Лабораторная работа №6
Управление выполнением программы.
Условные операторы. Операторы цикла

Условные операторы
Оператор if-else

В обобщенной форме этот оператор записывается следующим образом:
if (логическое выражение) оператор1; [else оператор2;]
Раздел else необязателен. На месте любого из *операторов* может стоять *составной оператор,* заключенный в фигурные скобки. *Логическое выражение* — это любое выражение, возвращающее значение типа boolean.

А вот полная программа, в которой для определения, к какому времени года относится тот или иной месяц, используются операторы if-else.

```
class IfElse {
public static void main(String args[]) { int month = 4;
String season;
if (month == 12 || month == 1 || month == 2) {
season = "Winter";
} else if (month ==3 || month == 4 || month == 5) {
season = "Spring";
} else if (month == 6 || month == 7 || month == 8) {
season = "Summer";
} else if (month == 9 || month == 10 || month == 11) {
season = "Autumn";
} else {
season = "Bogus Month";
}
System.out.println( "April is in the " + season + ".");
} }
```

После выполнения программы вы должны получить следующий результат:

April is in the Spring.

break

В *языке* Java отсутствует оператор goto. Для того, чтобы в некоторых случаях заменять goto, в Java предусмотрен оператор break. Этот оператор сообщает исполняющей среде, что следует прекратить выполнение именованного блока и передать управление оператору, следующему за данным блоком. Для именования блоков в языке Java используются метки. Оператор break при работе с циклами и в операторах switch может использоваться без метки. В таком случае подразумевается выход из текущего блока.
Например, в следующей программе имеется три вложенных блока, и у каждого своя уникальная метка. Оператор break, стоящий во внутреннем блоке, вызывает переход на оператор, следующий за блоком b. При этом пропускаются два оператора println.

```
class Break {
public static void main(String args[]) { boolean t = true;
a: { b: { c: {
System.out.println("Before the break"); // Перед break
if (t)
break b;
System.out.println("This won't execute"); // Не будет выполнено }
System.out.println("This won't execute"); // Не будет выполнено }
System.out.println("This is after b"); //После b
} } }
```

В результате исполнения программы вы получите следующий резуль-тат:

```
Before the break
This is after b
```

Вы можете использовать оператор break только для перехода за один из текущих вложенных блоков. Это отличает break от оператора goto языка C, для которого возможны переходы на произвольные метки.

Оператор switch

Оператор switch обеспечивает ясный способ переключения между различными частями программного кода в зависимости от значения одной переменной или выражения. Общая форма этого оператора такова:

```
switch ( выражение ) {
case значение1:
break;
case значение2:
break;
case значением:
break;
default:
}
```

Результатом вычисления *выражения* может быть значение любого простого типа, при этом каждое из значений, указанных в операторах case, должно быть совместимо по типу с выражением в операторе switch. Все эти значения должны быть уникальными литералами. Если же вы укажете в двух операторах case одинаковые значения, транслятор выдаст сообщение об ошибке.
Если же значению выражения не соответствует ни один из операторов case, управление передается коду, расположенному после ключевого слова default. Отметим, что оператор default необязателен. В случае, когда ни один из операторов case не соответствует значению выражения и в switch отсутствует оператор default выполнение программы продолжается с оператора, следующего за оператором switch.

Внутри оператора switch (а также внутри циклических конструкций, но об этом — позже) break без метки приводит к передаче управления на код, стоящий после оператора switch. Если break отсутствует, после текущего раздела case будет выполняться следующий. Иногда бывает удобно иметь в операторе switch несколько смежных разделов case, не разделенных оператором break.

```
class SwitchSeason { public static void main(String args[]) {
int month = 4;
String season;
switch (month) {
case 12:
case 1:
case 2:
season = "зимаг";
break;
case 3:
case 4:
case 5:
season = "весна";
break;
case 6:
case 7:
case 8:
season = "лето";
break;
case 9:
case 10:
case 11:
season = "осень";
break;
default:
season = "несуществующий номер месяца";
}
System.out.println("Месяц № "+ month + "- это " + season + ".");
} }
```

return

Подпрограмма main, которую мы использовали до сих пор — это статический метод соответствующего класса-примера. В любом месте программного кода метода можно поставить оператор return, который приведет к немедленному завершению работы и передаче управления коду, вызвавшему этот метод. Ниже приведен пример, иллюстрирующий использование оператора return для немедленного возврата управления, в данном случае — исполняющей среде Java.

```
class ReturnDemo {
public static void main(String args[]) {
boolean t = true;
System.out.println("Before the return"); //Перед оператором return
if (t) return;
System.out.println("This won't execute"); //Это не будет выполнено
} }
```

Зачем в этом примере использован оператор if (t)? Дело в том, не будь этого оператора, транслятор Java догадался бы, что последний оператор println никогда не будет выполнен. Такие случаи в Java считаются ошибками, поэтому без оператора if оттранслировать этот пример нам бы не удалось.

Тернарный оператор if-then-else

Некоторые операторы требуют одного операнда, их называют *унарными*. Одни операторы ставятся перед операндами и называются *префиксными*, другие — после, их называют *постфиксными* операторами. Большинство же операторов ставят между двумя операндами, такие операторы называются *инфиксными* и *бинарными* операторами. Существует *тернарный* оператор, работающий с тремя операндами. Общая форма оператора такова:

выражение1? выражение2***: выражение3***

В качестве первого операнда — «выражение1» — может быть использовано любое выражение, результатом которого является значение типа boolean. Если результат равен true, то выполняется оператор, заданный вторым операндом, то есть, «выражение2». Если же первый операнд равен false, то выполняется третий операнд — «выражение3». Второй и третий операнды, то есть «выражение2» и «выражение3», должны возвращать значения одного типа и не должны иметь тип void.

В приведенной ниже программе этот оператор используется для проверки делителя перед выполнением операции деления. В случае нулевого делителя возвращается значение 0.

```
class Ternary {
public static void main(String args[]) {
int a = 42;
int b = 2;
int c = 99;
int d = 0;
int e = (b == 0) ? 0 : (a / b);
int f = (d == 0) ? 0 : (c / d);
System.out.println("a = " + a);
System.out.println("b = " + b);
System.out.println("c = " + c);
System.out.println("d = " + d);
```

```
System.out.println("a / b = " + e);
System.out.println("c / d = " + f);
} }
```

При выполнении этой программы исключительной ситуации деления на нуль не возникает и выводятся следующие результаты:

```
a = 42
b = 2
c = 99
d = 0
a / b = 21
c / d = 0
```

Операторы цикла

Оператор while

Любой цикл можно разделить на 4 части — *инициализацию, тело, итерацию и условие завершения.* В Java есть три циклические конструкции: while (с пред-условием), do-while (с пост-условием) и for (с параметром или счетчиком).

Этот цикл многократно выполняется до тех пор, пока значение логического выражения равно true. Ниже приведены общие формы оператора while:

- упрощенные формы

while (условие завершения) оператор;

или

```
while (условие завершения)
        {
        <группа операторов>
        }
```

- расширенные формы

```
[ инициализация; ]
while ( завершение завершения)
{
тело;
[итерация;] }
```

Инициализация и итерация необязательны. Ниже приведен пример цикла while для вывода десяти чисел.

```
class WhileDemo {
public static void main(String args[]) {
int n = 10;
while (n > 0) {
System.out.println("Число n = " + n);
n--;
}
} }
```

Оператор do-while

Иногда возникает потребность выполнить тело цикла по крайней мере один раз — даже в том случае, когда логическое выражение с самого начала принимает значение false. Для таких случаев в Java используется циклическая конструкция do-while. Ее общие формы записи таковы:

- упрощенные формы

do оператор while (условие завершения) ;

или

```
do  {
        <группа операторов>
    } while  (условие завершения);
```

- расширенные формы

```
[ инициализация; ]
do { тело; [итерация;] }
while  ( завершение );
```

В следующем примере тело цикла выполняется до первой проверки условия завершения. Это позволяет совместить код итерации с условием завершения:

```
class DoWhile {
public static void main(String args[]) {
int n = 10;
do {
System.out.println("Число n =  " + n);
} while (--n > 0);
} }
```

Оператор for

В этом операторе предусмотрены места для всех четырех частей цикла. Ниже приведена общая форма оператора записи for.

for (инициализация; условие завершения; итерация)
тело;

Любой цикл, записанный с помощью оператора for, можно записать в виде цикла while, и наоборот. Если начальные условия таковы, что при входе в цикл условие завершения не выполнено, то операторы тела и итерации не выполняются ни одного раза. В каноническая форме цикла **for** происходит увеличение целого значения счетчика с минимального значения до определенного предела.

```
class ForDemo {
public static void main(String args[]) {
for (int i = 1; i <= 10; i++)
System.out.println("i = " + i);
} }
```

Следующий пример — вариант программы, ведущей обратный отсчет.

```
class ForTick {
public static void main(String args[]) {
for (int n = 10; n > 0; n--)
System.out.println("Число n =   " + n);
} }
```

Обратите внимание — переменные можно объявлять внутри раздела инициализации оператора for. Переменная, объявленная внутри оператора for, действует в пределах этого оператора.

Следующий пример — нахождение суммы, произведения и количества элементов массива.

```
  int i;
 float av=0;
 int  s=0;
double  p=1;
int  nn=0;

/* заполнение массива числами */
 int Ar[]={2,-4,1,-8,11,-3};

for(i=0;i<6;i++)
          {
          s=s+Ar[i];
           p=p*Ar[i];
          nn++;
          }
av=(float)s/nn;

System.out.println("Среднее арифметическое = "+av);
System.out.println("Произведение = "+p);
```

Иногда возникают ситуации, когда разделы инициализации или итерации цикла for требуют нескольких операторов. Поскольку составной оператор в фигурных скобках в заголовок цикла for вставлять нельзя, Java предоставляет альтернативный путь. Применение запятой (,) для разделения нескольких операторов допускается только внутри круглых скобок оператора for. Ниже приведен тривиальный пример цикла for, в котором в разделах инициализации и итерации стоит несколько операторов.

```
class Comma {
public static void main(String args[]) {
int a, b;
for (a = 1, b = 4; a < b; a++, b--) {
System.out.println("a = " + a);
System.out.println("b = " + b);
}
} }
```

Вывод этой программы показывает, что цикл выполняется всего два раза.

```
a = 1
b = 4
a = 2
b = 3
```

Оператор continue
В некоторых ситуациях возникает потребность досрочно перейти к выполнению следующей итерации, проигнорировав часть операторов тела цикла, еще не выполненных в текущей итерации. Для этой цели в Java предусмотрен оператор **continue**. Ниже приведен пример, в котором оператор continue используется для того, чтобы в каждой строке печатались два числа.

```
class ContinueDemo {
public static void main(String args[]) {
for (int i=0; i < 10; i++) {
System.out.print(i + " ");
if (i % 2 == 0) continue;
```

```
System.out.println("");
}
}}
```

Если индекс четный, цикл продолжается без вывода символа новой строки. Результат выполнения этой программы таков:

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

Для операторов break и continue можно задавать метку, указывающую, в каком из вложенных циклов вы хотите досрочно прекратить выполнение текущей итерации. Для иллюстрации служит программа, использующая оператор continue с меткой для вывода треугольной таблицы умножения для чисел от 0 до 9:

```
class ContinueLabel {
public static void main(String args[]) {
outer: for (int i=0; i < 10; i++) {
for (int j = 0; j < 10; j++) {
if (j > i) {
System.out.println("");
continue outer;
}
System.out.print(" " + (i * j));
}
}
}}
```

Оператор continue в этой программе приводит к завершению внутреннего цикла со счетчиком j и переходу к очередной итерации внешнего цикла со счетчиком i. В процессе работы эта программа выводит сле-дующие строки:

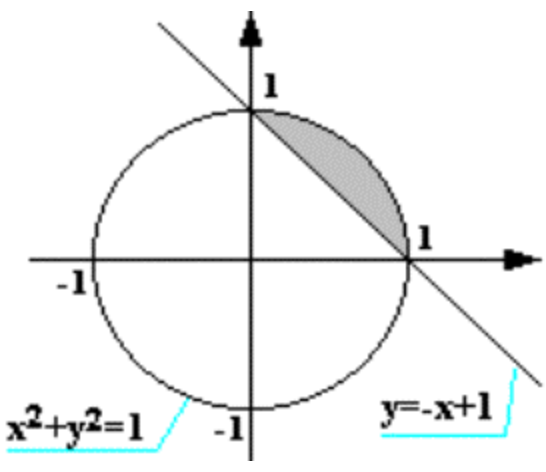
```
0
0 1
0 2 4
0 3 6 9
0 4 8 12 16
0 5 10 15 20 25
0 6 12 18 24 30 36
0 7 14 21 28 35 42 49
0 8 16 24 32 40 48 56 64
0 9 18 27 36 45 54 63 72 81
```

Задание 1. Ввести целое число – оценку студента по пятибальний системе. С помощью оператора switch … case и if…else вивести слово, соответстующее этой оценке.

Задание 2. С помощью тернарного оператора определить, четным или нечетным является заданное целое число.

Задание 3. С помощью тернарного оператора найти наибольшее из трех заданных чисел.

Задание 4. С помощью оператора ***if...else*** определить, попадает ли точка с координатами (x, y) в заштрихованую область на рисунке.



Задание 5.

Вычислить произведение и сумму, используя три оператора цикла

$$X=\prod_{i=1}^{4}(2i+xi^2),$$

$$Y=\sum_{n=3(2)}^{10}\frac{n^3}{n^2+1}$$