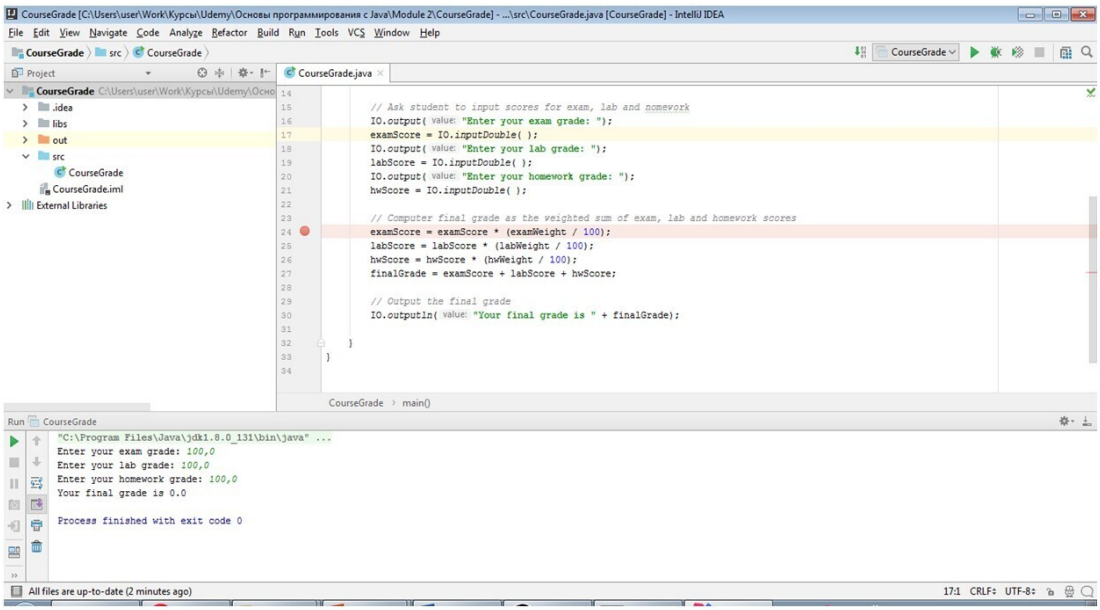
И в результате, по какой-то причине, итоговая оценка вернулась программой как 0.0.

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.

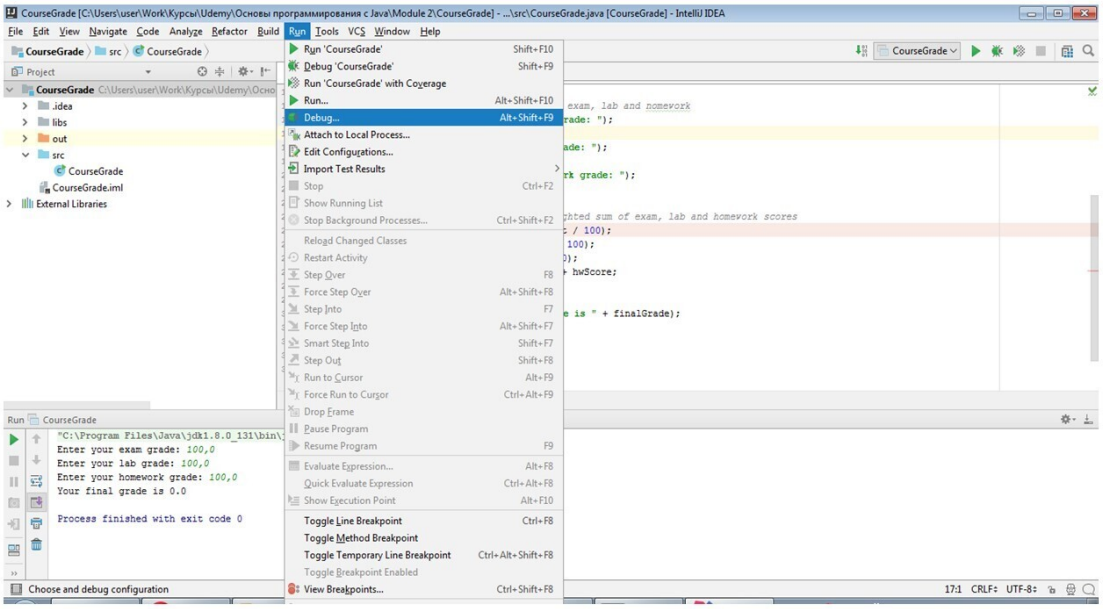


Для этого просто нажмем на соответствующее место вдоль правого столбца в окне редактора.

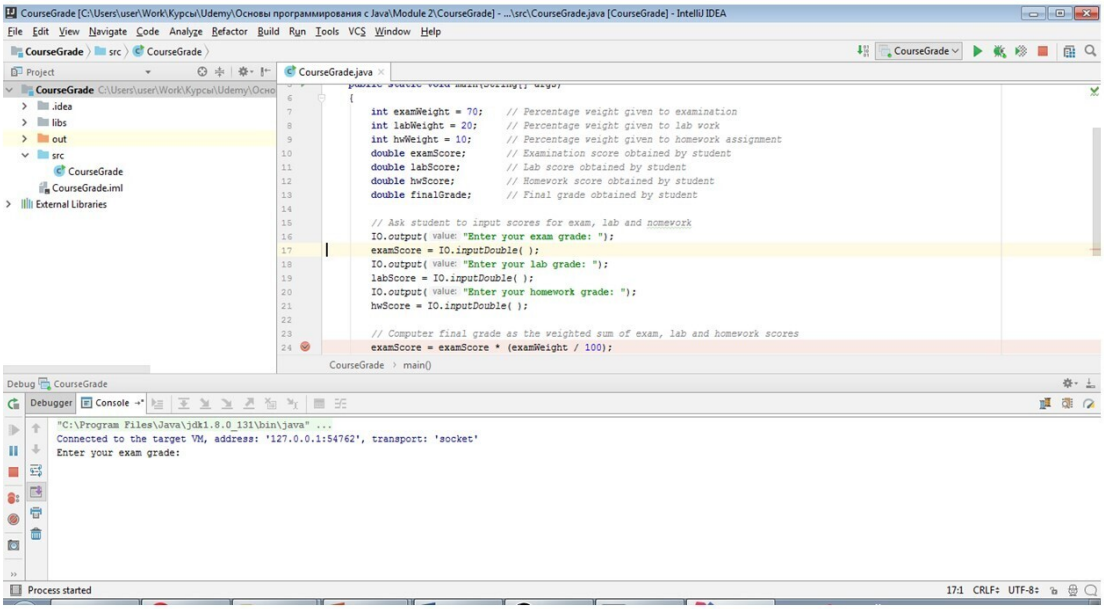
Вы можете нажимать в этих местах, чтобы установить или очистить точки останова.

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.



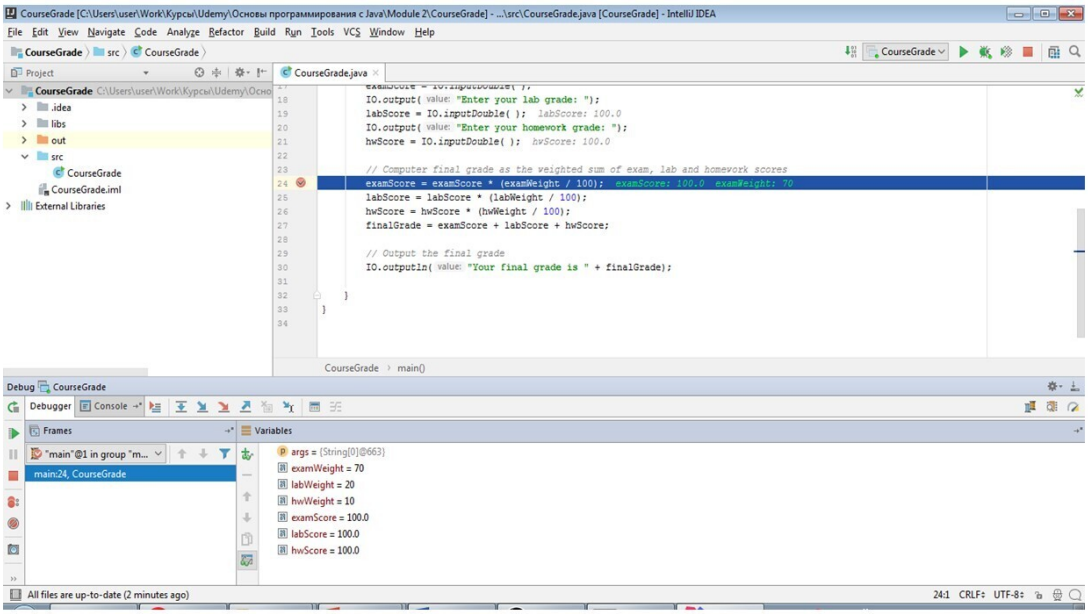
Давайте попробуем для домашних работ введем просто 100 без 0,0, чтобы сделать преобразование неявного типа, так как переменная hwScore имеет double тип и диапазон значений для int представляет собой подмножество для double.

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.

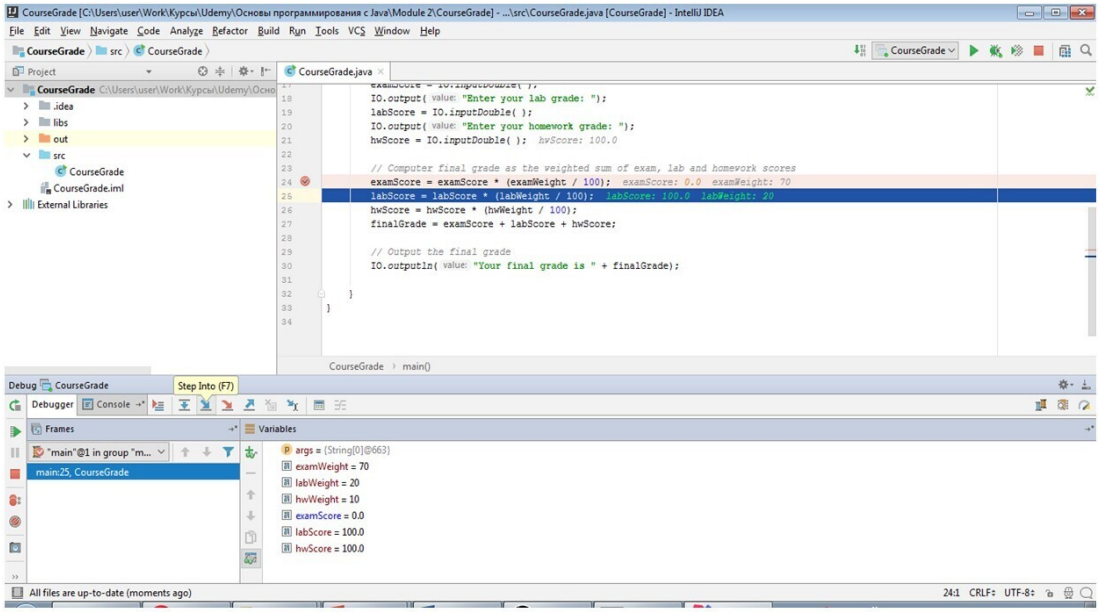
Вы можете продолжить выполнение программы, нажав на кнопку 'step', которая будет выполнять следующую инструкцию в программе.

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.

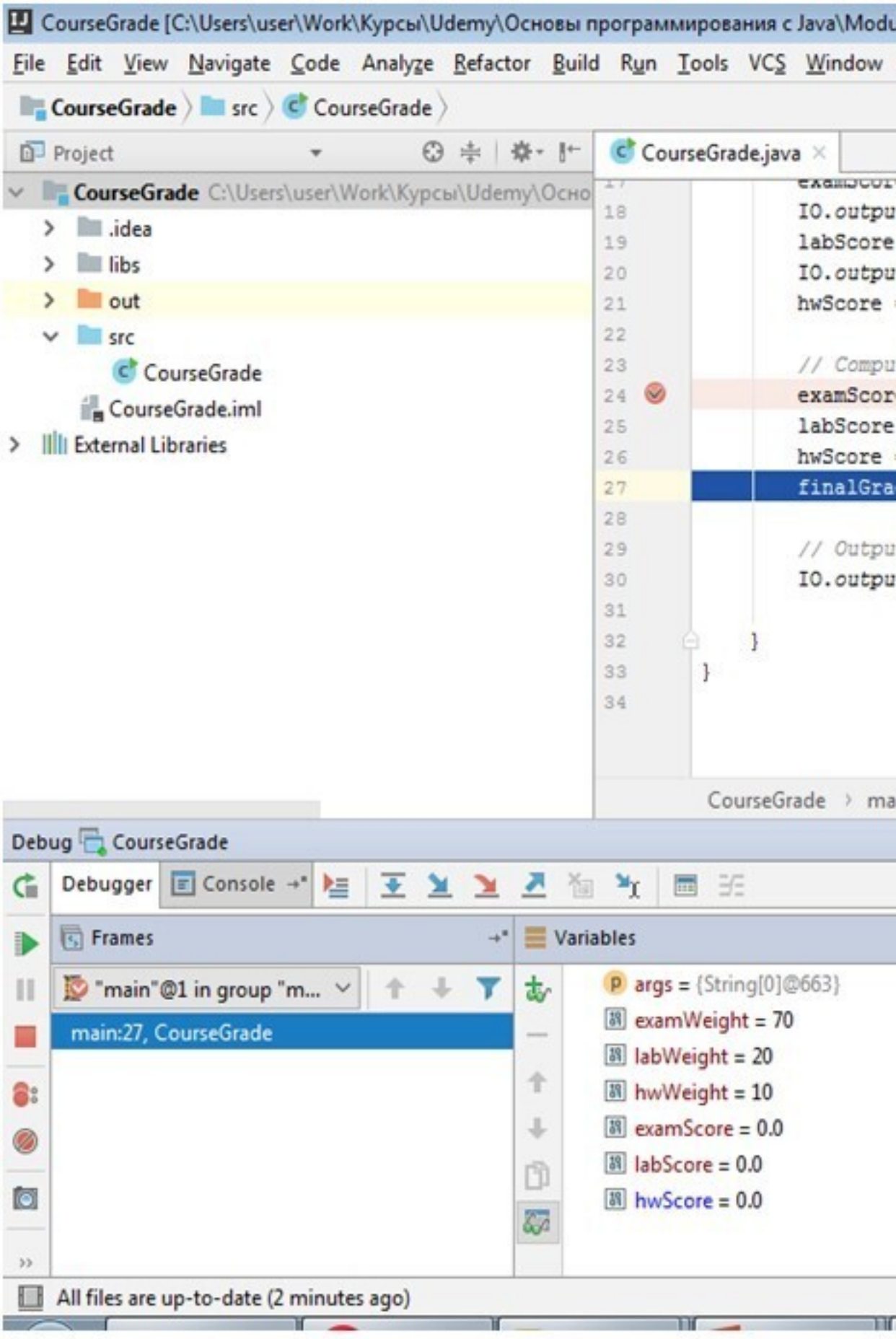
Это означает, что программа только что закончила выполнение инструкции для вычисления examScore.



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.

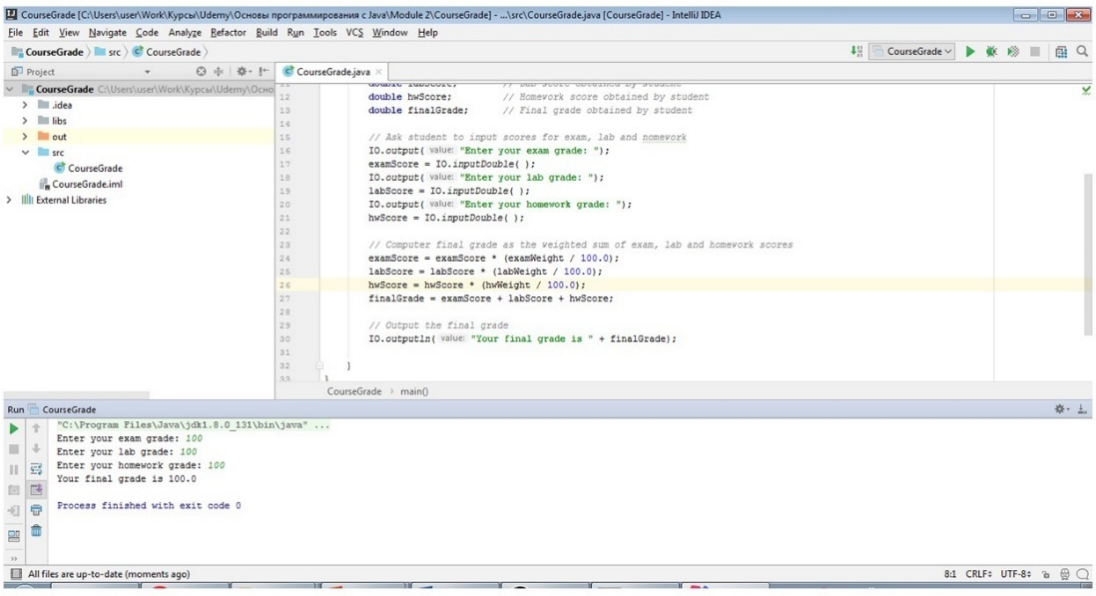
И эту проблему легко исправить. Для этого нужно заменить 100 на 100,0, и, если вы помните правило преобразования типов, которое мы только что обсудили, когда целое делится на число с плавающей точкой, результат будет преобразован в число с плавающей точкой.

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.

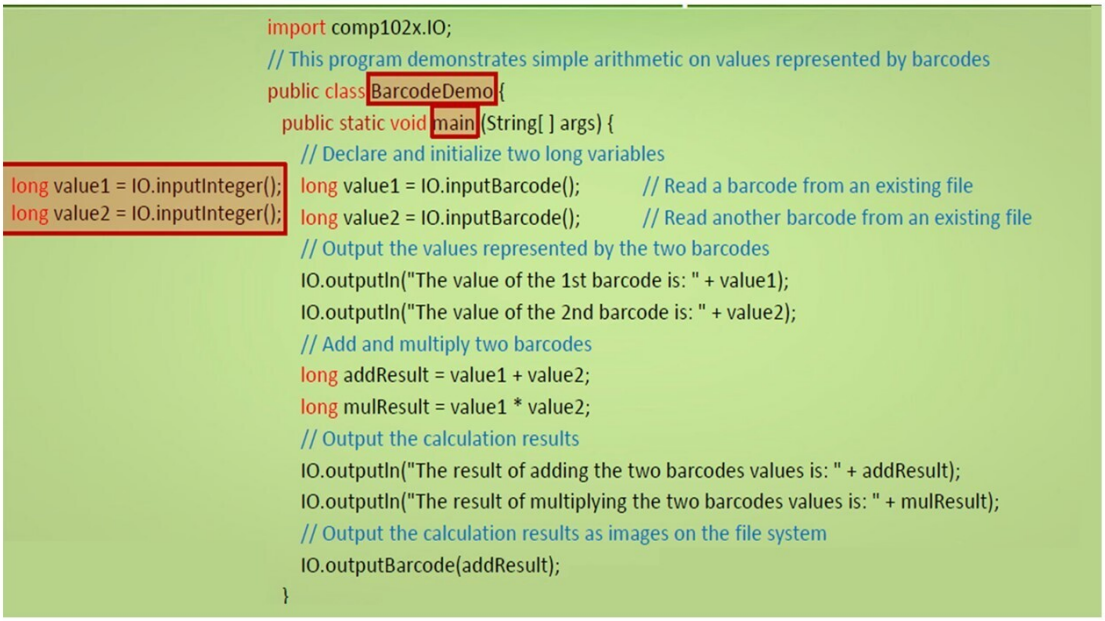
Я хочу отметить, что в следующем примере, мы используем абстракцию данных для работы со штрих-кодом.

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.

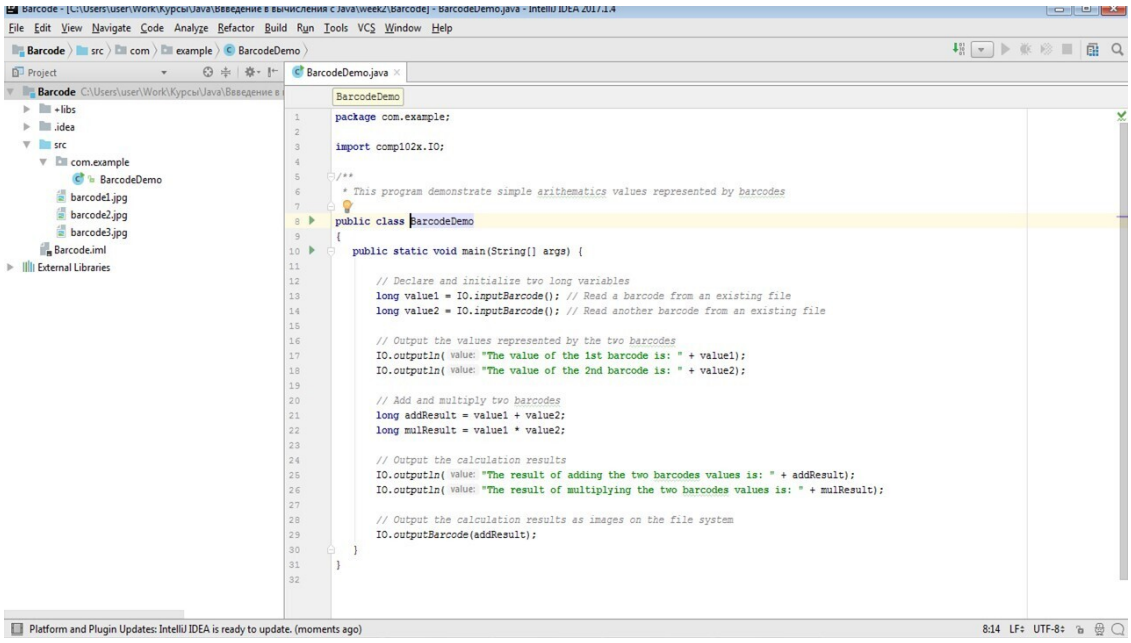
Второй сгенерирован компьютером, и мы не можем сказать, какие числа этот штрих-код представляет без помощи считывателя штрих-кодов.

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.

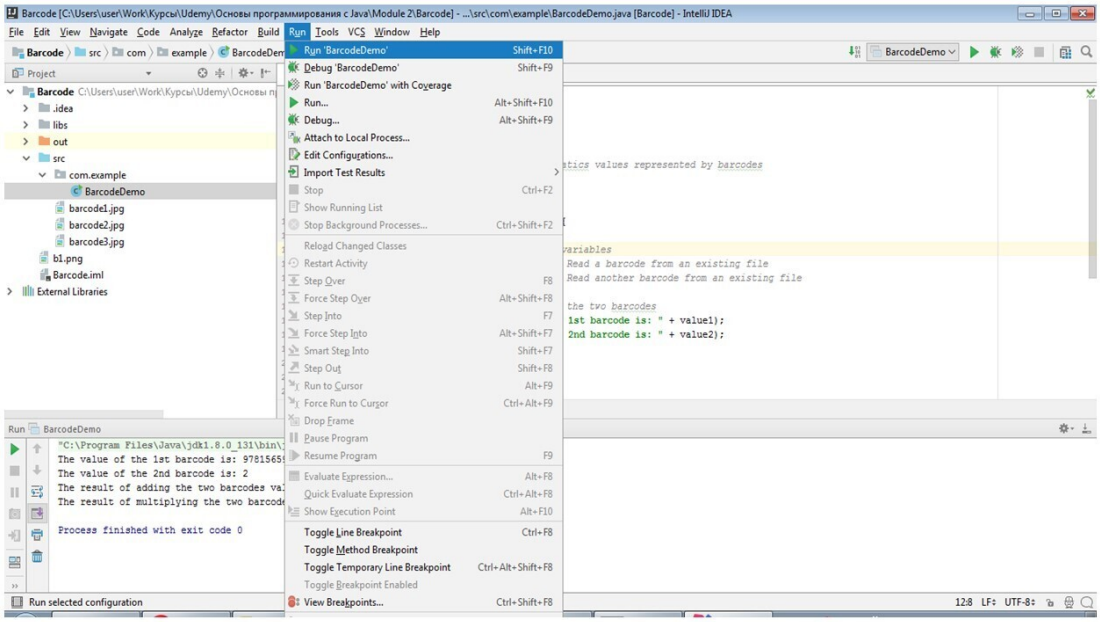
Теперь вы можете увидеть, что появилось окно диалога, которое запрашивает местоположение первого изображения штрих-кода.

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.

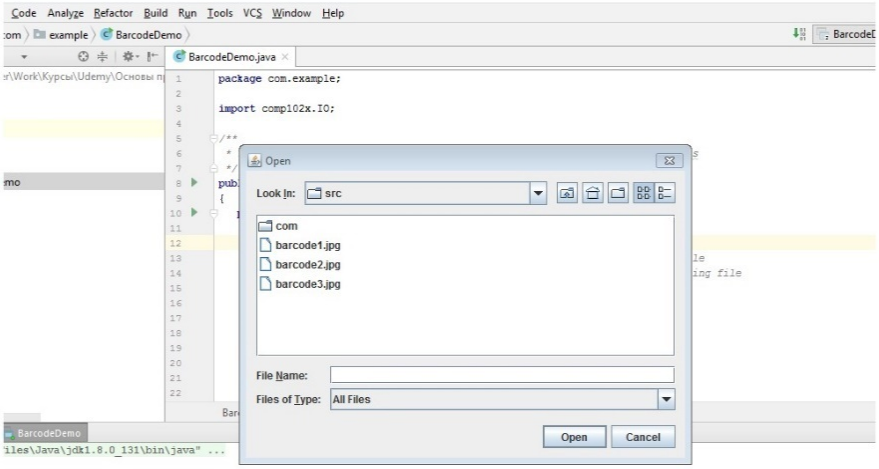
Значение первого штрих-кода огромное число из 12 цифр.

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.

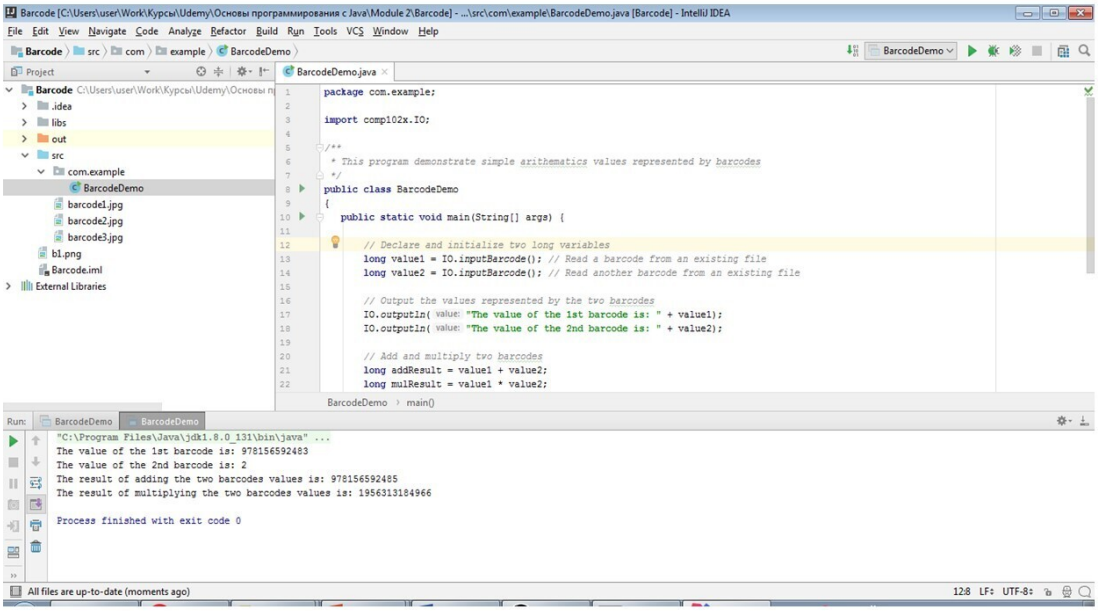
Обратите внимание, что мы еще не сделали – есть еще всплывающее окно, которое запрашивает имя файла.



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

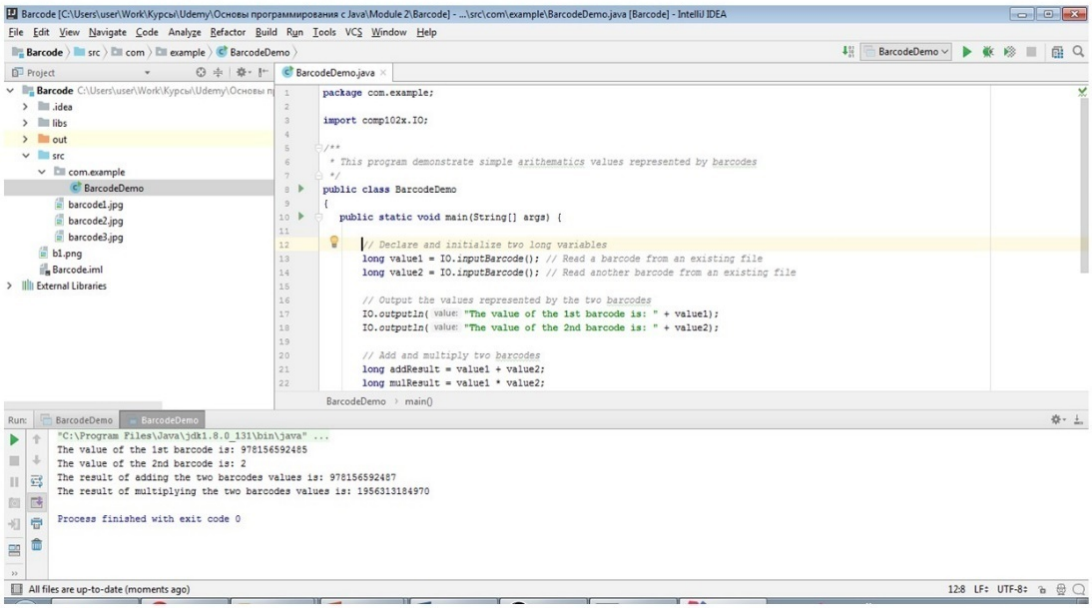
Давайте использовать название b1 в качестве выходного файла.

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.

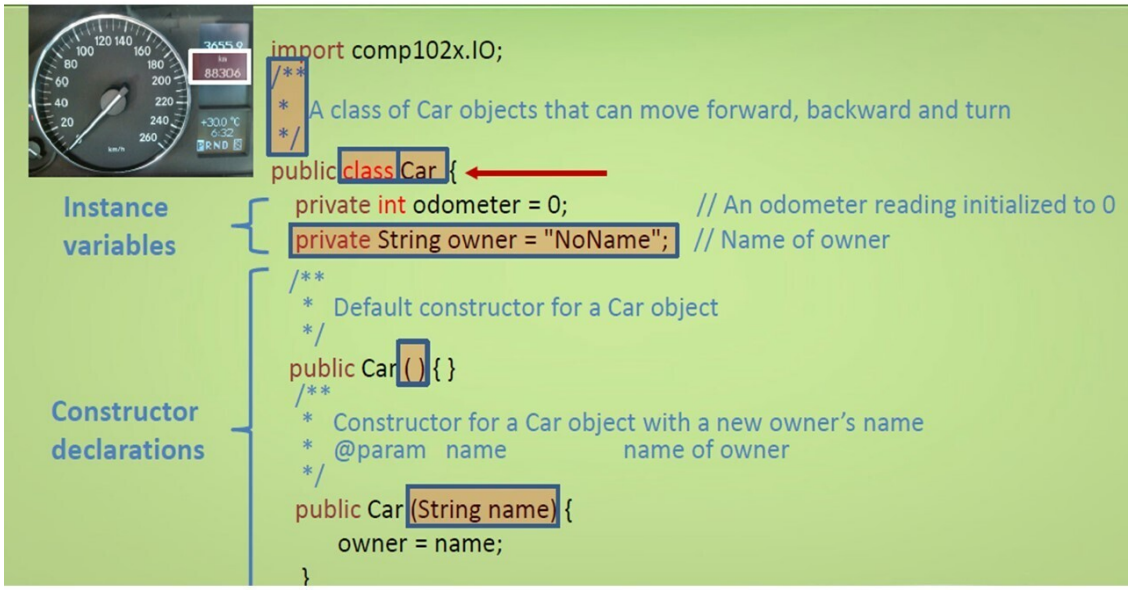
Например, чтобы изменить положение автомобиля нужно двигаться вперед или назад, или перевести смартфон в спящий режим, или увеличить громкость.

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.

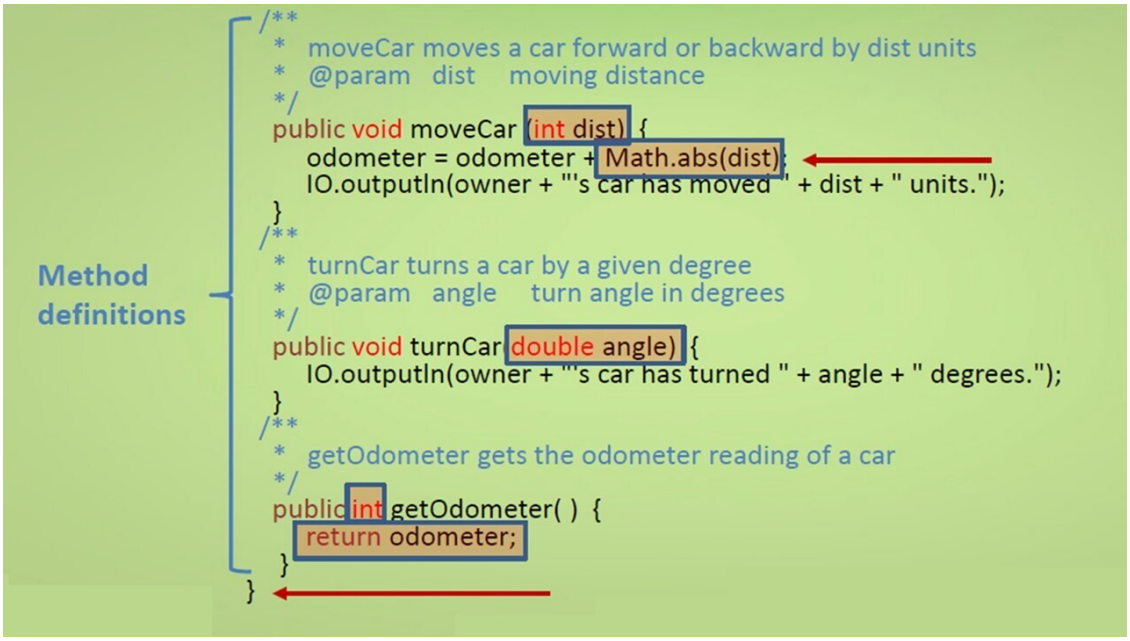
Здесь определены два конструктора.

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.

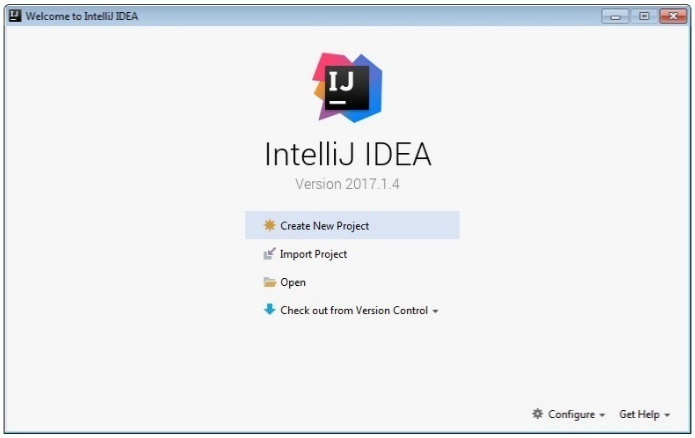
Я буду обсуждать метод, который возвращает значение, детально позже.

Программа заканчивается закрывающей фигурной скобкой.

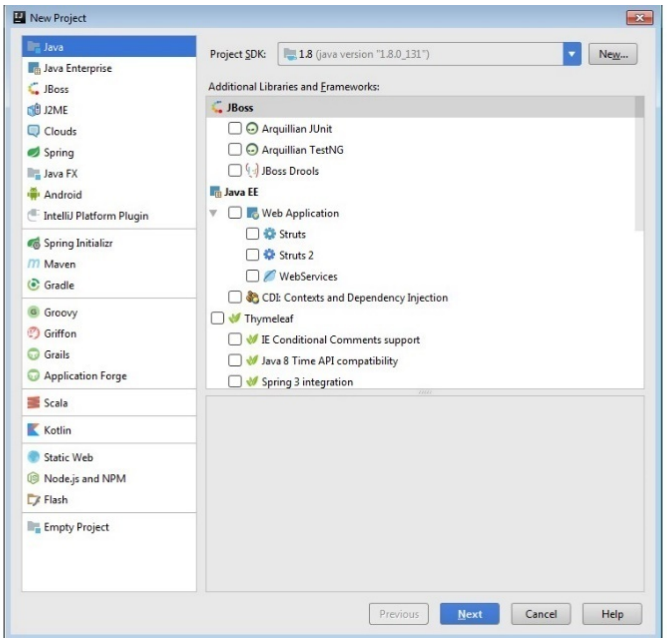
Теперь давайте вернемся к среде IntelliJ IDEA, чтобы увидеть, как этот класс автомобилей работает.

Демонстрация примера

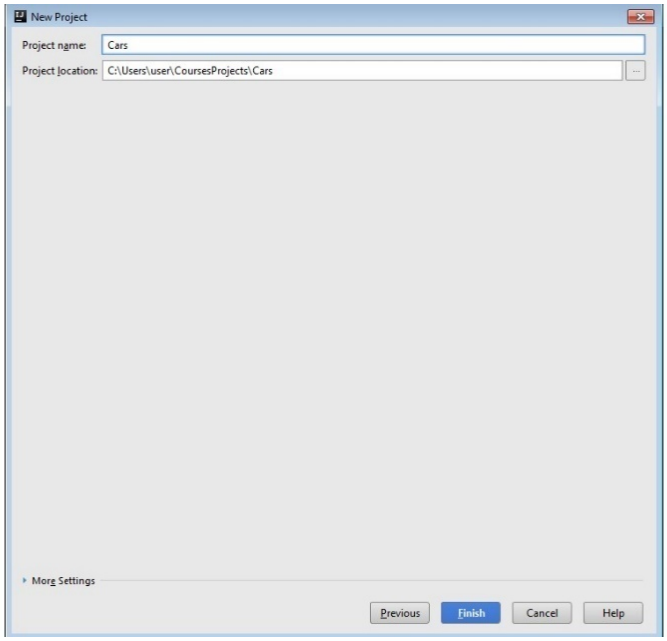
Давайте откроем IntelliJ IDEA.



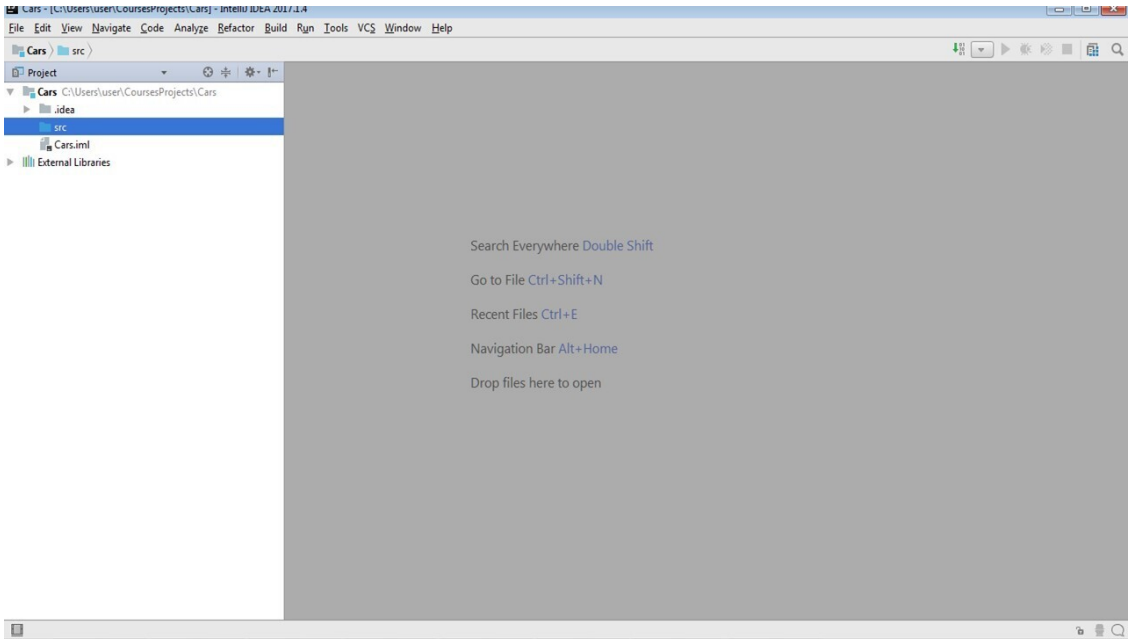
Вы можете создать новый проект, выбрав "Create New Project".



Выберем создание Java проекта.

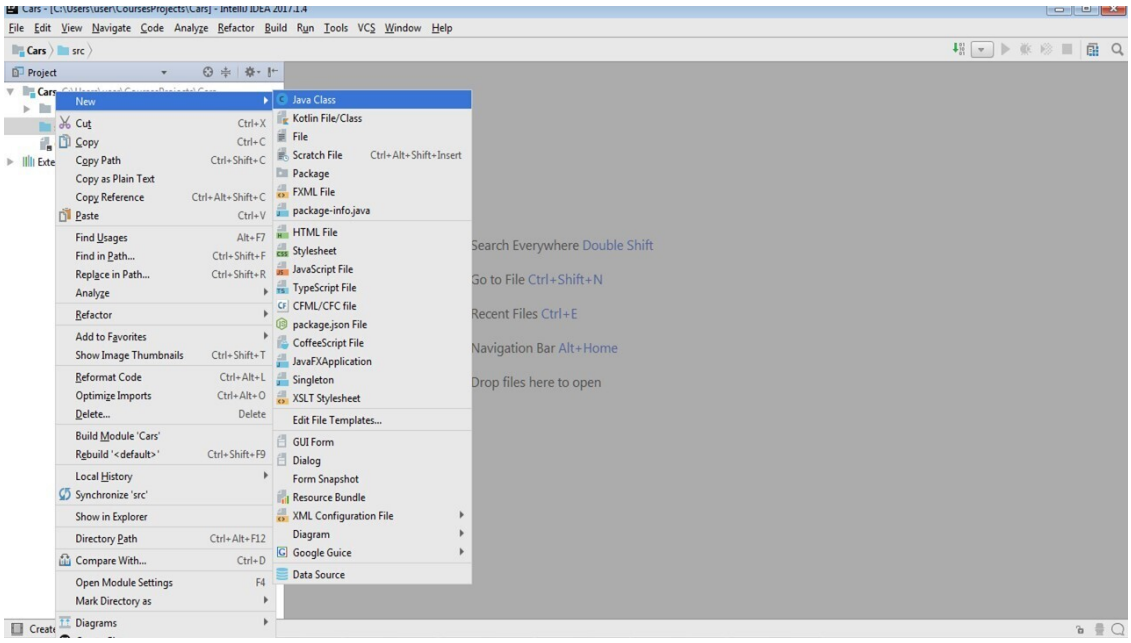


Вы можете выбрать место, где вы хотите сохранить ваш проект, а затем дать ему имя.
Давайте просто назовем проект «Cars».
И нажмем кнопку "Finish".

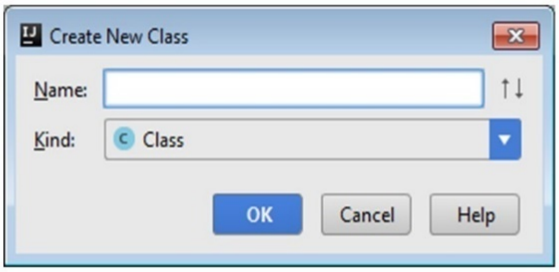


Появится окно нового проекта, который не содержит пока никакого кода.

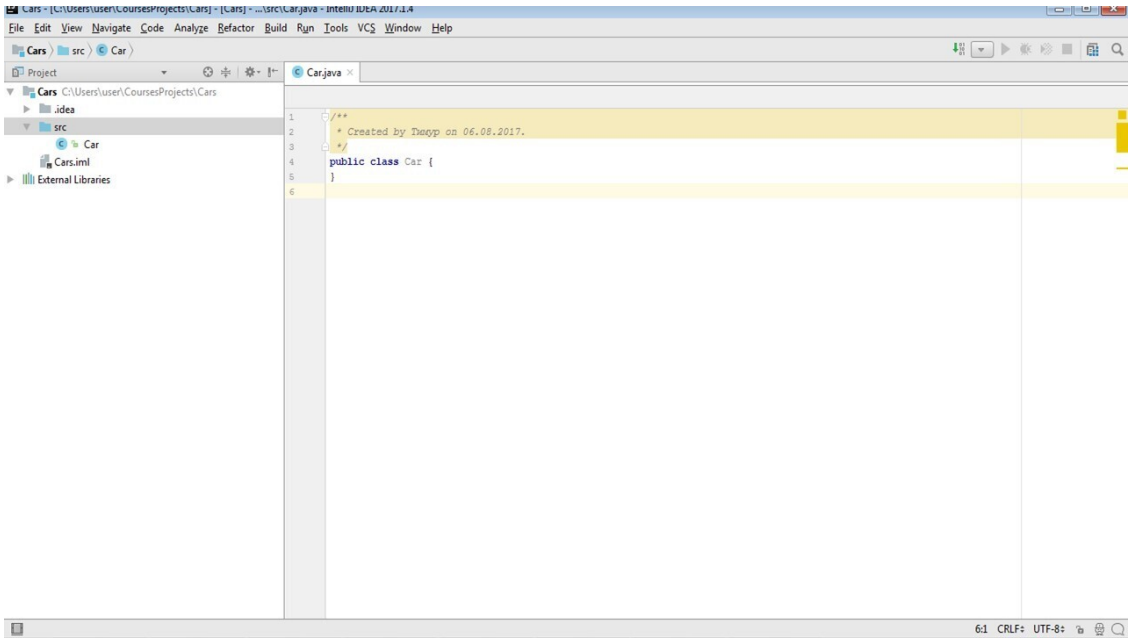
Мы можем начать писать нашу программу, создав новый класс, так как все программы Java это классы.



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.

Конец ознакомительного фрагмента.