**Синтаксис языка Java**

Синтаксис Java — это основа языка, все основные правила, команды, а также конструкции для написания программ.

Каждая Java-программа — это по сути набор объектов, которые включают данные (переменные) и поведение (функции или методы). Также программа Java представляет собой класс или несколько классов.

**Классы и Объекты**

Объект — это экземпляр класса. Класс — это модель объекта реального мира.

Например: Велосипед и легковой автомобиль это транспорт. Велосипед и автомобиль это реальные предметы, экземпляры, но в реальном мире нет такого предмета как “транспорт” - любой транспорт будет являться определенным предметом. Таким образом транспорт - это класс объединяющий множество предметов, а велосипед и автомобиль это экземпляры класса со своими уникальными свойствами, состоянием и поведением.

**Методы**

Методы предназначены для описания логики, работы с данными и выполнения всех действий. В методах обычно находятся основные функции которые может выполнять класс и в частности и программа в целом. Каждый метод определяет поведение. Класс может содержать множество методов.

**Переменные экземпляра**

У каждого объекта есть уникальный набор переменных экземпляра. Состояние объекта обычно формируется значениями, которые присваиваются этим переменным экземпляра.

Пример простой программы на Java:

public class Hello {

public static void main(String[] args) {

System.*out*.println("Hello, Java!");

}

}

Эта программа выводит в консоли строку «Hello, Java!».

Каждая программа на Java представляет собой класс или, чаще всего, множество классов. Строка public class Hello означает что программа располагается в классе Hello.

Пара фигурных скобок {} обозначает блок, группу программируемых формулировок, которые рассматриваются как единое целое. Где { означает начало блока, а } его конец. Блоки могут быть вложены друг в друга, или они могут идти последовательно.

Строчка public static void main(String[] args) - это начато основного (main) метода:

* public — модификатор доступа. К переменной, методу или классу, отмеченным модификатором public, можно получить доступ из любой точки программы;
* void — возвращаемый тип метода. Void значит, что он не возвращает никакого значения;
* main — это название метода;
* String[] args — это main (основной) аргумент метода. Пока достаточно знать, что почти у каждой Java-программы есть метод main: он запускает программу и объявляется как public static void main(String[] args).

Строка System.out.println("Hello, Java!") выполняет метод println объекта out. Объект out объявляется в классе OutputStream и статически инициализируется в классе System..

**Основные правила синтаксиса Java**

Есть несколько основных правил синтаксиса, которым нужно следовать при программировании на Java:

* имя файла должно совпадать с именем класса;
* чаще всего каждый класс находится в отдельном файле с расширением .java. Файлы классов обычно сгруппированы в папки. Эти папки называются пакетами;
* символы чувствительны к регистру. String не равно string;
* метод (процедура, функция) представляет собой последовательность команд. Методы определяют поведение над объектом;
* рекомендуется чтобы первая буква имени класса была в верхнем регистре. Если вы используете несколько слов, указывайте прописные буквы для первой буквы каждого слова (например, «MyFirstJavaClass»);
* рекомендуется чтобы имена всех методов в синтаксисе Java начинались со строчной буквы. При использовании нескольких слов последующие буквы пишутся заглавными (public void myFirstMethodName ());
* файлы сохраняются с именем класса и расширением .java (MyFirstJavaClass.java);
* в синтаксисе Java есть разделители {...}, которые обозначают блок кода и новую область кода;
* каждый оператор кода должен заканчиваться точкой с запятой.

**Переменные и типы данных**

Переменные — это специальные сущности, используемые для хранения данных. В Java все данные хранятся в переменных. У каждой переменной есть свой тип данных, имя (идентификатор) и значение.

Некоторые типы:

* целые числа: byte, short, int, long;
* дробные числами: float и double;
* логические значения: boolean;
* символьные значения (для представления букв и цифр): char;

Пример переменных в Java:

int s;

s = 5;

char myChar = 'a';

В этом коде создали целочисленную переменную s (пустой контейнер), а затем поместили в нее значение 5.

В переменной myChar создали переменную с типом данных char и определили её как букву a. В данном случае мы создали переменную и одновременно присвоили ей значение. Синтаксис Java позволяет вам сделать это таким образом.

**Объявление переменных**

Объявление — это процесс выделения памяти для переменной определенного типа и присвоения ей имени.

Вы также можем объявить инициализацию переменной с помощью оператора присваивания (=). Это означает, что вы помещаете определенное значение в выделенную память. Это можно сделать прямо в момент объявления или позже.

String str;

int i = 5;

str = "here is my string";

Если вы объявляете переменную без инициализации, она все равно получит какое-то значение по умолчанию. Для int это значение равно 0, для String или любого другого ссылочного типа это специальный идентификатор null.

**Идентификаторы в Java**

Идентификаторы — это имена компонентов Java — классов, переменных и методов. У всех компонентов Java должны быть имена.

class Violin {

int age;

String masterName;

}

* Violin — идентификатор класса.
* age и masterName — идентификаторы переменных.

Вот некоторые правила идентификаторов Java:

* все идентификаторы начинаются с латинской буквы (от A до Z или от a до z), символа валюты ($) или символа подчеркивания (\_);
* после первого символа у идентификаторов может быть любая комбинация символов;
* ключевое слово Java не может быть идентификатором (о ключевых словах вы узнаете чуть позже);
* идентификаторы чувствительны к регистру.

Допустимые идентификаторы: java, $mySalary, \_something

Недопустимые идентификаторы: 1stPart, -one

**Модификаторы в Java**

Модификаторы — это специальные элементы языка Java, которые вы можете использовать для изменения элементов (классов, методов, переменных). В Java есть две категории модификаторов: модификаторы доступа и другие модификаторы.

Примеры модификаторов доступа

В Java есть четыре модификатора доступа:

* public. Открытый элемент. Доступ к нему можно получить из класса, снаружи класса, внутри и снаружи пакета;
* К элементу с модификатором по умолчанию — default (пустым) — есть доступ только внутри пакета;
* protected модификатор — можно получить доступ внутри и снаружи пакета через дочерний класс;
* private — элемент доступен только внутри объявленного им класса.

Другие модификаторов (для классов, полей, методов, интерфейсов и так далее):

* static
* final
* abstract
* synchronized
* transient
* volatile
* native

**Ключевые слова в Java (Java Keywords)**

Ключевые слова Java — это специальные слова для использования в Java, которые действуют как ключ к коду. Они также известны как зарезервированные слова: их нельзя использовать для идентификаторов переменных, методов, классов и т. д. Перечислять их все не имеет смысла, любой текст который воспринимается компилятором за специальные слова нельзя использовать для создания идентификаторов, например: модификаторы доступа, циклы, типы и т.д.

**Комментарии**

Java поддерживает однострочные и многострочные комментарии. Все символы доступны внутри любого комментария и игнорируются компилятором Java. Разработчики используют их, чтобы объяснить код или вспомнить что-то.

*//однострочный комментарий*

*/\*а вот многострочный комментарий. Как видите, в нем используются слеши и звездочки в начале и в конце.\*/*

**Массивы**

Массив представляет набор однотипных значений. Объявление массива похоже на объявление обычной переменной, которая хранит одиночное значение, причем есть два способа объявления массива:

тип\_данных название\_массива[];

// либо

тип\_данных[] название\_массива;

Например, определите массив чисел:

int nums[];

int[] nums2;

После объявления массива вы можете инициализировать его:

int nums[];

nums **=** new int[4];  // массив из 4 чисел

Создание массива производится с помощью следующей конструкции: new тип\_данных[количество\_элементов], где new - ключевое слово, выделяющее память для указанного в скобках количества элементов. Например, nums = new int[4]; - в этом выражении создается массив из четырех элементов int, и каждый элемент будет иметь значение по умолчанию - число 0.

Также можно сразу при объявлении массива инициализировать его:

int nums[] **=** new int[4];    // массив из 4 чисел

int[] nums2 **=** new int[5];   // массив из 5 чисел

При подобной инициализации все элементы массива имеют значение по умолчанию. Для числовых типов (в том числе для типа char) это число 0, для типа boolean это значение false, а для остальных объектов это значение null. Например, для типа int значением по умолчанию является число 0, поэтому выше определенный массив nums будет состоять из четырех нулей.

Однако также можно задать конкретные значения для элементов массива при его создании:

// эти два способа равноценны

int[] nums **=** new int[] { 1, 2, 3, 5 };

int[] nums2 **=** { 1, 2, 3, 5 };

Стоит отметить, что в этом случае в квадратных скобках не указывается размер массива, так как он вычисляется по количеству элементов в фигурных скобках.

После создания массива мы можем обратиться к любому его элементу по индексу, который передается в квадратных скобках после названия переменной массива:

int[] nums **=** new int[4];

// устанавливаем значения элементов массива

nums[0] **=** 1;

nums[1] **=** 2;

nums[2] **=** 4;

nums[3] **=** 100;

// получаем значение третьего элемента массива

System.out.println(nums[2]);    // 4

Индексация элементов массива начинается с 0, поэтому в данном случае, чтобы обратиться к четвертому элементу в массиве, нам надо использовать выражение nums[3].

И так как у нас массив определен только для 4 элементов, то мы не можем обратиться, например, к шестому элементу: nums[5] = 5;. Если мы так попытаемся сделать, то мы получим ошибку.

**Длина массива**

Важнейшее свойство, которым обладают массивы, является свойство length, возвращающее длину массива, то есть количество его элементов:

int[] nums **=** {1, 2, 3, 4, 5};

int length **=** nums.length;   // 5

Нередко бывает неизвестным последний индекс, и чтобы получить последний элемент массива, мы можем использовать это свойство:

int last **=** nums[nums.length**-**1];

**Многомерные массивы**

Ранее мы рассматривали одномерные массивы, которые можно представить как цепочку или строку однотипных значений. Но кроме одномерных массивов также бывают и многомерными. Наиболее известный многомерный массив - таблица, представляющая двухмерный массив:

int[] nums1 **=** new int[] { 0, 1, 2, 3, 4, 5 };

int[][] nums2 **=** { { 0, 1, 2 }, { 3, 4, 5 } };

Визуально оба массива можно представить следующим образом:

Одномерный массив nums1

| 0 | 1 | 2 | 3 | 4 | 5 |
| --- | --- | --- | --- | --- | --- |

Двухмерный массив nums2

| 0 | 1 | 2 |
| --- | --- | --- |
| 3 | 4 | 5 |

Поскольку массив nums2 двухмерный, он представляет собой простую таблицу. Его также можно было создать следующим образом: int[][] nums2 = new int[2][3];. Количество квадратных скобок указывает на размерность массива. А числа в скобках - на количество строк и столбцов. И также, используя индексы, мы можем использовать элементы массива в программе:

// установим элемент первого столбца второй строки

nums2[1][0]**=**44;

System.out.println(nums2[1][0]);

Объявление трехмерного массива могло бы выглядеть так:

int[][][] nums3 **=** new int[2][3][4];

**Зубчатый массив**

Многомерные массивы могут быть также представлены как "зубчатые массивы". В вышеприведенном примере двухмерный массив имел 2 строчки и три столбца, поэтому у нас получалась ровная таблица. Но мы можем каждому элементу в двухмерном массиве присвоить отдельный массив с различным количеством элементов:

int[][] nums **=** new int[3][];

nums[0] **=** new int[2];

nums[1] **=** new int[3];

nums[2] **=** new int[5];

**foreach**

Специальная версия цикла for предназначена для перебора элементов в наборах элементов, например, в массивах и коллекциях. Она аналогична действию цикла foreach , который имеется в других языках программирования. Формальное ее объявление:

**for** (тип\_данных название\_переменной : контейнер){

    // действия

}

Например:

int[] array **=** new int[] { 1, 2, 3, 4, 5 };

**for** (int i : array){

    System.out.println(i);

}

В качестве контейнера в данном случае выступает массив данных типа int. Затем объявляется переменная с типом int

То же самое можно было бы сделать и с помощью обычной версии for:

int[] array **=** new int[] { 1, 2, 3, 4, 5 };

**for** (int i **=** 0; i < array.length; i**++**){

    System.out.println(array[i]);

}

В то же время эта версия цикла for более гибкая по сравнению for (int i : array). В частности, в этой версии мы можем изменять элементы:

int[] array **=** new int[] { 1, 2, 3, 4, 5 };

**for** (int i**=**0; i<array.length;i**++**){

    array[i] **=** array[i] **\*** 2;

    System.out.println(array[i]);

}

**Перебор многомерных массивов в цикле**

int[][] nums **=** new int[][]

{

    {1, 2, 3},

    {4, 5, 6},

    {7, 8, 9}

};

**for** (int i **=** 0; i < nums.length; i**++**){

**for**(int j**=**0; j < nums[i].length; j**++**){

        System.out.printf("%d ", nums[i][j]);

    }

    System.out.println();

}

Сначала создается цикл для перебора по строкам, а затем внутри первого цикла создается внутренний цикл для перебора по столбцам конкретной строки. Подобным образом можно перебрать и трехмерные массивы, и наборы с большим количеством размерностей.

**Задание для самостоятельной работы:**

1. Сколько элементов имеет следующий массив?

int[][][] nums **=** new int[3][2][3];

1. Что будет выведено на консоль в результате выполнения следующего кода:

int[][] nums **=**

{

    {1, 2},

    {3, 4},

    {5, 6}

};

System.out.println(nums[3][2]);

1. Дан следующий массив

int[][] nums **=** {{ 1, 2 }, { 3, 4, 5 }, { 6, 7, 8, 9, 10 }};

1. Что будет выведено на консоль в следующем случае и почему?

int[][] nums **=** new int[3][3]{{ 1, 2, 3 }, { 3, 4, 5 }, { 6, 7, 8}};

System.out.println(nums[2][1]);