Язык Java – Лекция 6. Массивы и другие структуры данных в Java

**6.1 Определение массива**

Массив — это структура данных, в которой хранятся элементы одного типа. Его можно представить, как набор пронумерованных ячеек, в каждую из которых можно поместить какие-то данные (один элемент данных в одну ячейку). Доступ к конкретной ячейке осуществляется через её номер. Номер элемента в массиве также называют **индексом**[[1]](#footnote-1).

В случае с Java массив однороден, то есть во всех его ячейках будут храниться элементы одного типа. Так, например, массив целых чисел содержит только целые числа.

**6.2 Объявление и создание массива**

Как и любую переменную, массив в Java нужно объявить. Сделать это можно одним из двух способов. Они равноправны, но первый из них лучше соответствует стилю Java. Второй же — наследие языка Си (многие Си-программисты переходили на Java, и для их удобства был оставлен и альтернативный способ). В таблице приведены оба способа объявления массива в Java (рисунок 6.1):

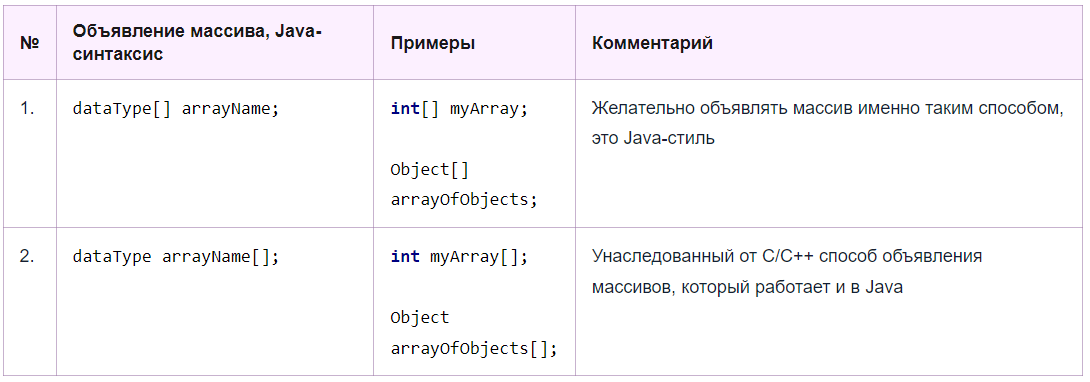


Рисунок 6.1.

В обоих случаях **dataType** — тип переменных в массиве. В примерах мы объявили два массива. В одном будут храниться целые числа типа int, в другом — объекты типа Object. Таким образом при объявлении массива у него появляется имя и тип (тип переменных массива). **arrayName** — это имя массива.

**Создать массив Java – это значит зарезервировать под него место в памяти**. Это можно сделать, как и с другими объектами, с помощью оператора **new**.

**new** typeOfArray [length].

Где **typeOfArray** — это тип массива, а **length** — его длина (то есть, количество ячеек), выраженная в целых числах (int). Однако здесь мы только выделили память под массив, но не связали созданный массив ни с какой объявленной ранее переменной. Обычно массив сначала объявляют, а потом создают, например,

**int**[] myArray; // объявление массива

myArray = **new** **int**[10]; // создание, то есть, выделение памяти для массива на 10 элементов типа int.

Здесь мы объявили массив целых чисел по имени myArray, а затем сообщили, что он состоит из 10 ячеек (в каждой из которых будет храниться какое-то целое число). Однако гораздо чаще массив создают сразу после объявления с помощью такого сокращённого синтаксиса:

**int**[] myArray = **new** **int**[10]; // объявление и выделение памяти одновременно.

***Что важно****! После создания массива с помощью* ***new****, в его ячейках записаны значения по умолчанию. Для численных типов (как в нашем примере) это будет 0, для boolean — false, для* ссылочных типов — null*.*

Таким образом после операции **int**[] myArray = **new** **int**[10]; мы получаем массив из десяти целых чисел, и, пока это не изменится в ходе программы, в каждой ячейке записан 0.

**6.3 Длина массива**

Как мы уже говорили выше, длина массива — это количество элементов, под которое рассчитан массив. **Длину массива нельзя изменить после его создания.** В Java элементы массива нумеруются с нуля. То есть, если у нас есть массив на 10 элементов, то первый элемент массива будет иметь индекс 0, а последний — 9. Получить доступ к длине массива можно с помощью **переменной** **length**.

**int**[] myArray = **new** **int**[10]; // создали массив целых чисел на 10 элементов и присвоили ему имя myArray

System.out.println(myArray.length); // вывели в консоль длину массива, то есть количество элементов,

Т. к. массив всегда хранит множество значений, то для именования переменной используется:

* **существительное во множественном числе**, например, *cats, cars, resumes, playerAttempts*
* **местоимение**, например, *allNumbers, myBooks*
* **прилагательное**, например, *physicalConstants*

**6.4** **Инициализация массива и доступ к его элементам**

Инициализация массива — это заполнение его конкретными данными (**не по умолчанию**!).

Два способа:

Первый:

String [] seasons = **new** String[4]; /\* объявили и создали массив. Java выделила память под массив из 4 строк, и сейчас в каждой ячейке записано значение **null** (поскольку строка — ссылочный тип) \*/

seasons[0] = "Winter"; /\* в первую ячейку, то есть, в ячейку с нулевым номером мы записали строку Winter. Тут мы получаем доступ к нулевому элементу массива и записываем туда конкретное значение \*/

seasons[1] = "Spring"; // проделываем ту же процедуру с ячейкой номер 1 (второй)

seasons[2] = "Summer"; // ...номер 2

seasons[3] = "Autumn"; // и с последней, номер 3

Второй:

String[] seasons = **new** String[] {"Winter", "Spring", "Summer", "Autumn"};

Более того, оператор new можно опустить:

String[] seasons = {"Winter", "Spring", "Summer", "Autumn"};

Массив может хранить не только значения примитивных, но и ссылочных типов. И сам он тоже является ссылочным типом данных. В строке ***String[] fullNames;*** объявляется переменная массива строк, а в строке ***short[] nums*;** — переменная для хранения примитивных целых значений.

Забегая вперед, необходимо отметить, что **переменные массивов хранят не сам массив, а ссылку на него**, которая, в свою очередь, указывает на начало области памяти, содержащую его значения.

Схема работы по инициализации массива по команде ***int****[] array =* ***new******int****[10]*; (рисунок 6.2):

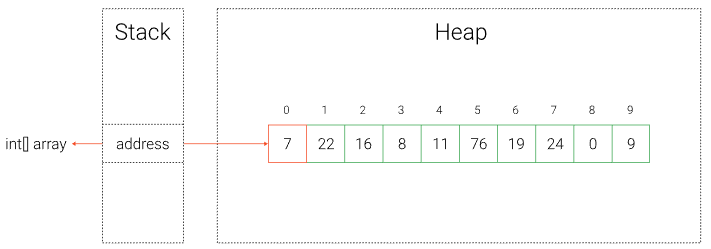


Рисунок 6.2

**6.6 Вывод массива**

Вывести элементы массива на экран (то есть, в консоль) можно, например, с помощью цикла for.

String[] seasons = **new** String[] {"Winter", "Spring", "Summer", "Autumn"};

**for** (**int** i = 0; i < 4; i++) {

System.out.println(seasons[i]);

}

**6.7 Многомерные массивы**

Массив массивов называется двумерным (рисунок 6.3).

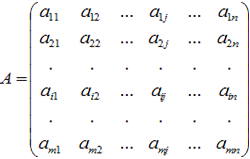


Рисунок 6.3

Многомерный массив объявляется и создается следующим образом:

Int **[][]** myTwoDimentionalArray = **new** **int** [8][8];

Data\_type[dimension1][dimension2][]..[dimensionN] arrayName=**new** data\_type[size1][size2]….[sizeN];

Пример инициализации:

**int** [][] twoDimArray = {{5,7,3,17}, {7,0,1,12}, {8,1,2,3}};

***Вывод двумерного массива на экран***

**int** [][] twoDimArray = {{5,7,3,17}, {7,0,1,12}, {8,1,2,3}};//объявили массив и заполнили его элементами

**for** (**int** i = 0; i < 3; i++) { //идём по строкам

**for** (**int** j = 0; j < 4; j++) { //идём по столбцам

System.out.print(" " + twoDimArray[i][j] + " "); //вывод элемента

}

System.out.println(); //перенос строки ради визуального сохранения табличной формы

}

**6.9 Массив – это не класс, но объект!**

Как написано в документации, массив является [объектом](https://docs.oracle.com/javase/specs/jls/se18/html/jls-4.html#jls-4.3.1): *«An object is a* ***class instance*** *or an array»*.

Но хоть массив и является таковым, но при этом не имеет класса, который описывал бы его. Мы просто используем массив, как есть. Но это «как есть» хранит в себе множество скрытых процессов и тайн, которые мы по возможности приоткроем.

Хоть тип массива и не является классом, например *int[]*, но все же имеет ассоциируемый с ним класс, который **автоматически создает виртуальная машина** (Java Virtual Machine, JVM).

При этом данный класс неявно наследуется от *java.lang.Object*, что делает доступным все его методы (кроме clone) и интерфейсы *Cloneable* и *Serializable*. Но у программиста нет доступа к его коду, его нельзя посмотреть глазами.

Ассоциируемые классы создаются для каждого типа хранимой в массиве информации, например, для *boolean[]*, или *Player[]*, или *String* и т. д. И мы можем узнать, как эти классы называются.

Они-то и являются реальными типами массивов. Чтобы увидеть их названия, необходимо воспользоваться Java-магией (рефлексией), которая позволит узнать настоящий тип массива, его суперкласс (от которого он наследуется) и список доступных методов.

Создадим массив нулевой длины (пустой) и отобразим нужную нам информацию (не забудьте сделать импорт *java.lang.reflect.Method*) (рисунок 6.3):

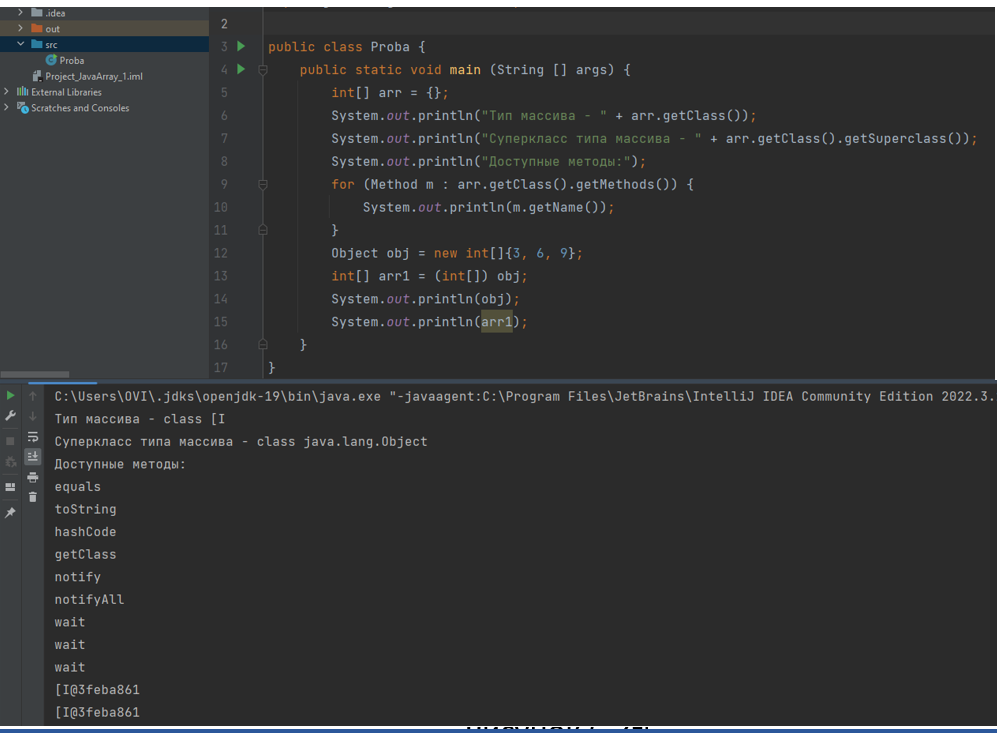


Рисунок 6.3

Что нам вывел код:

* [I — это сигнатура типа (класса, который JVM создает во время выполнения) для объекта «массив с элементами типа int». Это и есть настоящий тип данных массива, как объекта
* [говорит, что это одномерный массив
* I, что он содержит целые числа
* Имя суперкласса, записанного с помощью его полного имени — *java.lang.Object*

Все возможные сигнатуры типов имеют следующий формат (рисунок 6.4):

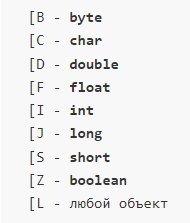


Рисунок 6.4

Можно сделать вывод, что тип массива и тип хранимых им значений — это формально разные типы. В нашем примере типом массива является тип [I, а типом хранимых значений — *int*.

Разберем еще одно доказательство того, что *Object* является суперклассом для массивов.

В строке 12 объявляется и инициализируется массив, ссылка на который присваивается переменной типа *Object*. И при этом никакой ошибки не возникнет. В строке 13 происходит приведение *Object* к *int[]*. В итоге обе переменные (их ссылки) указывают на один и тот же массив. Поэтому строки 14 и 15 выдают одинаковый результат, так как известно, если попытаться вывести, например, в *println* значение **объекта**, просто записав его имя в метод в качестве аргумента, то неявно будет вызван метод ***toString()***. А так как массивы его не переопределяют, т. е. у них нет **своей реализации** этого метода, то используется реализация по умолчанию, записанная в классе [Object](https://docs.oracle.com/en/java/javase/18/docs/api/java.base/java/lang/Object.html#toString()).

Этот метод выводит результат согласно следующего кода:

***getClass().getName() + '@' + Integer.toHexString(hashCode())***

В итоге строка состоит из имени класса, в нашем случае — [I, разделителя в виде @ и шестнадцатеричного представления хеш-кода объекта. Для человека эта информация не несет никакой полезной нагрузки.

Нам же сейчас важно, что обе переменные указывают на один и тот же массив, т. е. хранят одну и ту же ссылку на него. Это говорит также о том, что на один массив может указывать (ссылаться!) любое количество переменных.

***6.9.1 О представлении массива в памяти***

Как в памяти представляются следующие массивы:

* **String**[] array1 = {};
* **char**[] array2 = **new** **char**[0];
* **short**[] array3 = **null**;

Массив с нулевым размером представляет собой объект, у которого есть заголовок объекта (*Object header*) и размерность, но нет места в памяти, выделенного для его элементов. При этом ссылка на такой массив является рабочей и сохраняется в переменную.

Массив нулевой длины не равен *null*.

Для третьего варианта *null*— это указатель в никуда, который является маркером отсутствия указателя на объект.

**6.10 О длине массива**

В [спецификации](https://docs.oracle.com/javase/specs/jls/se7/html/jls-10.html#jls-10.7) написано: *«Поле length является public final (общедоступным и неизменяемым) и содержит в себе количество элементов массива (длина может быть положительной или нулевой)».*

Это поле **вычисляется один раз при создании массива**, т. к. его размер никогда не меняется.

Вызов этого поля часто путают с вызовом метода *length()* получения длины *String*. **Но** **length — не метод, а поле**. **Доступ к нему можно получить напрямую**, поскольку оно является общедоступным и неизменяемым (хотя тут, возможно, что это просто плохое архитектурное решение из далекого прошлого).

Так как данное поле является частью объекта и не меняется, то это позволяет JVM не пересчитывать количество элементов массива всякий раз, когда в коде используется его длина — **она просто берет его из заголовка объекта, когда это требуется**.

Чтобы вас окончательно запутать, сошлюсь на документацию, в которой написано: **«Длина массива не является частью его типа»**. Значит *length* не хранится в ассоциируемом классе!

«Да, это какой-то вынос мозга», — скажете вы: **класса нет**, но массив — это объект. При этом он наследуется от *Object*. А *length* — вроде бы, поле, но не совсем.

**А где тогда хранится длина массива?**

Она размещается в заголовке объекта. Что такое заголовок? Это часть любого объекта, содержащая метаинформацию о нем. Давайте взглянем на этот заголовок, используя утилиту *Java Object Layout (JOL)*.

Чтобы программа ниже у вас заработала, необходимо в IDEA нажать *Ctrl + Alt + Shift + S*, выбрать *Modules*, а затем в *Dependencies* нажать *+ → Library… → From Maven…* В строке поиска ввести *org.openjdk.jol*.

<https://topjava.ru/blog/rukovodstvo-po-massivam-v-java-ch1#:~:text=Написанный%20выше%20код%20отобразит%20следующий%20результат%3A>

**6.11** **Класс Java Arrays**

Массив не предоставляет никаких методов, облегчающих с ним работу — все приходится реализовывать самостоятельно. Это связано с тем, что массив не имеет как такового класса своей реализации. Вызвать какой-либо собственный метод у него мы не можем (методы *Object* не в счет) — их просто нет.

Подобные «издевательства» над разработчиками продолжалось до тех пор, пока в JDK 1.2 не появился специальный утилитный класс [Arrays](https://docs.oracle.com/en/java/javase/19/docs/api/java.base/java/util/Arrays.html), призванный упростить работу с массивами.

*Утилитный класс — это класс, имеющий набор* ***статических*** *методов, никак не связанных между собой и не обладающий состоянием. Характерной особенностью таких классов является использование множественного числа в их имени, например, Collections, Objects, а также слов Util и Helper, например, ArrayUtils.*

Для использования *Arrays* в своих классах необходимо прописать импорт:

**import** java.util.Arrays;

Класс ***Arrays*** содержит методы, упрощающие выполнение стереотипных (часто повторяющихся) операций с массивами: отображение, копирование, сортировка, поиск элементов массива и т. д.

***Быстрый вывод***

int[][] myArray = {{18,28,18},{28,45,90},{45,3,14}};

System.out.printLn(Arrays.deepToString(myArray));

1. <https://topjava.ru/blog/rukovodstvo-po-massivam-v-java-ch1#:~:text=Написанный%20выше%20код%20отобразит%20следующий%20результат%3A> [↑](#footnote-ref-1)