Массивы

Необходимость массивов

Давайте представим, что вы преподаватель. В конце года вам нужно обработать оценки 100 студентов

Как можно записать? Ну пусть так:

```
int gradeStudent1 = 5;
int gradeStudent2 = 4;
// ...
int gradeStudent100 = 5;
```

Круто? Нет, не круто… А почему?

1. ОООООЧень много дублирования кода
2. А что если кол-во студентов изменится/в системе у одного студента отобразилась неверная оценка
3. А как, например, найти средний балл студентов?

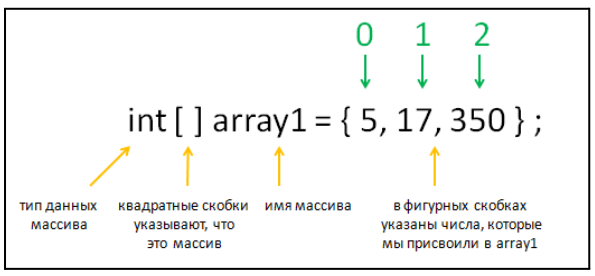
Все это не ахти, поэтому умные люди придумали массивы!

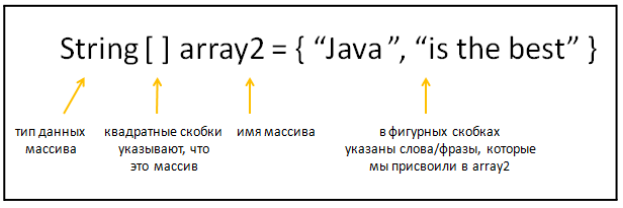
```
int[] grades = new int[100]; // мы создали одну переменную для 100 оценок
```

Вот это уже имбушечка!)

Что вообще такое массив?

Массив — это структура данных, которая хранит упорядоченную коллекцию элементов одного типа





Создание массива

Вообще создание массива состоит из двух шагов:

1. Объявляем переменную

```
int[] myArray; // тут мы типа говорим джавке “создай переменную с именем myArray, которая будет массивом целых чисел”
```

Но если честно, то массив это ссылочный тип данных, поэтому myArray пока что у нас null (аналог None в Python), но в будущем мы дадим ему ссылку, через которую он будет ссылаться на массив

2. Само создание массива (на самом деле с точки зрения внутрянки это выделение памяти на этот массив)

```
myArray = new int[10]; // myArray теперь ссылается на массив из 10 целых чисел
```

Оператор **new** выделяет в памяти непрерывный блок ячеек для хранения нужного количества элементов. Возвращается адрес этого блока – как раз ссылка, которую мы и присваиваем нашей переменной.

Но чаще всего, если иного не требуют условия, эти два шага можно объединить:

```
int[] myArray = new int[10];
```

Хранение массива в памяти

Давайте чуть глубже копнем, чтобы лучше шарить за массивы, и поймем, что это вообще за ссылка и что вообще происходит…

- Массив – это **объект**
- Когда мы пишем new int[10], в куче JVM выделяется непрерывный блок памяти размером 10 * sizeof(int)

// Куча (Heap) JVM - это область памяти, которая создается при запуске нашего приложения. Там хранятся все наши прекрасные объекты

- Переменная myArray хранит не сами данные, а ссылку-указатель на этот блок памяти в куче
- Элементы массива изначально инициализируются значениями по умолчанию:
  - int / byte / short / long -> 0
  - double / float -> 0.0
  - boolean -> false
  - char -> '\u0000' (пустой символ)
  - Для ссылок на объекты (включая String) -> null

!ИМЕННО ПОЭТОМУ НЕВЕРНО ГОВОРИТЬ, ЧТО МАССИВ ИЗНАЧАЛЬНО ПУСТОЙ!

В нашем случае с myArray после тех операций будет что-то типа этого:

```
myArray -> [0][0][0][0][0][0][0][0][0][0]
```

Обращение к элементу массива

Доступ к элементу осуществляется всегда по его **индексу. Индекс** - это порядковый номер конкретного элемента

Важно! *Мы не забываем, что индексация начинается с 0 (нуля), а не с 1 (единички) => первый элемент массива имеет индекс 0, последний - *(длина массива - 1)**

Пример обращения к элементу массива:

```
myArray[0] = 5;    // записываем 5 в первую ячейку
myArray[1] = 10;   // записываем 10 во вторую ячейку
int x = myArray[0]; // читаем значение из первой ячейки, x = 5


System.out.println(myArray[1]); // выведет 10 в консоль
```

---

*ЕСЛИ ВДРУГ ВЫ КОМПИЛИРУЕТЕ И У ВАС ВЫЛЕТАЕТ ИСКЛЮЧЕНИЕ **“ArrayIndexOutOfBoundsException”**, ТО ЭТО ЗНАЧИТ, ЧТО ВЫ НАХУЛИГАНИЛИ С ИНДЕКСАМИ И В КАКОЙ-ТО МОМЕНТ ОБРАЩАЕТЕСЬ К ИНДЕКСУ В МАССИВЕ, КОТОРОГО ВООБЩЕ НЕ СУЩЕСТВУЕТ*

Обход массива

Для обработки всех элементов массива используются циклы (то есть while или for). Из нюансов - в while не забывает самостоятельно увеличивать счетчик. В for мы это изначально прописали в блоке и он сам там разберется

- Цикл **for** (честно говоря, самый частый и самый контролируемый способ)

```
for (int i = 0; i < myArray.length; i++) {
    System.out.println("Элемент №" + i + ": " + myArray[i]);
}
```

---

*!Тут обратите внимание на использование **length** для определения размера массива*

- Цикл **for-each** (это, можно сказать, улучшенный for, но есть нюанс!)

```
for (int element : myArray) {
    System.out.println(element);
}
```

Можно увидеть, что тут нет индекса, соответственно, конкретный элемент мы ну никак изменить не сможем (element — это копия значения, а не оно само) => данный способ удобен, например, для чтения и вообще тогда, когда индекс нам не нужен (например, вывод в консоль, подсчет статистики и прочего)

- Цикл **while / do-while** (используется реже, но почему бы и да)

```
int i = 0;
while (i < myArray.length) {
    System.out.println(myArray[i]);
    i++;
}
```

Многомерные массивы

Многомерный массив – это массивы массивов… Да уж… Вдох-выдох…

Чаще всего это двумерные массивы (с точки зрения математики – матрицы) – массив массивов. Вы будете чаще всего в будущем работать с ними

- Объявление + создание матрицы

```
int[][] matrix = new int[3][4]; // матрица 3x4 (3 строки, 4 столбца)
```

В памяти это массив из 3-х ссылок, каждая из которых ссылается на массив из 4-х целых чисел

- Обращение к элементу

```
matrix[0][0] = 1; // верхний левый угол
matrix[2][3] = 9; // нижний правый угол
```

- Обход по матрице

```
for (int i = 0; i < matrix.length; i++) {       // loop по строкам
     for (int j = 0; j < matrix[i].length; j++) { // loop по столбцам i-ой строки
         System.out.print(matrix[i][j] + " ");
    }
    System.out.println();
}
```

Ступенчатые массивы

Ступенчатые массивы – это частный случай многомерного массива, где вложенные массивы могут быть разной длины

```
// создаем массив для хранения ссылок на подмассивы
int[][] jaggedArray = new int[3][]; // указываем только количество строк
// создаем каждую строку отдельно, с нужной длиной
jaggedArray[0] = new int[5]; // в первой строке 5 элементов
jaggedArray[1] = new int[2]; // во второй - 2
jaggedArray[2] = new int[3]; // в третьей - 3
// инициализация
jaggedArray[0][4] = 55;
jaggedArray[1][1] = 77;
```

В целом мы с ними работаем точно также :)