**#1. Вводный курс по языку программирования JAVA. Hello World**

**Java** — объектно-ориентированный язык программирования. В нем существуют классы (**class**) и объекты (**object**). Объекты в **Java** представляют собой экземпляры класса.

Например, модель android смартфона **Samsung Galaxy s6** вообще — это класс, а экземпляр Galaxy s6, котрый вы закажете на Ebay и он придет к вам по почте — это конкретный объект, экземпляр класса, и вы можете делать с ним все, что хотите.

Давайте напишем программу **Hello World**, которая просто выводит на экран надпись **«Hello, World!».**

|  |  |
| --- | --- |
| 1  2  3  4  5 | public class Main {      public static void main(String[] args) {          System.out.println("Hello, World!");      }  } |
|  |  |

Первая строка объявляет класс под названием **Main**.

|  |  |
| --- | --- |
| **1** | **public class Main {** |

В **Java** каждая строка кода, которая может выполняться должна находиться внутри класса. Эта строка объявляет класс **Main**, модификатор доступа **public** означает что класс общедоступен и любой другой класс может получить доступ к нему. На данный момент это не важно, так что не волнуйтесь. Для начала просто напишем наш код в классе **Main**, а про объекты поговорим позже.

Обратите внимание, что, когда мы объявляем общедоступный класс (**public**), мы должны объявить его в файле с тем же именем (**Main.java**), иначе мы получим ошибку при компиляции.

Следующая строка:

|  |  |
| --- | --- |
| **1** | **public static void main(String[] args) {** |

Это точка входа нашей **Java** программы. Метод **main** должен иметь точно такую же сигнатуру, как показано, иначе программа не будет работать.

* public снова же означает, что метод общедоступен
* static означает что вы можете выполнить этот метод без создания экземпляра класса Main
* void означает, что метод не возвращает никакого значения
* main — имя метода

При помощи этой строки мы выводим на экран «Hello, World!».

|  |  |
| --- | --- |
| **1** | **System.out.println("Hello, World!");** |

Это массив строк. Мы будем использовать его в [следующем](http://www.fandroid.info/tutorial-po-osnovam-yazyka-programmirovaniya-java-dlya-nachinayushhih/) уроке, так что не волнуйтесь, если сейчас вы не все понимаете.

Пока потренируйтесь выводить различный текст, ибо только практика сделает из вас программиста! =)

# #2. Вводный курс по языку программирования JAVA. Переменные и типы

Несмотря на то, что язык **Java** объектно-ориентирован, не все типы — объекты. Существуют так называемые примитивы (**primitives**). Вот список всех примитивов в **Java**:

* byte (число, 1 байт)
* short (число, 2 байта)
* int (число, 4 байта)
* long (число, 8 байтов)
* float (число с плавающей точкой, 4 байта)
* double (число с плавающей точкой, 8 байтов)
* char (символ, 2 байта)
* boolean (true (истина) или false (ложь), 1 байт)

**Java** — строго типизированный язык, это означает, что мы должны объявить переменные , прежде чем будем их использовать.

### Числа

Чтобы объявить и присвоить число используйте следующий синтаксис:

|  |  |
| --- | --- |
| **1**  **2** | **int myNumber;**  **myNumber = 5;** |

**=** это оператор присваивания.

Вы можете объединить эти операции:

|  |  |
| --- | --- |
| **1** | **int myNumber = 5;** |

Чтобы объявить число с плавающей точкой, используйте следующий синтаксис:

|  |  |
| --- | --- |
| **1**  **2** | **double d = 4.5;**  **d = 3.0;** |

Если вы хотите использовать float, то:

|  |  |
| --- | --- |
| **1** | **float f = (float) 4.5;** |

Или:

|  |  |
| --- | --- |
| **1** | **float f = 4.5f** |

(f — более короткий способ объявить float)

### Символы и строки

В Java символ — свой собственный тип, и это не просто число. Синтаксис:

|  |  |
| --- | --- |
| **1** | **char c = 'g';** |

String — не примитив. Это реальный тип. Вот несколько способов использования строки:

Создание строки с помощью конструктора

|  |  |
| --- | --- |
| **1** | **String s1 = new String("Who let the dogs out?");** |

С помощью двойных кавычек (» «).

|  |  |
| --- | --- |
| **1** | **String s2 = "Who who who who!";** |

В Java присутсвует конкатенация (объединение) строк при помощи оператора +.

|  |  |
| --- | --- |
| **1** | **String s3 = s1 + s2;** |

В Java нет перегрузки операторов! Оператор + определен только для строк, вы никогда не увидите его с другими объектами, только с примитивами.

|  |  |
| --- | --- |
| **1**  **2** | **int num = 5;**  **String s = "I have " + num + " cookies";** |

Заметьте, что кавычки с примитивами не используются.

### boolean

Каждый оператор сравнения в Java возвращает булевскую переменную (boolean), которая может принять только два значения: true (истина) или false (ложь).

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** | **boolean b = false;**  **b = true;**    **boolean toBe = false;**  **b = toBe || !toBe;**  **if (b) {**  **System.out.println(toBe);**  **}** |

Оператор || - это логическое «или».

А например, такой код не будет работать по причине несовместимости типов:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **int children = 0;**  **b = children;  // Не будет работать, требуется boolean, а найден int**  **if (children) {  // Не будет работать, требуется boolean, а найден int**  **// Не будет работать, требуется boolean, а найден int**  **}** |

# #3. Вводный курс по языку программирования JAVA. Условные операторы

**Java** использует булевские (логические) переменные, чтобы оценивать условия. Значение **true** или **false** возвратится, после того как выражение будет оценено. Например:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **int a = 4;**  **boolean b = a == 4;**  **if (b) {**  **System.out.println("It's true!");**  **}** |

Конечно, мы обычно не присваиваем условное выражение булевской переменной, мы просто используем короткую версию:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4** | **int a = 4;**  **if (a == 4) {**  **System.out.println("Ohhh! So a is 4!");**  **}** |

### Логические операторы

Есть не так много операторов, которые можно использовать в условиях. Вот они:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **int a = 4;**  **int b = 5;**  **boolean result;**  **result = a < b;  // истина result = a > b;  // ложь**  **result = a <= 4; // меньше или равно - истина result = b >= 6;  // больше или равно - ложь**  **result = a == b;  // равно - ложь**  **result = a != b;  // неравно - истина**  **result = a > b || a < b;  // логическое ИЛИ - истина**  **result = 3 < a && a < 6;  // логическое И - истина**  **result = !result;  // Логическое НЕ - ложь** |

### Оператор if — else

Синтаксис оператора **if — else** довольно прост:

|  |  |
| --- | --- |
| **1**  **2**  **3** | **if (a == b) {**  **// Тело метода. Выполняется если a и b равны.**  **}** |

Так же мы можем добавить еще одно выражение, на случай, если условие не выполняется:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **if (a == b) {**  **// Мы уже знаем эту часть**  **} else {**  **// a и b не равны... :/**  **}** |

Если тело метода можно разместить в одну строку, можно не использовать { }

|  |  |
| --- | --- |
| **1**  **2** | **if (a == b) System.out.println("Yeah!");**  **else System.out.println("Ohhh...");** |

Или

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4** | **if (a == b)**  **System.out.println("Another line Wow!");**  **else**  **System.out.println("Double rainbow!");** |

Несмотря на то, что такой метод мог бы сделать ваш код короче, мы строго рекомендуем новичкам не использовать короткую версию условного оператора

### Другая сторона if

Есть еще один способ записать **if — else** в одну строку — с помощью оператора **?** :

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** | **int a = 4;**  **int result = a == 4 ? 1 : 8;**  **// result будет равен 1**  **// Или обычная форма записи:**  **int result;**  **if (a == 4) {**  **result = 1;**  **} else {**  **result = 8;**  **}** |

Опять же, мы не рекомендуем новичкам использовать эту версию if.

### Операторы == и equals

Оператор **==** работает немного по-другому на объектах, нежели на примитивах. Когда вы используем объекты и хотите проверить, равны ли они, оператор **==** скажет что они равны, только если объекты одинаковы, но если вы хотите проверить их на логическое соответствие, используйте метод **equals**. Например:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **String a = new String("Wow");**  **String b = new String("Wow");**  **String sameA = a;**    **boolean r1 = a == b;      // Ложь, так как a и b не один и тот же объект**  **boolean r2 = a.equals(b); // Истина, так как a и b логически равны**  **boolean r3 = a == sameA;  // Истина, так как a и sameA действительно один и тот же объект** |

# #4. Вводный курс по языку программирования JAVA. Массивы

### Массивы

**Массив** **(англ. Array)** это объект, хранящий в себе фиксированное количество значений одного типа. Другими словами, массив — это нумерованный набор переменных. Переменная в массиве называется элементом массива, а ее позиция в массиве задается индексом.

Массивы в Java тоже представляют собой объекты. Они должны быть объявлены, а затем созданы. Чтобы объявить переменную, которая будет содержать массив целых чисел, мы используем следующий синтаксис:

|  |  |
| --- | --- |
| **1** | **int[] arr;** |

Обратите внимание, размер не указан, так что мы еще не создали массив.

|  |  |
| --- | --- |
| **1** | **arr = new int[10];** |

Теперь мы создали новый массив размером 10. Мы можем проверить размер массива, выведя на экран его длину:

|  |  |
| --- | --- |
| **1** | **System.out.println(arr.length);** |

Так же мы можем получить доступ к массиву и установить значения:

|  |  |
| --- | --- |
| **1**  **2** | **arr[0] = 4;**  **arr[1] = arr[0] + 5;** |

Счет элементов массива в Java начинается с нуля, то есть доступ к первому элементу можно получить по индексу 0 (например, arr[0]). Кроме того, как показано на примере, массив размером 5 будет заканчиваться на индексе 4, так как счет начинается с нуля.

|  |  |
| --- | --- |
| **1**  **2** | **int[] arr = new int[5]**  **arr[4] = 4;  // Получение доступа и присвоение значения последнему элементу** |

Есть также возможность создать массив с указанием значений в одну строку:

|  |  |
| --- | --- |
| **1** | **int[] arr = {1, 2, 3, 4, 5};** |

Кстати, если вы попытаетесь вывести массив целиком на экран, получите что-то в этом роде: [I@f7e6a96.

Для вывода всех значений массива используйте метод Arrays.toString(), преобразующий массив в строку.

|  |  |
| --- | --- |
| **1** | **System.out.println(Arrays.toString(arr));** |

Или напишите цикл, выводящий последовательно элементы массива.

|  |  |
| --- | --- |
| **1**  **2**  **3** | **for(int i=0; i<arr.length; i++) {**  **System.out.println(arr[i]);**  **}** |

IntellijIdea комбинация **CTRL + SHIFT + /** - выделяем текст, нажимаем эту комбинацию и текст у нас становится комментарием.

**ALT + Enter** - импортирует нужный нам класс

# #5. Вводный курс по JAVA. Циклы (Loops)

Есть два вида циклов в Java, for и while.

### For

Цикл for состоит из трех секций:

|  |  |
| --- | --- |
| **1** | **for (int i = 0; i < 3; i++) {}** |

Первая секция выполняется один раз, когда мы входим в цикл.  В нашем примере здесь задается начальное значение переменной i. Вторая секция проверяет логическое условие, если оно возвращает true, выполняются операторы в цикле, если false, выход из цикла. Вторая секция в первый раз запускается сразу после первой секции, и выполняется каждый раз, пока условие верно, вызывая третью секцию. Третья секция — заключительный оператор, его действие выполняется каждый раз при выполнении цикла. В нашем примере это инкремент, который при каждом выполнении увеличивает значение переменной на единицу.

Таким образом, цикл будет работать 3 раза. Вот порядок команд:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | **int i = 0;**  **i < 3 // 0 < 3 = true**  **// Inside of loop**  **i++ // i is now 1**  **i < 3 // 1 < 3 = true**  **// Inside of loop**  **i++ // i is now 2**  **i < 3 // 2 < 3 = true**  **// Inside of loop**  **i++ // i is now 3**  **i < 3 // 3 < 3 = false**  **// Loop is done...** |
|  |  |

Мы можем опустить первую и третью секции цикла (как бы странно это ни выглядело), и цикл все еще будет работать:

|  |  |
| --- | --- |
| **1** | **for (;i < 5;) {}** |

Для случаев, где нужно использовать цикл схожих повторяющихся действий, мы используем цикл while

### While

Синтаксис похож на предыдущий:

|  |  |
| --- | --- |
| **1** | **while (condition) {}** |

Условие будет работать впервые при вводе и каждый раз, когда вызывается цикл. Если условие возвратит **false**, то цикл не будет работать.  
Если мы хотим, чтобы цикл всегда выполнял по крайней мере одно действие, мы можем использовать **do-while**:

|  |  |
| --- | --- |
| **1**  **2**  **3** | **do {**    **} while(condition);** |

Не забудьте точку с запятой в конце.

### Foreach

Другая версия **for**, это **foreach**. Но в Java решили не добавлять новое ключевое слово **each**. Ключевое слово, которое мы используем, все еще **for**, но когда мы хотим выполнить действия над элементами массива, делаем так:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4** | **int[] arr = {2, 0, 1, 3};**  **for (int el : arr) {**  **System.out.println(el);**  **}** |

Это была короткая версия, эквивалентная следующей записи:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **int[] arr = {1, 9, 9, 5};**  **for (int i = 0; i < arr.length; i++) {**  **int el = arr[i];**  **System.out.println(el);**  **}** |

Заметьте, что, если вы хотите использовать индекс элемента в цикле, Вы должны использовать более длинную версию и не можете использовать **foreach**.

### break and continue

Эти два ключевых слова помогают нам управлять циклом из него. Оператор **break** останавливает цикл и переходит к оператору, следующему за ним:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | **int i;**  **for (i = 0; i < 5; i++) {**  **if (i >= 2) {**  **break;**  **}**  **System.out.println("Yuhu");**  **}**  **System.out.println(i);**  **// Output:**  **// Yuhu**  **// Yuhu**  **// 2** |

Оператор **continue** остановит текущую итерацию и переместится в следующую. Заметьте, что в цикле **for** действие в третьей секции будет выполнено при этом.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12**  **13**  **14**  **15**  **16**  **17**  **18** | **int i;**  **for (i = 0; i < 5; i++) {**  **if (i >= 3) {**  **break;**  **}**  **System.out.println("Yuhu");**  **if (i >= 1) {**  **continue;**  **}**  **System.out.println("Tata");**  **}**  **System.out.println(i);**  **// Yuhu**  **// Tata**  **// Yuhu**  **// Yuhu**  **// 3** |

### Упражнение

Используя цикл, выведите на экран все четные числа из списка чисел в порядке получения. Не выводите числа, идущие после числа 237 в последовательности.

# #6. Вводный курс по JAVA. Методы

Методы в Java — это законченная последовательность действий (инструкций), направленных на решение отдельной задачи. По сути, это функции (они же процедуры, подпрограммы) более ранних, не ООП языков. Только эти функции являются членами классов и для различия с обычными функциями, согласно терминологии объектно-ориентированного программирования, называются методами.

Методы определяются всегда внутри классов:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **public class Main {**  **public static void foo() {**  **// Тело метода**  **}**  **}** |

**foo** -  это метод, который мы определили в классе Main, давайте его рассмотрим.

* **public** — тип доступа (метод может вызываться из другого класса). Существуют и другие типы доступа, к примеру ***private*** (метод доступен только внутри класса) и ***protected*** (о нем мы будем говорить позже).
* **static**  означает что  метод статический, он принадлежит классу Main, а не конкретному экземпляру класса Main. Мы можем вызвать этот метод из другого класса так:  Main.foo().
* **void** значит, что этот метод не возвращает значение. Методы могут возвращать значение в Java и оно должно быть определено при объявлении метода. Однако, вы можете использовать return просто для выхода из метода.
* Этот метод не получает никаких аргументов, но методы java могут получать аргументы, как мы увидим далее на примерах.

Если тип возвращаемого значения не **void**, в теле метода должен быть хотя бы один оператор -

**return *выражение*;**

- где тип выражения должен совпадать с типом возвращаемого значения. Этот оператор возвращает результат вычисления выражения в точку вызова метода.  
Если тип возвращаемого значения – **void**, возврат из метода выполняется либо после выполнения последнего оператора тела метода, либо в результате выполнения оператора   
**return;**  
(таких операторов в теле метода может быть несколько).  
Пример объявления метода, возвращающего значение типа int – сумму двух своих параметров типа int:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **int sum(int a, int b){**  **int x;**  **x = a + b;**  **return x;**  **}** |

При вызове метода, например, sum(5, 3), параметры 5 и 3 передаются в метод, как значения соответственно a и b, и оператор вызова метода sum(5, 3)– заменяется значением, возвращаемым методом (8).

В отличие от языка C, в котором тип параметра, задаваемого при вызове, приводится к типу параметра в объявлении функции, тип задаваемого параметра в Java должен строго соответствовать типу параметра в объявлении метода, поэтому вызов метода **sum(1.5, 8)** приведет к ошибке при компиляции программы.

#### Не статические методы

Не статические методы в Java используются чаще, чем статические методы. Эти методы могут принадлежать любому объекту, экземпляру класса, а не всему классу.

Не статические методы могут получать доступ и изменять поля объекта.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9** | **public class Student {**  **private String name;**  **public String getName() {**  **return name;**  **}**  **public void setName(String name) {**  **this.name = name;**  **}**  **}** |

Вызов методов требует экземпляра класса Student.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6** | **Student s = new Student();**  **s.setName("Danielle");**  **String name = s.getName();**    **Student.setName("Bob"); // Не будет работать!**  **Student.getName(); // Не будет работать!** |

#### Перегруженные методы

В языке Java в пределах одного класса можно определить два или более ме­тодов, которые совместно используют одно и то же имя, но имеют разное количество параметров. Когда это имеет место, методы называют **перегру­женными**, а о процессе говорят как о **перегрузке метода** (method overloading).

Когда метод вызывается, то по количеству параметров и/или их типам среда выполнения Java определяет, какую именно версию перегруженного метода надо вызывать (тип возвращаемого значения во внимание не принимается, хотя, в принципе, он тоже может отличаться у разных версий перегруженных методов).

Например, метод

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5** | **double sum(double a, double b) {**  **double x;**  **x = a + b;**  **return x;**  **}** |

вместе с объявленным ранее методом **int** **sum(int** **a, int** **b)**составляют пару перегруженных методов и при вызове **sum(5, 8)** будет вызван первый метод, а при вызове **sum(5.0, 8.0)** будет вызван второй метод.

По умолчанию метод, как и переменная, доступен только классам в том же пакете (наборе классов), что и исходный класс. Если перед возвращаемым типом задан модификатор доступа **public**, то метод является глобальным и доступен любым объектам, а модификатор **private** означает, что метод доступен в том классе, в котором он был объявлен, т.е. метод инкапсулирован в данном классе.

#### Переопределение методов

Кроме перегрузки существует также замещение, или **переопределение методов** (англ. overriding). Замещение происходит, когда класс потомок (подкласс) определяет некоторый метод, который уже есть в родительском классе(суперклассе), таким образом новый метод заменяет метод суперкласса.  У нового метода подкласса должны быть те же параметры или сигнатура, тип возвращаемого результата, что и у метода родительского класса.

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | **public class Thought {**  **public void message() {**  **System.out.println("Я себя чувствую как стрекоза, попавшая в параллельную вселенную.");**  **}**  **}**  **public class Advice extends Thought {**  **@Override  // Аннотация @Override с Java 5 является необязательной, но весьма полезной**  **public void message() {**  **System.out.println("Внимание: Даты в календаре ближе, чем кажутся.");**  **}**  **}** |

 Класс **Thought** представляет собой суперкласс и обеспечивает вызов метода **message().** Подкласс, называемый **Advice**, наследует каждый метод класса **Though**t. Однако, класс **Advice** переопределяет метод **message()**, замещая функционал, описанный в классе **Thought**.

В Java, когда подкласс содержит метод, переопределяющий метод суперкласса, то он может помимо своего метода вызывать и метод суперкласса при помощи ключевого слова **super**.  
Например, нижеследующий вариант выводит оба сообщения при вызове метода подкласса:

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7** | **public class Advice extends Thought {**  **@Override**  **public void message() {**  **System.out.println("Внимание: Даты в календаре ближе, чем кажутся.");**  **super.message();  // Вызов версии метода родительского класса**  **}**  **}** |

 Существуют методы, которые подкласс не может переопределять. Например, в Java метод, объявленный с ключевым словом **final**, не может быть переопределён. Методы, объявленные как **private** или **static** не могут быть переопределены, поскольку это соответствует неявному использованию **final**.

#### Резюме

* Каждый java-метод должен быть внутри класса
* Статические методы принадлежат классу, а не статические методы принадлежат объектам, экземплярам класса
* В пределах одного класса может быть два и более методов с одинаковыми именами, но разным набором параметров (перегрузка метода)
* Класс-потомок может обеспечивать свою реализацию метода, уже реализованного в одном из родительских классов (переопределение метода)

#### Упражнение

Написать метод **printFullName** класса **Student,** который выводит полное ФИО студента.