1.6. Массивы, часть 2

> Привет!
> На связи шпаргалка урока, в которой мы продолжим изучение темы массивов и разберемся, как массивы сравнивать, какие ошибки бывают при работе с массивами и как их исправлять, а также как печатать и заполнять массив.
>
> **Время прочтения:** 15 минут.

Работа с массивами. Циклы

Так как массивы всегда упорядочены и хранят значение в ячейках от 0 до length − 1, мы можем работать с ними через цикл for.

Вот несколько самых популярных задач с массивами, что решаются через циклы.

Заполняем массив в цикле

Мы создали массив на 10 элементов.

Нам нужно его полностью заполнить определенными значениями.

Допустим, числами от 1 до 10.

Если бы мы отталкивались от тех данных, что нам уже известны, нам бы пришлось написать десяток строк подобного типа:

```
arr[0] = 1;
arr[1] = 2;
```

Это, конечно, задачу решает, но об удобстве можно забыть.

Но так как мы знаем циклы и умеем ими пользоваться, решение подобной задачи будет для нас достаточно простым.

```
for (int index = 0; index < arr.length; index++) {
          arr[index] = index + 1;
}
```

Разберемся с кодом выше.

В нем мы объявляем цикл for и создаем переменную index, которая будет увеличиваться от 0 до 9 включительно. Эти значения как раз соответствуют индексам (номерам) ячеек нашего массива.

Далее мы ячейкам массива arr с помощью созданной переменной index присваиваем значение, которое равно index + 1, так как index идет по значениям от 0 до 9, а нам нужно заполнить массив значениями от 1 до 10.

Но как нам проверить, что всё работает корректно?

Печать массивов

Чтобы напечатать массив, нам нужно снова пройти по всем элементам с помощью цикла.

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

for (int index = 0; index < arr.length; index++) {
  if (index == arr.length - 1) {
    System.out.println(arr[index]);
    break;
  }
```

```
  System.out.print(arr[index] + “ ”);
}
```

Разбираемся с кодом.

По прошлому примеру мы создали цикл, который пройдет по всем ячейкам массива с помощью увеличивающейся на каждом шаге переменной index.

Далее разберем последнюю строку в цикле.

В ней мы просто печатаем текущий элемент и пробел.

Так как вызывается метод print, все элементы будут напечатаны в одну строку.

Теперь вернемся к блоку if.

В нем мы проверяем, что индекс на данном шаге равен последнему элементу, и, если так, печатаем последний элемент (без пробела) в ту же строку, где уже напечатаны все предыдущие элементы, а затем делаем перевод на следующую строку.

С помощью оператора break мы еще до окончания шага прерываем цикл, не давая вызвать последнюю строку с печатью элемента и пробела.

Равенство массивов

Так как массивы являются объектами, проверить их на равенство мы не можем через оператор равенства (==).

О причинах вы узнаете в дальнейшем.

Сейчас же нам требуется узнать, как мы можем определить, что массивы являются идентичными друг другу.

Это делается с помощью цикла for:

```
int[] arr = new int[3]; // Объявляем массив на 3 элемента, заполняем значениями 1, 2 и 3
arr[0] = 1;
arr[1] = 2;
arr[2] = 3;
int[] arr2 = new int[3]; // Объявляем второй массив на 3 элемента, заполняем значениями 1, 2 и 3
arr2[0] = 1;
arr2[1] = 2;
arr2[2] = 3;
boolean arraysEqual = true; // Объявляем флаг, который хранит в себе результат равенства массивов. Изначально считаем, что массивы равны, п
if (arr.length != arr2.length) {
  arraysEqual = false;
} else {
for (int i = 0; i < arr.length; i++) {
    if (arr[i] != arr2[i]) {
      arraysEqual = false;
      break;
    }
}
}
System.out.print(“Массивы ”);
if (!arraysEqual) {
  System.out.print(“ не ”);
}
System.out.println(“равны”);
```

Разбираем код.

Мы создали два массива, которым сразу присвоили значения 1, 2 и 3.

Затем создали boolean-переменную arraysEqual, куда присвоили значение true.

Т. е. мы изначально считаем, что массивы равны.

Далее проверяем, что длина массивов соответствует.

Ведь не имеет никакого смысла сравнивать массивы по элементам, если их длина различна. Массивы в этом случае равными быть не могут.

Далее запускаем цикл по индексам первого массива (они должны соответствовать индексам второго массива, если массивы действительно равны) и сравниваем значения из двух массивов, которые лежат по одному индексу.

В случае различия присваиваем нашему флагу значение false.

Если после выполнения этого кода arraysEqual остается true, значит массивы равны.

Это и будет напечатано в консоль.

Если же arraysEqual равен false, то сработает блок if и будет напечатано, что массивы не равны.

Цикл foreach

Так как бывают случаи, когда нам нужно выполнить какое-то действие для каждого элемента массива (например, напечатать), индексы нам, скорее всего, не требуются.

Следовательно, цикл for является в данном случае избыточным, ведь он заставляет нас работать с некой переменной, особого смысла в которой нет.

Именно для этого был придуман способ записи цикла for, который позволяет нам проходить абсолютно по всем элементам массива без необходимости работать с индексами ячеек.

Выглядит цикл следующим образом:

```
for (тип_переменной имя : источник) {
 // действия с переменной, которая создана в первом блоке.

}
```

Возвращаемся к задаче с печатью массива.

В ней нам совершенно не требуются индексы элементов, так как мы работаем со всеми элементами массива, никак не меняя сам массив и значения, лежащие в его ячейках.

```
int[] arr = new int[3];
arr[0] = 1;
arr[1] = 2;
arr[2] = 3;

for (int i : arr) {
  System.out.print(i + " ");
}
System.out.println();
```

Разберем код.

Мы создали массив типа int, затем положили в него значения 1, 2 и 3.

Далее мы объявили цикл foreach.

В цикле, в первом блоке, была объявлена переменная i, в которую на каждом шаге цикла будет помещено следующее значение из массива, указанного во втором блоке.

В самом блоке кода, что выполняется на каждом шагу, мы можем делать с переменной i всё, что нам необходимо: суммировать, печатать и т. д.

Цикл выполнится ровно столько раз, сколько ячеек содержит массив arr.

Переменная i будет хранить на каждом шаге значение из следующей ячейки, начиная от 0-й ячейки на первом шаге и заканчивая 9-й ячейкой на 10-м шаге.

Считаем сумму элементов в массиве

Представим, что перед нами стоит задача подсчитать затраты за месяц в сумме.

Мы имеем массив на 30 элементов, где каждая ячейка хранит траты за день.

Ячейки будут иметь индексы от 0 до 29 включительно.

```
int[] costs = new int[30];
int sum = 0;

// Вариант решения с for
```

```
for (int index = 0; index < costs.length; index++) {
  sum += arr[index];
}

// Вариант решения с foreach
for (int element: arr) {
  sum += element;
}
```

Как мы видим, foreach выглядит проще, короче и красивее.

Ошибки при работе с массивами

Основной ошибкой при работе с массивами является, как ни странно, невнимательность.

Вследствие этой неприятной штуки мы временами вылезаем за границы массива.

В этом случае Java выбрасывает нам в консоль: ArrayIndexOutOfBoundsException.

Например. Напишем следующий код:

```
int[] arr = new int[1];
arr[1] = 3;
```

В результате выполнения этого кода у нас в консоль будет выведена следующая ошибка:

> Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1
> at Main.main(Main.java:5)

Это происходит потому, что массив arr имеет только 1 элемент по номеру ячейки 0.

При обращении к ячейке с номером 1, т. е. отсутствующей ячейке массива, мы выходим за его пределы, и Java нам об этом говорит в виде ошибки выше.

Работа со stacktrace

Разберем, что мы можем вынести из ошибки, которая представлена выше.

В консоли она выглядит следующим образом:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException Create breakpoint : Index 1 out of bounds for length 1
    at Main.main(Main.java:5)
```

Первым делом следует обратить внимание на текст, который находится сразу после Exception in thread “main”. А именно:

> java.lang.ArrayIndexOutOfBoundsException

Первые два слова, что указаны с маленькой буквы через точку, являются пакетом, где хранится класс, имя которого начинается с большой буквы. На скрине выше он отмечен серым.

Вы могли сталкиваться с пакетами при выполнении домашнего задания.

Они указаны в вашем файле java в самой верхней строке, после слова package.

Т. е. ваш класс является по такому же принципу не просто Main.java, а pro.sky.java.course1.lesson3.Main. Это называется полное имя класса.

Уникальность классов в приложении работает как раз за счет полного имени, вот поэтому вы можете в каждой папке создать файлы с одинаковыми именами.

Сам ArrayIndexOutOfBoundsException является классом из стандартной библиотеки Java (аналогом вашего класса Main, который вы создаете при выполнении домашнего задания), который выкидывает Java в случае выхода за пределы массивов в любой части вашего приложения.

Далее идет текст ошибки. В данном случае мы видим текст: Index 1 out of bounds for length 1. Как мы понимаем из текста, в коде был вызов элемента по индексу 1 для массива, длина которого 1, т. е. индекс находится за пределами массива.

Схожую ошибку мы можем наблюдать, если подадим в качестве индекса отрицательное число.

Затем идет адрес с указанием класса, метода и строки, которая эту ошибку выкинула.

Как мы видим, ошибка возникла в классе Main, его методе main, строка 5. А затем, в скобках, находится ссылка на строку с ошибкой (выделена синим). При нажатии на эту ссылку вы перейдете на ту строку кода, что спровоцировала выброс данной ошибки.

Более подробно с ошибками вы познакомитесь на следующем курсе.

Стандартные значения типов

Массив при создании формирует не просто ячейки типов, но еще и заполняет их стандартными значениями.

Это обусловлено тем, что нельзя создать ячейку типа, не поместив в нее некое значение этого типа.

Для этого в Java были введены значения типов «по умолчанию».

Таблица значений переменных по умолчанию (1)

Aa Тип переменной	Значение по умолчанию
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	0
String (или любой другой объект)	null
boolean	false

Это означает, что создавая массив типа int на 2 элемента, получаем массив на две ячейки, которые заполнены нулями, так как нуль является стандартным значением для типа int.

Получение любого элемента массива вернет нам именно нуль.

Например:

```
int[] arr = new int[2];
int i = arr[0];
```

В данном случае переменная i получит в себя значение 0.

Класс Arrays

У вас мог возникнуть вполне рациональный вопрос о том, почему работа с массивами настолько неудобна, что для достаточно простых операций приходится самостоятельно писать циклы.

У разработчиков Java тоже в свое время возник такой вопрос.

Так появился utility-класс Arrays, который содержит методы для работы с массивами.

Мы с вами пока не изучали структуры методов и то, как они работают, потому просто взглянем на методы класса Arrays и то, как ими пользоваться.

Большее понимание придет в дальнейшем.

Печатаем массив

Как мы помним, печать массива требует написания цикла.

И эта ситуация не меняется, если нам нужно вывести массив в том виде, который нам нравится.

Однако если формат вывода не принципиален, мы можем воспользоваться методом класса Arrays, который преобразует наш массив в его строчное представление и позволяет нам его напечатать без необходимости дополнительного написания циклов.

Например:

```
int[] arr = new int[2];
arr[0] = 1;
arr[1] = 2;

System.out.println(Arrays.toString(arr));
```

На выходе получим: [1, 2]

Также результат выполнения метода toString класса Arrays мы можем присвоить строковой переменной (как с ними работать узнаете на следующем уроке).

```
String arrString = Arrays.toString(arr);
```

Заполняем массив

Если нам необходимо заполнить массив одинаковыми значениями, мы можем избежать написания цикла.

Например, чтобы заполнить наш целочисленный массив единицами, нужно написать следующий код:

```
int[] arr = new int[10];
Arrays.fill(arr, 1); // Первым параметром передаем массив, вторым — значение, которым его нужно заполнить
```

Сравниваем массивы

Для сравнения двух массивов мы также можем использовать метод из класса Arrays.

Это позволяет нам сравнить стандартным способом (что мы писали выше в виде цикла) два массива.

Сразу стоит уточнить, что данный способ должен использоваться только в случае необходимости стандартного сравнения.

Если у вас есть какой-то свой алгоритм по сравнению, то следует писать собственный алгоритм через цикл.

В стандартном случае сравнение выглядит следующим образом:

```
int[] arr = new int[2];
arr[0] = 1;
arr[1] = 2;

int[] arr2 = new int[2];
arr2[0] = 1;
arr2[1] = 2;

System.out.println(Arrays.equals(arr, arr2));
```

В консоли печатается результат сравнения в формате boolean-значения, а именно true.

Также результат сравнения можно присвоить boolean-переменной.

```
boolean arraysEqual = Arrays.equals(arr, arr2);
```

Копирование массива

Чтобы создать копию уже существующего массива, можно использовать следующий метод:

```
int[] arr = new int[2];
arr[0] = 1;
arr[1] = 2;
int[] arr2 = Arrays.copyOf(arr, arr.length);
// Первым параметром идет сам массив, вторым — количество элементов,
// которые нужно скопировать в новый массив
```

Сортировка массива

Сортировка массива тоже бывает необходима. С помощью метода класса Arrays массив сортируется так же просто, как печатается или копируется.

```
int[] arr = new int[2];
arr[0] = 2;
arr[1] = 1;
Arrays.sort(arr);
System.out.println(Arrays.toString(arr));
```

В консоли напечатается уже отсортированная версия массива: [1, 2]

Дополнительная информация

▼ **Альтернативные способы инициализации массивов**

Если массив достаточно короткий, мы можем заполнить его значениями прямо в месте инициализации. Это помогает избежать написания строк, количество которых будет равно количеству необходимых элементов.

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

Это не значит, что будет создан массив, в котором мы можем менять количество значений. Размер массива будет вычислен на основе количества значений из {}. В данном примере будет создан массив с размером 3.

Еще более простой способ объявить массив:

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

▼ **Массивы в памяти**

Если у вас возник вопрос, почему массив не может расширяться, то давайте взглянем поглубже на способ его хранения в памяти.

Массив не может расширяться за счет того, что ячейки массива в памяти должны быть расположены по порядку.

Это делается для того, чтобы Java не хранила постоянно ссылки на каждую из ячеек, которые могут быть разбросаны в любой части памяти вашего приложения.

Т. к. если массив имеет миллион элементов, Java бы пришлось хранить в себе ссылки на весь миллион элементов. Это не выглядит хорошим решением. Потому предложили другое.

Было придумано, что есть одна ячейка со служебной информацией о массиве, где хранятся его длина, тип и ссылка на первый элемент, а каждый элемент отличный от первого находится путем обычной прибавки к индексу текущего элемента числа n, являющегося номером необходимой ячейки массива.

При необходимости расширить массив уже после его создания можно, но Java не может гарантировать, что следующие ячейки в памяти не заняты. Именно поэтому расширение массива невозможно.

▼ **Двумерные массивы (массивы массивов)**

Это дополнительная информация для общего понимания. Если она вам пока не совсем ясна, ничего страшного. В дальнейшем, немного поработав с массивами, вы ее усвоите.

Массивы могут хранить даже другие массивы.

Представим, что нам необходимо нарисовать матрицу 3x3.

Например, чтобы играть в крестики-нолики.

В этой ситуации нам потребуется создать массив массивов (char[][]).

Каждый внутренний массив будет строчкой, каждый его элемент будет колонкой.

Внешний массив же будет осуществлять контроль за этими строчками.

Т. е. поле должно выглядеть таким образом:

0. x x x // где 0 — номер ячейки внешнего массива
1. x x x // а все x — ячейки внутреннего массива
2. x x x // (следовательно, нам необходимо создать char[3][3])

Представим, что каждая ячейка массива является строкой, а каждая ячейка подмассива является клеточкой для крестика или нолика.

Чтобы создать такой массив, нам нужно написать следующее:

```
char[][] gameField = new char[3][3];
```

В первых скобках мы указываем размер внутренних массивов, а во вторых их количество (т. е. размер внешнего массива).

Выглядит сложно, но если разобрать эту строку, всё станет понятно.

Как мы помним, при создании массива пишется тип, а затем к нему добавляются []. Это показывает нам, что это не просто переменная данного типа, а именно массив данных этого типа.

В этой строке всё работает так же, только вместо маленького типа (как int) мы создаем массив данных типа char[], т. е. других массивов (подмассивов основного массива) типа char.

И теперь, если мы напечатаем каждый из массивов, воспринимая каждый внешний элемент массива как строку, а каждый элемент внутреннего массива как столбик, мы увидим, что печатается поле 3x3.

Это можно сделать следующим кодом:

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

Шпаргалка урока в PDF-формате:

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/fb20c22c-ea7c-4a4e-809f-5aa9ed1ec89c/1.6._-2.pdf