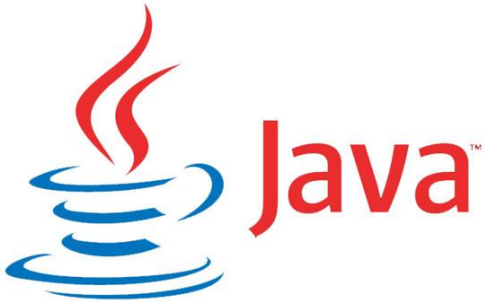
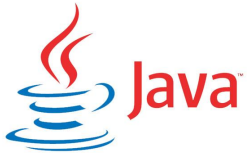
Тема 1.
Введение в язык программирования Java.

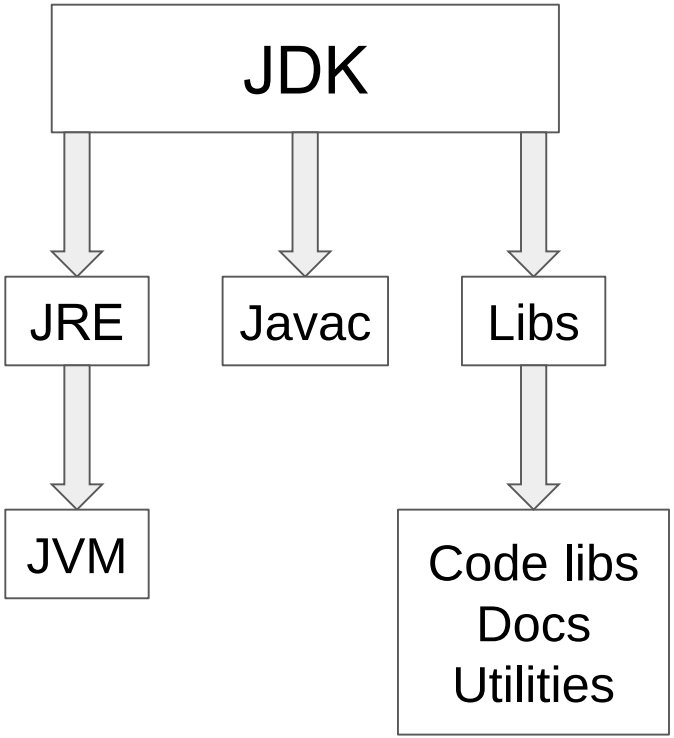


Основные качества:

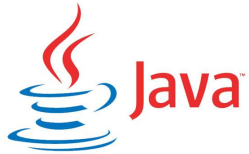


- Простота и мощь;
- Безопасность;
- Объектная ориентированность;
- Надежность;
- Интерактивность;
- Архитектурная независимость;
- Возможность интерпретации;
- Высокая производительность.

Компоненты Java



Компоненты Java

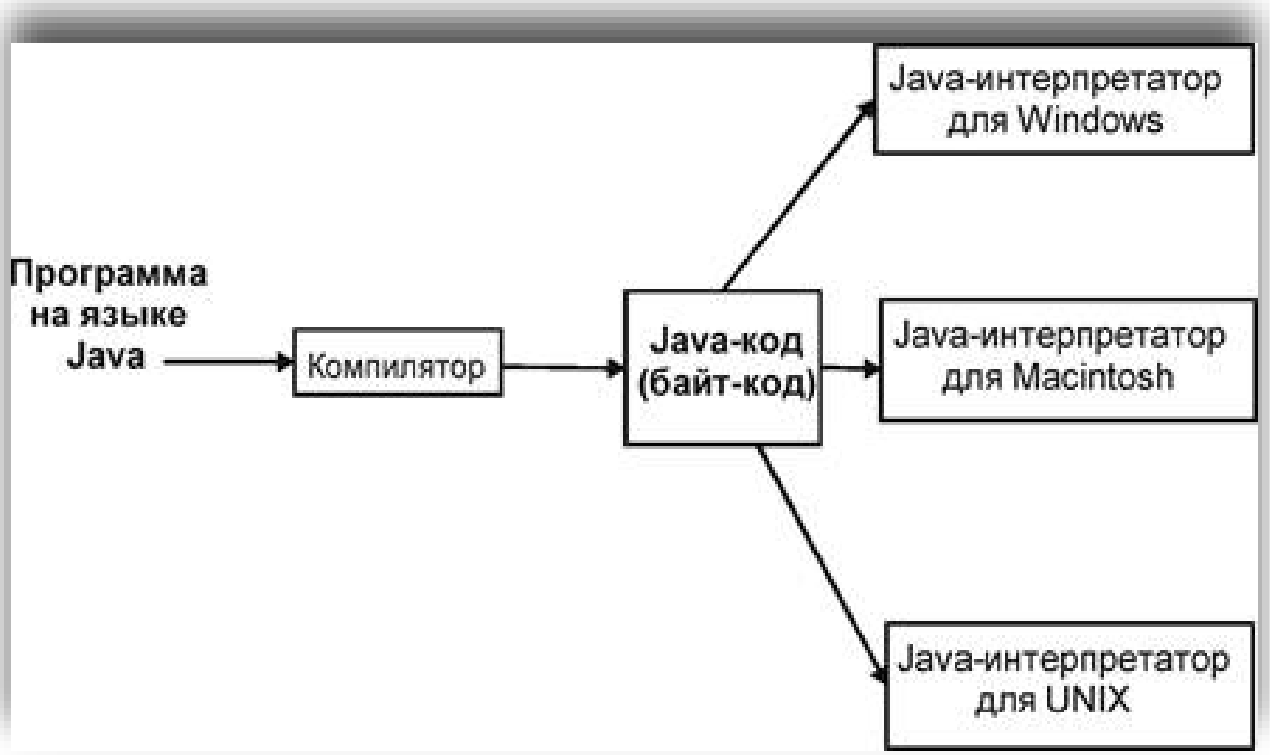


JDK (Java Development Kit) - комплект разработчика приложений на языке Java, включающий в себя **компилятор**, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и **исполнительную систему JRE.**

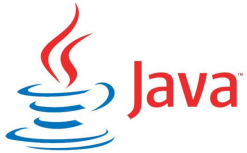
JRE (Java Runtime Environment) - минимальная реализация **виртуальной машины**, необходимая для исполнения Java -приложений, без компилятора и других средств разработки. Состоит из виртуальной машины и библиотек Java классов.

JVM (Java Virtual Machine) - виртуальная машина Java - основная часть исполняющей системы Java, так называемой Java Runtime Environment (JRE). Виртуальная машина Java исполняет байт-код Java, предварительно созданный из исходного текста Java-программы компилятором Java (javac). JVM обеспечивает платформо-независимый способ выполнения кода. Программисты могут писать код не задумываясь как и где он будет выполняться.

Java Virtual Machine



Java Virtual Machine



Виртуальная машина - это программное обеспечение, основанное на понятиях и идее воображаемого компьютера, который имеет логический набор команд, определяющих операции этого компьютера.

Это, можно сказать, небольшая операционная система.

Она формирует необходимый уровень абстракции, где достигается независимость от платформы и используемого оборудования.

Исполняет **байт-код Java**, предварительно созданный из исходного текста Java-программы компилятором Java (**javac**).

Java Virtual Machine

JVM является ключевым компонентом платформы Java.

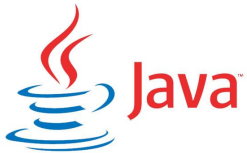
Так как виртуальные машины Java доступны для многих аппаратных и программных платформ, Java может рассматриваться и как связующее программное обеспечение, и как самостоятельная платформа. Использование одного байт-кода для многих платформ позволяет описать Java как **«скомпилировано однажды, запускается везде»** (compile once, run anywhere).

Виртуальные машины Java обычно содержат **Интерпретатор** байт-кода, однако, для повышения производительности во многих машинах также применяется **JIT (Just-in-time)** - компиляция часто исполняемых фрагментов байт-кода в машинный код.

Классификация платформ Java

- **Java SE** — Java Standard Edition, основное издание Java, содержит компиляторы, API, Java Runtime Environment; подходит для создания пользовательских приложений, в первую очередь — для настольных систем.
- **Java EE** — Java Enterprise Edition, представляет собой набор спецификаций для создания программного обеспечения уровня предприятия.
- **Java ME** — Java Micro Edition, создана для использования в устройствах, ограниченных по вычислительной мощности, например, в мобильных телефонах, КПК, встроенных системах;

Простота и мощь

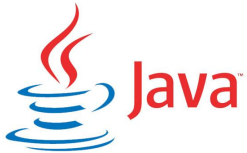


В языке Java для решения задачи имеется совсем немного вариантов.

Стремление к простоте зачастую приводило к созданию неэффективных и невыразительных языков типа командных интерпретаторов.

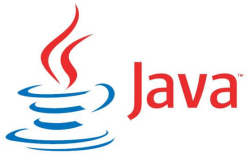
Java к числу таких языков не относится – для Вас вся мощность ООП и библиотек классов.

Надежность



- В Java накладывается ряд ограничений в нескольких наиболее важных областях, что вынуждает разработчиков выявлять ошибки на ранних этапах создания программы.
- Java избавляет от необходимости беспокоится по поводу наиболее часто встречающихся ошибок программирования
- Java строго типизированный язык, и проверка кода выполняется во время компиляции
- Проверка кода в Java выполняется и во время выполнения программы, в результате чего многие трудно обнаруживаемые программные ошибки, часто приводящие к с трудом воспроизводимым ситуациям, попросту невозможны
- Предсказуемость кода в разных ситуациях, одна из особенностей Java

Независимость от архитектуры



Основной задачей, которую ставили перед собой разработчики Java, было обеспечение долговечности и переносимости кода.

Одной из главных трудностей, стоявших перед разработчиками, когда они создавали Java, было отсутствие всяких гарантий что код, написанный сегодня, будет успешно выполняться завтра - даже на одном и том же компьютере.

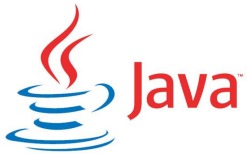
Независимость от архитектуры

Операционные системы и процессоры постоянно совершенствуются, и любые изменения в основных системных ресурсах могут стать причиной неработоспособности программ.

Пытаясь каким-то образом изменить это положение, разработчики приняли ряд жестких решений в самом языке и виртуальной машине Java. Они поставили перед собой следующую цель: **"написано однажды, выполняется везде, в любое время и всегда"**. И эта цель была в значительной степени достигнута.

Интерпретация

Интерпретация

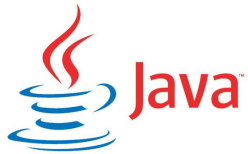


Необычайная способность Java исполнять свой код на любой из поддерживаемых платформ достигается тем, что ее программы транслируются в некое промежуточное представление, называемое **байт-кодом (bytecode)**.

Байт-код, в свою очередь, может интерпретироваться в любой системе, в которой есть среда времени выполнения Java.

Большинство ранних систем, в которых пытались обеспечить независимость от платформы, обладало огромным недостатком — потерей производительности (Basic, Perl). Несмотря на то, что в Java используется интерпретатор, байт-код легко переводится непосредственно в “родные” машинные коды (Just In Time compilers) “на лету”. При этом достигается весьма высокая производительность.

Объектная ориентированность

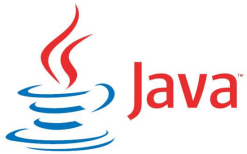


Java выделяется четким, практичным и прагматичным подходом к объектам.

Объектная модель Java проста и легко расширяема.

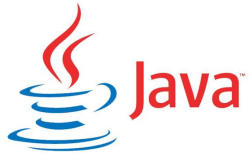
В то же время такие элементарные типы данных, как целочисленные, сохраняются в виде высокопроизводительных компонентов, не являющихся объектами.

Организация оперативной памяти в Java



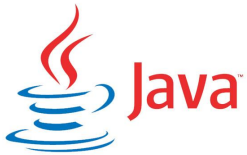
- В C++ есть прямой доступ к памяти, в Java – нет;
- Виртуальная машина Java самостоятельно управляет объектами в памяти;
- Сборщик мусора избавляет программиста Java от проблем управления памятью вручную. Он удаляет старые объекты, на которых уже нет ссылок в программе. Периодичность его работы динамична и заранее неизвестна;
- Запись объектов в память происходит при выполнении программы, а не при написании кода.

Безопасность



Программы на Java не могут вызывать глобальные функции и получать доступ к произвольным системным ресурсам, что обеспечивает в Java уровень безопасности, недоступный для других языков.

Основные библиотеки



java.lang - Классы ядра языка (типы, работа со строками, тригонометрические функции, обработка исключений)

java.io - Классы для различных типов ввода-вывода

java.system - Классы для работы с системой, консолью/командной строкой и т.д.

java.math - Классы для арифметических операций произвольной точности

java.net - Классы для работы в сети Интернет (сокеты, протоколы, URL)

java.util - Разноообразные полезные типы данных (стеки, словари, хэш-таблицы), даты, генератор случайных чисел

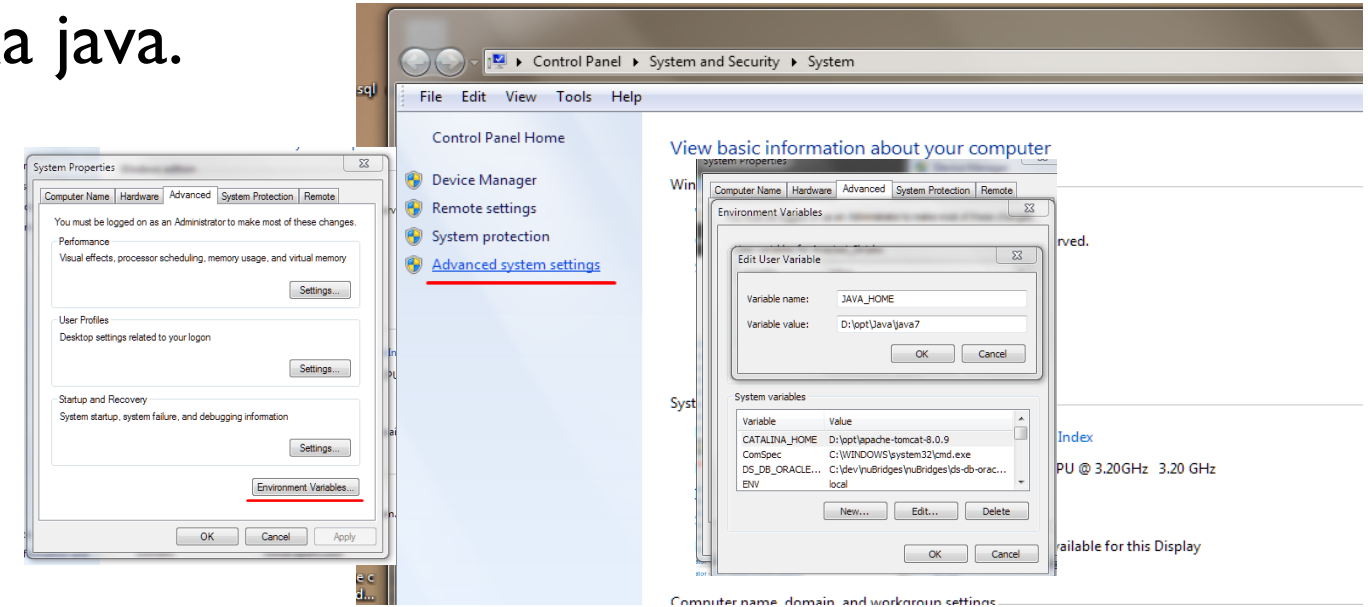
Установка Java

Скачать Java с официального сайта https://www.oracle.com/technetwork/java/javase/downloads

Windows x64 Installer	162.11 MB	jdk-14.0.2_windows-x64_bin.exe

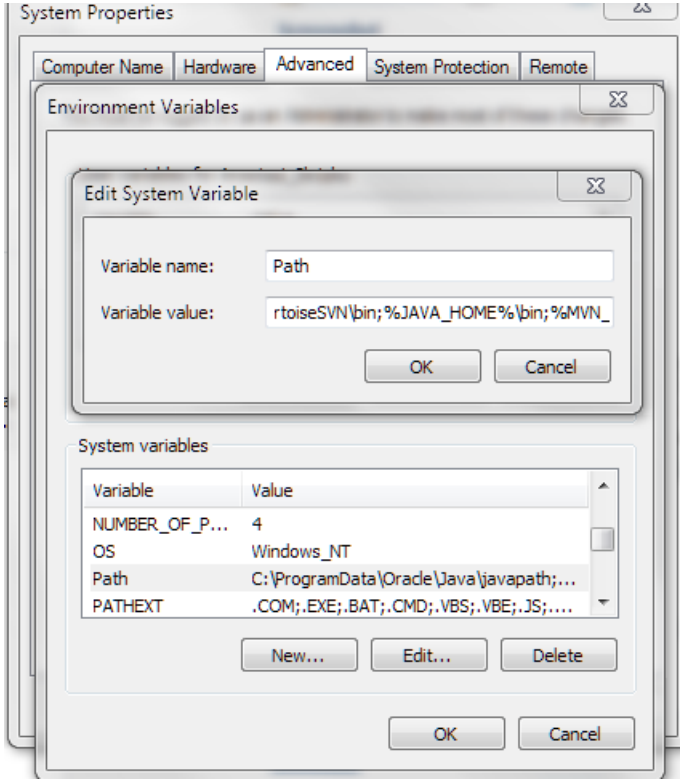
Установка Java

Создать новую переменную среду с именем JAVA_HOME, которая будет ссылаться на директорию, в которую была произведена установка java.



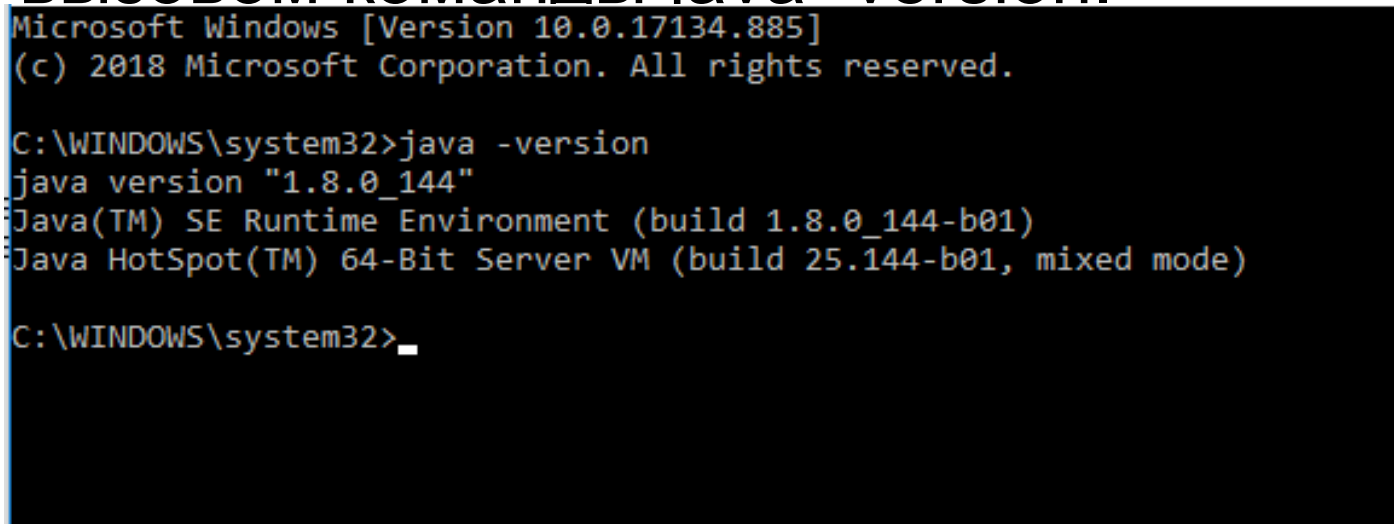
Установка Java

Добавить переменную JAVA_HOME в Path:



Установка Java

Проверить успешность установки вызовом команды java -version.



Среды разработки на Java

Интегри́рованная среда́ разрабо́тки (англ. Integrated development environment — IDE) — комплекс программных средств, используемый программистами для разработки программного обеспечения, включающий в себя:

- редактор кода,
- компилятор и/или интерпретатор,
- средства автоматизации сборки,
- отладчик.

Наиболее популярные среды разработки на Java

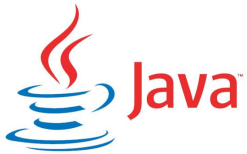
- **IntelliJ IDEA**

Одна из самых функциональных сред для java разработки, созданная компанией JetBrains.

Оснащена системой интеллектуальной помощи в написании кода, включает в себя огромное количество плагинов и надстроек под любую задачу, функцию автодополнения, имеет современный интерфейс.

Существуют две версии IntelliJ — **Community Edition**, которая является бесплатной, и **Ultimate Edition**, которая полностью признана и требует использования оплачиваемых лицензий.

Наиболее популярные среды разработки на Java



- **Eclipse**

Одна из самых популярных IDE, не только для Java, но и для C ++ с PHP, созданная компанией Eclipse Foundation.

Это инструмент с открытым исходным кодом, имеющий отличное сообщество разработчиков.

В нем имеется огромная библиотека плагинов, созданная самими пользователями, а также - множество версий, самая популярная из которых — Eclipse Oxygen.

Является бесплатной и не требует лицензии.

Наиболее популярные среды разработки на Java

Среди прочих можно выделить также:

- NetBeans
- JDeveloper
- Dr. Java
- BlueJ
- jCreator
- jGrasp
- Greenfoot
- Codenvy (облачная IDE)
- и др.

Что будем использовать?

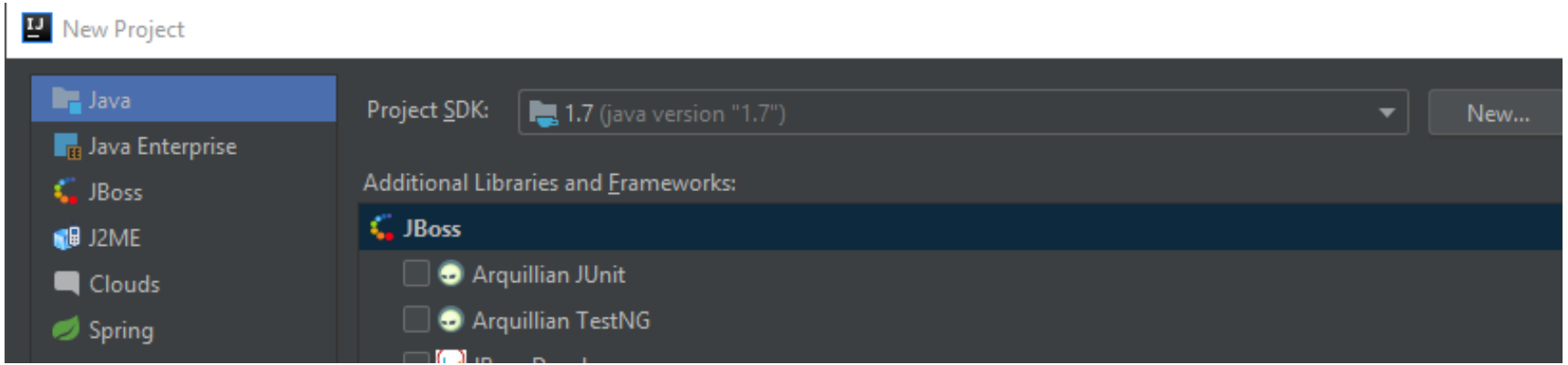
Для наших целей будем использовать IntelliJ IDEA Community версию.

Скачать IntelliJ IDEA Community можно с официального сайта пройдя по ссылке
https://www.jetbrains.com/idea/download/#section=windows

Далее необходимо запустить установочный файл и выполнить все шаги в открывшемся окне установки.

Создаем первый проект в IntelliJ IDEA Community

В верхнем левом углу жмем File / New / Project и в открывшемся окне New Project выбираем Java.



Создаем первый проект в IntelliJ IDEA Community

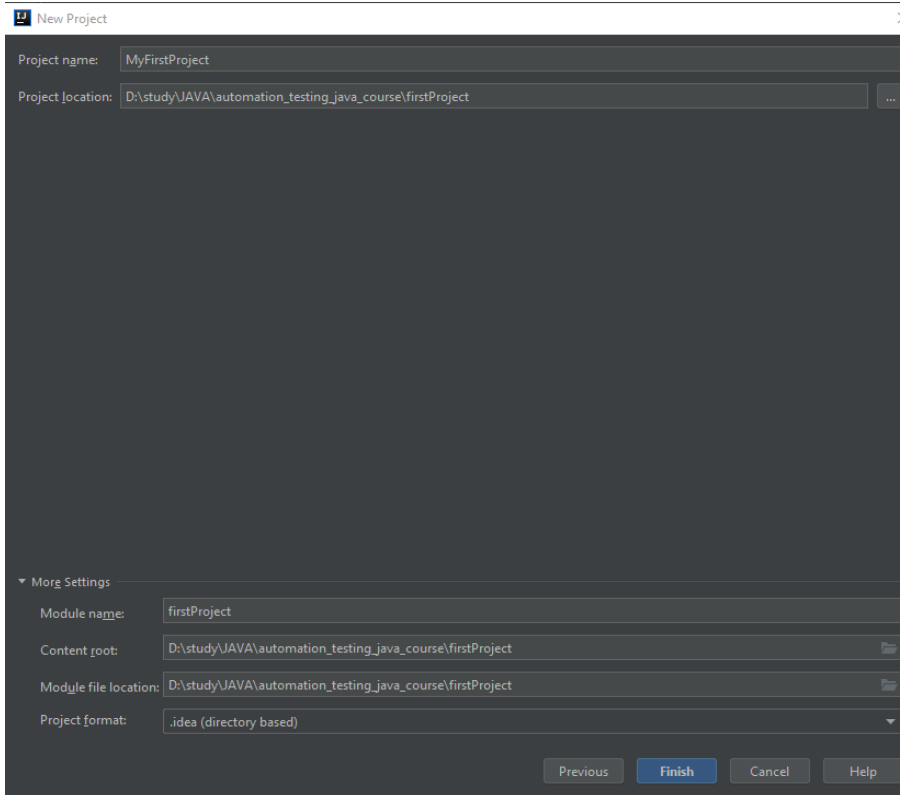
Жмем Next, оставляем все как есть в следующем окне и снова жмем Next.

На диске D: создаем папку и называем ее firstProject.

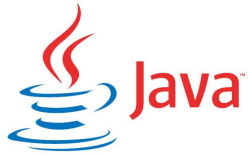
В открывшемся окне указываем имя проекта в поле Project Name - MyFirstProject

В поле Project Location указываем созданную Вами директорию, куда будем сохранять проект - firstProject.

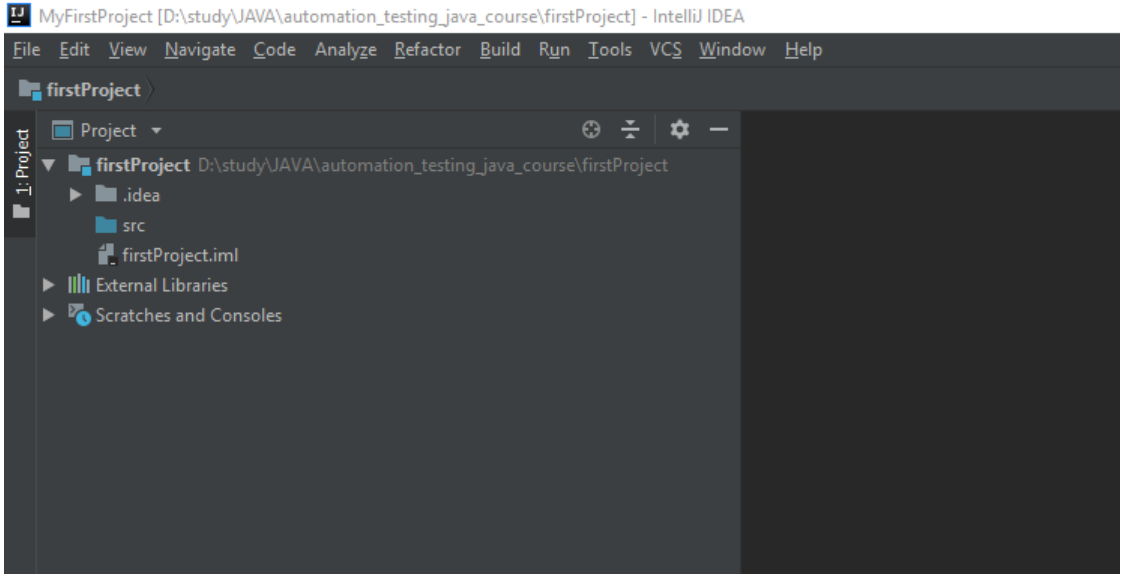
Создаем первый проект в IntelliJ IDEA Community



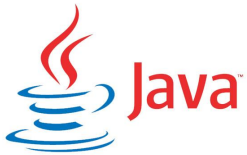
Создаем первый проект в IntelliJ IDEA Community



Нажимаем Finish, после чего созданный проект должен открыться в IDE.



Структура программы на Java

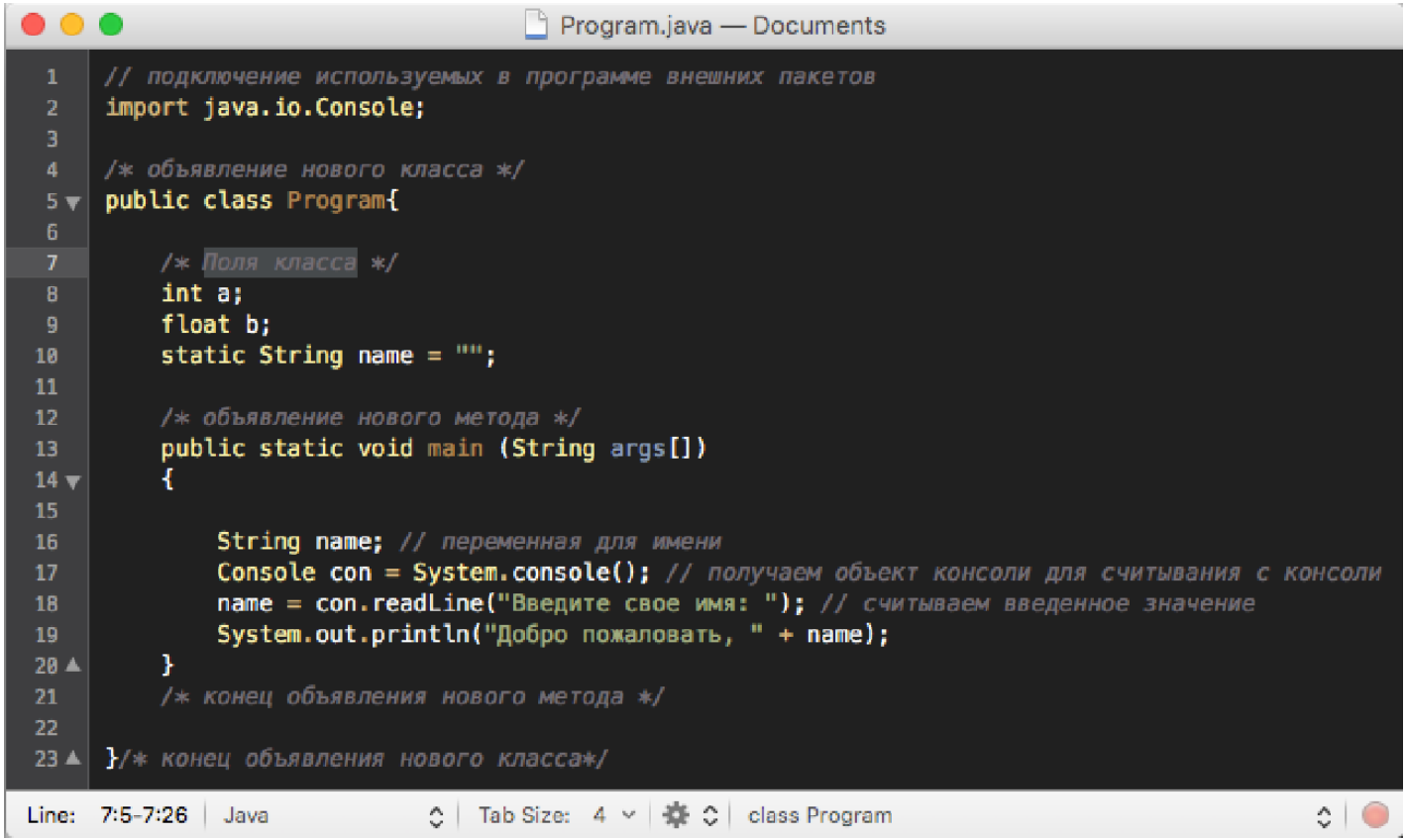


Любая программа на Java состоить из классов, которые, по мере надобности, а иногда и все сразу, загружаются в память JVM.

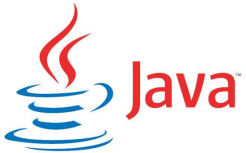
Классы описываются по определенному шаблону и имеют следующие основные блоки:

- Область подключения внешних пакетов
- Объявление класса
- Поля класса
- Описание конструктора класса
- Описание методов класса

Структура программы на Java

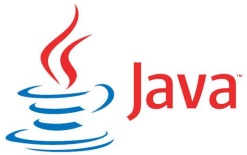


Структура программы на Java



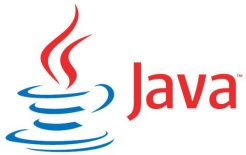
- В начале файла идет секция с подключенными внешними пакетами с помощью директивы import, после которой идут названия подключаемых **пакетов** и классов. **Пакеты** представляют собой организацию классов и интерфейсов в общие группы или блоки.
- Каждая строка завершается **точкой с запятой**, а каждый блок кода помещен в фигурные скобки.
- Далее идет определение класса программы. Классы объявляются следующим образом: модификатор доступа public, указывающий, что данный класс будет доступен из любого места программы и мы сможем запустить его из командной строки; затем идет ключевое слово class, а потом – название класса и тело самого класса в фигурных скобках.

Структура программы на Java



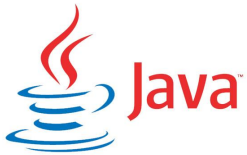
- Классы – это «кирпичики», из который строится программа на Java. К именам классов (идентификаторам) предъявляются определенные требования: именем класса может быть произвольная последовательность алфавитных и цифровых символов, а также символа подчеркивания, однако при этом названия не должны начинаться с цифры; идентификаторы не должны быть представлены зарезервированными ключевыми словами (такими, как class, int и т.д.)
- Класс может содержать различные переменные и методы.
- Выполнение любой программы начинается с метода main, который обязательно должен присутствовать в коде.

Структура программы на Java



- Метод main также имеет модификатор public.
- Слово static указывает, что метод main – статический, а слово void – что он не возвращает никакого значения.
- Все это станет более понятно чуть позже ☺
- Далее в скобках у нас идут параметры метода – String[] args – это массив args, который хранит значения типа String (строкового). В данном случае они нам пока не нужны, но в реальной программе – это те строковые параметры, которые передаются при запуске программы из командной строки.
- В фигурных скобках располагается тело метода, содержащее инструкции, которые будут выполняться при запуске программы.

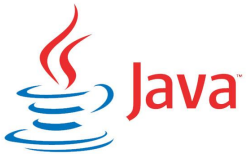
Создание первой программы



Чтобы создать простейшую программу на Java вы должны первым делом создать текстовый файл с именем **Hello.java**, и следующим содержанием:

```
1   import java.io.Console;
2
3   public class Program{
4       static String name = "";
5
6       public static void main(String[] args){
7           String name;
8           Console con = System.console();
9           name = con.readLine("Please, enter your name: ");
10          System.out.println("Welcome to Java, " + name);
11      }
12  }
```

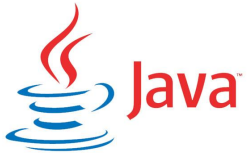
Структура программы на Java



- В начале тела класса объявляем переменную **name**, которая будет хранить строку (т.е. тип **String**). Java является регистрозависимым языком, поэтому следующие два объявления переменных будут не эквивалентны: **String name** и **String Name**. В данном случае будут объявлены 2 разные переменные.

- Далее идет создания переменной консоли, которая даст возможность взаимодействовать с консолью: **Console con = System.console();** . Так как класс **Console** находится в библиотеке классов в пакете **java.io**, то чтобы его использовать, необходимо этот пакет подключить, используя директиву import: **import java.io.Console**.

Структура программы на Java

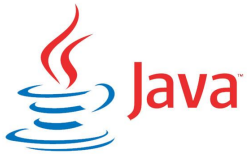


- Все это станет более понятно чуть позже ☺

- Далее в скобках у нас идут параметры метода – **String[] args** – это массив args, который хранит значения типа String (строкового). В данном случае они нам пока не нужны, но в реальной программе – это те строковые параметры, которые передаются при запуске программы из **командной строки**.

- В фигурных скобках располагается тело метода, содержащее инструкции, которые будут выполняться при запуске программы.

Создание первой программы



Далее необходимо в консоли набрать команду вида:

```
javac Hello.java
```

Для успешного выполнения этой команды вы должны находиться в том же каталоге, что и ваш файл.

Создание первой программы

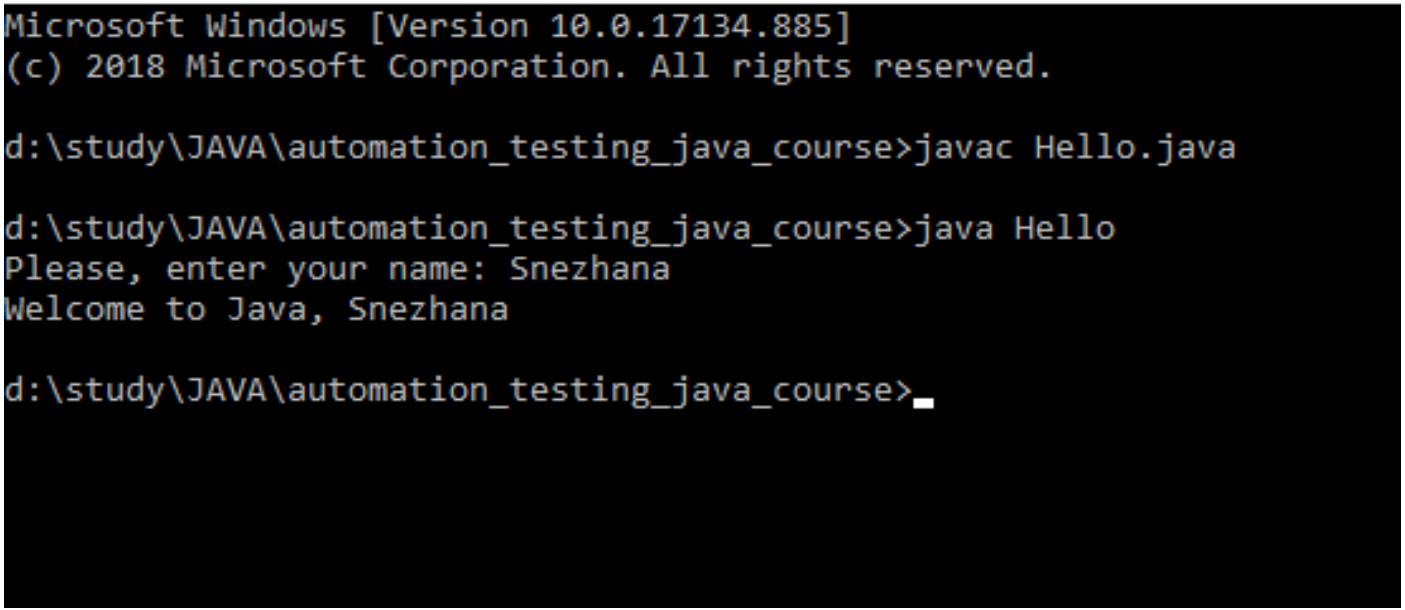
- В результате выполнения данной команды рядом с вашим файлом должен появиться другой файл с таким же именем и с расширением **.class**. Если он был успешно создан, программу можно запускать.

- Делается это командой:

- *java Hello*

Создание первой программы

- Если вы нигде не допустили ошибок, в консоли вы должны увидеть следующий текст:



Типы данных и переменные

Одной из основных особенностей Java является то, что данный язык является **строго типизированным**.

Это означает, что каждая переменная и константа представляет определенный тип и данный тип строго определен.

Тип данных определяет диапазон значений, которые может хранить переменная или константа.

Существует два вида типов данных в Java:

- базовые типы
- ссылочные типы

Базовые типы данных

Базовые (примитивные), в данном случае, означает, что эти типы доступны в рамках самого языка и для их использования не нужно создавать свои классы или обращаться к классам из библиотеки.

Переменные базовых типов не являются объектами.

В Java существует 8 базовых (примитивных) типов данных:

Базовые типы данных

Название	Область принимаемых значений	Занимаемая память
boolean	true, false	1 бит ???
byte	-128 до 127	1 байт
short	-32768 до 32767	2 байта
int	-2147483648 до 2147483647	4 байта
long	–9223372036854775808 до 9223372036854775807	8 байт
float	$-3.4*10^{38}$ до $3.4*10^{38}$	4 байта
double	$\pm4.9*10^{324}$ до $\pm1.8*10^{308}$	8 байт
char	0 до 65536	2 байта

Ссылочные типы данных

Ссылочный тип хранит в себе ссылку на объект какого-либо класса.

Создавая объект класса, его необходимо связывать с подобной ссылочной переменной.

Примером ссылочного типа является тип String, предназначенный для хранения строк текста или массив (например, int[] array – массив целых чисел).

Переменные

Переменная – это именованная область памяти, куда может быть записано или перезаписано и откуда может быть прочитано значение определенного типа.

Тип переменной и ее имя задаются при создании переменной, они не могут быть изменены далее по ходу программы.

Имя или идентификатор переменной – это последовательность из строчных и заглавных латинских букв, цифр, а также символов «$» и «_».

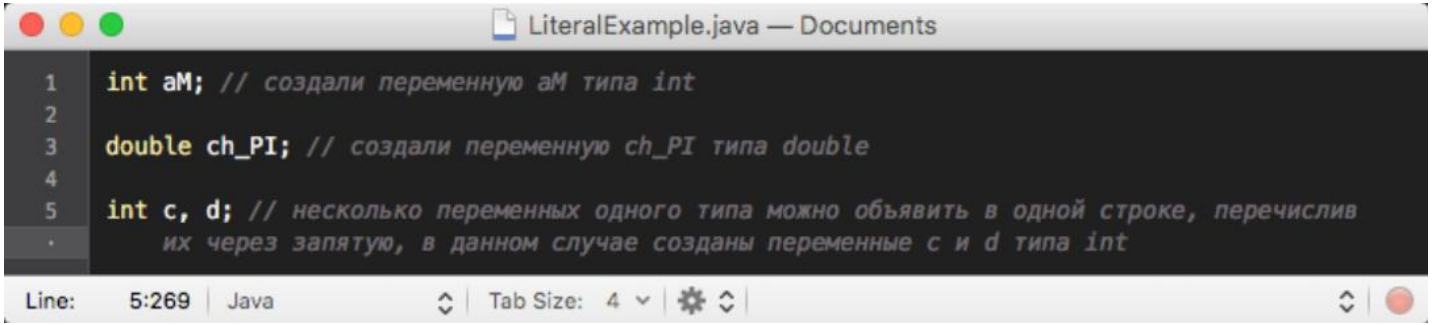
Имя переменной может начинаться с любого из перечисленных символов, кроме цифр.

Переменные

При создании или, как еще говорят, объявлении переменных нужно указывать имя и тип каждой из них.

Язык Java чувствителен к регистру символов в идентификаторах, т.е. идентификаторы Chis, CHIS и cHIS в Java различны.

Внутри одного блока не может существовать несколько элементов с одинаковыми идентификаторами.



Переменные

Использование суффиксов.

При присваивании переменной типа float значения с плавающей точкой Java автоматически рассматривает это значение как объект типа double. И, чтобы указать, что данное значение должно рассматриваться как float, нам необходимо использовать суффикс f.

```
1  float fl = 30.6f;
2  double db = 30.6;
```

И хотя в данном случае обе переменные имеют практически одинаковые значения, они будут рассматриваться по-разному и занимать разное место в памяти.

Переменные

Символы и строки.

В качестве значения, переменная символьного типа получает одиночный символ, заключенный в одинарные кавычки:

char ch = ‘e’;

Кроме того, переменной символьного типа также можно присвоить целочисленное значение **от 0 до 65536**.

В этом случае, переменная опять же будет хранить символ, а целочисленное значение будет указывать на номер символа в **таблице символов Unicode**.

```
1  char ch=102; // символ 'f'
2  System.out.println(ch);
```

Переменные

Символы и строки.

Еще одной формой задания символьных переменных является шестнадцатиричная форма: переменная получает значение в шестнадцатиричной форме, которая следует после символа “\u”.

Например: **char ch = ‘\uoo66’;** опять же будет хранить символ ‘f’.

Символьные переменные не стоит путать со **строками**:

‘a’ не идентично “a”.

Строковые переменные представляют объект String, который, в отличие от char или int не является примитивным типом в Java:

```
1  String hello = "Привет...";
2  System.out.println(hello);
```

Переменные

Константы.

Переменные можно объявить один раз в программе и затем несколько раз присваивать им различные значения, но в Java также имеются константы.

В отличии от переменных, константам можно присвоить значение только один раз.

Константа объявляется также, как и переменная, только вначале идет ключевое слово final.

```
1  int num=5;
2  num=57;
3  num=89;
```

Здесь можно менять значение переменной.

```
1  final int num=5;
2  num=57; // так мы уже не можем написать, так как num – константа
```

А вот здесь – нет.

Преобразование базовых типов данных

При рассмотрении типов данных указывалось, какие значения может иметь тот или иной тип и сколько байт в памяти он может занимать.

Мы можем написать, например, так:

```
byte x = 5;
byte y = x;
```

Но важно понимать, что эта запись не эквивалентна следующей (хотя результат будет одинаковым):

```
byte x = 5;
int y = x;
```

Преобразование базовых типов данных

В обоих случаях создается переменная типа byte, которая затем приравнивается другой переменной.

Однако, если в первом случае это простое приравнивание, а переменная y просто получает значение переменной x, то во втором примере происходит **преобразование типов**: *данные типа byte преобразуются к типу int*.

Данный тип преобразований называется **расширяющим**, так как значение типа byte расширяет свой размер до размера типа int.

Расширяющие преобразования проходят автоматически и, обычно, с этим никаких проблем не возникает.

Преобразование базовых типов данных

Подобным образом происходит преобразование от типа float к типу double или от типа int к типу long.

Кроме расширяющих преобразований есть еще и сужающие.

Сужающие преобразования позволяют привести данные к типу с меньшей разрядностью, например, от типа int, который занимает 4 байта в памяти, к типу byte, который занимает только 1 байт:

```
int a = 5;

byte b = a;
```

Несмотря на то, что значение переменной a – число 4 – укладывается в диапазон типа byte, мы все равно получим ошибку.

Преобразование базовых типов данных

И чтобы безошибочно провести преобразование от одного типа к другому, нам нужно применить операцию приведения типов.

Суть этой операции состоит в том, что в скобках указывается тип, к которому нужно привести данное значение:

```
int a = 4;
byte b = (byte) a;
```

Преобразование базовых типов данных

Потеря данных при преобразовании.

В предыдущей ситуации число 4 вполне укладывалось в диапазон значений типа byte.

Но что произойдет в следующем случае?

int a = 200;

byte b = (byte) a;

Результатом будет число **-56**. В данном случае число 200 вне диапазона для типа byte (от -128 до 127), поэтому произойдет усечение значения.

Преобразование базовых типов данных

Усечение рациональных чисел до целых.

При преобразовании значений с плавающей точкой к целочисленным значениям, происходит усечение дробной части:

```
double a = 56.9898;
int b = (int) a;
```

Здесь значение числа b будет равно 56, несмотря на то, что число 57 было бы ближе к 56.9898.

Чтобы избежать подобных казусов, надо применять функцию округления, которая есть в математической библиотеке Java:

```
double a = 56.9898;
int b = (int) Math.round(a);
```

Преобразование базовых типов данных

Преобразования при операциях.

Нередки ситуации, когда приходится применять различные операции, например, сложение и произведение, над значениями разных типов.

И здесь действуют некоторые правила:

- *Если один из операндов операции относится к типу double, то и второй операнд преобразуется к типу double*
- *Если предыдущее условие не соблюдено, а один из операндов операции относится к типу float, то и второй операнд преобразуется к типу float*
- *Если предыдущие условия не соблюдены, а один из операндов операции относится к типу long, то и второй операнд преобразуется к типу long*
- *Иначе все операнды операции преобразуются к типу int*

Преобразование базовых типов данных

Преобразования при операциях.

```
int a = 3;
double b = 4.6;
double c = a + b;
```

Так как в операции участвует значение типа double, то и другое значение приводится к типу double и сумма значений a + b будет тоже представлять тип double.

Преобразование базовых типов данных

Преобразования при операциях.

```
byte a = 3;
short b = 4;
byte c = (byte) a + b;
```

Две переменные типа byte и short (не double, float или long), поэтому при сложении они преобразуются к типу int, и их сумма a+b представляет значение типа int.

Поэтому если необходимо присвоить результат типу byte, то необходимо выполнить преобразование результата к byte.

Операции языка Java

Операция — это некое элементарное действие (например, сложение чисел), обозначаемое в языке Java заранее предопределенной последовательностью символов, которое может выполняться над одной или несколькими переменными и литералами.

Операнд – переменная или значение (например, число), участвующее в операции.

Выражение - фрагмент кода, где в одну конструкцию объединяется несколько операций.

Операции языка Java

Операции бывают:

- Унарные – выполняются над одним операндом
- Бинарные – над двумя операндами
- Тернарные – выполняются над тремя операндами

Операции языка Java

Арифметические

Операция	Значение
+	операция сложения двух чисел, например: z=x+y
-	операция вычитания двух чисел: z=x-y
*	операция умножения двух чисел: z=x*y
/	операция деления двух чисел: z=x/y
%	получение остатка от деления двух чисел: z=x%y
++(префиксный инкремент)	Предполагает увеличение переменной на единицу, например, z=++y (вначале значение переменной y увеличивается на 1, а затем ее значение присваивается переменной z)
++(постфиксный инкремент)	также, увеличение переменной на единицу, например, z=y++ (вначале значение переменной y присваивается переменной z, а потом значение переменной y увеличивается на 1)
--(префиксный декремент)	уменьшение переменной на единицу, например, z=--y (вначале значение переменной y уменьшается на 1, а потом ее значение присваивается переменной z)
--(постфиксный декремент)	z=y-- (сначала значение переменной y присваивается переменной z, а затем значение переменной y уменьшается на 1)

Операции языка Java

Арифметические

Примеры:

```
1   int a1 = 3 + 4; // результат равен 7
2   int a2 = 10 - 7; //результат равен 3
3   int a3 = 10 * 5; //результат равен 50
4   double a4 = 14.0 / 5.0; //результат равен 2.8
5   double a5 = 14.0 % 5.0; //результат равен 4
6   int b1 = 5;
7   int c1 = ++b1; // c1=6; b1=6
8   int b2 = 5;
9   int c2 = b2++; // c2=5; b2=6
10  int b3 = 5;
11  int c3 = --b3; // c3=4; b3=4
12  int b4 = 5;
13  int c4 = b4--; // c4=5; b4=4
14
15  //Операцию сложения также можно применять к строкам, в этом случае происходит
    конкатенация строк:
16
17  String hello = "hell to " + "world"; //результат равен "hell to world"
```

Операции языка Java

Операции сравнения

В операциях сравнения сравниваются два операнда, и возвращается значение типа **boolean** - *true*, если выражение верно, и *false*, если выражение неверно.

Операция	Значение
==	данная операция сравнивает два операнда на равенство: c=a==b; c равно true, если a равно b, иначе c будет равно false
!=	c=a!=b; (c равно true, если a **не** равно b, иначе c будет равно false)
<	c=a<b; (c равно true, если a меньше b, иначе c будет равно false)
>	c=a>b; (c равно true, если a больше b, иначе c будет равно false)
<=	c=a<=b; (c равно true, если a меньше или равно b, иначе c будет равно false)
>=	c=a>=b; (c равно true, если a больше или равно b, иначе c будет равно false)

Операции языка Java

Операции сравнения

Кроме, собственно, операций сравнения в Java также определены логические операторы, которые возвращают значение типа Boolean.

Выше мы рассматривали поразрядные операции над числами.

Теперь же рассмотрим эти же операторы при операциях над булевыми значениями:

Операции языка Java

Операции сравнения

Операция	Значение
|	c=a|b; (c равно true, если либо a, либо b (либо и a, и b) равны true, иначе c будет равно false)
&	c=a&b; (c равно true, если и a, и b равны true, иначе c будет равно false)
!	c!=b; (c равно true, если b равно false, иначе c будет равно false)
^	c=a^b; (c равно true, если либо a, либо b (но не одновременно) равны true, иначе c будет равно false)
||	c=a||b; (c равно true, если либо a, либо b (либо и a, и b) равны true, иначе c будет равно false)
&&	c=a&&b; (c равно true, если и a, и b равны true, иначе c будет равно false)

Операции языка Java

Операции сравнения

Здесь у нас две пары операций | и || (а также & и &&) выполняют похожие действия, однако же они не равнозначны.

Выражение c=a|b; будет вычислять сначала оба значения - a и b и на их основе выводить результат.

В выражении же c=a||b; вначале будет вычисляться значение a, и если оно равно true, то вычисление значения b уже смысла не имеет, так как у нас в любом случае уже с будет равно true. Значение b будет вычисляться только в том случае, если a равно false

То же самое касается пары операций &/&&. В выражении c=a&b; будут вычисляться оба значения - a и b.

В выражении же c=a&&b; сначала будет вычисляться значение a, и если оно равно false, то вычисление значения b уже не имеет смысла, так как значение c в любом случае равно false. Значение b будет вычисляться только в том случае, если a равно true

Таким образом, операции || и && более удобны в вычислениях, позволяя сократить время на вычисление значения выражения и тем самым повышая производительность. А операции | и & больше подходят для выполнения поразрядных операций над числами.

Операции языка Java

Примеры операций сравнения

```
1  boolean a1 = (5 > 6) || (4 < 6); // 5 > 6 – false, 4 < 6 – true, поэтому возвращается true
2  boolean a2 = (5 > 6) || (4 > 6); // 5 > 6 – false, 4 > 6 – false, поэтому возвращается false
3  boolean a3 = (5 > 6) && (4 < 6); // 5 > 6 – false, 4 < 6 – true, поэтому возвращается false
4  boolean a4 = (50 > 6) && (4 / 2 < 3); // 50 > 6 – true, 4/2 < 3 – true, поэтому возвращается true
5  boolean a5 = (5 > 6) ^ (4 < 6); // 5 > 6 – false, 4 < 6 – true, поэтому возвращается true
6  boolean a6 = (50 > 6) ^ (4 / 2 < 3); // 50 > 6 – true, 4/2 < 3 – true, поэтому возвращается false
```

Операции языка Java

Операции присваивания

В завершение рассмотрим операции присваивания, которые в основном представляют комбинацию простого присваивания с другими операциями:

Операция	Значение
=	просто приравнивает одно значение другому: c=b;
+=	c+=b; (переменной с присваивается результат сложения c и b)
-=	c-=b; (переменной с присваивается результат вычитания b из c)
*=	c*=b; (переменной с присваивается результат произведения c и b)
/=	c/=b; (переменной с присваивается результат деления c на b)
%=	c%=b; (переменной с присваивается остаток от деления c на b)
&=	c&=b; (переменной с присваивается значение c&b)
|=	c|=b; (переменной с присваивается значение c|b)
^=	c^=b; (переменной с присваивается значение c^b)
<<=	c<<=b; (переменной с присваивается значение c<<b)
>>=	c>>=b; (переменной с присваивается значение c>>b)
>>>=	c>>>=b; (переменной с присваивается значение c>>>b)