第 3 章　程序控制流：分支

控制流是程序执行动作的顺序。在本章之前，这个顺序都很简单，执行动作的顺序按照书写语句的先后顺序执行。本章开始展示如何编写具有更复杂控制流的程序。Java 以及其他绝大多数编程语言都使用两种类型的语句规范控制流：分支语句从两个或多个可能动作中选择一个动作执行；循环语句则反复地重复一个动作，直到满足某个停止条件为止。这两种类型的语句构成了程序的**控制结构**（control structure）。由于分支语句在多个动作中进行选择或者说抉择，因此称它们为决策结构。本章将讨论分支，第 4 章讨论循环。

本章内容

在学习完本章之后，你将可以：

- 在程序中使用 Java 分支语句 if-else 和 switch。
- 比较基本类型的值。
- 比较诸如字符串之类的对象。
- 使用基本数据类型 boolean。
- 在程序中使用简单的枚举。
- 在图形程序中使用颜色。
- 使用类 JOptionPane 创建有关是-或-否问题的对话框。

预备知识

在阅读本章之前，你需要熟悉第 2 章的内容。

3.1　if-else 语句

本章首先讨论在两个动作之间做出选择的一种 Java 语句。

3.1.1　基本 if-else 语句

程序就像日常生活一样，有时候也需要在两者之一中做出选择。如果在你的支票户头中有余额，某些银行将向你支付少量利息。但如果透支了你的支票户头，那么你的账户余额将会是负数，你将承担余额为负的后果。这一策略可以在银行记账程序中反映出来，使用下述 Java 语句实现，即 if-else 语句：

```
if ( balance >= 0 )
   balance = balance + (INTEREST_RATE * balance ) / 12;
else
   balance = balance - OVERDRAW_PENALTY;
```

符号对>=在 Java 中用于表示大于等于，原因在于键盘上没有符号≥。

if-else 语句的意义与按英文句子阅读它时的意义一样。当程序执行 if-else 语句时，首先检查关键字 if 后面括号中的表达式。这个表达式经计算后得到 true 或 false。如果为 true，就执行 else 之前的语句；如果表达式为 false，就执行 else 后面的语句。在前面的示例中，如果 balance 为正数或者零，那么就执行下述语句：

```
balance = balance + (INTEREST_RATE * balance ) / 12;
```

（由于只增加一个月的利息，因此必须把年利率除以 12）。但是，如果 balance 的值为负数，那么就执行下述语句：

```
balance = balance - OVERDRAW_PENALTY;
```

图 3.1 展示了这个 if-else 语句的操作，程序清单 3.1 给出了完整程序。

```
if (balance >= 0)
   balance = balance + (INTEREST_RATE * balance) / 12;
else
   balance = balance - OVERDRAWN_PENALTY;
```

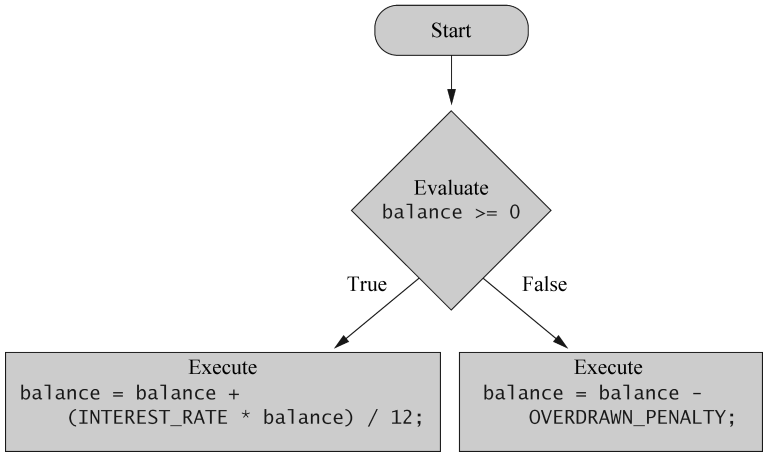

图 3.1　程序清单 3.1 中 if-else 语句的操作

表达式 balance >= 0 是**布尔表达式**（boolean express）的一个示例。这是一个其值要么为 true 要么为 false 的简单表达式。名称 boolean 来自 Geoge Boole，这是一位 19 世纪的英国逻辑学家和数学家，其工作与这类表达式密切相关。下一节将考察布尔表达式。

请注意，if-else 语句在其内包含两个较短一些的语句。例如，程序清单 3.1 中的 if-else 语句包含了下述两个较短语句：

```
balance = balance + (INTEREST_RATE * balance ) / 12;
balance = balance - OVERDRAW_PENALTY;
```

请注意，这些较短语句比 if 和 else 多缩进了一级。

编程技巧　缩进

第 2 章讨论了缩进。这是一个应该认真遵从的格式约定。尽管编译器会忽略缩进，

但不一致性地使用这一约定，会把阅读代码的人弄晕，并且也很可能让你自己出错。

如果要在每一个分支中包含多条语句，只需使用大括号{}把它们括起来就可以了。括在大括号中的几条语句可以认为是一条更大一些的语句。因此，下面语句是在其内具有两条更短语句的一条较大语句：

```
{
   System.out.println("Good for you. You earned interest. ")
   balance = balance + (INTEREST_RATE * balance ) / 12;
}
```

程序清单 3.1　使用 if-else 的程序

```
import java.util.Scanner;

public class BankBalance
{
   public static final double OVERDRAWN_PENALTY = 8.00;
   public static final double INTEREST_RATE = 0.02;//2% annually

   public static void main(String[] args)
   {
      double balance;

      System.out.print("Enter your checking account balance: $");
      Scanner keyboard =  new Scanner(System.in);
      balance = keyboard.nextDouble( );
      System.out.println("Original balance $" + balance);

      if (balance >= 0)
         balance = balance + (INTEREST_RATE * balance)/12;
      else
         balance = balance - OVERDRAWN_PENALTY;

      System.out.println("After adjusting for one month");
      System.out.println("of interest and penalties,");
      System.out.println("your new balance is $" + balance);
   }
}
```

样本输出 1

```
Enter your checking account balance: $505.67
Original balance $505.67
After adjusting for one month of interest and penalties,
your new balance is $506.51278
```

样本输出 2

```
Enter your checking account balance: $-15.53
Original balance $ -15.53
After adjusting for one month of interest and penalties,
your new balance is $ -23.53
```

用大括号把一系列语句括起来而构成的语句称为**复合语句**（compound statement）。一般很少单独使用这种语句，而是经常把它们用作为更大语句（比如 if-else 语句）的子语句。前面的复合语句可以以下述方式出现在 if-else 语句中：

```
if ( balance >= 0 )
{
   System.out.println("Good for you. You earned interest. ");
   balance = balance + (INTEREST_RATE * balance ) / 12;
}

else
{
   System.out.println("You will be charged penalty. ");
   balance = balance - OVERDRAW_PENALTY;
}
```

请注意，复合语句可以简化对 if-else 语句的描述。一旦掌握了复合语句的使用，那么就可以说，每一个 if-else 语句都具有如下形式：

```
if ( Boolean_Expression )
   statement_1
else
   statement_2
```

如果要让一条或两条分支包含几条语句而不是一条语句，那么可以把 statement_1 和（或）statement_2 编写为复合语句。图 3.2 概要给出了 if-else 语句的语义或意义。

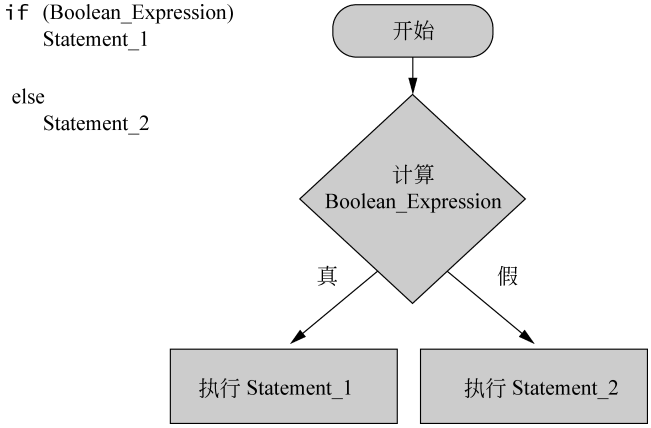

图 3.2 if-else 语句的语义

如果遗漏了 else 部分，当 if 语句的表达式为 false 时，程序简单地跳过 statement，如图 3.3 所示。例如，如果银行不透支罚款的话，前面给出的语句可以缩短为下述形式：

```
if ( balance >= 0 )
{
   System.out.println("Good for you. You earned interest. ");
   balance = balance + (INTEREST_RATE * balance ) / 12;
}
```

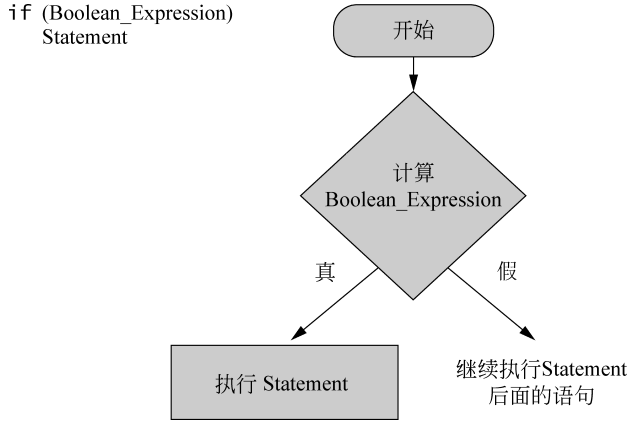

图 3.3　没有 else 的 if 语句的语义

要理解这条语句是如何工作的，让我们添加一些其他语句来增加以下上下文环境，如下所示，这里 keyboard 是通常类型的 Scanner 对象：

```
System.out.println("Enter your balance $");
balance = keyboard.nextDouble;
if ( balance >= 0 )
{
   System.out.println("Good for you. You earned interest. ");
   balance = balance + (INTEREST_RATE * balance ) / 12;
}
System.out.println("Your new balance is $" + balance);
```

当你的支票账户余额为$100 时，执行这些语句将生成下述交互结果：

```
Enter your balance $100.00
Good for you. You earned interest.
Your new balance is $100.16
```

由于表达式 balance >= 0 为 true，因此你可以获得一点利息。对这个示例来说，精确利息是多少无关紧要（这里使用了 2%的年利率，就像在程序清单 3.1 中那样）。

现在，假定你的账户透支，余额为负$50。那么输出将为如下所示：

```
Enter your balance $-50.00
Your new balance is $-50.00
```

在这种情况下，表达式 balance >= 0 为 false，但由于没有 else 部分，因此什么也没有发生，即余额没有被修改，程序只是前进到下一行语句，这是一条输出语句。

扼要重述 if-else 语句

语法（基本形式）

```
if ( Boolean_Expression )
   statement_1
else
   statement_2
```

如果表达式 Boolean_Expression 为 true，那么就执行 statement_1，否则就执行 statement_2。

示例

```
if ( time < limit )
   System.out.println("You made it.");
else
   System.out.println("You missed the deadline.");
```

语法（没有 else 部分）

```
if ( Boolean_Expression )
   statement
```

如果表达式 Boolean_Expression 为 true，那么就执行 statement；否则，statement 被忽略，程序前进到下一条语句。

示例

```
if ( weight > ideal )
   calorieAllotment = calorieAllotment -500;
```

复合语句

每一个 statement、statement_1、statement_2 都可以是一条复合语句。如果要让表达式的给定分支执行几条语句，那么就使用大括号把它们括起来，如下述示例所示：

```
if ( balance >= 0 )
{
   System.out.println("Good for you. You earned interest. ");
   balance = balance + (INTEREST_RATE * balance ) / 12;
}
else
{
   System.out.println("You will be charged penalty. ");
   balance = balance - OVERDRAW_PENALTY;
}
```

3.1.2　布尔表达式

在前面的 if-else 语句中已经使用了简单的布尔表达式。最简单的布尔表达式是比较两个较短的表达式，比如在下述示例中：

```
balance >= 0
```

和

```
time < limit
```

图 3.4 给出了可以用于比较两个表达式的各种 Java 比较运算符。

数学符号	名称	Java 符号	Java 示例
=	等于	==	`balance == 0` `answer == 'y'`
≠	不等于	!=	`income != tax` `answer != 'y'`
>	大于	>	`expenses > income`
≥	大于等于	>=	`points >= 60`
<	小于	<	`pressure < max`
≤	小于等于	<=	`expenses <= income`

图 3.4　Java 的比较运算符

请注意，布尔表达式不需要以小括号开始和结束。但是，当布尔表达式在 if-else 语句中使用时，确实需要把它括在小括号中。

疑难杂症　使用=代替==测试相等

运算符=是赋值运算符。这个符号在数学上意味着相等，但在 Java 中并不是这个意义。如果用 if (x = y)替代 if (x = = y)来测试 x 和 y 是否相等，将得到一条语法错误消息。

疑难杂症　使用==或!=比较浮点值

第 2 章中曾经警告过，必须树立浮点数是近似值的观念。具有小数部分的数值可以具有无限位数的数字。由于计算机只能存储有限位数的小数，因此浮点数通常都是不精确的。在计算机中执行各种计算之后，这些近似值的精度将会更低。

因此，如果计算两个浮点值，那么它们很有可能并不严格相等。几乎相等的两个值或许就是你所希望得到的。因此，不应该使用==比较两个浮点值。使用!=运算符也会引发相同的问题。

取而代之的是，要测试浮点值的“相等”，观察它们之间的差别是否十分微小，这些差别仅仅是由于近似特性引起的。如果是这样的话，就可以认为它们足够“靠近”，从而认为它们是相等的。

通过使用逻辑运算符&&（这是 Java 版本的“与”运算），可以把简单逻辑表达式拼接在一起，构造复杂的逻辑表达式。例如，考虑下述语句：

```
if (pressure > min ) && (pressure < max)
   System.out.println("Pressure is OK. ");
else
   System.out.println("Warning! Pressure is out of range.");
```

如果 pressure 的值大于 min，并且 pressure 的值小于 max，那么输出为

```
Pressure is OK.
```

否则，输出为

```
Warning! Pressure is out of range.
```

请注意，不能写作为

```
min <pressure < max
```

取而代之的是，必须单独表示每一个不等式，并使用&&把这些不等式连接起来，如下所示：

```
(pressure > min ) && (pressure < max)
```

当通过使用&&把两个较短表达式连接起来构成更长的逻辑表达式时，只有在两个较短的表达式都为 true 时，整个较长的表达式才为 true。如果两个较短表达式中至少有一个表达式的值为 false，那么较长表达式的值就为 false。例如，假定(pressure > min) 和 (pressure < max)都为 true，那么

```
(pressure > min ) && (pressure < max)
```

为 true；否则，该表达式的值为 false。

扼要重述　将&&用于表示“与”运算

符号对&&在 Java 中意味着“与”运算。可以使用&&，由两个较短逻辑表达式构造出较长的逻辑表达式。

语法

```
(Sub_Expression_1)&&( Sub_Expression_2)
```

当且仅当 Sub_Expression_1 和 Sub_Expression_2 都为 true 时，这个表达式才为 true。

示例

```
if (pressure > min ) && (pressure < max)
   System.out.println("Pressure is OK. ");
else
   System.out.println("Warning! Pressure is out of " + "range.");
```

除了可以使用“与”来拼接逻辑表达式之外，也可以使用“或”拼接它们。表达“或”的 Java 方法是 ||，它可以通过两根竖线得到它（符号 | 在某些系统上显示为一条空线）。运算符 || 的意义本质上与英文单词 or（即“或”）相同。例如，考虑

```
if (salary > expenses) || (savings > expenses)
   System.out.println("Solvent");
else
   System.out.println("Bankrupt");
```

如果 salary 的值大于 expenses 的值或 savings 的值大于 expenses 的值，或者它们都为 true，那么输出就为 Solvent；否则，输出为 Bankrupt。

扼要重述　将 || 用于表示“或”

符号对 || 在 Java 中意味着“或”。可以使用 ||，由两个较短逻辑表达式构造出较长的逻辑表达式。

语法

```
(Sub_Expression_1) || ( Sub_Expression_2)
```

如果 Sub_Expression_1 或 Sub_Expression_2 中至少有一个为 true，那么该表达式就为 true。

示例

```
if (salary > expenses) || (savings > expenses)
   System.out.println("Solvent");
else
   System.out.println("Bankrupt");
```

正像前面曾经提醒的那样，if-else 语句中的布尔表达式必须括在小括号中。使用&&运算符的 if-else 语句通常按下述方式使用括号：

```
if ((pressure > min ) && (pressure < max))
```

(pressure > min)和(pressure < max)中的括号并不要求，但是，我们通常使用这些括号，以便提高可读性。在包含运算符 || 的表达式中，括号的用法与包含运算符&&的表达式中括号的用法相同。

在 Java 中，可以通过在布尔表达式前面放上一个 ！符号来表示“非”的意义。例如，

```
if (!(number>= min ))
   System.out.println("Too small");
else
   System.out.println("OK");
```

如果 number 不大于等于 min，那么输出为 Too small；否则，输出为 OK。

通常应避免使用运算符 !，而且也建议应避免使用它。例如，上述 if-else 语句等价于

```
if (number < min )
    System.out.println("Too small");
else
    System.out.println("OK");
```

!(A op B)	等价于 (A op B)
<	>=
<=	>
>	<=
>=	<
==	!=
!=	==

图 3.5 避免使用非运算符

图 3.5 给出了如何把下述形式的表达式

```
!(A Comparison_Operator B)
```

转换为没有非运算符的表达式。

如果能不使用运算符 !，你的程序就更容易理解。但是，会有需要使用这个运算符的情形。特别地，我们可以在循环的上下文中使用运算符 !（第 4 章将介绍）来取反布尔变量的值，这一点将在本章后面讨论。

扼要重述　运算符 ! 用于表示“非”

在 Java 中，符号 ! 表示“非”。可以使用 ! 取反布尔表达式的值。当表达式为一个布尔变量时，这种做法更常见。也可以重写布尔表达式，以避免使用取反操作。

语法

```
!Boolean_Expression
```

该表达式的值是 Boolean_Expression 值的相反值：如果 Boolean_Expression 的值为 false，那么该表达式的值为 true；如果 Boolean_Expression 的值为 true，那么该表达式的值为 false。

示例

```
if (!(number <0 ))
    System.out.println("OK ");
else
    System.out.println("Nagtive");
```

图 3.6 总结了以上 3 个逻辑运算符。依据图 3.7 中给出的规则，Java 可以组合使用 true 和 false 的值。例如，如果布尔表达式 A 的值为 true，布尔表达式 B 的值为 false，那么 A&&B 的值为 false。

名称	Java 符号	Java 示例
逻辑与	&&	(sum > min) && (sum < max)
逻辑或	||	(answer == 'y') || (answer == 'Y')
逻辑非	!	!(number < 0)

图 3.6 Java 逻辑运算符

A 的值	B 的值	A&&B 的值	A‖B 的值	!(A) 的值
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

图 3.7　布尔运算符&&（与）、‖（或）、！（非）对布尔值的操作结果

3.1.3　比较字符串

要测试两个基本数据类型的值（比如两个数值）是否相等，可以使用相等运算符= =。但是，当作用于对象时，= =具有不同的意义。请回忆一下，字符串是 String 类的对象，因此，= =作用到两个字符串上时，并不是测试这些字符串是否相等。要查看两个字符串是否具有相等的值，必须使用方法 equals 而不是使用= =。

例如，如果字符串 s1 和 s2 具有相等的值，那么布尔表达式 s1.equals(s2)返回 true，否则返回 false。程序清单 3.2 中的程序展示了 equals 方法的用法。请注意，下面两个表达式

```
s1. equals(s2)
s2. equals(s1)
```

是等价的。

程序清单 3.2 还展示了 String 类的 equalsIgnoreCase 方法。这个方法与 equals 类似，只不过 equalsIgnoreCase 是把同一个字母的大写和小写形式看作为相同的字母。例如，"Hello"和"hello"不相等，原因在于它们的第一个字符'H'和'h'是不同的字符。但是，方法 equalsIgnoreCase 会认为它们是相等的。例如，下述语句将显示 Equal：

```
if ("Hello".equalsIgnoreCase("hello"))
   System.out.println("Equal");
```

请注意，在调用 equalsIgnoreCase 中使用引号括起来的字符串"Hello"是完全有效的。用引号括起来的字符串是一个 String 类型的对象，并且具有 String 类型其他对象所具有的所有方法。

程序清单 3.2　测试字符串是否相等

```
import java.util.Scanner;

public class StringEqualityDemo
{
   public static void main(String[] args)
   {
      String s1, s2;
```

```
        System.out.println("Enter two lines of text:");
        Scanner keyboard = new Scanner(System.in);
        s1 = keyboard.nextLine( );
        s2 = keyboard.nextLine( );

        if (s1.equals(s2))
            System.out.println("The two lines are equal.");
        else
            System.out.println("The two lines are not equal.");
        if (s2.equals(s1))
            System.out.println("The two lines are equal.");
        else
            System.out.println("The two lines are not equal.");
        if (s1.equalsIgnoreCase(s2))
            System.out.println(
                    "But the lines are equal, ignoring case.");
        else
            System.out.println(
                    "Lines are not equal, even ignoring case.");
    }
}
```

对 equals 方法的这两种调用是等价的

样本输出

```
Enter two lines of text:
Java is not coffee.
Java is NOT COFFEE.
The two lines are not equal.
The two lines are not equal.
But the lines are equal, ignoring case.
```

疑难杂症　在字符串中使用==

当运算符==作用于两个字符串（或其他任何两个对象）时，表示的是测试它们是否保存在相同的内存位置上。第 5 章讨论这个概念，这里只需知道运算符==不是用于测试两个字符串是否相等就可以了。

对于本章介绍的应用程序类型，使用运算符==来测试字符串的相等性可以给出正确答案。但是，这样做是一个危险的习惯。应该使用方法 equals 而不是==来测试字符串的相等性。类似的说明也适用于其他比较运算符，比如<和方法 compareTo，后面将介绍这些运算符。

扼要重述　方法 equals 和 equalsIgnoreCase

要测试字符串的相等性，或者使用 equals 方法，或者使用 equalsIgnoreCase 方法。

语法

```
String.equals(Other_String)
String.equalsIgnoreCase(Other_String)
```

示例

```
String s1 = keyboard.next();    //keyboard 是一个 Scanner 对象
if (s1.equals("Hello"))
   System.out.println("The string is Hello. ");
else
   System.out.println("The string is not Hello. ");
```

程序经常需要比较两个字符串，看哪一个字符串以字母为序时排在另一个字符串的前面。正像不能使用==测试两个字符串是否相等一样，也不能使用诸如<和>这样的运算符测试字母排序。取而代之的是，应该使用 String 类的方法 compareTo，第 2 章的图 2.5 描述过这个方法。

方法 compareTo 测试两个字符串，确定它们的字典顺序。字典顺序类似于字母顺序，并且有时候（但并不总是）与字母顺序相同。在字典顺序中，字母和其他字符按照它们的 Unicode 编码次序排序。在本书后面的附录中给出了 Unicode 编码。

如果 s1 和 s2 是 String 类型的两个变量，并且已被赋值，那么，方法调用

```
s1.compareTo(s2)
```

将比较两个字符串的字典顺序，并且

- 如果 s1 在 s2 前面，返回负数；
- 如果两个字符串相等，返回零；
- 如果 s1 在 s2 后面，返回正数。

因此，对于如下的布尔表达式

```
s1.compareTo(s2) < 0
```

如果 s1 按字典顺序排在 s2 前面，那么返回 true，否则返回 false。例如，下述代码将生成正确输出：

```
if (s1.compareTo(s2) < 0)
   System.out.println(s1 + " precedes " + s2 +
                      " in lexicographic ordering ");
else if (s1.compareTo(s2) > 0)
   System.out.println(s1 + " follows " + s2 +
                       " in lexicographic ordering ");
else // s1.compareTo(s2) ==0
   System.out.println(s1 + " equals " + s2);
```

但是，假定想要检查字母顺序，而不是字典顺序。如果查看一下 Unicode 编码图，可以看到，在字典顺序中，所有大写字母都出现在所有小写字母前面。例如，按字典顺序，

'Z'出现在'a'前面。因此，当比较两个由大写和小写字母混合组成的字符串时，字典顺序和字母顺序并不相同。但是，在 Unicode 编码中，所有小写字母都按字母顺序排序，所有大写字母也都按字母顺序排序。因此，对于由所有小写字母组成的两个字符串来说，字典顺序与常规的字母顺序相同。类似地，对于由所有大写字母组成的两个字符串来说，字典顺序与常规的字母顺序相同。因此，要按照常规字母顺序比较两个包含字母的字符串，只需把这两个字符串都转换为大写字母或小写字母，之后按照字典顺序比较就可以。让我们考察一下它在 Java 中的具体细节。

假设把包含字母的两个字符串 s1 和 s2 转换为大写字母形式，然后使用 compareTo 来测试大写版本 s1 和 s2 的字典顺序。下述语句也能生成正确的结果：

```
String upperS1 = s1.toUpperCase();
String upperS2 = s2.toUpperCase();
if (upperS1.compareTo(upperS2) < 0)
   System.out.println(s1 + " precedes " + s2 +
                      " in ALPHABETIC ordering ");
else if (upperS1.compareTo(upperS2) > 0)
   System.out.println(s1 + " follows " + s2 +
                     " in ALPHABETIC ordering ");
else // s1.compareTo(s2) == 0
   System.out.println(s1 + " equals " + s2 +
                      " ignoring case ");
```

编程技巧　字母顺序

要知道两个由字母组成的字符串是否按字母排序，在使用方法 compareTo 比较两个字符串之前，必须确保所有字母都为统一的大写或小写。可以使用 String 的 toUpperCase 或 toLowerCase 方法完成这个任务。如果遗漏了这个步骤，那么使用 compareTo 方法按字典顺序比较（并不是按字母顺序比较）。

自测题

1. 假定 goals 是一个 int 类型的变量。请编写一条 if-else 语句，如果变量 goals 的值大于 10，显示单词 Wow，如果 goals 的值最多为 10，那么就显示词汇 Oh well。
2. 假定 goals 和 errors 是 int 类型的变量。请编写一条 if-else 语句，如果变量 goals 的值大于 10 并且 errors 的值为 0，显示单词 Wow，否则，应显示词汇 Oh well。
3. 假定 salary 和 deduction 是 double 类型的变量，它们已经具有给定的值。请编写一条 if-else 语句，当 salary 至少与 deduction 一样多时，显示 OK，并将变量 net 设置为 salary 减去 deduction。但是，如果 salary 小于 deduction，if-else 语句只显示词汇 No way，且不修改任何变量的值。
4. 假定 speed 和 visibility 都是 int 类型的变量。请编写一条 if 语句，当 speed 的值超过 25 并且 visibility 的值小于 20 时，将变量 speed 的值设置为 25，并显示单词 Caution。没有

else 语句部分。

5. 假定 salary 和 bonus 都是 double 类型的变量。请编写一条 if-else 语句，如果 salary 大于等于 MIN_SALARY 或 bonus 大于等于 MIN_BONUS，显示单词 OK。否则，应该显示 Too low。MIN_SALARY 和 MIN_BONUS 都是具名常量。
6. 假定 nextWord 是一个 String 类型的变量，它已给定了一个全部由字母组成的值。请编写一些 Java 代码，如果 nextWord 按字母顺序排在字母 N 前面，那么显示消息 First half of the alphabet。如果按字母顺序 nextWord 不是排在字母 N 前面，你的代码应该显示 Second half of the alphabet。在表示字母 N 时，要小心谨慎。这里需要一个 String 值，而不是 char 值。
7. 假定 x1 和 x2 是两个具有给定值的变量。当这些变量为 int 类型时，如何测试这两个值是否相等？当这些变量为 String 类型时，又该如何测试？

3.1.4　嵌套 if-else 语句

if-else 语句中可以在其内包含任何类型的语句。特别地，可以把一条 if-else 语句嵌套在另一条 if-else 语句中，如下语句所示：

```
if (balance >= 0)
   if (INTEREST_RATE >= 0)
      balance = balance + (INTEREST_RATE * balance) / 12;
   else
      System.out.println("Cannot have a negative interest. ");
else
   balance = balance - OVERDRAWN_PENALTY;
```

如果 balance 的值大于或等于零，执行下面整条 if-else 语句：

```
if (INTEREST_RATE >= 0)
   balance = balance + (INTEREST_RATE * balance) / 12;
else
   System.out.println("Cannot have a negative interest. ");
```

通过添加大括号可以让嵌套语句更加清晰，比如：

```
if (balance >= 0)
{
   if (INTEREST_RATE >= 0)
      balance = balance + (INTEREST_RATE * balance) / 12;
   else
      System.out.println("Cannot have a negative interest. ");
}
else
   balance = balance - OVERDRAWN_PENALTY;
```

在这个示例中，大括号有助于提高代码的清晰性，但严格地讲，这里也可以不要。

在其他情况下，大括号是必要的。例如，如果省略了 else，那么事情就会变得有点棘

手。下面两条语句外观上看起来它们的差别仅仅在于一条语句包含了一对大括号，但它们并不相同：

```
//第一个版本：带大括号
if (balance >= 0)
{
   if (INTEREST_RATE >= 0)
      balance = balance + (INTEREST_RATE * balance) / 12;
}
else
   balance = balance - OVERDRAWN_PENALTY;

//第二个版本：无大括号
if (balance >= 0)
   if (INTEREST_RATE >= 0)
      balance = balance + (INTEREST_RATE * balance) / 12;
else
   balance = balance - OVERDRAWN_PENALTY;
```

在 if-else 语句中，每一个 else 都与前面未匹配的 if 相匹配。在第二个版本（没有大括号的版本）中，else 是与第二个 if 匹配的，尽管编写代码时给出了误导性的缩进。因此，其意义为：

```
//等价于第二个版本
if (balance >= 0)
{
   if (INTEREST_RATE >= 0)
      balance = balance + (INTEREST_RATE * balance) / 12;
   else
      balance = balance - OVERDRAWN_PENALTY;
}
```

为了更清晰地看出它们之间的区别，考虑 balance 大于等于零时发生了什么。在第一个版本中，发生下述动作：

```
if (INTEREST_RATE >= 0)
   balance = balance + (INTEREST_RATE * balance) / 12;
```

如果 balance 不大于等于零，在第一个版本中，发生下述动作：

```
balance = balance - OVERDRAWN_PENALTY;
```

在第二个版本中，如果 balance 大于等于零，执行下述整条 if-else 语句：

```
if (INTEREST_RATE >= 0)
   balance = balance + (INTEREST_RATE * balance) / 12;
else
   balance = balance - OVERDRAWN_PENALTY;
```

如果 balance 不大于等于零，在第二个版本中，没有发生任何动作。

编程技巧　匹配 else 与 if

在 if-else 语句中，每一个 else 都要与其前面最靠近的 if 配对。使用与语句意义一致的缩进可以清晰表达你的意图。但请谨记，编译器忽略缩进。当存在疑问时，使用大括号可以让语句的意义明确化。

3.1.5　多分支 if-else 语句

如果具备分支两条路径的能力，那么就有分支四条路径的能力。只需要分出两条路径，并对这两条路径中的每一条路径再分出两条路径。使用这一技巧，就可以嵌套 if-else 语句，生成产生任意多个可能的多路径分支。程序员具有完成这项任务的标准方法。事实上，将其处理为一种新的分支语句而不仅仅是几个嵌套语句已经成为一种标准。让我们以一个示例作为开始。

假定 balance 是一个变量，它保存了你的账户的余额，你想知道你的余额是正值、负值（透支）还是零。为了避免有关精度的问题，假定 balance 是你账户中的美元数额，而美分部分被忽略。也就是说，假定 balance 是 int 类型的变量。为了得出你的余额是正数、负数还是零，应使用下述嵌套 if-else 语句：

```
if (balance > 0)
   System.out.println("Positive balance ");
else
   if (balance < 0)
      System.out.println("Negative balance ");
   else
      if (balance == 0)
          System.out.println("Zero balance ");
```

与这条语句等价但更清晰、更受欢迎的编写方式如下：

```
if (balance > 0)
   System.out.println("Positive balance ");
else if (balance < 0)
   System.out.println("Negative balance ");
else if (balance == 0)
   System.out.println("Zero balance ");
```

我们称这种形式的语句为多分支 if-else 语句。这个语句就是普通的嵌套 if-else 语句，该语句的缩进方式反映了多分支 if-else 语句的思维方式。

当执行多分支 if-else 语句时，计算机一个接一个地测试布尔表达式，从顶部开始。当第一个为 true 的布尔表达式找到后，执行跟在该布尔表达式后面的语句。例如，如果 balance 大于零，那么上面代码将显示 Positive balance。如果 balance 小于零，那么显示 Negative

balance。最后，如果 balance 等于零，那么显示 Zero balance。这里正好生成三个可能输出中的某一个，具体产生哪一个输出，与变量 balance 的值有关。

在这个示例中，有三种可能性。也可以有任何多个可能性，只须添加更多的 else-if 部分即可。而且，这个示例中的可能性是互斥的。也就是说，在给定时刻，只有一个可能性为 true。也可以使用任意逻辑表达式，即使它们不是互斥的表达式也如此。如果多个逻辑表达式都为 true，那么只有与第一个为 true 的逻辑表达式相对应的动作才会被执行。多分支 if-else 语句永远也不会执行多个动作。

如果没有任何逻辑表达式为 true，那么什么也不发生。但是，在末尾增加一条 else 子句（没有任何 if）是一种良好的实践，它将在没有布尔表达式为 true 时执行。事实上，我们可以以这种形式重新编写前面的余额检查程序。我们知道，如果 balance 既不是正值，也不是负值，那么它一定是零。因此，不需要如下的最后一条测试：

```
if (balance == 0)
```

于是，前面的多分支 if-else 语句等价于下面的语句：

```
if (balance > 0)
   System.out.println("Positive balance ");
else if (balance < 0)
   System.out.println("Negative balance ");
else
   System.out.println("Zero balance ");
```

扼要重述　多分支 if-else 语句

语法

```
if (Boolean_Expression_1)
   Action_1
else if  (Boolean_Expression_2)
   Action_2
⋮
else if  (Boolean_Expression_n)
   Action_n
else
   Default_Action
```

这些 Action 都是 Java 语句。一个接一个地测试布尔表达式，从顶部的逻辑表达式开始测试。一旦找到某个逻辑表达式为 true，就执行跟在该 true 逻辑表达式后面的动作，后继的测试和动作被忽略。如果没有逻辑表达式为 true，那么执行 Default_Action。图 3.8 给出了这种形式的 if-else 语句的语义。

示例

```
if (number < 10)
   System.out.println("number < 10");
```

```
else if (number < 50)
    System.out.println("number >= 10 and number < 50 ");
else if (number < 100)
    System.out.println("number >= 50 and number < 100 ");
else
    System.out.println("number >= 100");
```

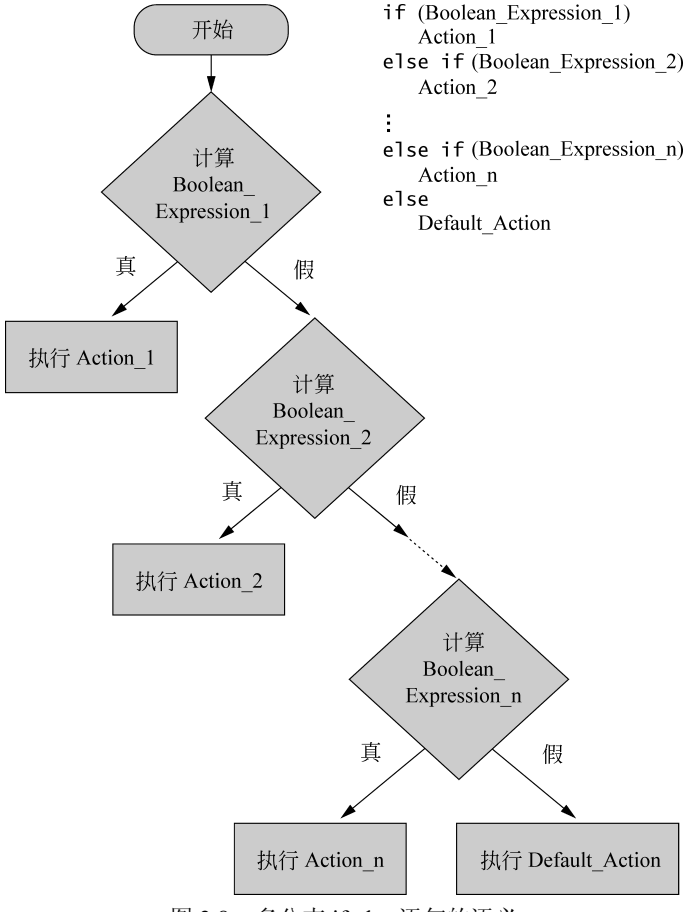

图 3.8　多分支 if-else 语句的语义

编程示例　转换字母等级

程序清单 3.3 包含了一个根据传统规则转换字母等级的程序：90 级以上分数为 A，80 分数段的分数为 B，以此类推。

请注意，与使用任何多分支 if-else 语句一样，布尔表达式依次被计算，因此，除非第一个表达式被计算为 false，否则第二个逻辑表达式不会被计算。因此，当且仅当第二个逻辑表达式被计算时，我们知道第一个逻辑表达式为 false，因此我们知道分数小于 90。

因此，如果把

```
(score >=80)
```

替换为

```
(score >=80 && score <90)
```

那么该多分支 if-else 语句意义不变。

对每一个布尔表达式使用相同类型分型方法，可以看到程序清单 3.3 中的多分支 if-else 语句等价于下述语句：

```
if (score >=90)
   grade = 'A';
else if ((score >=80) && (score <90))
   grade = 'B';
else if ((score >=70) && (score <80))
   grade = 'C';
else if ((score >=60) && (score <70))
   grade = 'D';
else
   grade = 'F';
```

绝大多数程序员会使用程序清单 3.1 中的版本，原因在于它的效率更高一些，也更秀气，但这两个版本都是可以接受的。

程序清单 3.3 使用多分支 if-else 语句转换字母等级

```
import java.util.Scanner;

public class Grader
{
   public static void main(String[] args)
   {
      int score;
      char grade;

      System.out.println("Enter your score: ");
      Scanner keyboard = new Scanner(System.in);
      score = keyboard.nextInt( );

      if (score >= 90)
         grade = 'A';
      else if (score >= 80)
         grade = 'B';
      else if (score >= 70)
         grade = 'C';
      else if (score >= 60)
         grade = 'D';
      else
         grade = 'F';
```