Модуль 3

Управляющие структуры

Важно!..

- Ввод символов с клавиатуры
- Общий вид выражения if
- Использование выражения switch
- Цикл for
- Цикл while
- Цикл do-while
- Использование оператора break для выхода из цикла
- Использование оператора break в качестве оператора goto
- Использование оператора continue
- Вложенные циклы

В данном модуле вы ознакомитесь с выражениями, которые позволяют управлять последовательностью выполнения программных инструкций. Существуют три категории управляющих выражений: операторы выбора, к которым относятся `if` и `switch`, операторы итераций, включающие в себя циклы `for`, `while` и `do-while`, и операторы перехода, `break`, `continue` и `return`. Все эти выражения, за исключением `return`, о котором речь пойдет далее в этой книге, будут подробно рассмотрены в данном модуле. Начнем же мы с обсуждения простых средств организации ввода данных с клавиатуры.

Ввод символов с клавиатуры

Перед тем как приступать к изучению управляющих выражений Java, уделим немного внимания средствам, которые позволят вам писать интерактивные программы. До сих пор в примерах, рассмотренных в этой книге, информация отображалась, но пользователь не имел возможности вводить данные. Так, мы применяли консольный вывод, но не консольный ввод. Причина того, что мы до сих пор не уделяли внимание получению данных, заключается в том, что средства ввода информации базируются на достаточно сложной системе классов. Чтобы использовать их, необходимо иметь определенные знания, например представлять себе обработку исключений и работу с классами. Эти вопросы мы обсудим далее в этой книге. Если метод `println()` использовать несложно, то подобного, достаточно простого метода, предназначенного для ввода данных пользователем, попросту не существует. Механизм консольного ввода в языке Java сложен и освоить его гораздо труднее, чем может показаться на первый взгляд. Но и необходимость в нем возникает сравнительно редко. В большинстве реальных Java-программ и аплетов используется оконный графический интерфейс. По этой причине в данной книге консольному вводу уделяется минимальное внимание. Однако есть задача, связанная с консольным вводом, решить которую достаточно просто. Это чтение символов с клавиатуры. Поскольку ввод символов используется в нескольких примерах, приведенных в данном модуле, мы обсудим этот вопрос сейчас.

Проще всего прочитать символ с клавиатуры, вызвав метод `System.in.read()`. `System.in` можно рассматривать как дополнение к `System.out`. Это объект ввода, связанный с клавиатурой. Метод `read()` ожидает нажатия клавиш пользователем, после чего возвращает результат. Символ представляется в виде целого числа, поэтому, чтобы присвоить его символьной переменной, надо выполнить явное приведение к типу `char`. По умолчанию данные, вводимые с консоли, буферизуются, поэтому они становятся доступны программе лишь после того, как пользователь нажмет клавишу <Enter>. Ниже приведен код программы, читающей символы, введенные с клавиатуры.

```
// Чтение символа с клавиатуры.
class KbIn {
  public static void main(String args[])
        throws java.io.IOException {
    char ch;

    System.out.print("Press a key followed by ENTER:");

    // Процесс получения символа, введенного с клавиатуры.
    ch = (char) System.in.read();

    System.out.println("Your key is:" + ch);
  }
}
```

Ниже приведен пример сеанса работы с программой.

```
Press a key followed by ENTER: t
Your key is: t
```

Обратите внимание на то, что метод `main()` начинается следующими строками:

```
public static void main(String args[])
      throws java.io.IOException {
```

Поскольку в программе используется метод `System.in.read()`, нужно включить в нее выражение `throws java.io.IOException`. Это выражение необходимо для обработки ошибок, которые могут возникнуть в процессе ввода. Оно является частью механизма обработки исключений Java, который мы рассмотрим в модуле 9. Сейчас же не обращайте на данное выражение внимания и не задумывайтесь о его назначении.

Буферизация ввода в `System.in` часто приводит к недоразумениям. При нажатии клавиши <Enter> во входной поток записывается последовательность, состоящая из символов возврата каретки и перевода строки. Эти символы ждут обработки. Для некоторых выражений может понадобиться удалить возврат каретки и перевод строки из входного потока, перед тем как переходить к следующей операции. Для этого достаточно прочитать их. С примерами подобных решений вы ознакомитесь далее в этом модуле.

Вопросы для текущего контроля[1]

1. Что такое `System.in`?
2. Как прочитать символ, введенный с клавиатуры?

[1] 1. `System.in` — это объект ввода, связанный со стандартным входным потоком, в роли которого обычно выступает клавиатура.

2. Для того чтобы прочитать символ, надо вызвать метод `System.in.read( )`.

Оператор if

В модуле 1 вы ознакомились с выражением `if`. Сейчас мы подробно обсудим его. Полностью формат оператора `if` выглядит следующим образом:

```
if(условие) выражение;
else выражение;
```

Фрагмент `else` может отсутствовать. В данном примере предполагается, что после `if(`*условие*`)` и после `else` присутствует одно выражение. Однако вместо выражения можно использовать и блок. Формат записи оператора `if`, в котором применяются программные блоки, приведен ниже.

```
if(выражение)
{
  последовательность выражений
}
else
{
 последовательность выражений
}
```

Если выражение, заданное в качестве условия, принимает значение `true`, выполняется выражение или программный блок, указанный после `if`, в противном случае выполняется выражение или блок, следующий за `else`. Ситуация, в которой выполнились бы и выражение после `if`, и выражение после `else`, возникнуть не может. Условное выражение в операторе `if` должно возвращать значение типа `boolean`.

Для того чтобы продемонстрировать действия оператора `if` (а также некоторых других управляющих выражений), разработаем простую игру, основанную на угадывании. Возможно, она понравится вашим детям. В первой версии игры программа предложит пользователю угадать задуманную букву в диапазоне от A до Z. Если пользователь правильно угадает букву и нажмет на клавиатуре соответствующую клавишу, программа выведет сообщение `** Right **`. Код программы приведен ниже.

```
// Программа угадывания буквы.
class Guess {
  public static void main(String args[])
      throws java.io.IOException {

    char ch, answer = 'K';

    System.out.println("I'm thinking of a letter between A and Z.");
    System.out.print("Can you guess it:");

    ch = (char) System.in.read(); // Чтение символа с клавиатуры.

    if(ch == answer) System.out.println("** Right **");
```

```
  }
}
```

Данная программа отображает для пользователя сообщение, а затем читает символ с клавиатуры. Используя выражение `if`, она сравнивает символ с задуманным ответом (в данном случае это символ K). Если введена буква K, то отображается сообщение об успехе. Используя программу, не забывайте, что символ K надо ввести в верхнем регистре.

В следующей версии программы используется выражение `else`, предназначенное для вывода сообщения о том, что пользователь не угадал букву:

```
// Программа угадывания буквы, вторая версия.
class Guess2 {
  public static void main(String args[])
      throws java.io.IOException {

    char ch, answer = 'K';

    System.out.println("I'm thinking of a letter between A and Z.");
    System.out.print("Can you guess it:");

    ch = (char) System.in.read(); // Получение символа

    if(ch == answer) System.out.println("** Right **");
    else System.out.println("...Sorry, you're wrong.");
  }
}
```

Вложенные операторы if

Вложенные операторы `if` — это языковая конструкция, в которой после `if` (*условие*) или после `else` указан оператор `if`. Подобные конструкции очень часто встречаются в программах. Используя их, надо помнить, что в языке Java `else` всегда относится к ближайшему оператору `if`, находящемуся в том же блоке, за исключением тех случаев, когда этому оператору уже соответствует `else`. Рассмотрим следующий пример:

```
if(i == 10) {
  if(j < 20) a = b;
  if(k > 100) c = d;
  else a = c; // Это выражение else относится к if(k > 100).
}
else a = d; // Это выражение else относится к if(i == 10).
```

Как следует из комментариев, последнее `else` не имеет отношения к `if(j < 20)`, поскольку не находится с ним в одном блоке (несмотря на то, что этот оператор `if` — ближайший, не имеющий `else`). Следовательно, последнее `else` относит-

ся к `if(i == 10)`. Внутри блока `else` связано с `if(k > 100)`, поскольку этот оператор — самый близкий из всех `if`, находящихся в том же блоке.

Используя вложенные операторы `if`, можно модифицировать игру, рассматриваемую здесь в качестве примера. Теперь пользователю в случае неудачной попытки предоставляется дополнительная информация.

```
// Программа угадывания буквы, третья версия.
class Guess3 {
  public static void main(String args[])
      throws java.io.IOException {

    char ch, answer = 'K';

    System.out.println("I'm thinking of a letter between A and Z.");
    System.out.print("Can you guess it: ");

    ch = (char) System.in.read(); // Получение символа

    if(ch == answer) System.out.println("** Right **");
    else {
      System.out.print("...Sorry, you're ");

      // Вложенный оператор if
      if(ch < answer) System.out.println("too low");
      else System.out.println("too high");
    }
  }
}
```

Пример сеанса работы с программой выглядит следующим образом:

```
I'm thinking of a letter between A and Z.
Can you guess it: Z
...Sorry, you're too high
```

Цепочка if-else-if

На практике часто используется цепочка `if-else-if`, базирующаяся на вложенных операторах `if`. Эта цепочка выглядит следующим образом:

```
if(условие)
  выражение;
else if(условие)
  выражение;
else if(условие)
  выражение;
.
.
```

```
.
else
  выражение;
```

Условные выражения вычисляются “сверху вниз”. Как только одно из условий вернет значение `true`, выполнится соответствующее выражение и остальная часть цепочки будет пропущена. Если значения всех условий равны `false`, то выполнится выражение, следующее за последним `else`. Последнее `else` выполняет роль условия по умолчанию; соответствующее ему выражение получает управление лишь тогда, когда проверка всех условий окончилась неудачей. Если последнее `else` отсутствует, значение `false` всех условий приведет к тому, что ни одно действие выполнено не будет.

Ниже приведен текст программы, демонстрирующей использование цепочки `if-else-if`.

```
// использование цепочки if-else-if.
class Ladder {
  public static void main(String args[]) {
    int x;

    for(x=0; x<6; x++) {
      if(x==1)
        System.out.println("x is one");
      else if(x==2)
        System.out.println("x is two");
      else if(x==3)
        System.out.println("x is three");
      else if(x==4)
        System.out.println("x is four");
      else
        // Выражение по умолчанию.
        System.out.println("x is not between 1 and 4");
    }
  }
}
```

В процессе работы программа генерирует следующие выходные данные:

```
x is not between 1 and 4
x is one
x is two
x is three
x is four
x is not between 1 and 4
```

Нетрудно заметить, что выражение, следующее за последним `else`, выполняется лишь тогда, когда проверка всех условий окончилась неудачей.

Вопросы для текущего контроля[2]

1. Каким должен быть тип условного выражения, управляющего выполнением оператора `if`?
2. Какому оператору `if` соответствует компонент `else`?
3. Что такое цепочка `if-else-if`?

Оператор switch

Помимо `if`, к операторам выбора в языке Java относится также `switch`. Этот оператор реализует множественное ветвление, предоставляя возможность выбора между несколькими альтернативными вариантами. Несмотря на то что многократные проверки можно реализовать и с помощью вложенных операторов `if`, во многих случаях использование `switch` является более эффективным решением. Данный оператор работает следующим образом: значение выражения проверяется на совпадение с несколькими предопределенными константами. При совпадении с одной из констант выполняется последовательность выражений, предусмотренная для этого случая. Формат записи оператора `switch` выглядит следующим образом:

```
switch(выражение) {
  case константа_1:
    последовательность выражений
    break;
  case константа_2:
    последовательность выражений
    break;
  case константа_3:
    последовательность выражений
    break;
...
  default:
    последовательность выражений
}
```

[2] 1. Выражение, управляющее выполнением оператора `if`, должно возвращать значение типа `boolean`.

2. Выражение `else` всегда соответствует ближайшему оператору `if`, расположенному в том же блоке, за исключением случаев, когда этот оператор имеет свой компонент `else`.

3. Цепочка `if-else-if` — это последовательность вложенных выражений `if-else`.

Выражение, используемое в операторе `switch`, может возвращать значение типа `char`, `byte`, `short` или `int`. Другие типы, например значения с плавающей точкой, недопустимы. Чаще всего выражение, управляющее работой `switch`, состоит из одной переменной. Константа, указанная после ключевого слова `case`, должна быть литералом типа, совместимого с типом выражения. В одном операторе `switch` не должны встречаться одинаковые константы.

Последовательность выражений, заданная после ключевого слова `default`, выполняется в том случае, если значение выражения не совпало ни с одной из констант. Элемент `default` может отсутствовать, в этом случае, если проверки оканчиваются неудачей, никакие действия не выполняются. Если обнаружено совпадение выражения с одной из констант, начинает выполняться последовательность выражений, следующая за соответствующим элементом `case`. Выражения выполняются одно за другим до тех пор, пока не встретится оператор `break`. Если управление получили выражения, следующие за ключевым словом `default` или за последним элементом `case`, то выполнение продолжится до конца оператора `switch`.

Ниже приведен код программы, демонстрирующей использование оператора `switch`.

```
// Пример использования оператора switch.
class SwitchDemo {
  public static void main(String args[]) {
    int i;

    for(i=0; i<10; i++)
      switch(i) {
        case 0:
          System.out.println("i is zero");
          break;
        case 1:
          System.out.println("i is one");
          break;
        case 2:
          System.out.println("i is two");
          break;
        case 3:
          System.out.println("i is three");
          break;
        case 4:
          System.out.println("i is four");
          break;
        default:
          System.out.println("i is five or more");
      }
  }
}
```

Выходные данные, генерируемые этой программой, имеют следующий вид:

```
i is zero
i is one
i is two
i is three
i is four
i is five or more
i is five or more
i is five or more
i is five or more
i is five or more
```

Как видите, на каждой итерации выполняется выражение, указанное после элемента `case`, который содержит константу, совпадающую со значением выражения. Остальные элементы оператора `switch` не получают управления. Когда значение `i` равно или больше пяти, оно не совпадает ни с одной из констант, поэтому управление получает выражение, следующее за ключевым словом `default`.

В принципе оператор `break` может отсутствовать, но, как правило, в реальных приложениях он используется. При выполнении оператора `break` оператор `switch` завершает работу и управление передается выражению, следующему за ним. Если последовательность выражений, следующая за ключевым словом `case`, не содержит оператора `break`, то все выражения выполняются, после чего начинают выполняться выражения, соответствующие следующему элементу `case`. Это продолжается до тех пор, пока не встретится оператор `break` или не будет достигнут конец оператора `switch`.

Рассмотрим в качестве примера следующую программу. Сможете ли вы предсказать, как будут выглядеть данные, выведенные этой программой на экран?

```
// Выполнение оператора switch, не содержащего оператора break.
class NoBreak {
  public static void main(String args[]) {
    int i;

    for(i=0; i<=5; i++) {
      switch(i) {
        case 0:
          // Это и все последующие выражения будут выполняться
          // одно за другим.
          System.out.println("i is less than one");
        case 1:
          System.out.println("i is less than two");
        case 2:
          System.out.println("i is less than three");
        case 3:
          System.out.println("i is less than four");
        case 4:
```

```
            System.out.println("i is less than five");
        }
        System.out.println();
    }
  }
}
```

Данные, генерируемые в процессе работы программы, выглядят следующим образом:

```
i is less than one
i is less than two
i is less than three
i is less than four
i is less than five

i is less than two
i is less than three
i is less than four
i is less than five

i is less than three
i is less than four
i is less than five

i is less than four
i is less than five

i is less than five
```

Как вы уже знаете, при отсутствии оператора `break` управление передается выражениям, соответствующим следующему элементу `case`.

При необходимости вы можете использовать пустые элементы `case`, как показано в следующем примере:

```
switch(i) {
  case 1:
  case 2:
  case 3: System.out.println("i is 1, 2 or 3");
    break;
  case 4: System.out.println("i is 4");
    break;
}
```

В данном фрагменте кода, если переменная i имеет значение 1, 2 или 3, вызывается первый метод `println()`. Если значение равно 4, вызывается второй метод `println()`. Последовательность пустых элементов `case` часто используется тогда, когда нескольким таким элементам должен соответствовать один и тот же набор выражений.

Вложенные операторы switch

Оператор `switch` может быть одним из выражений, входящих в другой оператор `switch`. Таким образом, формируются вложенные операторы данного типа. При этом конфликты не возникают, даже если константа в составе элемента `case` внешнего оператора совпадает с такой константой во внутреннем операторе. Например, следующий фрагмент кода является допустимым:

```
switch(ch1) {
  case 'A': System.out.println("This A is part of outer switch.");
    switch(ch2) {
      case 'A':
        System.out.println("This A is part of inner switch");
        break;
      case 'B': // ...
    } // Окончание внутреннего оператора switch
    break;
  case 'B': // ...
```

Вопросы для текущего контроля[3]

1. Какого типа должно быть выражение, управляющее поведением оператора `switch`?
2. Что происходит, когда выражение, используемое в операторе `switch`, совпадает с константой, входящей в состав одного из элементов `case`?
3. Что происходит в случае, если последовательность выражений, следующих за элементом `case`, не оканчивается оператором `break`?

---

[3] 1. Выражение, управляющее поведением оператора `switch`, может иметь тип `char`, `short`, `int` или `byte`.

2. Когда выражение, используемое в операторе `switch`, совпадает с константой, входящей в состав одного из элементов `case`, управление получает выражение, следующее за этим элементом `case`.

3. Если последовательность выражений, следующих за элементом `case`, не оканчивается оператором `break`, то начинают выполняться выражения, находящиеся после следующего элемента `case` (если таковой имеется).

Проект 3.1.

`Help.java`

В рамках данного проекта мы создадим простую справочную систему, предоставляющую сведения о синтаксисте управляющих выражений Java. Данная программа отображает меню, содержащее названия операторов и ожидает выбора одного из них. После того, как пользователь выберет один из пунктов, выводятся сведения о синтаксисе соответствующего выражения. Первая версия программы предоставляет сведения только об операторах `if` и `switch`. В последующих проектах будут добавлены остальные выражения.

Последовательность действий

1. Создайте файл `Help.java`.
2. В начале работы программы отображается следующее меню.

```
Help on:^n  1. if
  2. switch
Choose one:
```

   Для того, чтобы вывести эту информацию потребуются выражения, представленные ниже.

```
System.out.println("Help on:");
System.out.println("  1. if");
System.out.println("  2. switch");
System.out.print("Choose one: ");
```

3. Далее программа получает сведения о выборе пользователя. Для этого вызывается метод `System.in.read()`.

```
choice = (char) System.in.read();
```

4. После этого вступает в действие оператор `switch`, посредством которого отображаются сведения о выбранном выражении.

```
switch(choice) {
  case '1':
    System.out.println("The if:\n");
    System.out.println("if(condition) statement;");
    System.out.println("else statement;");
    break;
  case '2':
    System.out.println("The switch:\n");
    System.out.println("switch(expression) {");
    System.out.println(" case constant:");
    System.out.println(" statement sequence");
    System.out.println(" break;");
    System.out.println(" // ...");
    System.out.println("}");
```

```
    break;
  default:
    System.out.print("Selection not found.");
}
```

Обратите внимание на то, что сведения о неправильном выборе отображают выражение, соответствующее элементу `default`. Например, если пользователь введет значение 3, оно не совпадет ни с одной из констант, и управление будет передано коду, соответствующему элементу `default`.

5. Код программы, содержащейся в файле `Help.java`, имеет следующий вид:

```
/*
  Проект 3.1.

  Простая справочная система.
*/
class Help {
  public static void main(String args[])
      throws java.io.IOException {
    char choice;

    System.out.println("Help on:");
    System.out.println("  1. if");
    System.out.println("  2. switch");
    System.out.print("Choose one: ");
    choice = (char) System.in.read();

    System.out.println("\n");

    switch(choice) {
      case '1':
        System.out.println("The if:\n");
        System.out.println("if(condition) statement;");
        System.out.println("else statement;");
        break;
      case '2':
        System.out.println("The switch:\n");
        System.out.println("switch(expression) {");
        System.out.println(" case constant:");
        System.out.println(" statement sequence");
        System.out.println(" break;");
        System.out.println(" // ...");
        System.out.println("}");
        break;
      default:
        System.out.print("Selection not found.");
    }
  }
```

6. Данные, выводимые программой, выглядят так:

```
Help on:
  1. if
  2. switch
Choose one: 1

The if:

if(condition) statement;
else statement;
```

Спросим у опытного программиста

Вопрос. В каких случаях следует вместо оператора `switch` использовать цепочку `if-else-if`?

Ответ. Цепочка `if-else-if` уместна в тех случаях, когда проверка условий не сводится к выяснению совпадения или несовпадения выражения с одиночным значением. Рассмотрим, например, следующую последовательность `if-else-if`:

```
if(x < 10) // ...
else if(y != 0) // ...
else if(!done) // ...
```

В данном случае поставленную задачу не удастся решить с помощью оператора `switch`, поскольку в трех условиях используются разные переменные и разные типы сравнений. Цепочку `if-else-if` также целесообразно использовать при проверке значений с плавающей точкой и других объектов, типы которых отличаются от типов, предусмотренных для выражения `switch`.

Цикл for

Простая форма цикла `for` встречалась вам еще в модуле 1. Ознакомившись детально с данным циклом, вы увидите, насколько это гибкий инструмент и какие обширные возможности он предоставляет. Начнем рассмотрение цикла `for` с его основных, наиболее часто используемых вариантов.

Формат записи цикла `for` приведен ниже.

`for(`*инициализационное_выражение; условие; выражение_итерации*`)` *выражение;*

Вместо одного выражения также может использоваться блок кода.

```
for(инициализационное_выражение; условие; выражение_итерации)
{
  последовательность выражений
}
```

Инициализационное выражение обычно задает исходное значение переменной цикла. Эта переменная, как правило, используется в качестве счетчика. Условие — это логическое выражение, определяющее, до каких пор должно повторяться тело цикла. Итерационное выражение в большинстве случаев изменяет переменная цикла при очередном повторении на определенную величину. Обратите внимание на то, что три элемента в круглых скобках должны разделяться точкой с запятой. Выполнение цикла `for` продолжается до тех пор, пока значение логического выражения, указанного в качестве условия, равно `true`. Как только это выражение примет значение `false`, цикл завершается и выполнение программы продолжится, начиная с выражения, следующего за оператором `for`.

Ниже приведен код программы, в которой цикл `for` используется для вывода значений квадратного корня чисел в интервале от 1 до 99. В данной программе также отображается ошибка округления, допущенная при вычислении квадратного корня.

```
// Отображение квадратного корня чисел от 1 до 99
// и ошибки округления.
class SqrRoot {
  public static void main(String args[]) {
    double num, sroot, rerr;

    for(num = 1.0; num < 100.0; num++) {
      sroot = Math.sqrt(num);
      System.out.println("Square root of " + num +
                         " is " + sroot);

      // Вычисление ошибки округления.
      rerr = num - (sroot * sroot);
      System.out.println("Rounding error is " + rerr);
      System.out.println();
    }
  }
}
```

Заметьте, что ошибка округления вычисляется путем возведения в квадрат квадратного корня числя. Полученное значение отнимается от исходного числа.

Переменная цикла может как увеличиваться, так и уменьшаться, а величина приращения может выбираться произвольно. Например, в приведенном ниже

фрагменте кода выводятся числа от 100 до –95, а на каждом шаге переменная цикла уменьшается на 5.

```
// Цикл for с уменьшающейся переменной цикла.
class DecrFor {
  public static void main(String args[]) {
    int x;

    // Переменная цикла уменьшается на 5 при каждой итерации
    for(x = 100; x > -100; x -= 5)
      System.out.println(x);
  }
}
```

Важной особенностью циклов `for` является тот факт, что выражение, выбранное в качестве условия, вычисляется перед началом каждой итерации. Это означает, что если изначально значение выражения равно `false`, код в теле цикла ни разу не выполнится. Это иллюстрирует следующий пример:

```
for(count=10; count < 5; count++)
  x += count; // Это выражение не получит управления.
```

Тело цикла не выполняется, так как переменная цикла `count` изначально больше пяти. В результате при вычислении выражения условия `count < 5` получается значение `false` и цикл завершается, не выполнив ни одной итерации.

Некоторые разновидности цикла for

Цикл `for` — одна из самых гибких языковых конструкций Java; он имеет большое число разновидностей. Например, вы можете использовать несколько переменных цикла. Рассмотрим следующую программу:

```
// Использование запятых в цикле for.
class Comma {
  public static void main(String args[]) {
    int i, j;

    // В данном случае используются две переменные цикла.
    for(i=0, j=10; i < j; i++, j--)
      System.out.println("i and j: " + i + " " + j);
  }
}
```

Выходные данные программы выглядят следующим образом:

```
i and j: 0 10
i and j: 1 9
i and j: 2 8
```

```
i and j: 3 7
i and j: 4 6
```

В рассматриваемой программе запятыми разделяются два инициализационных выражения и два выражения итерации. В начале цикла инициализируется как i, так и и j. При каждой новой итерации значение i инкрементируется, а значение j декрементируется. В ряде случаев использование нескольких переменных цикла очень удобно и позволяет упростить алгоритм решения задачи. В одном цикле может присутствовать произвольное количество инициализационных выражений и выражений итерации, но на практике, если цикл содержит более двух или трех таких выражений, он выглядит слишком громоздким.

Условием завершения цикла может быть любое корректное логическое выражение. В нем не обязательно должна присутствовать переменная цикла. В следующем примере тело цикла выполняется до тех пор, пока пользователь не введет с клавиатуры букву S:

```
// Выполнение цикла до появления символа S.
class ForTest {
  public static void main(String args[])
      throws java.io.IOException {

    int i;

    System.out.println("Press S to stop.");

    for(i = 0; (char) System.in.read() != 'S'; i++)
      System.out.println("Pass #" + i);
  }
}
```

Пропуск элементов в цикле for

В программах могут использовать циклы `for`, в которых некоторые элементы отсутствуют. В языке Java разрешается не указывать инициализационное выражение, условие, выражение итерации и даже все эти элементы одновременно. Рассмотрим в качестве примера следующую программу:

```
// Некоторые элементы цикла for могут быть пустыми.
class Empty {
  public static void main(String args[]) {
    int i;

    // Выражение итерации отсутствует.
    for(i = 0; i < 10; ) {
      System.out.println("Pass #" + i);
```

```
      i++; // Иекрементирование переменной цикла
    }
  }
}
```

В данном случае в цикле `for` отсутствует выражение итерации. Вместо этого переменная цикла `i` изменяется в теле цикла. Это означает, что в начале очередной итерации переменная `i` проверяется на равенство значению 10, но никакие другие действия не предпринимаются. Поскольку `i` инкрементируется в теле цикла, цикл выполняется корректно и программа выводит следующие выходные данные:

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

В следующем примере выражение инициализации вынесено за пределы цикла `for`:

```
// Вынос элементов за пределы цикла.
class Empty2 {
  public static void main(String args[]) {
    int i;

    i = 0; // Инициализационное выражение за пределами цикла.
    for(; i < 10; ) {
      System.out.println("Pass #" + i);
      i++; // increment loop control var
    }
  }
}
```

В данном случае `i` инициализируется до начала цикла. В принципе наиболее разумным решением является инициализация переменной цикла в рамках самого цикла. За пределы цикла инициализацию следует выносить только в том случае, если она связана со сложными вычислениями и соответствующий фрагмент кода невозможно или неудобно включить в выражение `for`.

Бесконечный цикл

С помощью оператора `for` можно создать бесконечный цикл (цикл, который никогда не завершится). Для этого надо использовать пустое выражение усло-

вия. Например, большинство программистов, использующих язык Java, формируют бесконечный цикл так, как показано ниже.

```
for(;;) // Бесконечный цикл.
{
  //...
}
```

Это цикл никогда не завершается. Несмотря на то что существуют задачи, для решения которых необходим бесконечный цикл (в качестве примера можно привести командный процессор операционной системы), в подавляющем большинстве случаев в таких циклах предусмотрены средства для их прерывания. В конце данного модуля вы увидите, как осуществляется остановка бесконечного цикла (для этой цели используется оператор `break`).

Отсутствующее тело цикла

В языке Java тело цикла (`for` или любого другого) может быть пустым. Дело в том, что пустое выражение, или выражение `null`, считается допустимым. Циклы, тело которых отсутствует, часто оказываются полезными. Так, например, в приведенной ниже программе в цикле производится сложение чисел от 1 до 5.

```
// Тело цикла может быть пустым.
class Empty3 {
  public static void main(String args[]) {
    int i;
    int sum = 0;

    // Несмотря на то что тело цикла отсутствует,
    // в цикле производится суммирование чисел.
    for(i = 1; i <= 5; sum += i++) ;

    System.out.println("Sum is " + sum);
  }
}
```

Выходные данные программы выглядят следующим образом:

```
Sum is 15
```

Заметьте, что процесс суммирования полностью происходит в рамках выражения `for` и тело цикла попросту не нужно. Обратите особое внимание на выражение итерации.

```
sum += i++
```

Поначалу оно может показаться непонятным, но со временем вы увидите, насколько удобно применять подобные приемы. Они часто встречаются в профессиональных Java-программах. Разобраться в данном выражении несложно, если

разделить его на составные части. В данном случае переменной sum присваивается значение sum плюс i, после чего i увеличивается на единицу. Тот же результат даст приведенная ниже последовательность выражений.

```
sum = sum + i;
i++;
```

Объявление переменной цикла в составе выражения for

Очень часто переменная, управляющая выполнением цикла for, используется исключительно для этой цели; за пределами цикла она не нужна. Существует возможность объявить переменную цикла в инициализационном выражении. Например, в приведенном ниже фрагменте кода производится суммирование чисел от 1 до 5 и вычисление факториала. Переменная цикла объявляется в выражении for.

```
// Объявление переменной цикла в составе цикла for.
class ForVar {
  public static void main(String args[]) {
    int sum = 0;
    int fact = 1;

    // Вычисление факториала.
    // Переменная i объявляется в цикле for.
    for(int i = 1; i <= 5; i++) {
      sum += i; // i is known throughout the loop
      fact *= i;
    }

    // В данной точке программы переменная i не определена.

    System.out.println("Sum is " + sum);
    System.out.println("Factorial is " + fact);
  }
}
```

Объявляя переменную в выражении for, необходимо помнить, что область видимости этой переменной ограничивается текущим циклом. За пределами цикла переменная не существует. Таким образом, в предыдущем примере после закрывающей фигурной скобки, которой оканчивается тело цикла, переменная i не определена. Если вам необходимо использовать переменную цикла в других частях программы, ее нельзя объявлять в выражении for.

Перед тем как перейти к изучению дальнейшего материала, желательно поэкспериментировать с разновидностями цикла for. Как видите, эта языковая конструкция предоставляет обширные возможности и очень удобна в использовании.

Расширенный цикл for

С недавних пор в распоряжение разработчиков Java-программ предоставлен так называемый расширенный цикл `for`, обеспечивающий специальные средства для перебора объектов из набора, например из массива. Расширенный цикл `for` будет обсуждаться в модуле 5, когда вы ознакомитесь с массивами.

Вопросы для текущего контроля[4]

1. Могут ли элементы цикла `for` быть пустыми?
2. Как сформировать с помощью выражения `for` бесконечный цикл?
3. Какова область видимости переменной, объявленной в составе выражения `for`?

Цикл while

Помимо цикла `for`, в языке Java также поддерживается цикл `while`. Формат записи оператора `while` выглядит следующим образом:

`while(`*условие*`)` *выражение;*

В качестве тела цикла может использоваться как одно выражение, так и программный блок. Условие управляет выполнением цикла и должно представлять собой корректное логическое выражение. Цикл выполняется до тех пор, пока значение выражения-условия равно `true`. Когда это выражение принимает значение `false`, управление передается выражению, следующему после цикла.

Ниже приведен простой пример использования цикла `while` для вывода латинского алфавита.

```
// Пример использования цикла while.
class WhileDemo {
  public static void main(String args[]) {
    char ch;

    // Вывод латинского алфавита с помощью цикла while.
    ch = 'a';
```

4 1. Да. Все три элемента цикла `for` — выражение инициализации, условие и выражение итерации — могут быть пустыми.

2. `for(;;)`

3. Область видимости переменной, объявленной в цикле `for`, ограничена самим циклом. За его пределами эта переменная не существует.

```
    while(ch <= 'z') {
      System.out.print(ch);
      ch++;
    }
  }
}
```

Переменная ch инициализируется кодом буквы a. На каждой итерации значение ch выводится, а затем увеличивается на единицу. Процесс продолжается до тех пор, пока значение ch не станет больше кода буквы z.

Как и в случае цикла for, условие проверяется в начале цикла, а это значит, что возможна ситуация, при которой тело цикла ни разу не выполнится. Такое поведение цикла while избавляет разработчика от необходимости осуществлять специальную проверку до начала цикла. Ниже приведен код программы, иллюстрирующий данное свойство цикла. Эта программа вычисляет степени двойки: от нулевой до девятой.

```
// Вычисление степеней двойки.
class Power {
  public static void main(String args[]) {
    int e;
    int result;

    for(int i=0; i < 10; i++) {
      result = 1;
      e = i;
      while(e > 0) {
        result *= 2;
        e--;
      }

      System.out.println("2 to the " + i +
                         " power is " + result);
    }
  }
}
```

Выходные данные программы выглядят следующим образом:

```
2 to the 0 power is 1
2 to the 1 power is 2
2 to the 2 power is 4
2 to the 3 power is 8
2 to the 4 power is 16
2 to the 5 power is 32
2 to the 6 power is 64
2 to the 7 power is 128
2 to the 8 power is 256
2 to the 9 power is 512
```

Заметьте, что в данном случае цикл `while` выполняется только тогда, когда `e` больше 0. Таким образом, если `e` равно нулю (как это имеет место на первой итерации цикла `for`), цикл `while` пропускается.

Спросим у опытного программиста

Вопрос. В языке Java циклы обладают большой гибкостью. Каков в этих условиях критерий выбора цикла? Другими словами, как выбрать цикл, наиболее подходящий для решения конкретной задачи?

Ответ. Если число итераций известно заранее, лучше использовать цикл `for`. Цикл `do-while` подходит тогда, когда необходимо, чтобы была выполнена как минимум одна итерация. Цикл `while` наиболее целесообразен тогда, когда число повторений тела цикла неизвестно заранее.

Цикл do-while

Последний из циклов, используемых в языке Java, — это `do-while`. В отличие от циклов `for` и `while`, в которых условие проверяется перед выполнением тела цикла, `do-while` проверяет условие в конце каждой итерации. Это означает, что тело цикла будет выполнено по крайней мере один раз. Формат записи выражения `do-while` выглядит следующим образом:

```
do {
 выражение;
} while(условие);
```

Если тело цикла состоит из одного выражения, то фигурные скобки необязательны, однако в большинстве случаев программисты используют их, чтобы сделать код программы более удобочитаемым. Цикл `do-while` выполняется до тех пор, пока выражение, указанное в качестве условия, равно `true`.

В приведенном ниже фрагменте кода цикл выполняется, пока пользователь не введет букву `q`.

```
// Пример использования цикла do-while.
class DWDemo {
  public static void main(String args[])
    throws java.io.IOException {

    char ch;

    do {
      System.out.print("Press a key followed by ENTER: ");
      ch = (char) System.in.read(); // get a char
```

```
    } while(ch != 'q');
  }
}
```

Используя цикл `do-while`, мы можем модернизировать игру, созданную в начале данного модуля. На этот раз выполнение цикла будет продолжаться до тех пор, пока пользователь не угадает букву.

```
// Четвертая версия игры угадывания буквы.
class Guess4 {
  public static void main(String args[])
      throws java.io.IOException {

    char ch, answer = 'K';

    do {
      System.out.println("I'm thinking of a letter between A and Z.");
      System.out.print("Can you guess it: ");

      // Чтение буквы (возврат каретки и перевод строки игнорируются).
      do {
        ch = (char) System.in.read(); // Получение символа.
      } while(ch == '\n' | ch == '\r');

      if(ch == answer) System.out.println("** Right **");
      else {
        System.out.print("...Sorry, you're ");
        if(ch < answer) System.out.println("too low");
        else System.out.println("too highv);
        System.out.println("Try again!\n");
      }
    } while(answer != ch);
  }
}
```

Данные, выводимые программой, выглядят так:

```
I'm thinking of a letter between A and Z.
Can you guess it: A
...Sorry, you're too low
Try again!

I'm thinking of a letter between A and Z.
Can you guess it: Z
...Sorry, you're too high
Try again!

I'm thinking of a letter between A and Z.
Can you guess it: K
** Right **
```

Обратите внимание на интересную особенность данной программы. Для получения очередной буквы используется цикл `do-while`, посредством которого игнорируются символы возврата каретки и перевода строки, которые могут присутствовать во входном потоке.

```
// Чтение буквы (возврат каретки и перевод строки игнорируются).
do {
  ch = (char) System.in.read(); // Получение символа.
} while(ch == '\n' | ch == '\r');
```

Именно по этой причине нужен цикл. Как было сказано ранее, `System.in` — это буферизированный входной поток, и введенные с клавиатуры символы становятся доступны лишь после нажатия клавиши <Enter>. А при нажатии этой клавиши во входной поток помещаются символы возврата каретки и перевода строки. Цикл `do-while` позволяет избавиться от них.

Вопросы для текущего контроля[5]

1. В чем состоит основное отличие между циклами `while` и `do-while`?
2. Выражение, управляющее выполнением цикла `while`, может быть любого типа. Да или нет?

Проект 3.2.

`Help2.java`

В рамках данного проекта мы расширим справочную систему Java, созданную нами в проекте 3.1. В этой версии программы будет добавлена информация о циклах `for`, `while` и `do-while`. Кроме того, мы реализуем проверку действий пользователя, работающего с меню; цикл будет повторяться до тех пор, пока пользователь не введет допустимое значение.

Последовательность действий

1. Скопируйте файл `Help.java` и сохраните его под именем `Help2.java`.
2. Измените часть программы, ответственную за отображение вариантов, предлагаемых пользователю на выбор. Реализуйте ее с помощью циклов

[5] 1. В цикле `while` условие проверяется в начале цикла. В цикле `do-while` эта проверка выполняется в конце цикла. Таким образом, тело цикла `do-while` будет выполнено по крайней мере один раз.

2. Нет. Условие должно иметь тип `boolean`.

так, как показано ниже.

```
do {
  System.out.println("Help on:");
  System.out.println(" 1. if");
  System.out.println(" 2. switch");
  System.out.println(" 3. for");
  System.out.println(" 4. while");
  System.out.println(" 5. do-while\n");
  System.out.print("Choose one: ");
  do {
    choice = (char) System.in.read();
  } while(choice == '\n' | choice == '\r');
} while( choice < '1' | choice > '5');
```

Заметьте, что вложенные циклы `do-while` используются для того, чтобы избавиться от нежелательных символов возврата каретки и перевода строки, которые могут присутствовать во входном потоке. После внесения данных изменений программа будет отображать меню в цикле до тех пор, пока пользователь не введет значение в интервале от 1 до 5.

3. Дополните выражение `switch` с тем, чтобы учесть циклы `for`, `while` и `do-while`.

```
switch(choice) {
  case '1':
    System.out.println("The if:\n");
    System.out.println("if(condition) statement;");
    System.out.println("else statement;");
    break;
  case '2':
    System.out.println("The switch:\n");
    System.out.println("switch(expression) {");
    System.out.println(" case constant:");
    System.out.println(" statement sequence");
    System.out.println(" break;");
    System.out.println(" // ...");
    System.out.println("}");
    break;
  case '3':
    System.out.println("The for:\n");
    System.out.print("for(init; condition; iteration)");
    System.out.println(" statement;");
    break;
  case '4':
    System.out.println("The while:\n");
```

```
      System.out.println("while(condition) statement;");
      break;
    case '5':
      System.out.println("The do-while:\n");
      System.out.println("do { ");
      System.out.println(" statement;");
      System.out.println("} while (condition);");
      break;
}
```

Заметьте, что в данном варианте выражения `switch` отсутствует элемент `default`. Поскольку меню повторно предлагается пользователю до тех пор, пока он не введет допустимые данные, нет необходимости в обработке некорректных значений.

4. Полностью код программы, содержащейся в файле `Help2.java`, имеет следующий вид:

```
/*
  Проект 3.2.

   Модифицированная справочная система, использующая цикл
   do-while для обработки действий пользователя.
*/
class Help2 {
  public static void main(String args[])
      throws java.io.IOException {
    char choice;

    do {
      System.out.println("Help on:");
      System.out.println(" 1. if");
      System.out.println(" 2. switch");
      System.out.println(" 3. for");
      System.out.println(" 4. while");
      System.out.println(" 5. do-while\n");
      System.out.print("Choose one: ");
      do {
        choice = (char) System.in.read();
      } while(choice == '\n' | choice == '\r');
    } while( choice < '1' | choice > '5');

    System.out.println("\n");

    switch(choice) {
      case '1':
```

```
        System.out.println("The if:\n");
        System.out.println("if(condition) statement;");
        System.out.println("else statement;");
        break;
      case '2':
        System.out.println("The switch:\n");
        System.out.println("switch(expression) {");
        System.out.println(" case constant:");
        System.out.println(" statement sequence");
        System.out.println(" break;");
        System.out.println(" // ...");
        System.out.println("}");
        break;
      case '3':
        System.out.println("The for:\n");
        System.out.print("for(init; condition; iteration)");
        System.out.println(" statement;");
        break;
      case '4':
        System.out.println("The while:\n");
        System.out.println("while(condition) statement;");
        break;
      case '5':
        System.out.println("The do-while:\n");
        System.out.println("do {");
        System.out.println(" statement;");
        System.out.println("} while (condition);");
        break;
    }
  }
}
```

Использование оператора break для прерывания цикла

По мере необходимости выполнение цикла можно немедленно прекратить, запретив к тому же передачу управления последующим операторам, содержащимся в теле цикла. Для этой цели используется выражение `break`. Если оно встречается в цикле, то цикл завершается и управление передается выражению, следующему после цикла. Простой пример использования оператора `break` приведен ниже.

```
// Использование оператора break для выхода из цикла.
class BreakDemo {
```

```
  public static void main(String args[]) {
    int num;

    num = 100;

    // Цикл продолжается до тех пор, пока квадрат i
    // меньше значения num
    for(int i=0; i < num; i++) {
      if(i*i >= num) break; // Цикл завершается, если i*i >= 100
      System.out.print(i + " ");
    }
    System.out.println("Loop complete.");
  }
}
```

Данные, генерируемые в процессе работы программы, выглядят следующим образом:

```
0 1 2 3 4 5 6 7 8 9 Loop complete.
```

Несмотря на то что в цикле `for` предусмотрен перебор значений от 0 до `num` (в данном случае до 100), выражение `break` завершает цикл раньше, а именно в тот момент, когда значение `i` становится больше или равным квадратному корню из `num`.

Выражение `break` может быть использовано в любом цикле, предусмотренном в языке Java, в том числе в бесконечном. Например, в приведенном ниже фрагменте производится чтение из входного потока до тех пор, пока пользователь не введет символ `q`.

```
// Чтение из входного потока до получения символа q.
class Break2 {
  public static void main(String args[])
      throws java.io.IOException {

    char ch;

    // Бесконечный цикл, завершаемый с помощью оператора break.
    for( ; ; ) {
      ch = (char) System.in.read(); // Получение символа
      if(ch == 'q') break;
    }
    System.out.println("You pressed q!");
  }
}
```

Если выражение `break` встречается во вложенных циклах, оно завершает лишь текущий цикл. Например:

```
// Использование оператора break во вложенных циклах.
class Break3 {
  public static void main(String args[]) {
```

```
    for(int i=0; i<3; i++) {
      System.out.println("Outer loop count: " + i);
      System.out.print(" Inner loop count: ");

      int t = 0;
      while(t < 100) {
        if(t == 10) break; // Если t равно 10, цикл завершается
        System.out.print(t + " ");
        t++;
      }
      System.out.println();
    }
    System.out.println("Loops complete.");
  }
}
```

Выходные данные программы выглядят так:

```
Outer loop count: 0
    Inner loop count: 0 1 2 3 4 5 6 7 8 9
Outer loop count: 1
    Inner loop count: 0 1 2 3 4 5 6 7 8 9
Outer loop count: 2
    Inner loop count: 0 1 2 3 4 5 6 7 8 9
Loops complete.
```

Как видите, оператор `break` во внутреннем цикле вызывает завершение только этого цикла. На выполнение включающих циклов он не оказывает влияния.

Следует сделать два замечания, касающихся оператора `break`. Во-первых, в цикле может присутствовать несколько выражений `break`. Однако при этом следует соблюдать осторожность. Слишком часто встречающиеся операторы `break` могут нарушить структуру кода, и программа станет трудной для восприятия. Во-вторых, если оператор `break` встречается в выражении `switch`, он воздействует только на это выражение и не затрагивает включающие циклы.

Оператор break в качестве оператора goto

Помимо выражения `switch` и циклов, оператор `break` может быть использован как "цивилизованный" вариант оператора `goto`. В языке Java оператор `goto` отсутствует. Причина в том, что возможность произвольного перехода в любую точку программы способствует созданию плохо структурированного кода. Программы, в которых часто используется оператор `goto`, обычно сложны для восприятия и поддержки. Однако в некоторых случаях оператор `goto` может оказаться полезен. Например, его удобно использовать для экстренного выхода из многократно вложенных циклов. Для решения подробных задач в языке Java определена

расширенная форма оператора `break`. Используя этот вариант выражения `break`, вы можете, например, выйти за пределы одного или нескольких блоков кода. Эти блоки должны быть частью циклов или оператора `switch`. Кроме того, вам надо явно указать точку, с которой должно быть продолжено выполнение программы; для этого в расширенной форме оператора `break` предусмотрена метка. Как вы вскоре увидите, оператор `break` позволяет воспользоваться всеми преимуществами оператора `goto` и в то же время избежать проблем, присущих последнему.

Оператор `break` с меткой имеет следующий формат:

```
break метка;
```

Обычно метка — это имя, идентифицирующее блок кода. При выполнении расширенного оператора `break` управление передается за пределы именованного блока кода. Оператор `break` с меткой может содержаться непосредственно в именованном блоке кода или в одном из блоков, входящих в состав именованного блока. Таким образом, вы можете использовать этот вариант `break` для выхода из набора вложенных блоков. Однако данное языковое средство не позволяет передать управление в блок кода, не содержащий оператор `break`.

Для того чтобы присвоить блоку имя, надо поставить перед ним метку. Именовать можно как независимый блок, так и блок, входящий в какое-либо выражение. Роль метки может выполнять любой допустимый идентификатор Java, сопровождаемый двоеточием. Пометив блок, вы можете использовать метку в качестве целевой в выражении `break`. При выполнении такого оператора `break` управление будет передано в конец именованного блока. Например, в приведенном ниже фрагменте кода используются три вложенных блока.

```
// Использование оператора break с меткой.
class Break4 {
  public static void main(String args[]) {
    int i;

    for(i=1; i<4; i++) {
one:  {
two:    {
three:    {
            System.out.println("\ni is " + i);

            if(i==1) break one;
            if(i==2) break two;
            if(i==3) break three;

            // Следующая строка кода
            // никогда не получит управление
            System.out.println("won't print");
          }
```

```
        System.out.println("After block three.");
      }
      System.out.println("After block two.");
    }
    System.out.println("After block one.");
  }
  System.out.println("After for.");
 }
}
```

Выходные данные программы выглядят следующим образом:

```
i is 1
After block one.

i is 2
After block two.
After block one.

i is 3
After block three.
After block two.
After block one.
After for.
```

Рассмотрим подробнее приведенную выше программу, чтобы лучше понять, каким образом генерируются именно такие выходные данные. Когда в переменной `i` содержится значение 1, условие первого выражения `if` становится истинным и управление передается в конец блока с меткой `one`. В результате выводится сообщение `After block one`. Если `i` равно 2, то успешно завершается проверка во втором выражении `if` и выполнение программы продолжается с конца блока с меткой `two`. В результате выводятся сообщения `After block two` и `After block one`. Когда `i` принимает значение 3, истинным становится выражение в третьем операторе `if`. Управление передается в конец блока с меткой `three`. В этом случае выводятся все три сообщения.

Теперь рассмотрим еще один пример. На этот раз оператор `break` будет использован для выхода за пределы нескольких вложенных циклов. Когда во внутреннем цикле выполняется выражение `break`, управление передается в конец блока, соответствующего внешнему циклу; этот блок помечен меткой `done`. В результате происходит выход из всех трех циклов.

```
// Очередной пример использования оператора break с меткой.
class Break5 {
  public static void main(String args[]) {

done:
    for(int i=0; i<10; i++) {
      for(int j=0; j<10; j++) {
```

```
        for(int k=0; k<10; k++) {
          System.out.println(k + " ");
          if(k == 5) break done; // Переход по метке done.
        }
        System.out.println("After k loop"); // Не должно выполняться.
      }
      System.out.println("After j loop"); // Не должно выполняться.
    }
    System.out.println("After i loop");
  }
}
```

Выходные данные программы приведены ниже.

```
0
1
2
3
4
5
After i loop
```

Расположение метки очень важно, особенно когда речь идет о циклах. Рассмотрим в качестве примера следующий фрагмент кода:

```
// Позиция метки имеет большое значение.
class Break6 {
  public static void main(String args[]) {
    int x=0, y=0;

// Здесь метка расположена перед выражением.
stop1: for(x=0; x < 5; x++) {
         for(y = 0; y < 5; y++) {
           if(y == 2) break stop1;
           System.out.println("x and y: " + x + " " + y);
         }
       }

       System.out.println();

// Теперь метка расположена непосредственно
// перед открывающей фигурной скобкой
       for(x=0; x < 5; x++)
stop2:   {
           for(y = 0; y < 5; y++) {
             if(y == 2) break stop2;
             System.out.println("x and y: " + x + " " + y);
           }
         }

  }
}
```

В процессе работы программа отображает приведенные ниже данные.

```
x and y: 0 0
x and y: 0 1

x and y: 0 0
x and y: 0 1
x and y: 1 0
x and y: 1 1
x and y: 2 0
x and y: 2 1
x and y: 3 0
x and y: 3 1
x and y: 4 0
x and y: 4 1
```

Наборы вложенных циклов в этой программе идентичны за исключением одной детали. В первом наборе метка предшествует внешнему циклу `for`. Тогда при выполнении оператора `break` управление передается в конец всего выражения `for`, причем оставшиеся итерации внешнего цикла пропускаются. Во втором наборе метка находится перед открывающей фигурной скобкой блока, соответствующего телу внешнего цикла. В результате при выполнении выражения `break stop2` управление передается в конец тела внешнего цикла `for`, после чего выполняется очередная итерация.

Помните, что вы не имеете права использовать в операторе `break` метку, не определенную во включающем цикле. Например, приведенный ниже фрагмент кода некорректен и не будет компилироваться.

```
// Данный код содержит ошибку.
class BreakErr {
  public static void main(String args[]) {

    one: for(int i=0; i<3; i++) {
      System.out.print("Pass " + i + ": ");
    }

    for(int j=0; j<100; j++) {
      if(j == 10) break one; // WRONG
      System.out.print(j + " ");
    }
  }
}
```

Поскольку блок, помеченный меткой `one`, не содержит выражения `break`, управление не может быть передано этому блоку.

Спросим у опытного программиста

Вопрос. Вы говорили, что оператор `goto` нарушает структуру программы и что оператор `break` с меткой является гораздо лучшим решением. Но ведь переход в конец внешнего цикла, т.е. в точку, далеко отстоящую от оператора `break`, также ухудшает восприятие программы.

Ответ. Вы совершенно правы. Однако если явный переход все-таки необходим, то передача управления в конец блока сохраняет хоть какое-то подобие структуры, чего нельзя сказать об операторе `goto`.

Использование оператора continue

Существует возможность преждевременно завершить очередную итерацию цикла, нарушив нормальный ход выполнения команд. Это позволяет сделать оператор `continue`. Данное выражение вызывает принудительный переход на следующую итерацию цикла, причем код между оператором `continue` и выражением, проверяющим условие завершения цикла, игнорируется. Таким образом, `continue` является своеобразным дополнением оператора `break`. Например, в следующей программе оператор `continue` применяется для того, чтобы обеспечить вывод четных чисел в диапазоне от 0 до 100:

```
// Использование оператора continue.
class ContDemo {
  public static void main(String args[]) {
    int i;

    // Вывод четных чисел в интервале от 0 до 100.
    for(i = 0; i<=100; i++) {
      if((i%2) != 0) continue;
      System.out.println(i);
    }
  }
}
```

В данном случае выводятся только четные числа. Это происходит потому, что нечетные числа вызывают преждевременное завершение очередной итерации и метод `println()` не вызывается.

В циклах `while` и `do-while` выражение `continue` вызывает переход непосредственно к выражению проверки условия, после чего выполнение цикла продолжается. В случае цикла `for` проверяется завершение цикла и вычисляется выражение итерации, после чего цикл продолжается.

Подобно выражению `break`, в операторе `continue` может присутствовать метка, описывающая, на какой из включающих циклов этот оператор должен воздействовать. Ниже приведен пример программы, использующий оператор `continue` с меткой.

```
// Использование оператора continue с меткой.
class ContToLabel {
  public static void main(String args[]) {

outerloop:
    for(int i=1; i < 10; i++) {
      System.out.print("\nOuter loop pass " + i +
                       ", Inner loop: ");
      for(int j = 1; j < 10; j++) {
        if(j == 5) continue outerloop; // Выражение continue
                                       // для внешнего цикла.
        System.out.print(j);
      }
    }
  }
}
```

Выходные данные программы выглядят следующим образом:

```
Outer loop pass 1, Inner loop: 1234
Outer loop pass 2, Inner loop: 1234
Outer loop pass 3, Inner loop: 1234
Outer loop pass 4, Inner loop: 1234
Outer loop pass 5, Inner loop: 1234
Outer loop pass 6, Inner loop: 1234
Outer loop pass 7, Inner loop: 1234
Outer loop pass 8, Inner loop: 1234
Outer loop pass 9, Inner loop: 1234
```

Как видно из данного примера, при выполнении оператора `continue` управление передается внешнему циклу и оставшиеся итерации внутреннего цикла пропускаются.

Ситуации, в которых оператор `continue` абсолютно необходим, встречаются редко. Однако данный оператор — часть богатого набора средств управления циклами, который предоставляется разработчику, использующему язык Java. Этот оператор удобно применять в тех случаях, когда необходимо преждевременно завершить очередную итерацию.

Вопросы для текущего контроля[6]

1. Что произойдет, если в теле цикла будет выполнен оператор `break` (без метки)?
2. Что происходит при выполнении оператора `break` с меткой?
3. Какие действия выполняет оператор `continue`?

Проект 3.3.

`Help3.java`

В рамках данного проекта мы завершим работу над справочной системой Java, начатую нами в предыдущих проектах. В данной версии будет учтен синтаксис операторов `break` и `continue`. Она также позволит пользователю запрашивать данные о синтаксисе нескольких выражений. Это достигается путем добавления внешнего цикла, который выполняется до тех пор, пока пользователь не введет вместо номера пункта меню символ q.

Последовательность действий

1. Скопируйте файл `Help2.java` и сохраните его под именем `Help3.java`.
2. Поместите весь имеющийся код программы в бесконечный цикл `for`. Выход из этого цикла будет осуществляться посредством оператора `break`, который получит управление тогда, когда пользователь введет букву q. Поскольку цикл включает в себя весь код, выход из этого цикла означает завершение программы.
3. Измените цикл работы с меню так, как показано ниже.

```
do {
  System.out.println("Help on:");
  System.out.println(" 1. if");
  System.out.println(" 2. switch");
```

6 1. Оператор break без метки в составе цикла взывает преждевременное прекращение этого цикла. Выполнение программы продолжается со строки кода, следующей за циклом.

2. При выполнении оператора break с меткой управление передается в конец помеченного блока.

3. Выражение `continue` вызывает немедленный переход к следующей итерации цикла; код, оставшийся в теле цикла, игнорируется. Если в составе оператора `continue` присутствует метка, то продолжается помеченный цикл.

```
    System.out.println(" 3. for");
    System.out.println(" 4. while");
    System.out.println(" 5. do-while");
    System.out.println(" 6. break");
    System.out.println(" 7. continue\n");
    System.out.print("Choose one (q to quit): ");
    do {
      choice = (char) System.in.read();
    } while(choice == '\n' | choice == '\r');
  } while( choice < '1' | choice > '7' & choice != 'q');
```

Заметьте, что теперь цикл включает выражения `break` и `continue`. Кроме того, символ q является допустимым.

4. Дополните выражение `switch` с тем, чтобы учесть операторы `break` и `continue`.

```
case '6':
  System.out.println("The break:\n");
  System.out.println("break; or break label;");
  break;
case '7':
  System.out.println("The continue:\n");
  System.out.println("continue; or continue label;");
  break;
```

5. Полностью код программы, содержащейся в файле `Help3.java`, имеет следующий вид:

```
/*
  Проект 3.3.

  Окончательный вариант справочной системы, поддерживающей
  многократные запросы.
*/
class Help3 {
  public static void main(String args[])
      throws java.io.IOException {
    char choice;

    for(;;) {
      do {
        System.out.println("Help on:");
        System.out.println(" 1. if");
        System.out.println(" 2. switch");
        System.out.println(" 3. for");
        System.out.println(" 4. while");
        System.out.println(" 5. do-while");
```

```
    System.out.println(" 6. break");
    System.out.println(" 7. continue\n");
    System.out.print("Choose one (q to quit): ");
    do {
      choice = (char) System.in.read();
    } while(choice == '\n' | choice == '\r');
  } while( choice < '1' | choice > '7' & choice != 'q');

  if(choice == 'q') break;

  System.out.println("\n");

  switch(choice) {
    case '1':
      System.out.println("The if:\n");
     System.out.println("if(condition) statement;");
      System.out.println("else statement;");
      break;
    case '2':
      System.out.println("The switch:\n");
      System.out.println("switch(expression) {");
      System.out.println(" case constant:");
      System.out.println(" statement sequence");
      System.out.println(" break;");
      System.out.println(" // ...");
      System.out.println("}");
      break;
    case '3':
      System.out.println("The for:\n");
     System.out.print("for(init; condition;
                  iteration)");
      System.out.println(" statement;");
      break;
    case '4':
      System.out.println("The while:\n");
      System.out.println("while(condition)
                  statement;");
      break;
    case '5':
      System.out.println("The do-while:\n");
      System.out.println("do {");
      System.out.println(" statement;");
      System.out.println("} while (condition);");
      break;
```

```
        case '6':
          System.out.println("The break:\n");
          System.out.println("break; or break label;");
          break;
        case '7':
          System.out.println("The continue:\n");
          System.out.println("continue; or continue
                             label;");
          break;
      }
      System.out.println();
    }
  }
}
```

6. Данные, выводимые программой, выглядят так:

```
Help on:
  1. if
  2. switch
  3. for
  4. while
  5. do-while
  6. break
  7. continue

Choose one (q to quit): 1

The if:

if(condition) statement;
else statement;

Help on:
  1. if
  2. switch
  3. for
  4. while
  5. do-while
  6. break
  7. continue

Choose one (q to quit): 6

The break:
```

```
break; or break label;

Help on:
  1. if
  2. switch
  3. for
  4. while
  5. do-while
  6. break
  7. continue

Choose one (q to quit): q
```

Вложенные циклы

Как было видно из предыдущих примеров, один цикл может быть вложен в другой. С помощью вложенных циклов решаются самые разнообразные задачи. Поэтому, перед тем как завершить разговор о циклах в языке Java, уделим дополнительное внимание вложенным циклам. Ниже приведен код программы, содержащей вложенные циклы `for`. С помощью этих циклов для каждого числа от 2 до 100 определяется набор значений, на которые данное число делится без остатка.

```
/*
  Использование вложенных циклов для нахождения делителей
  чисел в диапазоне от 2 до 100.
*/
class FindFac {
  public static void main(String args[]) {

    for(int i=2; i <= 100; i++) {
      System.out.print("Factors of " + i + ": ");
      for(int j = 2; j < i; j++)
        if((i%j) == 0) System.out.print(j + " ");
      System.out.println();
    }
  }
}
```

Ниже приведен фрагмент выходных данных, генерируемых программой.

```
Factors of 2:
Factors of 3:
Factors of 4: 2
Factors of 5:
Factors of 6: 2 3
Factors of 7:
Factors of 8: 2 4
```

```
Factors of 9: 3
Factors of 10: 2 5
Factors of 11:
Factors of 12: 2 3 4 6
Factors of 13:
Factors of 14: 2 7
Factors of 15: 3 5
Factors of 16: 2 4 8
Factors of 17:
Factors of 18: 2 3 6 9
Factors of 19:
Factors of 20: 2 4 5 10
```

В данном случае во внешнем цикле переменная `i` последовательно принимает значения до 2 до 100. Во внутреннем цикле для каждого числа от 2 до `i` выполняется проверка, делится ли `i` на это число без остатка. В качестве задания для самостоятельной работы попробуйте найти способ сделать работу программы более эффективной. (Подсказка: число итераций во внутреннем цикле можно уменьшить.)

Тест для самоконтроля по модулю 3

1. Напишите программу, которая читала бы символы с клавиатуры до тех пор, пока не встретится точка. Предусмотрите в программе счетчик числа пробелов. Информация о количестве пробелов должна выводиться в конце работы программы.
2. В каком формате записывается структура `if-else-if`?
3. Дан следующий фрагмент кода:

```
if(x < 10)
  if(y > 100) {
    if(!done) x = z;
    else y = z;
  }
else System.out.println("error"); // Для какого if?
```

   Какому из операторов `if` соответствует последнее выражение `else`?
4. Напишите цикл `for`, в котором бы перебирались значения от 1000 до 0 с шагом –2.
5. Корректен ли следующий фрагмент кода?

```
for(int i = 0; i < num; i++)
  sum += i;

count = i;
```

6. Какие действия выполняет оператор `break`? Опишите оба варианта данного оператора.
7. Какое сообщение будет выведено после выполнения оператора `break` в приведенном ниже фрагменте кода?

```
for(i = 0; i < 10; i++) {
  while(running) {
    if(x<y) break;
      // ...
  }
  System.out.println("after while");
}
System.out.println("After for");
```

8. Какие данные будут выведены в результате выполнения следующего фрагмента программы?

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

9. Итерационное выражение для цикла `for` не обязательно должно изменять переменную цикла на фиксированную величину. Эта переменная может принимать произвольные значения. Напишите программу, использующую цикл `for`, которая выводила бы геометрическую прогрессию 1, 2, 4, 8, 16, 32 и т.д.
10. Код ASCII-символов нижнего регистра отличается от кода соответствующих символов верхнего регистра на 32. Таким образом, для преобразования строчной буквы в прописную, надо уменьшить ее код на 32. Используйте эти сведения для написания программы, осуществляющей чтение символов с клавиатуры. При выводе результатов данная программа должна преобразовать строчные буквы в прописные, а прописные — в строчные. Остальные символы не должны изменяться. Работа программы должна завершаться тогда, когда пользователь введет точку. И наконец, сделайте так, чтобы программа отображала число символов, для которых был изменен регистр.
11. Что такое бесконечный цикл?
12. При использовании оператора `break` с меткой должна ли метка быть определена в блоке, содержащем данный оператор?