Операторы управления выполнением программы

Условный оператор if-else

if(логическое выражение) оператор1; [else оператор2;]

```
class IfElse {
public static void main(String args[]) {
        String m = args[0];
        String season;
if ((m == “12”) || (m == “1”) || (m ==“2”) ) {
        season = "Winter";
} else if ((m ==“3”) || (m == “4”) || (m ==“5”) ) {
        season = "Spring";
} else if ((m == “6”) || (m == “7”) || (m == “8”) ) {
        season = "Summer";
} else if ((m == “9”) || (m == “10”) || (m == “11”) ) {
        season = "Autumn";
} else {
        season = "Bogus Month";
}
System.out.println( "This month is in the " + season);
} }
```

После выполнения программы вы должны получить следующий результат:

C: \> java IfElse 4

This month is in the Spring.

Операторы управления выполнением программы

Оператор **return**

Ключевое слово **return** используется в методах для указания того, какое значение возвращает метод (если он имеет возвращаемое значение не **void**) и показывает место завершения метода.
Для методов, не возвращающих значений значащих типов (спецификатор **void**), оператор **return** без значения в общем случае необязателен, но может использоваться для указания места принудительного завершения метода.

```
public class IfElse2 {
 static int test(int testval, int target) {
   int result = 0;
     if(testval > target)  return +1;
     else if(testval < target)   return -1;
      else      return 0; // Совпадает
}
public static void main(String args[ ]) {
   System.out.println(test(10, 5));
   System.out.println(test(5, 10));
   System.out.println(test(5, 5));
/* здесь можно было поставить return;
    однако метод main завершится по выходу на границу
*/
}}
```

Операторы управления выполнением программы

Операторы цикла

Оператор **while** – (с предусловием)

while(Логическое выражение) {
операторы
}

Логическое выражение вычисляется каждый раз перед каждой будущей итерацией. Операторы выполняются, если логическое выражение равно **true**.

//: c03:WhileTest.java

```
public class WhileTest {
public static void main(String args[ ]) {
 double r = Math.random();
while(r <= 0.77d) {
    r = Math.random();
    System.out.println(r);
 }
}}
```

Здесь используется **статический** метод **random( )** из библиотеки **Math**, который генерирует значения типа **double** в пределах от 0 до 1. (Это включает 0, но не включает 1.) Логическое выражение для **while** говорит, “продолжать выражение этого цикла, пока не встретится число 0.77 или больше”. Всякий раз запуск программы, будет давать разные списки чисел (возможно список будет пуст).

Операторы управления выполнением программы

Операторы цикла

Оператор **do-while** – (с постусловием)

do {
операторы
}
while(Логическое выражение);

Логическое выражение проверяется только после первой итерации и далее после каждой следующей. Операторы выполняются хотя бы один раз, а далее – если логическое выражение равно **true**.

//: c04:DoWhileTest.java

```
public class DoWhileTest {
public static void main(String args[ ]) {
 double r = Math.random();
do {
    r = Math.random();
    System.out.println(r);
}
while(r <= 0.77d);
}}
```

Оператор работает аналогично (список хотя бы из одного значения будет выведен всегда).

Операторы управления выполнением программы

Операторы цикла

Оператор **for** – (со счетчиком)

for(инициализация счетчика; условие завершения; шаг счетчика){
операторы
}

Цикл **for** выполняет инициализацию переменной любого целого типа (и типа char) перед первой итерацией. Затем он проверяет условие завершения, а в конце каждой итерации выполняется “шаг” переменной цикла. Выражение проверяется перед каждой итерацией, и как только при вычислении получится **false**, выполнение продолжится со строки, следующей за инструкцией **for**. В конце каждого цикла выполняется *шаг*.

```
//: c03:ListCharacters.java
// Демонстрация цикла "for" для составления списка всех ASCII символов.

public class ListCharacters {
public static void main(String args[ ]) {
for(char c = 0; c < 128; c++) {
        if (c != 26 )  // ANSI Очистка экрана
            System.out.println("value: " + (int)c +   “====> character: " + c);
   }
 }
}
```

Операторы управления выполнением программы

Операторы **break** и **continue**

- Управление выполнением циклов: **break** прерывает текущий цикл и передает управление на следующий за циклом оператор; **continue** останавливает выполнение текущей итерации и возвращается к началу цикла, начиная следующую итерацию.

- Передача управления по метке (метка – это идентификатор, после которого стоит символ «:», метками можно снабжать циклы и программные блоки {…..}) : **break** с меткой прерывает внутренний цикл или блок и передает управление на оператор, следующий за помеченным циклом (блоком); **continue** с меткой прерывает текущую итерацию внутреннего цикла (выполнение внутреннего блока) и передает управление на следующую итерацию внешнего помеченного цикла (начинает выполнение внешнего помеченного блока).

```
label1:
outer-iteration {
    inner-iteration {
//...
break; // 1
//...
continue;  // 2
//...
continue label1; // 3
//...
break label1;  // 4
}
//…
}
```

- **break** без метки используется также в операторе **switch-case**

Операторы управления выполнением программы

Оператор **switch-case**

```
switch(выражение) {
case значение1: {группа операторов;
                 break; }
case значение2: {группа операторов;
                 break; }
case значение3: {группа операторов;
                 break; }
…
default: {группа операторов;}
}
```

Выражение – типы **int** или **char** , при этом значения в **case** – того же типа и не должны совпадать. Если значению *выражения* не соответствует ни один из конструкций **case**, выполняется код после ключевого слова **default**. Конструкция **default** необязательна. Когда ни один из **case** не соответствует значению *выражения* и в **switch** отсутствует конструкция **default,** выполнение программы продолжается со следующего за **switch** оператором.

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**.

Операторы управления выполнением программы

Пример: В случайном порядке создаются буквы и проверяются, являются ли они гласными или согласными.

Math.random( ) генерирует значения в пределах от 0 до 1. Номера букв в таблице символов ASCII начинаются с 27. Чтобы получать случайным образом номера букв необходимо умножить результат **Math.random( )** на верхний предел границы чисел - 26 и прибавлять смещение для установки нижней границы номеров букв. Поскольку **Math.random( )** генерирует числа типа **double** необходимо провести явное приведение типа (char).

```
public class VowelsAndConsonants {
  public static void main(String[] args) {
     for(int i = 0; i < 100; i++) {
        char c = (char)(Math.random() * 26 + 'a');
        System.out.print(c + ": ");
       switch(c) {
              case 'a':
              case 'e':
              case 'i':
              case 'o':
              case 'u':    { System.out.println("гласная"); break;}
                     case 'y':
                     case 'w':    { System.out.println("  иногда гласная");  break;}
              default:     { System.out.println("согласная");  }
       }
       }
}}
```

Хотя здесь используется переключение для символов (char), инструкция **switch** на самом деле использует целые значения номеров символов в таблице ASCII. Символы в одинарных кавычках в инструкциях **case** также производят целочисленные значения, которые и используются для сравнения.

Несколько **case**‘ов друг над другом без **break** обеспечивают выполнение одного и того же действия при разных значениях переключателя.

Операторы управления выполнением программы

Оператор «запятая»

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, в Java применяется оператор «запятая» («,»), который не следует путать с разделителем «запятая». Использование оператора «запятая» допускается только внутри круглых скобок оператора **for**.

```
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.

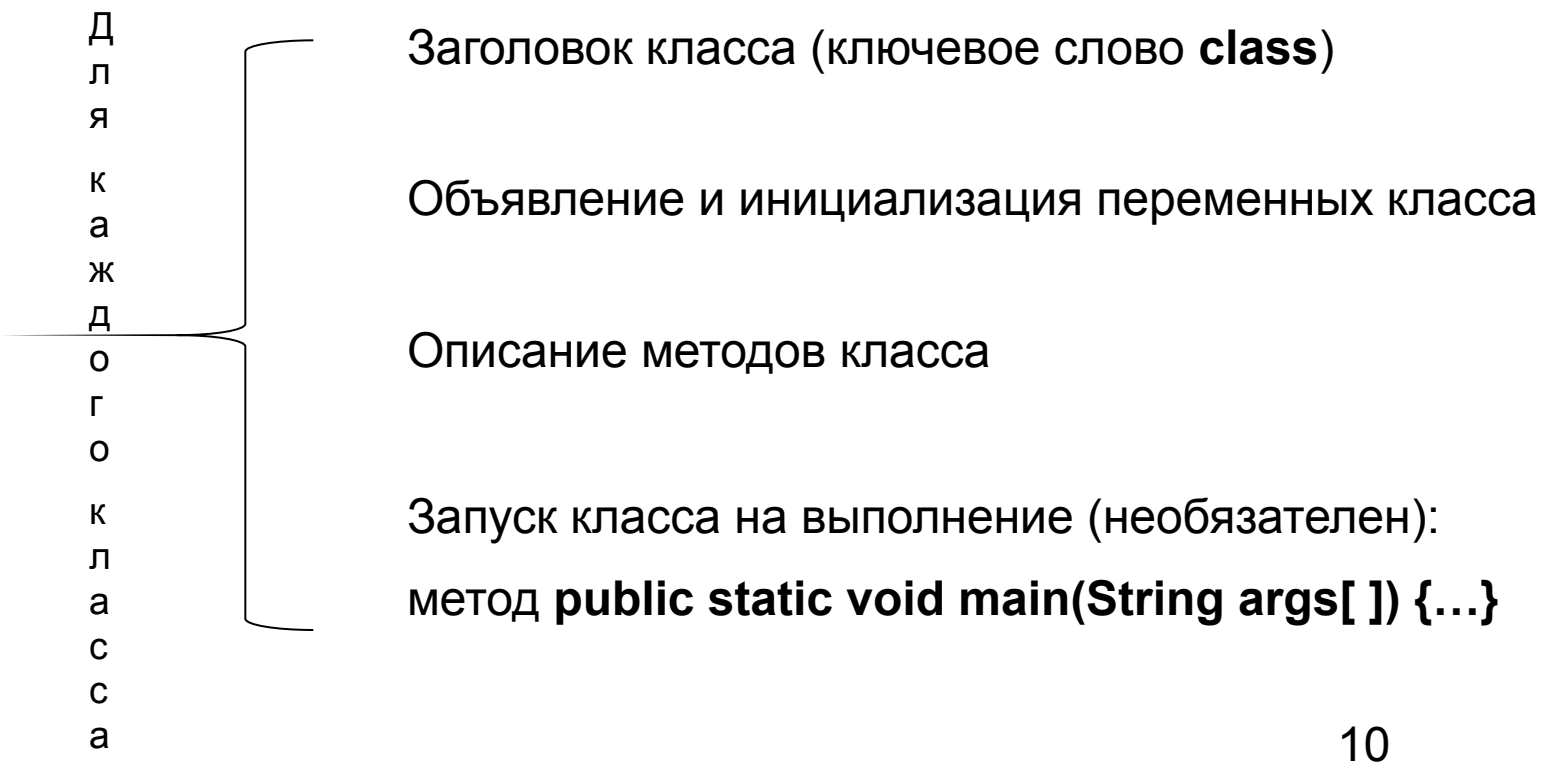
```
C: \> java Comma
a = 1
b = 4
a = 2
b = 3
```

Общая структура программы на языке Java

Блок идентификации пакетов и подключения внешних библиотек классов (операторы **package** и **import**)

Блок описания уникальных интерфейсов (ключевое слово **interface**)

Описание классов (в файле может быть несколько описаний классов)



Пример: Написать на Java программу, которая проверяет есть ли в массиве заданное число.

```
// Primer011.java
import java.util.*;

class Primer011 {
	public static boolean proverka(int a[], int b) {
	boolean f = false;
		for (int i=0; i<=a.length-1; i++) {
			if (a[i] == b) { f = true;
			}
		 }
	return f;
	}

	public static void main(String args[]) {
		int aa[] = {33, 10, 1, 7, 9, 20, 0, 12, 20, 99, 56};
		int bb = (int) (Math.random()*100 + 1);
		if (proverka(aa, bb)) {
				System.out.println(“Есть - " + bb);
			}  else {
				System.out.println(“Нет - " + bb);
				 }
	} // main
} // class
```

Задание на самостоятельную практику:

Update Your Java Runtime Environment

Написать на Java программу, для поиска максимального элемента массива.

```
/* Primer021.java
* Поиск максимума в массиве
*/
class Primer021 {
	public static double myMaxArray(double a[]) {
		double b = a[0];
		for (int i=1; i<=a.length-1; i++) {
			if (a[i] >= b) { b = a[i];
			}
		 }
	return b;
	}

	public static void main(String[] args) {
		double aa[] = {-33, -10, -1, -7, -9, 20, 0, 12, -20};
		double bb;
		bb = myMaxArray(aa);
	System.out.println("max = " + bb);
	}
}
```