Практика

Массивы, разбор практических примеров использования базовых элементов языка Java, работа с консолью

[Массивы](#_17dp8vu)

[Одномерные массивы](#_lrb398yue5sp)

[Двумерные массивы](#_sb7sgau2l6se)

[Нерегулярные массивы](#_pxqxm8o7b60r)

[Многомерные массивы](#_864eah3vuomf)

[Альтернативный синтаксис объявления массивов](#_brxndb2ylqbo)

[Получение длины массива](#_d9ek3br2qlzg)

[Ввод данных из консоли](#_svy5ibwww1y0)

[Полезные примеры](#_3dy6vkm)

[Так делать нельзя](#_tv70etlyje8s)

[Домашнее задание](#_nqaqj8crzsre)

[Дополнительные материалы](#_3rdcrjn)

[Используемая литература](#_2iapxbubv9y2)

[Подсказки по домашнему заданию](#_44sinio)

# Массивы

Массив представляет собой набор однотипных переменных с общим именем.

## Одномерные массивы

Для объявления одномерного массива обычно применяется следующая форма.

| тип\_данных[] имя\_массива = new тип\_данных[размер\_массива]; |
| --- |

При создании массива сначала объявляется переменная, ссылающаяся на него. Затем выделяется память для массива, в Java динамически распределяется с помощью оператора new; ссылка на неё присваивается переменной. В следующей строке кода создается массив типа int, состоящий из 5 элементов, ссылка на него присваивается переменной arr.

| int[] arr = new int[5]; |
| --- |

В переменной arr сохраняется ссылка на область памяти для массива оператором new. Этой памяти должно быть достаточно для размещения в ней 5 элементов типа int. Доступ к отдельным элементам массива осуществляется с помощью индексов. Индекс обозначает положение элемента в массиве, индекс первого элемента равен нулю. Если массив arr содержит 5 элементов, их индексы находятся в пределах от 0 до 4. Индексирование массива осуществляется по номерам его элементов, заключенным в квадратные скобки. Например, для доступа к первому элементу массива arr следует указать arr[0], а для доступа к последнему элементу этого массива — arr[4]. В приведенном ниже примере программы в массиве arr сохраняются числа от 0 до 4.

| public static void main(String args[]) {  int[] arr = new int[5];  for(int i = 0; i < 5; i++) {  arr[i] = i;  System.*out*.println("arr[" + i + "] = " + arr[i]);  } }  **Результат:**  arr[0] = 0  arr[1] = 1  arr[2] = 2  arr[3] = 3  arr[4] = 4 |
| --- |

| **arr[0]** | **arr[1]** | **arr[2]** | **arr[3]** | **arr[4]** |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |

Заполнять созданные массивы можно последовательным набором операторов.

| public static void main(String args[]) {  int[] nums = new int[4];  nums[0] = 5;  nums[1] = 10;  nums[2] = 15;  nums[3] = 15; } |
| --- |

В приведённом выше примере массив nums заполняется через четыре оператора присваивания. Существует более простой способ решения этой задачи: заполнить массив сразу при его создании.

| тип\_данных[] имя\_массива = {v1, v2, v3, ..., vN} ; |
| --- |

Здесь v1-vN обозначают первоначальные значения, которые присваиваются элементам массива слева направо по порядку индексирования, при этом Java автоматически выделит достаточный объем памяти. Например.

| public static void main(String args[]) {  int[] nums = { 5, 10, 15, 20 }; } |
| --- |

Границы массива в Java строго соблюдаются. Если обратиться к несуществующему элементу массива, будет получена ошибка. Пример.

| public static void main(String args[]) {  int[] arr = new int[10];  for(int i = 0; i < 20; i++) {  arr[i] = i;  } } |
| --- |

Как только значение переменной i достигнет 10, будет сгенерировано исключение ArraylndexOutOfBoundsException и выполнение программы прекратится.

Распечатать одномерный массив в консоль можно с помощью конструкции Arrays.toString().

| import java.util.Arrays;  public class MainClass {  public static void main(String args[]) {  String[] arr = {"A", "B", "C", "D"};  System.*out*.println(Arrays.*toString*(arr));  }  }  **Результат:** [A, B, C, D] |
| --- |

## 

## Двумерные массивы

Среди многомерных массивов наиболее простыми являются двумерные. Двумерный массив – это ряд одномерных массивов. При работе с двумерными массивами проще их представлять в виде таблицы, как будет показано ниже. Объявим двумерный целочисленный табличный массив table размером 10x20.

| int[][] table = new int[10][20]; |
| --- |

В следующем примере создадим двумерный массив размером 3х4, заполним его числами от 1 до 12 и отпечатаем в консоль в виде таблицы.

| public static void main(String args[]) {  int counter = 1;  int[][] table = new int[3][4];  for (int i = 0; i < 3; i++) {  for (int j = 0; j < 4; j++) {  table[i][j] = counter;  System.*out*.print(table[i][j] + " ");  counter++;  }  System.*out*.println();  } } |
| --- |

|  | j = 0 | j = 1 | j = 2 | j = 3 |
| --- | --- | --- | --- | --- |
| i = 0 | 1 | 2 | 3 | 4 |
| i = 1 | 5 | 6 | 7 | 8 |
| i = 2 | 9 | 10 | 11 | 12 |

При работе с отладкой и двумерными массивами для их распечатки можно пользоваться следующим методом. На вход метода необходимо подать ссылку на любой двумерный целочисленный массив. Первый индекс массива указывает на строку, второй – на столбец.

| public static void printArr(int[][] arr) {  for (int i = 0; i < arr.length; i++) {  for (int j = 0; j < arr[i].length; j++) {  System.*out*.print(arr[i][j]);  }  System.*out*.println();  } } |
| --- |

## 

## Нерегулярные массивы

Выделяя память под многомерный массив, достаточно указать лишь первый (крайний слева) размер. Память под остальные размеры массива можно выделять по отдельности.

| int[][] table = new int[3][]; table[0] = new int[1]; table[1] = new int[5]; table[2] = new int[3]; |
| --- |

Поскольку многомерный массив является массивом массивов, существует возможность установить разную длину массива по каждому индексу. В некоторых случаях такие массивы могут значительно повысить эффективность работы программы и снизить потребление памяти, например, если требуется создать очень большой двумерный массив, в котором используются не все элементы.

## Многомерные массивы

В Java допускаются n-мерные массивы, ниже показана форма объявления.

| тип\_данных[][]...[] имя\_массива = new тип\_данных[размер1][размер2]...[размерN]; |
| --- |

В качестве примера ниже приведено объявление трехмерного целочисленного массива размерами 2x3x4.

| int[][][] mdarr = new int[2][3][4]; |
| --- |

Многомерный массив можно инициализировать. Инициализирующую последовательность нужно заключить в отдельные фигурные скобки.

| тип\_данных[][] имя\_массива = {  { val, val, val, ..., val },  { val, val, val, ..., val },  { val, val, val, ..., val }  }; |
| --- |

## Альтернативный синтаксис объявления массивов

Помимо рассмотренной выше общей формы для объявления массива можно также пользоваться следующей формой.

| тип\_данных имя\_массива[]; |
| --- |

Два следующих объявления массивов равнозначны.

| public static void main(String[] args) {  int arr[] = new int[3];  int[] arr2 = new int[3]; } |
| --- |

## 

## Получение длины массива

При работе с массивами имеется возможность программно узнать его размер. Для этого можно воспользоваться записью *имя\_массива.length*. Это удобно использовать, когда нужно пройти циклом for по всему массиву.

| public static void main(String[] args) {  int[] arr = {2, 4, 5, 1, 2, 3, 4, 5};  System.*out*.println("arr.length: " + arr.length);  for (int i = 0; i < arr.length; i++) {  System.*out*.print(arr[i] + " ");  } }  **Результат:**  arr.length: 8  2 4 5 1 2 3 4 5 |
| --- |

# Ввод данных из консоли

Для ввода данных из консоли можно воспользоваться объектом класса Scanner (вопрос, что такое классы и объекты, будет подробно рассмотрен на 5 занятии).

| public static void main(String[] args) {  Scanner sc = new Scanner(System.in); // создание объекта класса Scanner  int a = sc.nextInt(); // чтение целого числа в переменную a  String b = sc.nextLine(); // чтение введенной строки  String c = sc.next(); // слово до следующего пробела  sc.close(); // после завершения работы со сканером его необходимо закрыть, } |
| --- |

Пример программы, запрашивающей у пользователя ввод целого числа и выводящей в консоль число в 2 раза больше.

| import java.util.Scanner; public class MainClass {  public static void main(String[] args) {  Scanner sc = new Scanner(System.*in*);  System.*out*.println("Введите число: ");  int a = sc.nextInt();  a \*= 2;  System.*out*.println("Введенное вами число, умноженное на 2, равно " + a);  sc.close();  } } |
| --- |

Как же сделать ввод данных в заданных пределах?

| import java.util.Scanner; public class MainClass {  public static Scanner *sc* = new Scanner(System.*in*);   public static void main(String[] args) {  int d = *getNumberFromScanner*("Введите число в пределах от 5 до 10", 5, 10);  System.*out*.println("d = " + d);  }   public static int getNumberFromScanner(String message, int min, int max) {  int x;  do {  System.*out*.println(message);  x = *sc*.nextInt();  } while (x < min || x > max);  return x;  } }  **Результат:**  Введите число в пределах от 5 до 10  8  d = 8 |
| --- |

Метод getNumberFromScanner() будет запрашивать у пользователя целое число до тех пор, пока оно не окажется в пределах от min до max включительно. Перед каждым запросом будет выводится сообщение, которое передано в message. Повторный запрос осуществляется с помощью цикла do/while. Мы будем запрашивать у пользователя ввод числа до тех пор, пока он будет пытаться указать число меньше минимального или больше максимального.

# Полезные примеры

Напишем метод, который принимает в качестве параметра одномерный массив и печатает его в консоль. По завершению печати ставится перенос строки. При необходимости можно вместо пробела поставить любой символ-разделитель.

| public static void print1DArray(int[] arr) {  for (int i = 0; i < arr.length; i++) {  System.*out*.print(arr[i] + " ");  }  System.*out*.println(); } |
| --- |

Печать двумерного прямоугольного массива с нумерацией строк и столбцов.

| public static void print2DArray(int[][] arr) {  for (int i = 0; i <= arr[0].length; i++) {  System.*out*.print(i + " ");  }  System.*out*.println();  for (int i = 0; i < arr.length; i++) {  System.*out*.print(i + 1 + " ");  for (int j = 0; j < arr[i].length; j++) {  System.*out*.print(arr[i][j] + " ");  }  System.*out*.println();  } } |
| --- |

Первый цикл отвечает за печать шапки таблицы. После него стоит оператор System.out.println() для перевода строки. После этого открывается двойной цикл для печати самого массива, i отвечает за номер строки, j за номер столбца. Сам же цикл j отвечает за печать элементов массива. Перед печатью строки массива прописываем номер этой строки System.*out*.print(i + 1 + " ").

Посчитать сумму элементов в массиве можно с помощью следующего кода.

| public static int arrSum(int[] arr) {  int sum = 0;  for (int i = 0; i < arr.length; i++) {  sum += arr[i];  }  return sum; } |
| --- |

Для расчёта суммы вводим временную переменную sum, к ней в цикле будем прибавлять значения элементов массива. Как только пройдём по всем элементам массива, в переменной sum будет находиться сумма всех элементов. По аналогии можно решить задачу подсчета элементов массива, удовлетворяющих какому-либо условию, например, количество чисел 5 в массиве – пробегаем по всему массиву и увеличиваем счетчик, если нашли число 5.

Для формирования случайного числа нужно создать объект класса Random и вызвать у него метод nextInt(n), который возвращает случайное целое число в пределах от 0 до n – 1 включительно. В примере ниже в x могут попасть числа 0, 1, 2, 3, ..., 19.

| public class MainClass {  public static void main(String[] args) {  Random rand = new Random();  intx = rand.nextInt(20);  } } |
| --- |

Можно напечатать текст в консоль с форматированием с помощью метода System.out.printf(). Вначале вводится форматируемая строка с вставками вида %d, %f, %s, %c, на месте которых затем подставляются значения, взятые из аргументов метода.

| public static void main(String[] args) {  System.*out*.printf("Слово: %s, Число с плавающей запятой: %f, Целое число: %d, Символ: %c", "Java", 2.5f, 20, 'e'); }  **Результат:**  Слово: Java, Число с плавающей запятой: 2,500000, Целое число: 20, Символ: e |
| --- |

Сравнение строк должно осуществляться с помощью метода equals(), как показано в примере ниже. Смысл такого сравнения будет пояснен на занятиях по ООП.

| public static void main(String[] args) {  String str1 = "A";  String str2 = "A";  String str3 = "B";  System.*out*.println(str1.equals(str2));  System.*out*.println(str1.equals(str3)); }  **Результат:**  true  false |
| --- |

# Так делать нельзя

В данном разделе перечислены мелкие ошибки, встречающиеся у студентов, начинающих изучать язык Java и программирование в целом.

После закрывающейся круглой скобки в операторах **if** и **for** точку с запятой ставить нельзя:

| public static void main(String[] args) {  int x = 10;  if (x < 20); { *// <- вот тут*  System.out.println(1);  }  for (int i = 0; i < 5; i++); { *// <- и вот тут*  System.out.println(i);  } } |
| --- |

Нельзя объявлять методы внутри методов .

| public static void main(String[] args) {  public static void method2() { *// <-* } } |
| --- |

При вызове метода внутри скобок нельзя объявлять переменные.

| public static void main(String[] args) {  *method*(int z = 5); *// <-* } public static void method(int x) {  System.*out*.println(x); } |
| --- |

В приведённом ниже случае и во многих похожих случаях оператор continue не нужен, цикл и без него перейдет на следующий шаг, после того как дойдет до последней строки тела цикла.

| public static void main(String args[]) {  for (int i = 0; i < 5; i++) {  if (i < 3) {  System.*out*.println("e");  } else continue;  } } |
| --- |

Следите за скобками. Каждая открывающаяся фигурная скобка должна быть закрыта.

| public class MainClass {  public static void main(String[] args) {  // <- тут не хватает закрытой фигурной скобки } |
| --- |

В методе с возвратом не должно быть ситуаций, при которых ни один return не сработает. Если методу подать число x = 20, мы не сможем выйти из него, поэтому такой код даже не скомпилируется.

| public static boolean wrongReturn(int x) {  if (x < 10) {  return true;  } } |
| --- |

# Домашнее задание

1. Задать целочисленный массив, состоящий из элементов 0 и 1. Например: [ 1, 1, 0, 0, 1, 0, 1, 1, 0, 0 ]. С помощью цикла и условия заменить 0 на 1, 1 на 0;
2. Задать пустой целочисленный массив длиной 100. С помощью цикла заполнить его значениями 1 2 3 4 5 6 7 8 … 100;
3. Задать массив [ 1, 5, 3, 2, 11, 4, 5, 2, 4, 8, 9, 1 ] пройти по нему циклом, и числа меньшие 6 умножить на 2;
4. Создать квадратный двумерный целочисленный массив (количество строк и столбцов одинаковое), и с помощью цикла(-ов) заполнить его диагональные элементы единицами (можно только одну из диагоналей, если обе сложно). Определить элементы одной из диагоналей можно по следующему принципу: индексы таких элементов равны, то есть [0][0], [1][1], [2][2], …, [n][n];
5. Написать метод, принимающий на вход два аргумента: **len** и **initialValue**, и возвращающий одномерный массив типа int длиной **len**, каждая ячейка которого равна **initialValue**;
6. \* Задать одномерный массив и найти в нем минимальный и максимальный элементы ;
7. \*\* Написать метод, в который передается не пустой одномерный целочисленный массив, метод должен вернуть true, если в массиве есть место, в котором сумма левой и правой части массива равны.   
   **Примеры**:   
   checkBalance([2, 2, 2, 1, 2, 2, ||| 10, 1]) → true, **т.е. 2 + 2 + 2 + 1 + 2 + 2 = 10 + 1**  
   checkBalance([1, 1, 1, ||| 2, 1]) → true, **т.е. 1 + 1 + 1 = 2 + 1**  
   **граница показана символами |||, эти символы в массив не входят и не имеют никакого отношения к ИЛИ**.
8. \*\*\* Написать метод, которому на вход подается одномерный массив и число n (может быть положительным, или отрицательным), при этом метод должен сместить все элементы массива на n позиций. Элементы смещаются циклично. Для усложнения задачи нельзя пользоваться вспомогательными массивами. Примеры: [ 1, 2, 3 ] при n = 1 (на один вправо) -> [ 3, 1, 2 ]; [ 3, 5, 6, 1] при n = -2 (на два влево) -> [ 6, 1, 3, 5 ]. При каком n в какую сторону сдвиг можете выбирать сами.

*Если выполнение задач вызывает трудности, можете обратиться к последней странице методического пособия. Для задач со \* не нужно искать решение в интернете, иначе нет смысла их выполнять..*

# Дополнительные материалы

1. **Видео: GeekBrains. Массивы. База (дополнительный разбор)**: <https://www.youtube.com/watch?v=0TJFdyZFTKw>

# Используемая литература

1. Брюс Эккель Философия Java // 4-е изд.: Пер. с англ. – СПб.: Питер, 2016. – 1168 с.
2. Г. Шилдт Java 8. Полное руководство // 9-е изд.: Пер. с англ. – М.: Вильямс, 2015. – 1376 с.
3. Г. Шилдт Java 8: Руководство для начинающих. // 6-е изд.: Пер. с англ. – М.: Вильямс, 2015. – 720 с.

# Подсказки по домашнему заданию

1. Вариант 1:  
   public static void invertArray() {  
    int[] arr = { 1, 0, 1, 0, 0, 1 };  
    for (int i = 0; i < arr.length; i++) {  
    // …  
    }   
   }  
   Вариант 2:  
   public static void invertArray() {  
    int[] arr = { 1, 0, 1, 0, 0, 1 };  
    for (int i = 0; i < arr.length; i++) {  
    if (…) {  
    // ...  
    } else {  
    // ...  
    }  
    }   
   }
2. Вариант 1:  
   public static void fillArray() {  
    int[] arr = new int[100];  
    for (int i = 0; i < arr.length; i++) {  
    // ...  
    }  
   }  
   Вариант 2:  
   public static void fillArray() {  
    int[] arr = new int[100];  
    arr[0] = 0;  
    for (int i = 1; i < arr.length; i++) {  
    // ...  
    }  
   }  
   Вариант 3:  
   public static void fillArray() {  
    int[] arr = new int[100];  
    for (int i = 0, ...; i < arr.length; i++, ...) {  
    // ...  
    }  
   }  
   И еще есть несколько вариантов...
3. public static void changeArray() {  
    int[] arr = { 1, 5, 3, 2, 11, 4, 5, 2, 4, 8, 9, 1 };  
    for (int i = 0; i < arr.length; i++) {  
    if (...) {  
    // ...  
    }  
    }  
   }
4. Вариант 1:  
   public static void fillDiagonal() {  
    int[][] arr = new int[4][4];  
    for (int i = 0; i < 4; i++) {  
    // ...  
    }  
   }  
   Вариант 2:  
   public static void fillDiagonal() {  
    int[][] arr = new int[4][4];  
    for (int i = 0; i < 4; i++) {  
    for (int j = 0; j < 4; j++) {  
    // ...  
    }  
    }  
   }

*Вместо … подставляете ваш код. Варианты 1-n означает, что можно выполнить задачу несколькими способами. Представлены не все существующие решения, возможно, вы найдете свое.*