



Урок 2

Основные конструкции

Оператор switch, циклы, кодовые блоки, массивы

[Оператор switch](#)

[Циклы for](#)

[Пример цикла с отрицательным приращением счётчика](#)

[Цикл for с несколькими управляющими переменными](#)

[Бесконечный цикл](#)

[Цикл foreach](#)

[Вложенные циклы](#)

[Циклы while](#)

[Кодовые блоки](#)

[Массивы](#)

[Одномерные массивы](#)

[Двумерные массивы](#)

[Нерегулярные массивы](#)

[Многомерные массивы](#)

[Альтернативный синтаксис объявления массивов](#)

[Получение длины массива](#)

[Домашнее задание](#)

[Дополнительные материалы](#)

[Подсказки по домашнему заданию](#)

Оператор switch

Оператор switch позволяет делать выбор между несколькими вариантами дальнейшего выполнения программы. Выражение последовательно сравнивается с списка значений оператора switch и при совпадении, выполняется набор операторов, связанных с этим условием. Если совпадений не было, выполняется блок default (блок default является необязательной частью оператора switch).

```
switch (выражение) {  
    case значение1:  
        набор_операторов1;  
        break;  
    case значение 2:  
        набор_операторов2;  
        break;  
    ...  
    default:  
        набор_операторов;  
}
```

Например, последовательность if-else-if-...:

```
public static void main(String[] args) {  
    int a = 3;  
    if(a == 1) {  
        System.out.println("a = 1");  
    } else if (a == 3) {  
        System.out.println("a = 3");  
    } else if (a == 5) {  
        System.out.println("a = 5");  
    } else {  
        System.out.println("Ни одно из условий не сработало");  
    }  
}
```

Может быть заменена на:

```
public static void main(String[] args) {  
    int a = 3;  
    switch (a) {  
        case 1:  
            System.out.println("a = 1");  
            break;  
        case 3:  
            System.out.println("a = 3");  
            break;  
        case 5:  
            System.out.println("a = 5");  
            break;  
        default:  
            System.out.println("Ни один из case не сработал");  
    }  
}
```

Циклы for

Циклы позволяют многократно выполнять последовательность кода.

```
for (инициализация; условие; итерация) {  
    набор_операторов;  
}
```

Инициализация представлена переменной, выполняющей роль счётчика и управляющей циклом (например, `int i = 0;`); условие определяет необходимость повторения цикла; итерация задаёт шаг изменения переменной, управляющей циклом. Эти три части **должны быть** разделены точкой с запятой, **заметьте, что после закрывающейся круглой скобки точки с запятой нет**. Выполнение цикла `for` продолжается до тех пор, пока проверка условия даёт истинный результат. Пример:

```
public static void main(String args[]) {  
    for (int i = 0; i < 5; i++) {  
        System.out.println("i = " + i);  
    }  
    System.out.println("end");  
}
```

Результат:

```
i = 0  
i = 1  
i = 2  
i = 3  
i = 4  
end
```

В начале каждого шага цикла проверяется условие `i < 5`. Если это условное выражение верно, вызывается тело цикла (в котором прописан метод `System.out.println(...)`), после чего выполняется итерационная часть цикла. Как только условное выражение примет значение `false`, цикл закончит свою работу.

Пример цикла с отрицательным приращением счётчика

Ниже приведён пример цикла с отрицательным приращением цикла. Еще одной особенностью цикла является «вынос» объявления управляющей переменной до начала цикла, но, как правило, она объявляется внутри `for`.

```
public static void main(String args[]) {  
    int x; // объявление управляющей переменной вынесено до начала цикла  
    for (x = 10; x >= 0; x -= 5) { // Шаг -5  
        System.out.print(x + " ");  
    }  
}
```

Результат:

```
10 5 0
```

Условное выражение цикла `for` всегда проверяется в самом начале цикла. Это означает, что код в цикле может вообще не выполняться, если проверяемое условие с самого начала оказывается ложным. Пример:

```
public static void main(String args[]) {
    int x = 0;
    for (int count = 10; count < 5; count++) {
        x += count; // этот оператор не будет выполнен, так как 10 > 5
    }
}
```

Этот цикл вообще не будет выполняться, поскольку начальное значение переменной `count` больше 5, а значит, что условное выражение `count < 5` оказывается ложным с самого начала.

Цикл `for` с несколькими управляющими переменными

Для управления циклом можно использовать одновременно несколько переменных, в примере ниже за одну итерацию переменная `i` увеличивается на 1, а `j` уменьшается на 1.

```
public static void main(String args[]) {
    for (int i = 0, j = 10; i < j; i++, j--) {
        System.out.println("i-j: " + i + "-" + j);
    }
}
```

Результат:

```
i-j: 0-10
i-j: 1-9
i-j: 2-8
i-j: 3-7
i-j: 4-6
```

Бесконечный цикл

При использовании следующей записи цикла `for`, можно получить бесконечный цикл, хотя большинство таких циклов на самом деле требуют специальное условие для своего завершения.

```
for (;;) {
    ...
}
```

Выход из работающего цикла осуществляется оператором **`break`**, который «моментально» выходит из цикла, не дожидаясь выполнения всего кода из тела цикла, поэтому в результате нет числа 4 (тело цикла не успело выполниться до конца).

```
public static void main(String[] args) {
    for(int i = 0; i < 10; i++) {
        if (i > 3) {
            break;
        }
        System.out.println("i = " + i);
    }
}
```

Результат:

```
i = 0
i = 1
i = 2
i = 3
```

Цикл foreach

Ещё одной разновидностью цикла for является цикл foreach, который в большинстве случаев используется для прохождения по всем элементам массива или коллекции без необходимости знать индекс проверяемого элемента. В приведённом ниже примере мы проходим по элементам массива sm типа String, и каждому такому элементу присваиваем временное имя o, то есть «в единицу времени» o указывает на один элемент массива.

```
public static void main(String[] args) {
    String[] sm = {"A", "B", "C", "D"};
    for (String o : sm) {
        System.out.print(o + " ");
    }
}
```

Результат:

```
A B C D
```

Вложенные циклы

Циклы, работающие внутри других циклов, называют вложенными. Внимательно разберите последовательность исполнения таких циклов, на полное выполнение внутреннего цикла приходится одна итерация внешнего. Пример:

```
public static void main(String args[]) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            System.out.print(" " + i + j);
        }
    }
}
```

Результат:

```
00 01 02 10 11 12 20 21 22
```

Циклы while

Цикл `while` работает до тех пор, пока указанное условие истинно. Как только условие становится ложным, управление программой передается строке кода, следующей непосредственно после цикла. Если заранее указано условие, которое не выполняется, программа в тело цикла даже не попадет.

```
while (условие) {  
    набор_операторов;  
}
```

Цикл `do-while` очень похож на ранее рассмотренные циклы, только в отличие от `for` и `while`, в которых условие проверялось в самом начале (предусловие), в цикле `do-while` условие выполнения проверяется в самом конце (постусловие), это означает, что цикл `do-while` всегда выполняется хотя бы один раз.

```
do {  
    набор_операторов;  
while (условие);
```

Кодовые блоки

Кодовый блок представляет собой группу операторов. Для оформления в виде блока они помещаются между открывающей и закрывающей фигурными скобками. Созданный кодовый блок становится единым логическим блоком. В частности, кодовый блок можно использовать при работе с `if` и `for`. Пример:

```
public static void main(String args[]) { // <- начало кодового блока main  
    int w = 1, h = 2, v = 0;  
    if (w < h) { // <- Начало кодового блока if  
        v = w * h;  
        w = 0;  
    } // <- Конец кодового блока if  
} // <- Конец кодового блока main
```

В данном примере оба оператора в блоке выполняются в том случае, если значение переменной `w` меньше значения переменной `h`. Эти операторы составляют единый логический блок, и ни один из них не может быть выполнен без другого. Кодовые блоки позволяют оформлять многие алгоритмы в удобном для восприятия виде. Ниже приведён пример программы, в которой кодовый блок предотвращает деление на ноль.

```
public static void main(String args[]) {
    double a = 5.0, b = 10.0, c = 0.0;
    if (b != 0) {
        System.out.println("b не равно нулю");
        c = b / a;
        System.out.print("b / a равно " + c);
    }
}
```

Результат:

b не равно 0
b / a равно 2.0

Области видимости переменных в кодовых блоках:

```
public static void main(String args[]) { // Кодовый блок метода main()
    int x = 10; // эта переменная доступна для всего кода в методе main
    if (x == 10) { // Кодовый блок тела if
        int y = 20; // Эта переменная доступна только в данном кодовом блоке
        // Обе переменные, x и y, доступны в данном кодовом блоке
        System.out.println("x & y: " + x + " " + y);
        x = y * 2;
    }
    // y = 100; // Ошибка! Переменная y недоступна за пределами тела if
    System.out.println("x = " + x); // Переменная x по-прежнему доступна
}
```

Ещё один пример объявления переменных в цикле:

```
public static void main(String args[]) {
    for (int i = 0; i < 3; i++) {
        int y = -1; // переменная y инициализируется при каждом входе в блок
        System.out.println("y = " + y); // всегда выводится значение -1
        y++;
        System.out.println("y = " + y);
    }
}
```

Массивы

Массив представляет собой набор однотипных переменных с общим для обращения к ним 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 массив, затем выделяется память для массива, а ссылка на неё присваивается переменной массива. Следовательно, память для массивов в Java динамически распределяется с помощью оператора new. В следующей строке

кода создается массив типа `int`, состоящий из 5 элементов, а ссылка на него присваивается переменной `arr`:

```
int[] arr = new int[5];
```

В переменной `arr` сохраняется ссылка на область памяти, выделяемую 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. Так, 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};
```

где `v1-vN` обозначают первоначальные значения, которые присваиваются элементам массива по очереди, слева направо и 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, при попытке 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 `ArrayIndexOutOfBoundsException` и выполнение программы прекратится.

Распечатать одномерный массив в консоль можно с использованием конструкции вида:

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

Двумерные массивы

Среди многомерных массивов наиболее простыми являются двумерные. Двумерный массив, по существу, представляет собой ряд одномерных массивов. При работе же 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 3x4, 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 двумерными массивами и отладке можно пользоваться следующим методом для распечатки этого массива. На вход метода необходимо подать ссылку на любой двумерный целочисленный массив. При данной реализации 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. А 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, по сути, 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, заключая инициализирующую последовательность для каждого размера массива 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 размер массива, для этого можно воспользоваться записью *имя_массива.length*. Например, это удобно использовать при необходимости 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

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 размером 8. С помощью цикла заполнить его значениями 0 3 6 9 12 15 18 21;
- 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;
- 5 ** Задать одномерный массив и найти в нем минимальный и максимальный элементы (без помощи интернета);
- 6 ** 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([1, 1, 1, || 2, 1]) → true, checkBalance ([2, 1, 1, 2, 1]) → false, checkBalance ([10, || 10]) → true, граница показана символами ||, эти символы в массив не входят.
- 7 **** 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. Для усложнения задачи нельзя пользоваться вспомогательными массивами.

*Если выполнение задач вызывает трудности, можете обратиться к последней странице методического пособия. Для задач со * не нужно искать решение в интернете, иначе вы теряете весь смысл их выполнения. Если делаете 2+ задачи со *, обычные задачи можно не делать.*

Дополнительные материалы

1. К. Сьерра, Б. Бейтс Изучаем Java // Пер. с англ. – М.: Эксмо, 2012. – 720 с.
2. Кей С. Хорстманн, Гари Корнелл Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. - М.: Вильямс, 2014. - 864 с.
3. Брюс Эккель Философия Java // 4-е изд.: Пер. с англ. – СПб.: Питер, 2016. – 1168 с.
4. Г. Шилдт Java 8. Полное руководство // 9-е изд.: Пер. с англ. - М.: Вильямс, 2015. - 1376 с.
5. 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[8];  
    for (int i = 0; i < arr.length; i++) {  
        ...  
    }  
}
```

Вариант 2:

```
public static void fillArray() {  
    int[] arr = new int[8];  
    arr[0] = 0;  
    for (int i = 1; i < arr.length; i++) {  
        ...  
    }  
}
```

Вариант 3:

```
public static void fillArray() {  
    int[] arr = new int[8];  
    for (int i = 0, ...; i < arr.length; i++, ...) {  
        ...  
    }  
}
```

И еще есть несколько вариантов...

3)

```
public static void changeArray() {  
    int[] w = { 1, 5, 3, 2, 11, 4, 5, 2, 4, 8, 9, 1 };  
    for (int i = 0; i < w.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-н означает что можно выполнить задачу несколькими способами. Представлены не все существующие решения, возможно вы найдете свое.