2 Управляющие инструкции Java

Пошли, Скрипач, в открытый космос.

из к/ф «Кин-дза-дза»

В этой главе мы рассмотрим операторы цикла, условный оператор и оператор выбора. Все эти синтаксические конструкции позволяют создавать в программе точки ветвления и многократно выполнять блоки команд. Обычно их называют *управляющими инструкциями*. Без этих инструкций попросту невозможно создавать эффективные программные коды.

Условный оператор if

Хорошо, допустим, Земля вращается вокруг Солнца.

из к/ф «Приключения Шерлока Холмса и доктора Ватсона»

Как отмечалось в предыдущей главе, кроме тернарного оператора в Java имеется более функциональная конструкция. Речь об *условном операторе* со следующим синтаксисом:

```
if(условие){
   // Команды
}
else{
   // Команды
}
```

После ключевого слова `if` в круглых скобках указывается условие.

НА ЗАМЕТКУ

Условие, указанное в операторе `if`, представляет собой выражение, возвращающее результатом значение логического типа `boolean`. Это же замечание относится к условиям, используемым в операторах цикла.

Выполнение условного оператора начинается с проверки этого условия. Если условие истинно (значение выражения в круглых скобках после ключевого слова `if` равно `true`), выполняются команды, указанные в фигурных скобках сразу после инструкции `if(условие)`. Если условие ложно (значение выражения в круглых скобках равно `false`), то выполняются команды, размещенные в блоке (выделенном фигурными скобками) после ключевого слова `else`. После выполнения условного оператора управление передается следующей после него команде (рис. 2.1).

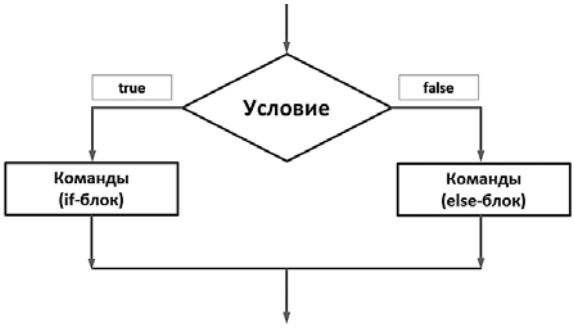

Рис. 2.1. Схема работы условного оператора

НА ЗАМЕТКУ

Условный оператор работает по следующей схеме: есть два блока команд и есть условие. Если условие истинно, то выполняется один блок команд. Если условие ложно, то выполняется другой блок команд.

Если любой из двух блоков команд (в `if`-блоке или в `else`-блоке) состоит из одной команды, то фигурные скобки для соответствующего блока можно не использовать. Тем не менее фигурные скобки улучшают читабельность программы.

У условного оператора есть упрощенная форма, в которой не используется `else`-блок (проще говоря, в условном операторе `else`-блок не является обязательным). Синтаксис упрощенной формы условного оператора имеет следующий вид:

```
if(условие){
   // Команды
}
```

В этом случае сначала проверяется на истинность условие, указанное в скобках после ключевого слова `if`. Если условие истинно, то выполняется расположенный далее блок команд. Если условие ложно, то ничего не происходит (рис. 2.2).

НА ЗАМЕТКУ

Принцип работы упрощенной формы условного оператора следующий: есть блок команд, которые выполняются, только если истинно некоторое условие.

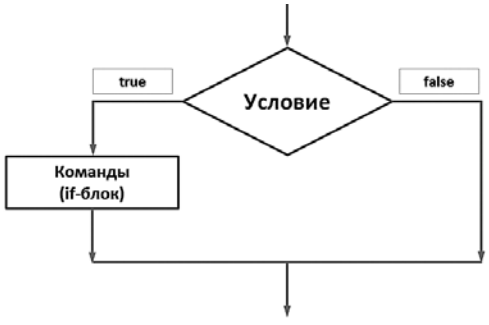

Рис. 2.2. Принцип работы упрощенной формы условного оператора

На практике нередко используются вложенные `if`-операторы. С точки зрения синтаксиса языка Java такая ситуация проста: в `else`-блоке условного оператора размещается другой условный оператор, а в его `else`-блоке размещается еще один условный оператор, и так далее. Но при этом вся конструкция выглядит достаточно элегантно:

```
if(условие){
   // Команды
}else if(условие){
   // Команды
}else if(условие){
   // Команды
}
// Продолжение
else if(условие){
   // Команды
}else{
   // Команды
}
```

Принцип выполнения вложенных условных операторов следующий. Сначала проверяется условие в первом (внешнем) `if`-операторе. Если оно истинно, то выполняются соответствующие команды. Если условие ложно, то проверяется условие во втором (внутреннем) условном операторе в `else`-блоке. При истинном условии в этом операторе выполняются команды в его `if`-блоке. Если условие ложно, то начинает выполняться третий условный оператор в `else`-блоке второго оператора, и так далее. Получается, что условия проверяются последовательно, одно за другим, если все предыдущие условия оказались ложными. Последний `else`-блок (если он есть) выполняется, если все условия оказались ложными.

В листинге 2.1 представлен пример несложной программы, в которой используется условный оператор. В программе генерируется случайное число (в диапазоне значений от `1` до `5` включительно). Пользователю предлагается угадать это число (указывается в поле ввода диалогового окна). В зависимости от того, угадал пользователь или нет, появляется сообщение соответствующего содержания.

Листинг 2.1. Знакомство с условным оператором

```
import static javax.swing.JOptionPane.*;
import static java.lang.Integer.parseInt;
import static java.lang.Math.random;
class Demo{
   public static void main(String[] args){
      int num,ans,icon;
      String txt;
      // Случайное целое число (от 1 до 5):
      num=(int)(5*random())+1;
      // Считывание целого числа:
      ans=parseInt( // Преобразование текста в число
         showInputDialog(null, // Родительское окно
            // Текст над полем ввода:
            "Угадайте число (от 1 до 5):",
            "Число", // Заголовок окна
            QUESTION_MESSAGE // Тип пиктограммы
         )
      );
      // Условный оператор:
      if(ans==num){
         txt="Вы угадали! Это число "+num+"!";
         icon=INFORMATION_MESSAGE;
      }else{
         txt="Вы не угадали! Это число "+num+"!";
         icon=ERROR_MESSAGE;
      }
      showMessageDialog(null, // Родительское окно
         txt, // Текст сообщения
         "Результат", // Заголовок окна
         icon // Тип пиктограммы
      );
   }
}
```

При запуске программы появляется окно (рис. 2.3).

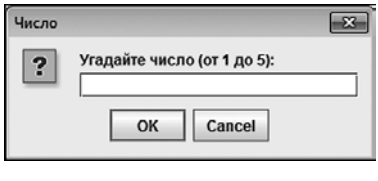

Рис. 2.3. Окно с полем для ввода числа

Пользователь должен ввести число в поле диалогового окна и нажать кнопку OK. Если пользователь угадал число, то появится диалоговое окно (рис. 2.4).

Если пользователь не угадал число, то окно будет иметь иной вид (рис. 2.5).

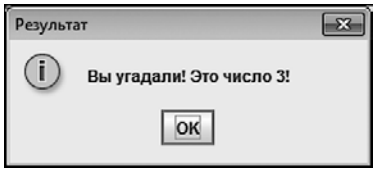

Рис. 2.4. Окно появляется, если пользователь угадал число

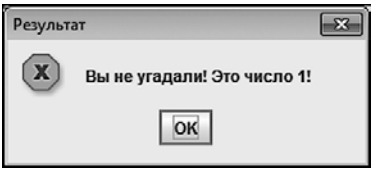

Рис. 2.5. Окно появляется, если пользователь не угадал число

Теперь проанализируем код, выполнение которого приводит к таким результатам.

В программе генерируется целое случайное число, и это число записывается в целочисленную переменную `num`. Мы использовали команду `num=(int)(5*random())+1`, задействовав статический метод `random()` из класса `Math` (для импорта метода использован статический импорт). Метод возвращает результатом случайное действительное число от `0` (включительно) до `1` (строго меньше). Умножив результат вызова метода на `5`, получим случайное действительное число от `0` (включительно) до `5` (строго меньше). После приведения этого значения к целочисленному формату (инструкция `(int)`) получаем целое число в диапазоне от `0` до `4` включительно. После прибавления единицы получаем число в диапазоне значений от `1` до `5`.

Число, которое вводит пользователь, записывается в целочисленную переменную `ans`. Для отображения диалогового окна с полем ввода мы вызываем статический метод `showInputDialog()` из класса `JOptionPane`. Но на этот раз методу передается не один (как было раньше), а четыре аргумента. Первый аргумент `null` является пустой ссылкой на родительское окно (такого окна нет). Второй текстовый аргумент определяет надпись над полем ввода. Третий текстовый аргумент задает заголовок окна. Наконец, четвертым аргументом передана статическая константа `QUESTION_MESSAGE` из класса `JOptionPane`. Эта константа определяет тип пиктограммы, отображаемой в окне (в данном случае — пиктограмма со знаком вопроса). Но это не все. Даже если пользователь вводит в поле диалогового окна целое число, оно все равно считывается как текст. Это так называемое текстовое представление числа, то есть речь о тексте (например, `"3"`), который содержит число. Чтобы извлечь число из текста, используем статический метод `parseInt()` из класса-оболочки `Integer`. Инструкция с вызовом метода `showInputDialog()` указана аргументом метода `parseInt()`.

НА ЗАМЕТКУ

Для использования статических методов и констант из класса `JOptionPane` использован статический импорт всех статических полей и методов этого класса. Также мы использовали статический импорт для метода `parseInt()` из класса `Integer`.

Далее в игру вступает условный оператор. В нем проверяется условие `ans==num`, которое истинно в случае, если введенное пользователем число равно сгенерирован-

ному случайному числу. При истинном условии значение текстовой переменной `txt` присваивается командой `txt="Вы угадали! Это число "+num+"!"`, а целочисленная переменная `icon` получает значение `INFORMATION_MESSAGE` (статическая константа из класса `JOptionPane`). Если условие ложно, то переменной `txt` присваивается значение `"Вы не угадали! Это число "+num+"!"`, а переменной `icon` присваивается значение `ERROR_MESSAGE` (статическая константа из класса `JOptionPane`).

ПОДРОБНОСТИ

Класс `JOptionPane`, кроме статических методов, содержит еще и статические поля — константы, используемые для определения вида пиктограмм в диалоговых окнах: `INFORMATION_MESSAGE` (информационная пиктограмма), `QUESTION_MESSAGE` (пиктограмма со знаком вопроса), `WARNING_MESSAGE` (пиктограмма предупреждения), `ERROR_MESSAGE` (пиктограмма ошибки), `PLAIN_MESSAGE` (пиктограмма отсутствует). Поскольку константы целочисленные, то их можно присваивать значениями целочисленным переменным.

После определения значений переменных `txt` и `icon` они используются как аргументы при вызове метода `showMessageDialog()`. Методу передаются такие аргументы: пустая ссылка `null` на родительское окно (его нет), текст сообщения (определяется значением переменной `txt`), текст для заголовка окна, тип пиктограммы (определяется значением переменной `icon`).

НА ЗАМЕТКУ

В зависимости от истинности или ложности условия в условном операторе поразному определяются значения переменных, определяющих текст сообщения и тип пиктограммы. После этого переменные используются при отображении сообщения.

Пример использования упрощенной формы условного оператора приведен в листинге 2.2.

Листинг 2.2. Упрощенная форма условного оператора

```
// Статический импорт:
import static javax.swing.JOptionPane.*;
// Импорт класса:
import javax.swing.ImageIcon;
class Demo{
   public static void main(String[] args){
      int res;
      // Отображается окно подтверждения:
      res=showConfirmDialog(null, // Родительское окно
         "Хотите увидеть красную панду?", // Сообщение
         "Вопрос", // Заголовок окна
         YES_NO_OPTION // Кнопки
      );
```

```
        // Упрощенная форма условного оператора:
        if(res==YES_OPTION){
            // Полный путь к файлу с изображением:
            String file="d:/Pictures/Animals/panda.png";
            // Создание объекта изображения:
            ImageIcon img=new ImageIcon(file);
            // Отображается окно с картинкой:
            showMessageDialog(null, // Родительское окно
                img, // Изображение
                "Красная панда", // Заголовок окна
                PLAIN_MESSAGE // Пиктограмма отсутствует
            );
        }
    }
}
```

При запуске программы появляется *окно подтверждения*: сообщение в окне содержит вопрос, и у окна есть две кнопки (Yes и No), как показано на рис. 2.6.

Если пользователь нажимает кнопку No, то окно закрывается и на этом выполнение программы завершается. Но если пользователь нажимает кнопку Yes, то появляется еще одно окно (рис. 2.7).

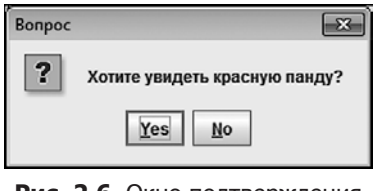

Рис. 2.6. Окно подтверждения с двумя кнопками

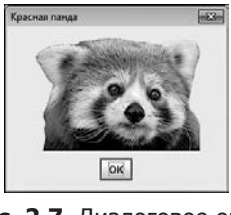

Рис. 2.7. Диалоговое окно с изображением красной панды

Окно содержит изображение красной панды.

Что касается самой программы, то она небольшая. Для отображения окна подтверждения мы используем статический метод `showConfirmDialog()` из класса `JOptionPane`. Аргументами методу при вызове передаются:

- Ссылка на родительское окно (указано значение `null`, поскольку такого окна нет).
- Текст сообщения `"Хотите увидеть красную панду?"`, которое отображается в диалоговом окне.
- Текст `"Вопрос"`, определяющий заголовок окна.
- Статическая константа `YES_NO_OPTION` из класса `JOptionPane`, определяющая количество кнопок в диалоговом окне.

ПОДРОБНОСТИ

Константа YES_NO_OPTION означает, что в окне отображаются кнопки Yes и No. Также можно использовать следующие константы: DEFAULT_OPTION (набор кнопок по умолчанию — отображается одна кнопка OK), YES_NO_CANCEL_OPTION (отображаются кнопки Yes, No и Cancel) или OK_CANCEL_OPTION (кнопки OK и Cancel).

Результат вызова метода записывается в целочисленную переменную res. Значение, возвращаемое методом showConfirmDialog(), зависит от того, как было закрыто окно. Если пользователь закрывает окно нажатием кнопки Yes, то результатом метод возвращает значение константы YES_OPTION.

ПОДРОБНОСТИ

Результат NO_OPTION возвращается, если пользователь нажал кнопку No. Результат CANCEL_OPTION возвращается, если была нажата кнопка Cancel. Результат OK_OPTION означает, что была нажата кнопка OK. А если окно закрыто нажатием системной пиктограммы, то результат определяется константой CLOSED_OPTION.

После того как первое окно закрыто, начинается выполнение условного оператора (в упрощенной форме — у этого оператора отсутствует else-блок). Проверяется условие res==YES_OPTION, истинность которого означает, что окно было закрыто нажатием кнопки Yes. Если так, то объявляется текстовая переменная file. Значением ей присваивается текст "d:/Pictures/Animals/panda.png". Это полный путь к файлу с изображением, которое мы собираемся показать в диалоговом окне. Но предварительно на основе изображения нужно создать объект. Объект изображения создается командой ImageIcon img=new ImageIcon(file).

ПОДРОБНОСТИ

Объект создается на основе класса, подобно тому как дом строится на основе проекта. Объект изображения создается на основе класса ImageIcon, который импортируется в программу (инструкция import javax.swing.ImageIcon). Для создания объекта используется инструкция new, после которой указывается название класса и, в круглых скобках, параметры, необходимые для создания объекта. В данном случае класс — ImageIcon, а параметр один, и это полный путь к файлу с изображением. Результатом выражения на основе инструкции new является ссылка на созданный объект. Эта ссылка записывается в объектную переменную (в нашем случае объектная переменная называется img). Объектная переменная объявляется как обычная переменная, но только типом переменной указывается класс, на основе которого создается объект.

Классы и объекты обсуждаются немного позже. Что касается самого изображения, то для корректной работы программы необходимо разместить файл panda.png в директорию D:\Pictures\Animals (или изменить в программе значение текстовой переменной file в соответствии с тем, где именно находится файл с картинкой). Сам графический файл сохранен в формате, позволяющем исполь-

зование прозрачной подложки (прозрачный фон), поэтому в диалоговом окне при отображении изображения создается впечатление, что оно наложено на фон окна. Вообще же использовано прямоугольное изображение шириной 200 пикселов и высотой 140 пикселов.

После создания объекта изображения отображается диалоговое окно с картинкой. Особенность соответствующей команды на основе метода `showMessageDialog()` в том, что вторым аргументом вместо текстового значения передается переменная `img` (объект изображения). То есть вместо текста мы указываем графический объект. Тип пиктограммы в окне определяется константой `PLAIN_MESSAGE` из класса `JOptionPane`, что на самом деле означает отсутствие пиктограммы у окна (две картинки в одном окне выглядят не очень эстетично).

Пример использования нескольких вложенных условных операторов представлен в листинге 2.3.

Листинг 2.3. Вложенные условные операторы

```
import java.util.Scanner;
class Demo{
   public static void main(String[] args){
      int a;
      // Создание объекта класса Scanner:
      Scanner input=new Scanner(System.in);
      System.out.print("Введите целое число: ");
      // Считывание целого числа:
      a=input.nextInt();
      if(a==0){ // Если введен ноль
         System.out.println("Вы ввели ноль!");
      }else if(a==1){ // Если введена единица
         System.out.println("Вы ввели единицу!");
      }else if(a%2==0){ // Если введено четное число
         System.out.println("Вы ввели четное число!");
      }else{ // В прочих случаях
         System.out.println("Вы ввели нечетное число!");
      }
      System.out.println("Спасибо!");
   }
}
```

В программе считывается целое число, введенное пользователем, и в зависимости от значения этого числа выводится то или иное сообщение. Для считывания числа мы используем консольный ввод.

НА ЗАМЕТКУ

Консольный ввод в Java реализован не самым простым образом. Причина в том, что язык Java разрабатывался не для создания консольных программ, хотя и такие программы успешно создаются.

Для считывания значений, которые пользователь вводит через окно вывода, нам нужен объект класса `Scanner`. Класс импортируется в программу инструкцией `import java.util.Scanner`. Объект класса `Scanner` создается командой `Scanner input=new Scanner(System.in)`.

ПОДРОБНОСТИ

Мы используем ту же схему, что и при создании объекта изображения, а именно: объект класса `Scanner` создается с помощью инструкции `new`, после которой указано название класса. Параметром, необходимым для создания объекта, является ссылка `System.in` на объект стандартного потока ввода (реализуется через статическое поле `in` класса `System`). Ссылка на созданный объект записывается в объектную переменную `input`.

Командой `System.out.print("Введите целое число: ")` в окно выводится сообщение.

НА ЗАМЕТКУ

В отличие от метода `println()`, методом `print()` выводится сообщение, но курсор в области вывода в новую строку не переводится — он остается в той же строке.

Командой `a=input.nextInt()` считывается целочисленное значение, введенное пользователем, которое записывается в переменную `a`.

ПОДРОБНОСТИ

Из объекта `input` вызывается метод `nextInt()`. Метод результатом возвращает целочисленное значение, которое ввел пользователь.

Происходит все так: появляется сообщение `Введите целое число:`, и курсор находится в области вывода. Пользователь вводит число, нажимает клавишу <Enter>, и программа считывает введенное пользователем значение.

Для считанного значения отслеживаются варианты действий: пользователь ввел ноль; пользователь ввел единицу; пользователь ввел четное число (делится без остатка на `2`); пользователь ввел нечетное число (не делится без остатка на `2`). Перебор всех возможных вариантов реализован через блок вложенных условных операторов. Перечисленные ранее условия проверяются по очереди, до первого выполненного условия. Если ни одно из условий не является истинным, то выполняются команды в последнем `else`-блоке вложенных условных операторов. Результат выполнения программы варьируется и может быть таким (здесь и далее жирным шрифтом выделено введенное пользователем значение):

Результат выполнения программы (из листинга 2.3)

```
Введите целое число: 0
Вы ввели ноль!
Спасибо!
```

Таким:

Результат выполнения программы (из листинга 2.3)

```
Введите целое число: 1
Вы ввели единицу!
Спасибо!
```

Таким:

Результат выполнения программы (из листинга 2.3)

```
Введите целое число: 8
Вы ввели четное число!
Спасибо!
```

Или таким:

Результат выполнения программы (из листинга 2.3)

```
Введите целое число: 9
Вы ввели нечетное число!
Спасибо!
```

Хотя с помощью блоков вложенных условных операторов можно реализовать практически любой алгоритм с точками ветвления, нередко более эффективным представляется использование оператора выбора `switch`.

Оператор выбора switch

— Утром деньги — вечером стулья. Вечером деньги — утром стулья.

— А можно так: утром стулья — вечером деньги?

— Можно. Но деньги вперед.

из к/ф «Двенадцать стульев»

Оператор выбора `switch` позволяет выполнять разные блоки команд в зависимости от значения, которое принимает некоторое выражение (в этом смысле он немного напоминает условный оператор). Синтаксис вызова оператора `switch` следующий:

```
switch(условие){
   case значение:
      // Команды
      break;
   case значение:
      // Команды
```