Лекция 4

Операторы в JAVA

Ввод информации с клавиатуры

Проще всего вводить числа с клавиатуры, пользуясь классом `Scanner`. Этот класс принадлежит пакету `java.util`. Следовательно, чтобы использовать его в своей программе, нужно в начале (до описания класса, но после `package`) написать

```
import java.util.Scanner;
```

После этого в методе main нужно написать

```
Scanner S=new Scanner (System.in);
```

Далее, чтобы считать с клавиатуры вещественное число и запомнить его в переменной `x` (которая, разумеется, должна быть объявлена ранее), нужно написать

```
x=S.nextDouble();
```

Если мы хотим считать не вещественное, а целое число, вместо x нужно использовать имя целой переменной, а вместо `nextDouble` писать `nextInt`.

Пример. Рассмотрим программу, вычисляющую синус числа, введенного с клавиатуры.

```
import java.util.Scanner;
public class sinchisla {
    public static void main (String[] args) {
    Scanner S = new Scanner (System.in);
    double x = S.nextDouble();
    System.out.println (Math.sin(x));
 }
}
```

Если запустить программу, в нижней части окна «`Output`» появляется строка текстового ввода «`Input`», куда нужно вводить информацию.

Замечание. Класс `Scanner` пользуется при своей работе настройками культурной среды и воспринимает в качестве разделителя целой и дробной частей не точку, а запятую.

Если нужно ввести с клавиатуры не одно число, а два и более, то нужно несколько раз писать `S.nextDouble()`. Каждая такая фраза выдает следующее введенное с клавиатуры вещественное число, которое можно записать в переменную или использовать в выражениях. При этом можно вводить в строке ввода несколько чисел, разделяя их пробелами.

Пример. Напишем программу, считывающую с клавиатуры два числа и печатающую их произведение.

```
import java.util.Scanner;
public class proizvedeniechisel {
    public static void main (String[] args) {
```

```
        Scanner S = new Scanner (System.in);
        double x = S.nextDouble();
        double y = S.nextDouble();
        System.out.println (x*y);
    }
}
```

При запуске этой программы можно либо ввести оба числа сразу, разделив их пробелами, либо вводить их по-одному, завершая ввод каждого числа нажатием клавиши Enter.

Условные операторы

Программам, получающим от внешнего мира какие-то данные, например, поступающие с клавиатуры или из параметров командной строки, часто приходится анализировать полученную информацию и выполнять различные действия в зависимости от истинности или ложности некоторых условий. Конструкция, существующая в языке Java для реализации такого поведения, называется условным оператором.

Прежде, чем мы перейдем к изучению условного оператора, рассмотрим вопрос о том, что такое **оператор**. Оператор это единица действия в программе. Существует несколько разновидностей операторов. Одну из них мы уже знаем это оператор, построенный из выражения.

Любое выражение можно превратить воператор, поставив после него точку с запятой. Надо заметить, что в Java, как и в C++, точка с запятой не разделяет операторы она превращает выражение в оператор. В частности, допустим пустой оператор, состоящий только из точки с запятой. Такой оператор ничего не делает и может встречаться везде, где структура языка требует наличия оператора.

Например, присваивание в Java это не оператор, это выражение; однако его можно превратить воператор, поставив после соответствующего выражения точку с запятой.

Следующая разновидность операторов - так называемый **составной оператор**. Это последовательность операторов, заключенная в фигурные скобки. Выполнение составного оператора состоит в последовательном выполнении входящих в него операторов в том порядке, в котором они встречаются в составном операторе. Составной оператор нужен тогда, когда есть необходимость поставить несколько операторов в то место программы, где структура языка допускает только один оператор.

Синтаксис, т. е. форма записи, **условного оператора** в языке Java следующий:

```
if (условие) оператор1 else оператор2
```

В языке Java условием может быть только выражение, имеющее логический результат (например, применение операции сравнения или логической операции).

Смысл условного оператора состоит в том, что сначала вычисляется условие; если оно истинно, выполняется `оператор1` (он называется положительной альтернативой), иначе выполняется `оператор2`

(отрицательная альтернатива). Ключевое слово `else` и отрицательная альтернатива могут отсутствовать - в этом случае при ложном условии не выполняется никаких действий.

Синтаксис языка не позволяет записывать несколько операторов ни в ветви `then`, ни в ветви `else`. При необходимости составляется блок операторов в фигурных скобках. Соглашения "Code Conventions" рекомендуют всегда использовать фигурные скобки и размещать оператор на нескольких строках с отступами, как в следующем примере:

```
if (a < x)   {
                    x = a + b;
               }
else           {
                    x = a – b;
               }
```

Очень часто одним из операторов является снова условный оператор, например:

```
if (n == 0)  {
                    sign = 0;
               }
else if (n < 0)  {
                         sign = -1;
                    }
    else {
         sign = 1;
          }
```

При этом может возникнуть такая ситуация ("dangling else"):

```
Int ind = 5, x = 100;
if (ind>= 10) if (ind<= 20) x = 0; else x = 1;
```

Сохранит переменная *x* значение 0 или станет равной 1? Здесь необходимо волевое решение, и общее для большинства языков, в том числе и Java. Правило таково: ветвь `else` относится к ближайшему слева условию `if`, не имеющему своей ветви `else`. Поэтому в нашем примере переменная *x* останется равной 0.

Изменить этот порядок можно с помощью блока:

```
if (ind> 10) {if (ind< 20) x = 0; else x = 1;}
```

Вообще не стоит увлекаться сложными вложенными условными операторами. Проверки условий занимают много времени. По возможности лучше использовать логические операции, например, в нашем примере можно написать

```
if (ind>= 10 && ind<= 20) x = 0; else x = 1;
```

В приведенном ниже листинге вычисляются корни квадратного уравнения $ax^2 + bx + c = 0$ для любых коэффициентов, в том числе и нулевых.

```
class QuadraticEquation {
public static void main(String[] args){
```

```
    double a = 0.5, b = -2.7, c = 3.5, d, eps=le-8;
    if (Math.abs(a) < eps)
    if (Math.abs(b) < eps)
    if (Math.abs(c) < eps) // Все коэффициенты равны
нулю
        System.out.println("Решение —любое число");
    else
        System.out.println("Решений нет");
    else
        System.out.println("xl = x2 = " +(-c / b) ) ;
    else{ // Коэффициенты не равны нулю
    if((d = b*b — 4*a*c)< 0.0){ // Комплексные корни
        d = 0.5 * Math.sqrt(-d) / a;
        a = -0.5 * b/ a;
        System.out.println("xl = " +a+ " +i " +d+
        ",x2 = " +a+ " -i " +d);
                                }
    else{
    // Вещественные корни
    d =0.5 * Math.sqrt(d) / a;
    a = -0.5 * b / a;
    System.out.println("x1 = " + (a + d) + ", x2 = "
+(a - d));}
    }
    }
```

В этой программе использованы методы вычисления модуля `abs()` и квадратного корня `sqrt()` вещественного числа из встроенного в Java API класса Math. Поскольку все вычисления с вещественными числами производятся приближенно, мы считаем, что коэффициент уравнения равен нулю, если его модуль меньше 0,00000001. Обратите внимание на то, как в методе println() используется сцепление строк, и на то, как операция присваивания при вычислении дискриминанта вложена в логическое выражение.

Оператор выбора

Следующая разновидность оператора, которую нам надо изучить, называется **оператор выбора**. Он предназначен для той же цели, что и условный оператор - для выполнения одного из фрагментов программы в зависимости от некоторых условий. Он поддерживает условия весьма специального вида, но может анализировать большое количество таких условий в одном операторе.

Оператор выбора анализирует значение целочисленного выражения и позволяет задавать последовательности операторов, выполняемых в случае различных конкретных значений этого выражения. Также можно задавать последовательность операторов, выполняемую в том случае, если значение выражения не совпадает ни с одним из явно предложенных вариантов.

Синтаксис (форма записи) оператора выбора следующий:

```
Switch (выражение) { тело }
```

Тело по структуре похоже на содержимое обычного составного оператора; оно также представляет собой последовательность операторов, но в нем между операторами присутствуют специальные конструкции вида `case число:`, а также не более одной конструкции `default:`. В конструкциях `case` допустимы только константы целого типа; в них нельзя использовать переменные.

Работает оператор выбора следующим образом: сначала вычисляется выражение, и ищется конструкция case с константой, равной значению выражения. Если такая есть, выполняются операторы из тела, начиная с оператора, следующего за найденной конструкцией `case`. Если такой нет, но есть конструкция `default`, выполнение начинается со следующего за ней оператора. Наконец, если подходящего варианта `case` и конструкции `default` нет, оператор выбора ничего не делает.

К сожалению, в одном варианте `case` можно указывать только одно конкретное значение; нельзя указать несколько значений через запятую или указать диапазон значений, т. е. указать начальное и конечное значения, имея в виду также все значения, расположенные между ними.

Если какая-то последовательность операторов должна выполняться при нескольких разных значениях выражения, единственный вариант — поставить перед ней несколько конструкций case.

Оператор выбора в языке Java, как и в C++, имеет одну особенность — конструкции case определяют только место начала выполнения операторов из тела оператора выбора. При этом операторы выполняются до тех пор, пока либо не кончится тело, либо не встретится оператор `break;` (специальный оператор, используемый для того, чтобы досрочно прекращать выполнение определенных управляющих конструкций, в число которых попал и оператор выбора). Попадающиеся при этом «по дороге» конструкции `case` и `default` просто игнорируются, поэтому если разные варианты должны быть взаимоисключающими, не нужно забывать ставить оператор `break;` перед конструкциями `case`.

Пример. По целому числу — параметру командной строки нужно напечатать словесное представление оценки. Текст метода «main» приведен ниже:

```
public static void main(String[] args) {
switch(Integer.parselnt(args[0])) {
case 5: System.out.println("excellent"); break;
case 4: System.out.println("good"); break;
case 3: System.out.println("satisfactory"); break;
case 2:System.out.println("unsatisfactory"); break;
case 1: System.out.println("bad"); break;
default:System.out.println("Unrecognized mark.");
}}
```

Важным частным случаем целочисленных выражений являются символьные. Для ввода и анализа символьных данных необходимо работать со строками. Завести переменную типа строка и заполнить ее строкой, введенной с клавиатуры, можно, написав следующее:

```
String str = S.next();
```

где `S` — переменная типа `Scanner`. Получить теперь символ такой строки с заданным номером можно, написав `str.charAt(номер-1)`.

Чтобы проанализировать первый символ строки `str` при помощи оператора выбора, нужно написать

```
switch(str.charAt(0)) {
case 'A': операторы_варианта_A
case 'B': операторы_варианта_B

default: операторы_по_умолчанию
}
```

Естественно, символьные константы под `case` могут быть любыми; приведенные здесь 'A' и 'B' взяты только для примера.

Аналогично, чтобы проанализировать второй символ строки `str`, нужно заменить в выражении под `switch` индекс 0 на 1, и т. д.

Увы, анализировать строки целиком оператор выбора не умеет, т. е. выражение под `switch` не может иметь результат типа строка, равно как и под case нельзя использовать строковые константы.