Урок 3

Циклы for и foreach

Наиболее удобные конструкции циклов.

Цикл for

For относится к типу циклов со счетчиком. Это классический пример красивой организации кода, и он выглядит похоже в большинстве языков программирования. Конструкция позволяет одной строкой определить поведение цикла:

```
for (инициализация; условие; завершающее выражение) {
   набор_операторов;
}
```

Инициализация выполняется перед началом цикла, обычно в ней создается управляющая переменная. **Условие** вычисляется в начале каждой итерации цикла. Это выражение ведет себя так же, как условие цикла **while**: если значением условия оказывается **true** — цикл продолжается, иначе — останавливается. **Завершающее выражение** вычисляется в конце каждой итерации и используется, как правило, для изменения значения управляющей переменной цикла.

Сам цикл выполняется согласно следующему алгоритму:

1. Выполнение инициализации.
2. Проверка на истинность условия.
3. Если условие истинно, выполняется тело цикла.
4. Выполнение завершающего выражения.

Цикл **for** выполняется до тех пор, пока проверка условия дает истинный результат. Выведем в цикле числа от 1 до 10 с применением **for**:

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

Наиболее часто **for** применяется именно так. С точки зрения кода мы создаем управляющую переменную-счетчик, устанавливаем ей границу роста и инкрементируем ее с каждой итерацией. Но это не единственный способ использования **for**.

Не обязательно указывать все три выражения для расчета. Так **for** может заменить **while**:

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

Первое выражение (инициация) не выполняется, на каждом шаге будет проверяться «условие» (нам надо знать, когда выходить из цикла). После каждой итерации не производим неявных манипуляций. Получаем цикл только с управляющим предусловием, что эквивалентно **while**.

Ниже приведен пример цикла с отрицательным приращением. Еще одна особенность цикла — объявление управляющей переменной выносится 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 + " ");
    }
}
```

Условное выражение цикла **for** всегда проверяется в начале цикла. Это означает, что код в цикле может вообще не выполняться, если проверяемое условие сразу оказывается ложным:

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

Этот цикл не будет выполняться совсем, поскольку начальное значение переменной **count** больше 5. Значит, условное выражение **count < 5** оказывается ложным с самого начала.

Для управления циклом можно использовать одновременно несколько переменных. Здесь 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);
    }
}
```

Бесконечный цикл и выход из шагов цикла

Частая ошибка неопытных программистов — бесконечные циклы. Ведь если в них войти, с каждой итерацией потребляется все больше памяти. В лучшем случае упадет программа, в худшем — весь сервер. Но бесконечный цикл может быть и полезен.

Бесконечными считаются циклы видов:

```
for (;;) {
   …
}
```

Как ими управлять, если условие выхода никогда не вернет **false**? Используется оператор **break**, который моментально выходит из цикла, не дожидаясь выполнения всего кода из его тела.

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

Работа с массивами и цикл foreach

Если хотим обработать каждый элемент массива, обходим его в цикле. Самый простой способ — использовать цикл **for**.

```
public static void main(String[] args) {
    String[] sm = {"A", "B", "C", "D"};
    for (int i = 0; i < sm.length; i++) {
        System.out.print(sm[i] + " ");
    }
}
```

Во многих языках существует более компактная форма **for** для перебора элементов массивов (и не только — чуть позже узнаем, что есть более сложные наборы элементов, основанные на объектах). Конструкция **foreach** не требует ручного изменения переменной-шага для перебора — цикл автоматически выполняет эту работу.

В Java решили не добавлять новое ключевое слово, а усовершенствовали цикл **for**:

```
for (тип итерируемый_элемент : имя_коллекции) {
   …
}
```

Перепишем первый пример с применением **foreach**:

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

Проходим по элементам массива **sm** типа **String**, и каждому присваиваем временное имя **o**. То есть «в единицу времени» **o** указывает на один элемент массива.

Вложенные циклы

Для многомерных массивов будет недостаточно одного цикла, чтобы последовательно обработать все элементы. Здесь на помощь приходят вложенные циклы (работающие внутри других). Рассмотрим последовательность их исполнения: одной итерации внешнего цикла будет соответствовать одно полное выполнение внутреннего.

Создадим двумерный массив размером 3x4, заполним его числами от 1 до 12 и напечатаем 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 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();
    }
}
```