Конспект лекций по Java. Занятие 4

В.Фесюнов

1. Операции (operators) в языке Java

Большинство операций Java просты и интуитивно понятны. Это такие операции, как +, -, *, /, <,> и др. Операции имеют свой порядок выполнения и приоритеты. Так в выражении

a + b * c

сначала выполняется умножение, а потом сложение, поскольку приоритет у операции умножения выше, чем у операции сложения. В выражении

a + b — c

сначала вычисляется a + b, а потом от результата вычитается c, поскольку порядок выполнения этих операций слева направо.

Но операции Java имеют и свои особенности. Не вдаваясь в детальное описание простейших операций, остановимся на особенностях.

Начнем с **присваивания** . В отличии от ряда других языков программирования в Java присваивание — это не оператор, а операция. Семантику этой операции можно описать так.

- Операция присваивания обозначается символом '='. Она вычисляет значение своего правого операнда и присваивает его левому операнду, а также выдает в качестве результата присвоенное значение. Это значение может быть использовано другими операциями. Последовательность из нескольких операций присваивания выполняется справа налево.

В простейшем случае все выглядит как обычно.

x = a + b;

Здесь происходит именно то, что мы интуитивно подразумеваем, — вычисляется сумма a и b, результат заносится в x. Но вот два других примера.

a = b = 1; a+b;

В первом сначала 1 заносится в b, результатом операции является 1, потом этот результат заносится в a. Во втором примере вычисляется сумма a и b и результат теряется. Это бессмысленно, но синтаксически допустимо.

Операции сравнения

Это операции >, <, >=, <=, != и ==. Следует обратить внимание, что сравнение на равенство обозначается двумя знаками '='. Операндами этих операций могут быть арифметические данные, результат — типа boolean.

Операции инкремента, декремента

Это операции ++ и --. Так y++ (инкремент) является сокращенной записью y = y +1, аналогично и с операцией декремента (--).

Но с этими операциями есть одна тонкость. Они существуют в двух формах - префиксной (++y) и постфиксной (y++). Действие этих операций одно и то же — они увеличивают (операции декремента — уменьшают) свой операнд на 1, а вот результат у них разный. Префиксная форма в качестве результата выдает уже измененное на 1 значение операнда, а постфиксна -значение операнда до изменения.

```
a = 5;
x = a++;
y = ++a;
```

В этом фрагменте x получит значение 5, а y — 7.

Операция целочисленного деления

Нужно учитывать, что деление одного целого на другое выдает целое, причем не округляет, а отбрасывает дробную часть.

Остаток от деления (значение по модулю)

В Java имеется операция %, которая обозначает остаток от деления.

Расширенные операции присваивания

Кроме обычной операции '=' в Java существуют операции +=, -=, *=, /= и др. Это сокращенные записи. Так a += b полностью эквивалентна a = a + b. Аналогично и с другими такими операциями.

Логические операции

```
!      — отрицание
&&     — логическое 'и'
||     — логическое 'или'
```

Операнды этих операций должны быть типа boolean, результат — boolean. Операции && и || имеют одну особенность — их правый операнд может и не вычислиться, если результат уже известен по левому операнду. Так, если левый операнд операции && — ложь (false), то правый операнд вычисляться не будет, т.к. результат все равно — ложь.

Это свойство нужно учитывать, особенно тогда, когда правый операнд содержит вызов некоторой функции.

Побитовые логические операции

Это операции

```
&   — побитовое 'и'
|   — побитовое 'или'
^   — побитовое 'исключающее или'
~   — побитовое отрицание
```

Они выполняются для каждой пары битов своих операндов.

Операции сдвига

```
<<    — сдвиг влево
>>    — сдвиг вправо
>>>   — беззнаковый сдвиг вправо
```

Эти операции сдвигают значение своего левого операнда на число бит, заданное правым операндом.

Условная операци

Это единственная тернарная операция, т.е. операция, имеющая три операнда. Соответственно, для нее используется не один знак операции, а два.

```
<условие> ? <выражение1> : < выражение2>
```

Если <условие> истинно, то результатом будет < выражение1>, иначе < выражение2>.

Например, "a < b ? a : b" вычисляет минимум из a и b.

Операция приведения типов

Это очень важная операция. По умолчанию все преобразования, которые могут привести к проблемам, в Java запрещены. Так, нельзя long-значение присвоить int-операнду. В тех случаях, когда это все же необходимо, нужно поставить явное преобразование типа.

Например, пусть метод f(...) выдает long.

int x = (int)f(10);

Здесь (int) — это операция преобразования типа. Операция преобразования типа обозначается при помощи имени типа, взятого в скобки.

Эта операция применима не только к базовым типам, но и к классам. Мы разберем это подробнее, когда будем рассматривать наследование.

2. Литералы (константы)

2.1. Арифметические

Примеры арифметических констант

```
- 10
- 010 — это 8
- 0123 — это 83  (1*64 + 2*8 + 3)
- 0x10 — это 16
- 0x123 — это 291 (1*256 + 2*16 +3)
- 1e5 — это 100000
- 1.23e-3 — это 0.00123
```

Для указания типа константы применяются суффиксы: l (или L) — long, f (или F) — float, d (или D) — double. Например,

1L — единица, но типа long.

2.2. Логические литералы

Логические литералы — это true (истина) и false (ложь)

2.3. Строковые литералы

Записываются в двойных кавычках, например

"это строка"

2.4. Символьные литералы

Записываются в апострофах, например 'F', 'ш'.

В строковых и символьных литералах есть правила для записи спец. символов. Во-первых, есть набор предопределенных спец. символов. Это

```
- '\n' — конец строки (перевод строки)
- '\r' — возврат каретки
- '\t' — табуляция
```

и ряд других.

Во-вторых, можно явно записать код символа (нужно только знать его). Запись кода обычно выполняется в восьмеричной системе: '\001' — символ с кодом 1 и т.п.

3. Операторы (statements)

3.1. Оператор — выражение

Синтаксис

<выражение>;

Такой оператор состоит из одного выражения, в конце стоит ';'. Его действие состоит в вычислении выражения, значение, вычисленное данным выражением, теряется. Т.е. обычно такое выражение содержит операцию присваивания, хотя это по синтаксису не обязательно.

Примеры

```
a = 0;
x = (a > b ? a : b);
cnt++;
```

3.2. Условный оператор (if)

Синтаксис

```
if ( <условие> )
       <оператор1>
[else
       <оператор2>]
```

Здесь <условие> — это логическое выражение, т.е. выражение, возвращающее true или false. Как видно из синтаксиса, часть else вляется необязательной. После if и после else стоит по одному оператору. Если нужно поместить туда несколько операторов, то нужно поставить **блок** . **Блок** начинается с '{' и заканчивается '}'.

В Java принято блок ставить всегда, даже если после if или else стоит один оператор.

Примеры

```
if ( a > b ) {
    x = a;
} else {
    x = b;
}
```

```
if ( flag ) {
    flag = false;
    init();
}
```

В последнем примере flag — логическая переменная или поле, init() -метод, вызываемый, если флаг равен true (говорят, "если flag установлен").

3.3. Оператор return (уже рассматривали)

3.4. Оператор цикла по предусловию (while)

Синтаксис

```
while ( <условие> )
        <оператор>
```

Как и в случае оператора if, в Java принято <оператор> заключать в фигурные скобки.

Пример

```
int i = 0;
while ( more ) {
    x /= 2;
```

```
    more = (++i < 10);
}
```

В этом примере more должна быть логической переменной или полем, x — некоторой арифметической переменной или полем. Цикл выполняется 10 раз.

3.5. Оператор цикла по постусловию (do while)

Синтаксис

```
do
        <оператор>
while ( <условие> );
```

Оператор цикла по постусловию отличается от оператора цикла по предусловию только тем, что в нем виток цикла выполняется всегда как минимум один раз, в то время как в операторе по предусловию может не быть ни одного витка цикла (если условие сразу ложно).

Пример

```
int i = 0;
do {
    x \= 2;
    more = (++i < 10);
} while (more );
```

Содержательно, данный пример эквивалентен предыдущему.

3.6. Оператор цикла "со счетчиком" (for)

Синтаксис

```
for (<инициализация>; <условие>; <инкремент> )
```

```
    <оператор>
```

Заголовок такого цикла содержит три выражения (в простейшем случае). Из наименования оператора можно понять, что он служит для организации цикла со счетчиком. Поэтому выражение <инициализация> выполняется один раз перед первым витком цикла. После каждого витка цикла выполняется выражение <инкремент>, а потом выражение <условие>. Последнее выражение должно быть логическим и служит для задания условия продолжения цикла. Т.е. пока оно истинно, витки цикла будут продолжаться.

Для удобства составления операторов цикла со счетчиком в данной конструкции введено множество расширений.

- <инициализация> может быть не выражением, а описанием с инициализацией типа "int i = 0".
- <инициализация> может быть списком выражений через запятую, например, "i = 0, r = 1" .
- <инкремент> тоже может быть списком выражений, например, "i++, r*=2"
- Все составляющие (<инициализация>, <условие> и <инкремент>) являются необязательными. Для выражения <условие> это означает, что условие считается всегда истинным (т.е. выход из цикла должен быть организован какими-то средствами внутри самого цикла).

Т.е. допустим и такой цикл (бесконечный цикл):

```
for (;;) {
   . . .
}
```

Пример

```
for (int i = 0; i < 10; i++)
    x /=2;
```

Это самая "экономичная" реализация цикла из предыдущих примеров.

3.7. Операторы break и continue

В Java нет операторов goto. Как известно, goto приводит к появлению неструктурированных программ. Операторы break и continue являются структурированными аналогами goto.

Они могут применяться в циклах, а break еще и в операторе выбора (switch). Выполнение оператора break приводит к немедленному завершению цикла. Оператор continue вызывает окончание текущего витка цикла и начало нового. Проверка условия в этом случае все же выполняется, так что continue может вызвать и окончание цикла.

Пример

```
for ( int i = 0; ;i++ ) {
    if ( oddOnly && i%2 == 0 )
          continue;
    y = (x + 1)/x;
    if ( y – x < 0.001 )
          break;
    x = y;
}
```

Здесь oddOnly — логическая переменная. Если она установлена, то все витки цикла с четными номерами пропускаются с использованием оператора continue;

Условие окончания цикла в данном примере проверяется в середине цикла и, если оно выполнено, то цикл прекращается при помощи оператора break.

При вложенных циклах операторы break и continue могут относиться не только к тому циклу, в котором они расположены, но и к охватывающему циклу. Для этого охватывающий оператор цикла должен быть помечен меткой, которая должна быть указана в операторе break или continue. Например,

```
lbl: while ( ... ) {
   . . .
   for ( ... ) {
      . . .
```

```
     if ( ... )
         break lbl;
   }
   . . .
}
```

Здесь оператор break вызовет прекращение как цикла for, так и while.

3.8. Оператор выбора (switch)

Служит для организации выбора по некоторому значению одной из нескольких ветвей выполнения.

Синтаксис

```
switch ( <выражение> ) {
       case <константа1>:
             <операторы1>
       case <константа2>:
             <операторы2>
       . . .
       [default:
             <операторы_D>]
}
```

Выражение должно выдавать целочисленное или символьное значение, константы должны быть того же типа, что и значение этого выражения.

Элементы "case <константа>:" являются метками перехода, если значение выражения совпадает с константой, то будет осуществлен переход на эту метку. Если значение выражения не совпадает ни с одной из констант, то все зависит от наличия фрагмента default. Если он есть, то переход происходит на метку default, если его нет, то весь оператор switch пропускается.

- В операторе switch фрагменты case не образуют какие-либо блоки. Если после последнего оператора данного case-фрагмента стоит следующий case, то выполнение будет продолжено, начиная с первого оператора этого case-фрагмента.

- В силу этого в операторе switch обычно применяется оператор break. Он ставится в конце каждого case-фрагмента.

Пример (файл SymbolTest.java)

Рассмотрим демонстрационную программу, в которой использованы операторы for и switch.

Эта программа генерирует случайным образом 100 символов латинского алфавита и классифицирует их как "гласные", "согласные" и "иногда гласные". В последнюю категорию отнесены символы 'y' и 'w'.

```
public class SymbolTest {

    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");
            }
        }

    }

}
```

В данном примере есть несколько новых для нас элементов.

- Используется метод random() класса Math. Посмотрим документацию по классу Math и разберемся, что он делает.

- В операторе

```
char c = (char)(Math.random()*26 + 'a');
```

  производится сложение арифметического значения с символом. При таком сложении в Java символ преобразуется в число, которое равно коду этого символа.
- В операторе

```
System.out.print(c + ": ");
```

  используется не println(...), а print(...). Метод print(...) отличается от println(...) только тем, что не переводит печать на новую строку, так что следующий оператор print(...) или println(...) продолжит печать в той же строке.

Следует также обратить внимание на фрагменты case. Формально здесь 7 таких фрагментов, но 5 из них не содержат никаких операторов. Так что, можно считать, что здесь 2 case-фрагмента, но каждый из них имеет несколько меток case.

4. Задание на дом

- **1** Изменить SymbolTest.java так, чтобы количество генерируемых символов задавалось параметром вызова программы.
- **2** Написать программу, которая в качестве параметров вызова принимает два числа - длины катетов прямоугольного треугольника, а в качестве результата печатает углы в градусах.