Модуль 1. Основы программирования

Тема 1.8. Цикл for. Массивы

2 часа

Оглавление

1.8. Цикл for. Массивы 2
1.8.1. Цикл for....2
Цикл for и его отличия от while 2
Оператор break с меткой 3
Оператор continue 4
1.8.2. Одномерные массивы 5
Объявление и создание массивов 5
Доступ к элементам массива....7
1.8.3. Цикл for each 8
1.8.4. Примеры программ обработки массивов....8
1.8.5. Основные итоги урока....10
Задание 1.8.1....10
Благодарности 11

1.8. Цикл for. Массивы

1.8.1. Цикл for

Цикл for и его отличия от while

Всю теорию предыдущего занятия по циклам (третье занятие модуля) можно пересказать фактически одним предложением:

"Для того, чтобы организовать цикл в программе, можно использовать конструкцию **while**, которая очень похожа на **if**".

Но не все так просто, конечно. В отличие от **if** при использовании **while** необходимо правильно организовать выход. Это можно организовать оператором **break**, но это именно "аварийный выход".

"Обычный" **while** в программе выглядит примерно так:

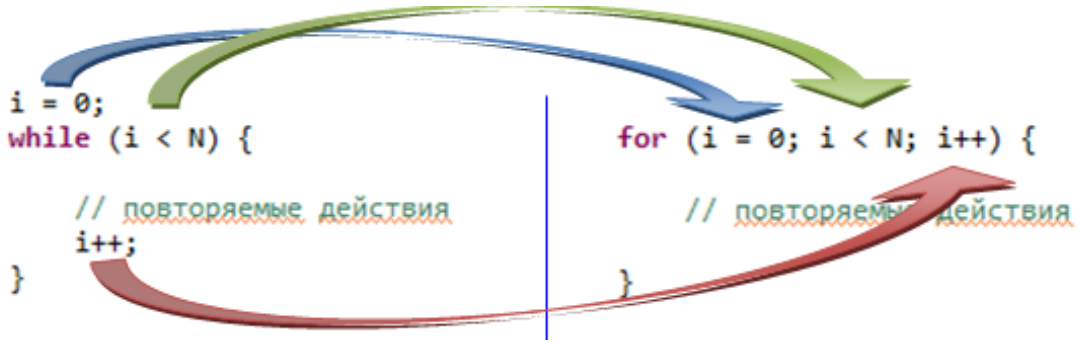
```
i = 0; // инициализация счетчика
while (i < N){ //условие цикла


      //... тело цикла - действия, которые выполняются многократно
      i++; //приращение счетчика
}
```

Все эти строки очень важны.

Конечно, никто не забывает написать в программе условие цикла. Но вот начальную инициализацию и приращение счетчика очень легко забыть написать. И от этого цикл "зависает" или наоборот не выполняется ни разу.

В конструкции **for** все действия, кроме собственно команд собраны в одной строке:



В конструкции **for**, как и в конструкции **while** "ненужные" вещи можно опускать. Например, для того, чтобы вычислить степень тройки, ближайшую к миллиону справа (минимальную, но превосходящую миллион) можно написать:

```
p = 1;
for (; p <= 1000000; ){
      p *= 3;
}
```

При этом две точки с запятой должны остаться в любом случае.

Впрочем, этот код лучше переписать так:

```
for (p = 1; p <= 1000000; p *= 3){
}
```

Оператор **break** работает в **for** точно так же, как и в **while**.

Счетчик цикла можно объявлять в самой строке **for**. В этом случае объявленная переменная будет существовать только внутри цикла. Например, программу, выводящую столбик чисел от 10 до 1 можно написать так:

```
for (int i = 10; i > 0; i--){
      out.println(i);
}
```

Если команду вывода вызвать еще раз после цикла,

```
for (int i = 10; i > 0; i--){
      out.println(i);
}
out.println(i);
```

то программа не скомпилируется: после цикла переменная **i** не существует.

Особенно повышается читаемость программы при использовании **for** вместо **while** при организации вложенных циклов ("один в другом"). Например, если надо вывести квадрат из N звездочек, можно написать так:

```
for (int i = 0; i < N; i++)
{
      for (int j = 0; j < N; j++)
      {
            out.print("*");
      }
      out.println();
}
```

Обратите внимание: в циклах – разные переменные-счетчики!

Оператор break с меткой

Выйти из вложенных циклов – самая настоящая проблема.

Оператор **break** не сильно помогает в этом, потому что он прерывает только внутренний – свой – цикл, и мы попадаем во внешний, который на следующей итерации… снова входит во внутренний.

В языке Си можно для этого использовать оператор goto – переход по метке на произвольную команду программы. Бьерн Страуструп, создатель языка C++, писал, что это, пожалуй, единственное обоснованное применение этого оператора. Вообще же, использование goto в программах крайне не рекомендуется, это обычно очень сильно затрудняет чтение программы.

Но все же тот, единственный полезный функционал goto – **выход из вложенных циклов, в языке Java реализовать можно при помощи оператора break с меткой**.

```
outer:
for (int i = 0; i < 3; i++) {
  System.out.print("Итерация " + i + ": ");
  for (int j = 0; j < 100; j++) {

    if (j == 10) {
      break outer; // выйти из обоих циклов
    }
    out.print(j + " ");
  }
  out.println("Эта строка никогда не будет выведена");
}
```

Программа выведет:

```
Итерация 0: 0 1 2 3 4 5 6 7 8 9
```

Итерацией называется один проход цикла.

При прерывании внутреннего цикла заканчиваются оба – и внутренний и внешний.

Оператор continue

Иногда полезно начать очередную итерацию цикла раньше. То есть нужно продолжить выполнение цикла, но прекратить обработку остатка кода в его теле для данной итерации. Такое действие выполняет оператор **continue**. Можно воспринимать его работу как **goto**-переход мимо следующих операций тела в конец блока цикла. В циклах **while** и **do while** оператор continue вызывает передачу управления непосредственно условному выражению, которое управляет циклом. В цикле **for** управление переходит сначала к итерационной части оператора **for** и затем к условному выражению. Для всех трех циклов любой промежуточный код обходится.

Оператор **continue** срабатывает на каждом четном i и перевод строки **не** происходит:

```
for (int i = 0; i < 10; i++) {
  System.out.print(i + " ");
  if (i % 2 == 0) {
    continue;
  }
  out.println();
}
```

Операцией % проверяется, является ли i четным. Если это так, цикл продолжается **без** перевода новой строки.

Вывод программы:

```
0 1
2 3
4 5
6 7
8 9
```

Как и в операторе **break**, в **continue** можно определить метку, указывающую, какой включающий цикл следует продолжить.

1.8.2. Одномерные массивы

Мы познакомились с примитивными типами данных языка Java и научились использовать переменные примитивных типов. Существует множество задач, в которых требуется рассматривать сразу целый набор значений одного и того же типа. Примерами таких задач могут служить поиск наивысшего балла в списке участников ЕГЭ по информатике, вычисление среднемесячной дневной температуры по результатам ее ежедневных измерений, построение топ-листа наиболее популярных исполнителей на основе опроса слушателей и т.д. Во всех этих случаях вместо большого числа однотипных переменных используются структура данных, называемая массивом.

Под структурой данных в программировании понимается множество значений одного или разных типов, определенным образом размещенных в памяти компьютера. Структура данных, рассматриваемая как целое, снабжается именем, а для доступа к отдельным входящим в нее значениям определяется специальный синтаксис.

Массив – это самая простая структура данных, которая представляет собой заданное количество значений одинакового типа – элементов массива, размещенных последовательно в ячейках памяти. Количество таких элементов называется размером массива, а тип элементов – типом массива. Ниже в качестве примеров будут использоваться массивы типа int, однако все сказанное применимо и к массивам других типов.

Объявление и создание массивов

При объявлении массива, как и при объявлении переменной, нужно указать имя массива и его тип, например:

```
int[] a;
```

Квадратные скобки после имени типа указывают на то, что объявляется именно массив, а не простая переменная. Можно одновременно объявить несколько массивов:

```
int [] a, b, c;
```

*Допустимо также объявление массивов в стиле языка C/C++, а именно, с записью квадратных скобок **после** имени массива:*

```
int a[];
```

В этом случае объявление одновременно нескольких массивов будет выглядеть более громоздко:

```
int a[], b[], c[];
```

Однако применение такого синтакиса позволяет одновременно объявить как массив, так и простую переменную:

```
int a[], i;
```

В отличие от переменной, объявления массива недостаточно для начала работы с ним. Следующим шагом является создание массива с заданием его размера. Создание массива осуществляется с помощью операции new и имеет следующий синтаксис:

```
имя_массива = new тип_массива [размер_массива]
```

Например:

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

Объявление массива можно совместить с его созданием, используя следующий синтаксис:

```
тип_массива имя_массива[] = new тип_массива [размер_массива]
```

Например:

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

При создании массива происходит его инициализация, т.е. присваивание начальных значений элементам массива. По умолчанию компилятор инициализирует числовые массивы нулевыми значениями, символьные – нулевым символом (символом с кодом 0), булевские – значением false. Имеется возможность инициализировать элементы массива другими значениями, добавив в оператор создания массива список выражений, заключенный в фигурные скобки:

```
тип_массива имя_массива[] = new тип_массива[] {список_выражений};
```

Например, если имеются ранее объявленные и инициализированные переменные x и y, то можно следующим образом создать и инициализировать массив **a**:

```
int a[] = new int [] {3, 11, x, 2*x, y - x};
```

При этом размер создаваемого массива не указывается, поскольку он устанавливается равным длине списка инициализации (в примере это 5). Более того, использование списка инициализации позволяет отказаться от операции new и создавать массив с использованием простого синтаксиса:

```
тип_массива имя_массива[ ] = {список_выражений};
```

Например:

```
int a[] = {3, 11, x, 2*x, y - x};
```

После создания массива его размер сохраняется в свойстве length, которое рекомендуется использовать в различных алгоритмах обработки массивов. Обращение к этому свойству имеет синтаксис:

```
имя_массива.length
```

Например, для массива a, объявленного в предыдущем примере, оператор

```
System.out.println(a.length);
```

выведет на экран значение 5.

Доступ к элементам массива

Элементы массива нумеруются, начиная с нулевого значения, и номер элемента называется его индексом. Таким образом, первому элементу массива соответствует значение индекса 0, второму – значение индекса 1, элементу с порядковым номером k – значение индекса k-1. Для доступа к отдельным элементам массива используется следующий синтаксис:

```
имя_массива[выражение_целого_типа]
```

Значение выражения в квадратных скобках рассматривается как индекс элемента массива и поэтому должно иметь значение в диапазоне от 0 до length – 1, включительно.

В случае, если в ходе выполнения Java программа пытается выйти за границы массива (обратиться к элементу массива с индексом вне диапазона от 0 до размерности массива минус 1), то возникнет ошибка - исключение ArrayIndexOutOfBoundsException

Поэтому необходимо внимательно отслеживать возможный диапазон значений, которые может принять индекс массива.

Как сделать программу более устойчивой и обрабатывать, возникающие по ходу выполнения программы исключения, более подробно будет рассмотрено в одной из тем Модуля 2.

В качестве примеров обращения к отдельным элементам массива рассмотрим циклы заполнения массива случайными числами с последующим выводом полученных значений на экран. Для этого нам потребуется функция, генерирующая последовательность случайных чисел. Такой функцией является метод random() класса Math.

Метод random() позволяет получить последовательность случайных значений типа double из диапазона [0, 1). Чтобы получить случайное число в диапазоне [0,**maxRange**), результат random() нужно умножить на **maxRange**.

Если же необходимо получить целое число из диапазона [0, **maxRange**], то результат random() нужно умножить на max+1, а затем полученный результат преобразовать к целому типу:

```
int maxRange = 100;
for (int i = 0; i < a.length; i++) {
      a[i] = (int) (Math.random() * (maxRange + 1));//Целое число
0..max
      System.out.print("\t" + a[i]);
}
```

В этом примере элементы массива будут получать случайные значения из интервала [0; 100]. Полученные значения выводятся в строку и разделяются символами табуляции. Меняя в программе значение **maxRange**, можно получить случайное число из нужного диапазона [0; **maxRange**].

В общем случае, чтобы получить число из диапазона [**minRange**; **maxRange**] можно использовать следующее выражение:

```
(int) (minRange + Math.random() * (maxRange - minRange + 1));
```

1.8.3. Цикл for each

В тех случаях, когда не требуется производить запись новых значений в элементы массива, можно воспользоваться модификацией цикла for, известной как цикл for each. В этом варианте цикл for имеет следующий синтаксис:

```
for (тип_массива имя_переменной: имя_массива) тело_цикла
```

В переменную, имя которой указано в заголовке цикла, последовательно копируются значения элементов массива. Эта переменная может использоваться только в теле цикла. Любые ее изменения никак не скажутся на элементе массива, который она представляет на очередном шаге цикла. Так, для выше рассмотренного массива a, с использованием такого цикла можно вычислить сумму элементов числового массива:

```
int sum = 0;
for (int x: a)
    sum += x;
```

1.8.4. Примеры программ обработки массивов

Программа, демонстрирующая способы создания, инициализации и заполнения массива.

```
public class myArray {

        public static void main(String[] args) {
           //TODO Auto-generated method stub

           System.out.println("Массив, заполненный случайными числами");
           // Объявление массива, определение максимума

           int [] a;
           int maxRange = 100;
           // Создание массива

           a = new int [10];
           // Заполнение массива случайными числами

           for(int i = 0; i < a.length; i++) {
                 a[i] = (int) (Math.random() * (maxRange + 1));
                 System.out.print("\t" + a[i]);
```

```
            }
            System.out.println();
            System.out.println("\nМассив, созданный списком инициализации");
            int x = 5;
            // Объявление и создание массива
            int b[] = {3, 11, x, 2 * x, a[1] - a[2]};
            for(int y: b)
                  System.out.print("\t" + y);
            System.out.println();
         }
}
```

Результат работы программы:

```
Массив, заполненный случайными числами
  82   95   12   48   47   25   50   36   50   53

Массив, созданный списком инициализации
  3   11   5   10   83
```

Программа, демонстрирующая алгоритм нахождения минимального и максимального значения в массиве.

```
public class myArray {

         public static void main(String[] args) {
            // TODO Auto-generated method stub
            System.out.println("Массив, заполненный случайными числами");
            int maxRange = 100;
            // Объявление и создание массива
            int[] a = new int[10];
            for (int i = 0; i < a.length; i++) {
                  a[i] = (int) (Math.random() * (maxRange + 1));
                  System.out.print("\t" + a[i]);
            }
            System.out.println();
            // Поиск минимального и максимального значений
            int min = a[0], max = a[0];
            int imin = 0, imax = 0;
            for (int i = 0; i < a.length; i++) {
                  if (a[i] < min) {
                        min = a[i];
                        imin = i;
                  }
                  else if (a[i] > max) {
                        max = a[i];
                        imax = i;
                  }
            }
       System.out.println("Минимальное значение " + min + " у элемента с
индексом " + imin);
```

```
            System.out.println("Максимальное значение " + max + " у
    элемента с индексом " + imax);
        }
}
```

Результат работы программы:

```
Массив, заполненный случайными числами
              16  50  66  0  9  72  22  35  44  88
Минимальное значение 0 у элемента с индексом 3
Максимальным значение 88 у элемента с индексом 9
```

1.8.5. Основные итоги урока

1. Массив – это набор значений одного и того же типа (элементов массива), размещенных в последовательных ячейках памяти.
2. Число элементов массива называется его размером или длиной. Размер массива хранится в свойстве length.
3. Существует две формы объявления массива.
4. Кроме объявления необходимо выполнить создание массива с указанием его размера.
5. Массив может быть создан с указанием списка инициализации; в этом случае размер массива задается неявным образом.
6. Массив может быть создан с помощью операции new или с помощью списка инициализации.
7. Доступ к отдельным элементам массива осуществляется с помощью целочисленного индекса. Индексация элементов начинается с 0.
8. Для заполнения массива после его создания можно использовать метод random() класса Math.
9. Перебор элементов массива в режиме «только чтение» удобно проводить в цикле for each.

Задание 1.8.1

Написать программу по обработке массива с выводом на экран полученного результата:

- переворот массива «задом наперед» (инверсия) без использования вспомогательного массива (задача № 69 на informatics)
- вычисление суммы элементов массива
- нахождение самого часто повторяющегося значения среди элементов массива (задача № 3173 на informatics)
- нахождение среднего арифметического значений элементов массива (задача № 112282 на informatics)
- заполнить массив значениями членов арифметической прогрессии, полученными по заданной учителем формуле (задача № 112271 на informatics)
- подсчитать количество положительных и отрицательных значений в массиве (задача № 3154 на informatics)

- переместить в начало массива все отрицательные значения задача, не меняя их относительного расположения.

Кроме того, решите задачи на informatics №№ 66, 70, 71, 72.

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю IT ШКОЛЫ SAMSUNG Хлебостроеву Виктору Григорьевичу и Ильину Владимиру Владимировичу.