Лекция 6

Массивы в Java

Массив − это конечная последовательность упорядоченных элементов одного типа, доступ к каждому элементу в которой осуществляется по его индексу.

Размер или длина массива − это общее количество элементов в массиве. Размер массива задаётся при создании массива и не может быть изменён в дальнейшем, т. е. нельзя убрать элементы из массива или добавить их туда, но можно в существующие элементы присвоить новые значения.

Индекс начального элемента − 0, следующего за ним − 1 и т. д. Индекс последнего элемента в массиве − на единицу меньше, чем размер массива.

В Java массивы являются объектами. Это значит, что имя, которое даётся каждому массиву, лишь указывает на адрес какого-то фрагмента данных в памяти. Кроме адреса в этой переменной ничего не хранится. Индекс массива, фактически, указывает на то, насколько надо отступить от начального элемента массива в памяти, чтоб добраться до нужного элемента.

Чтобы создать массив надо объявить для него подходящее имя, а затем с этим именем связать нужный фрагмент памяти, где и будут друг за другом храниться значения элементов массива.

Возможные следующие варианты объявления массива:

```
тип[] имя;
тип имя[];
```

где **тип** − это тип элементов массива, а **имя** − уникальный (незанятый другими переменными или объектами в этой части программы) идентификатор, начинающийся с буквы.

Примеры:

```
int[] a;
double[] ar1;
double ar2[];
```

В примере мы объявили имена для трёх массивов. С первом именем **a** сможет быть далее связан массив из элементов типа int, а с именами **ar1** и **ar2** далее смогут быть связаны массивы из вещественных чисел (типа double). Пока мы не создали массивы, а только подготовили имена для них.

Как и любой другой объект массив должен быть создан операцией **new**.

Создать (или как ещё говорят инициализировать) массивы можно следующим образом:

```
a=new int[10]; //массив из 10 элементов типа int
int n = 5;
ar1=new double[n]; //Массив из 5 элементов double
ar2={3.14, 2.71, 0, -2.5, 99.123}; //Массив из 6 элементов типа double
```

То есть при создании массива мы можем указать его размер, либо сразу перечислить через запятую все желаемые элементы в фигурных скобках (при

этом размер будет вычислен автоматически на основе той последовательности элементов, которая будет указана). Обратите внимание, что в данном случае после закрывающей фигурной скобки ставится точка с запятой, чего не бывает когда это скобка закрывает какой-то блок.

Если массив был создан с помощью оператора **new**, то каждый его элемент получает значение по умолчанию. Каким оно будет определяется на основании типа данных (0 для int, 0.0 для double и т.д.).

Наиболее частая ошибка у начинающих при работе с массивами классов примерно следующая. Создается сам массив, например,

```
A[] a1 = new A[10];
```

а потом сразу идет попытка работы с элементами этого массива. Но здесь построен только массив ссылок, а сами объекты еще не созданы.

Объявить имя для массива и создать сам массив можно было на одной строке по следующей схеме:

```
тип[] имя = new тип[размер];
тип[] имя = {эл0, эл1, …, элN};
```

Примеры:

```
int[] mas1 = {10,20,30};
int[] mas2 = new int[3];
```

Чтобы обратиться к какому-то из элементов массива для того, чтобы прочитать или изменить его значение, нужно указать имя массива и за ним индекс элемента в квадратных скобках. Элемент массива с конкретным индексом ведёт себя также, как переменная. Например, чтобы вывести последний элемент массива `mas1` мы должны написать в программе:

```
System.out.println("Последний элемент массива"+mas1[2]);
```

А вот так мы можем положить в массив mas2 тот же набор значений, что хранится в `mas1`:

```
mas2[0] = 10;
mas2[1]  = 20;
mas2[2] = 30;
```

Уже из этого примера видно, что для того, чтоб обратиться ко всем элементам массива, нам приходится повторять однотипные действия. Как вы помните для многократного повторения операций используются циклы. Соответственно, мы могли бы заполнить массив нужными элементами с помощью цикла:

```
for(int i=0; i<=2; i++)
    {
        mas2[i] = (i+1) * 10;
    }
```

Понятно, что если бы массив у нас был не из 3, а из 100 элементов, до без цикла мы бы просто не справились.

Длину любого созданного массива не обязательно запоминать, потому что имеется свойство, которое его хранит. Обратиться к этому свойству можно дописав **`.length`** к имени массива. Например:

```
int razmer = mas1.length;
```

Это свойство нельзя изменять (т.е. ему нельзя ничего присваивать), можно только читать. Используя это свойство можно писать программный код для обработки массива даже не зная его конкретного размера.

Например, так можно вывести на экран элементы любого массива с именем `ar2`:

```
for(int i = 0; i <= ar2.length - 1; i++)
    {
        System.out.print(ar2[i] + " ");
    }
```

Для краткости удобнее менять нестрогое неравенство на строгое, тогда не нужно будет вычитать единицу из размера массива. Давайте заполним массив целыми числами от 0 до 9 и выведем его на экран:

```
for(int i = 0; i < ar1.length; i++)
    {
        ar1[i] = Math.floor(Math.random() * 10);
        System.out.print(ar1[i] + " ");
    }
```

Обратите внимание, на каждом шаге цикла мы сначала отправляли случайное значение в элемент массива с i-ым индексом, а потом этот же элемент выводили на экран. Но два процесса (наполнения и вывода) можно было проделать и в разных циклах. Например:

```
for(int i = 0; i < ar1.length; i++)
    {
        ar1[i] = Math.floor(Math.random()* 9);
    }
for(int i = 0; i < ar1.length; i++)
    {
        System.out.print(ar1[i] + " ");
    }
```

В данном случае более рационален первый способ (один проход по массиву вместо двух), но не всегда возможно выполнить требуемые действия в одном цикле.

Для обработки массивов всегда используются циклы типа «n раз» (for) потому, что нам заранее известно сколько раз должен повториться цикл (столько же раз, сколько элементов в массиве).

Java жестко контролирует выход за пределы массива. При попытке обратиться к несуществующему элементу массива возникает `IndexOutOfBoundException`.

В Java есть как одномерные, так и многомерные массивы.

Массив может состоять не только из элементов какого-то встроенного типа (int, double и пр.), но и, в том числе, из объектов какого-то существующего класса и даже из других массивов.

Массив который в качестве своих элементов содержит другие массивы называется **многомерным массивом**.

Чаще всего используются двумерные массивы. Такие массивы можно легко представить в виде матрицы. Каждая строка которой является обычным одномерным массивом, а объединение всех строк — двумерным массивом в каждом элементе которого хранится ссылка на какую-то строку матрицы.

Трёхмерный массив можно представить себе как набор матриц, каждую из которых мы записали на библиотечной карточке. Тогда чтобы добраться до конкретного числа сначала нужно указать номер карточки (первый индекс трёхмерного массива), потому указать номер строки (второй индекс массива) и только затем номер элемент в строке (третий индекс).

Соответственно, для того, чтобы обратиться к элементу n-мерного массива нужно указать n индексов.

Многомерные массивы строятся по принципу "массив массивов". Массив является объектом. Двумерный массив − это массив ссылок на объекты-массивы. Трехмерный массив − это массив ссылок на массивы, которые, в свою очередь, являются массивами ссылок на массивы.

Создаются многомерные массивы в Java аналогичным способом. Количество квадратных скобок указывает на размерность.

```
int[] d1; //Обычный, одномерный
int[][] d2; //Двумерный
double[][] d3; //Трёхмерный
int[][][][][] d5; //Пятимерный
```

Вариант 1. (явное создание)

```
int ary[][] = new int[3][3];
```

Вариант 2. (использование списка инициализации)

```
int ary[][] = new int[][] {
                           {1, 1, 1},
                           {2, 2, 2},
                           {1, 2, 3},
                           };
```

Внимание: в варианте 1 массив создан, но его элементы имеют неопределенное значение. Если попытаться их использовать, возникнет `Exception`.

При создании массива можно указать явно размер каждого его уровня:

```
d2 = int[3][4]; // Матрица из 3 строк и 4 столбцов
```

Но можно указать только размер первого уровня:

```
int[][] dd2 = int[5][];
/* Матрица из 5 строк. Сколько элементов будет в каждой строке пока не известно. */
```

Для обработки двумерных массивов используются два вложенных друг в друга цикла с разными счётчиками.

Пример (заполняем двумерный массив случайными числами от 0 до 9 и выводим его на жкран в виде матрицы):

```
int[][] da = new int[6][4];
for(int i=0; i<da.length; i++)
    {
      for(int j=0; j<da[i].length; j++)
        {
            da[i][j] = (int)(Math.random()*10);
        }
    }
for(int i=0; i<da.length; i++)
    {
      for(int j=0; j<da[i].length; j++)
        {
            System.out.print(da[i][j] + "\t");
        }
      System.out.println();    //    Переходим    на
следующую строку
    }
```

Глядя на примеры со списком инициализации, можно задаться вопросом, что будет, если мы в одних строках зададим одно количество элементов, а в других — другое. Например.

```
int ary[][] = new int[][] {
                            {1, 1, 1, 1},
                            {2, 2, 2},
                            {1, 2, 3, 4, 5},
                            };
```

В Java такое допустимо и именно потому, что многомерный массив является массивом ссылок на массивы. Т.е. каждый массив следующего уровня является самостоятельным массивом и может иметь свой размер. Причем, создание таких "непрямоугольных" массивов возможно не только с использованием списка инициализации, но и явно.

Однако, не обязательно изначально указывать размер на всех уровнях, можно указать размер только на первом уровне.

```
int ary[][] = new int[3][];
ary[0] = new int[5];
ary[1] = new int[2];
ary[2] = new int[6];
```

- Увлекаться такими "непрямоугольными" массивами не стоит. На практике очень редко встречаются задачи, в которых подобные возможности могут потребоваться. Но знать о них нужно. Хотя бы для того, чтобы понять смысл ошибки, возникшей в результате непреднамеренного создания подобного массива.

Обычно всё же используются двумерные массивы с равным количеством элементов в каждой строке.

Пример.

```
int[][] a1 = new int[5][];// двумерный массив с 5 строками
```

В данном случае, пока неизвестно сколько будет элементов в каждой строке, это можно определить позже, причем, массив может содержать в каждой строке разное количество элементов, то есть быть **несимметричным**. Определим количество элементов в каждой строке для массива `a1`

```
a1[0] = new int [1];
a1[1] = new int [2];
a1[2] = new int [3];
a1[3] = new int [4];
a1[4] = new int [5];
```

В результате, при выводе на экран,

```
for(int i = 0; i<a1.length; i++){
    for(int j = 0; j<a1[i].length; j++){
        System.out.print(a1[i][j] + " ");
    }
    System.out.println();
}
```

массив будет иметь такой вид:

0
0 0
0 0 0
0 0 0 0
0 0 0 0 0

При создании массива его элементы автоматически инициализируются нулями, поэтому в это примере на экран выведены нули.

Мы могли создать массив явно указав его элементы. Например так:

```
int[][] ddd2 = {{1,2}, {1,2,3,4,5}, {1,2,3}};
```

При этом можно обратиться к элементу с индексом 4 во второй строке **ddd2[1][4]**, но если мы обратимся к элементу **ddd2[0][4**] или **ddd2[2][4]** — произойдёт ошибка, поскольку таких элементов просто нет. Притом ошибка это будет происходить уже во время исполнения программы (т. е. компилятор её не увидит).