Chapter 5
Conditional Control Structures

Statements that branch to perform one action or another depending on a condition are used to give an application decision-making capabilities. This chapter focuses on the `if` and `switch` conditional control structures.

The if Statement

conditional control structure

The `if` statement is a *conditional control structure*, also called a *decision structure*, which executes a set of statements when a condition is true. Conditional control structures are used to change program flow. The `if` statement takes the form:

```
if (<condition>) {
   <statements>
}
```

TIP Using = instead of == in an if statement condition generates an error.

For example, in the following `if` statement, `guess == 7` is the condition, and there is one statement that will be executed when this condition is true:

```
if (guess == SECRET_NUM) {
   System.out.println("You guessed it!");
}
```

The `==` relational operator determines if the value of `guess` is equal to the value of `SECRET_NUM`. If equal, the println() statement executes. If not, then program flow continues to the next statement after the closing brace of the `if` statement.

Boolean expression

relational operators

The condition of an `if` statement is a *Boolean expression*, which evaluates to either `true` or `false`. *Relational operators* can be used to form Boolean expressions. There are six relational operators:

Operator	Meaning
==	equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
!=	not equal

Comparing Objects

Because objects point to a location in memory, rather than directly storing a value, they cannot be compared using built-in relational operators. A class defines methods that are used to compare objects.

A `boolean` variable may also be used as the condition of an `if` statement because its value is `true` or `false`. For example, in the following statements, the message is displayed:

```
boolean gameOver = true;
if (gameOver) {
   System.out.println("Thanks for playing!");
}
```

Roundoff Error

The condition of an `if` statement should never make an equality comparison between floating point numbers because of the possibility of roundoff error. A *roundoff error* occurs when a floating point number cannot be exactly represented in binary notation by the computer. For example, the decimal number 0.8 is a repeating decimal in binary form (0.1100110011…). Since there are only a finite number of bits in memory that can be used to represent a number, a repeating decimal must be rounded off, preventing an exact representation.

For example, consider the following statements:

```
if ((4.80 * 100 – 480) == 0) {
   System.out.println("Zero.");
}
```

The condition seems to evaluate to `true`. However, the number 4.80 cannot be exactly represented in binary. Therefore, when 4.80 is multiplied by 100 the result is slightly different from 480. This difference is due to roundoff error. In this case, subtracting 480 from 4.80 * 100 results in a very small number that is not equal to 0 (–1.77635683940025E-14). The following statements are one way to take into consideration possible roundoff error:

```
final double VERY_SMALL_VALUE = 0.000001
if ((4.80 * 100 – 480) < sngVERY_SMALL_VALUE) {
   System.out.println("Zero.");
}
```

Review: SurfsUp – part 1 of 3

Create a SurfsUp application that prompts the user for the wave height and then displays "Great day for surfing!" when the waves are 6 feet and over.

The if-else Statement

The `if` statement can include an optional `else` clause that is executed when the `if` condition evaluates to false. The `if-else` statement takes the following form:

```
if (<condition>) {
   <statements>
} else {
   <statements>
}
```

For example, in the following `if-else` statement, different messages are displayed for correct and incorrect guesses:

```
if (guess == SECRET_NUM) {
   System.out.println("You guessed it!");
} else {
   System.out.println("Try again.");
}
```

The indentation and organization of the `if-else` is important for readability. The structure shown is a code convention that clearly indicates the actions for a true condition and the actions for a false condition.

Review: SurfsUp – part 2 of 3

Modify the SurfsUp application to display "Great day for surfing!" when the waves are 6 feet or over and "Go body boarding!" when the waves are less than 6 feet.

Review: CircleCircumference – part 2 of 2

Modify the CircleCircumference application from Chapter 4 so that the message "Negative radii are illegal." is displayed if a negative number is entered by the user for the radius value. Otherwise the application should calculate and display the circumference of the circle.

Nested Statements

An `if-else` statement can contain another `if-else` or `if` statement. Statements placed within the same type of statements are called *nested*. For example, the nested `if-else` gives a hint when the user does not guess the correct number:

```
if (guess == SECRET_NUM) {      //correct
   System.out.println("You guessed it!");
} else {
   if (guess < SECRET_NUM) {    //too low
       System.out.println("Too low.");
   } else {                     //too high
       System.out.println("Too high.");
   }
}
```

Carefully indenting the statements makes it clear which are nested.

Review: Stages

Create a Stages application that prompts the user for an age. For an age over 18, `adult` is displayed. For an age less than or equal to 18, `toddler` is displayed when the age is less than or equal to 5, `child` when the age is less than or equal to 10, `preteen` when the age is less than or equal to 12, and `teen` when the age is over 12.

The if-else if Statement

The `if-else if` statement is used to decide among three or more actions and takes the form:

```
if (<condition>) {
   <statements>
} else if (<condition>) {
   <statements>
} else {
   <statements>
}
```

There can be multiple `else if` clauses, and the last `else` clause is optional. For example, there are three possible decisions in the `if-else if` statement below:

```
if (guess == SECRET_NUM) {          //correct
   System.out.println("You guessed it!");
} else if (guess < SECRET_NUM) {    //too low
   System.out.println("Too low.");
} else {                            //too high
   System.out.println("Too high.");
}
```

The logic used in developing an `if-else if` statement is important. For example, when testing a range of numbers, `if` conditions must be properly ordered because statements are executed for the first true condition only and then program flow continues to the next statement after the `if-else if`.

When choosing between nested `if-else` statements and a single `if-else if` statement, the `if-else if` allows only one branch to execute and the conditions show a clear sequence. In general, the `if-else if` statement is easier to read and understand.

Commenting Complex Decision Structures

Decision structures with many branches can quickly become difficult to understand. Brief inline comments can make code much more readable. This is especially important for the last branch of a decision structure, which usually does not include an explicit condition.

Review: SurfsUp – part 3 of 3

Modify the SurfsUp application to display "Great day for surfing!" when the waves are 6 feet or over, "Go body boarding!" when the waves are between 3 and 6 feet, "Go for a swim." when the waves are from 0 to 3 feet, and "Whoa! What kind of surf is that?" otherwise.

Review: Discriminant

In mathematics, the quantity $b^2 - 4ac$ is called the "discriminant." Create a Discriminant application that prompts the user for the values of a, b, and c an then displays "No roots" if the discriminant is negative, "One root" is the discriminant is zero, and "Two roots" if the discriminant is positive. Application output should look similar to:

```
Enter the value for a: 7
Enter the value for b: -9
Enter the value for c: 2
Two roots
```

The switch Statement

The `switch` statement is a conditional control structure that uses the result of an expression to determine which statements to execute. The `switch` statement is sometimes preferable to the `if-else if` statement because code may be easier to read. The `switch` statement takes the form:

```
switch (<integer expression>) {
  case x:
      <statement>;
      break;
  ...
  default:
      <statements>;
      break;
}
```

case
break

The expression must evaluate to an integer. There can be multiple `case` clauses. The `break` statement is necessary to move program control to the next statement after the `switch` statement. The `default` code is optional and is executed when none of the previous cases are met. For example, when `score` is `5`, the `case 5` statement executes and then program control moves to the next statement after the `switch` (skipping the `case 10` statement):

```
switch (score) {
  case 0: System.out.println("Better luck next time."); break;
  case 5: System.out.println("Pretty good."); break;
  case 10: System.out.println("Great!"); break;
}
```

If the `break` statement is not included in a `case` clause, execution continues on to the next statement within the `switch` statement. This can be useful when the same set of statements applies to more than one situation:

```
switch (score) {
  case 0: System.out.println("Better luck next time."); break;
  case 1:
  case 2:
  case 3:
  case 4:
  case 5: System.out.println("Pretty good."); break;
  case 6:
  case 7:
  case 8:
  case 9:
  case 10: System.out.println("Great!"); break;
}
```

In this statement, “Pretty good.” is displayed when the score is 1, 2, 3, 4, or 5. Scores 6 though 10 display “Great”.

Review: Hurricane

The Saffir-Simpson Hurricane Scale provides a rating (a category) depending on the current intensity of a hurricane. Create a Hurricane application that displays the wind speed for the hurricane category entered by the user. Display the speed in miles per hour (mph), knots (kts), and kilometers per hour (km/hr). Refer to the Saffir-Simpson Hurricane Scale below for wind speeds:

Category 1: 74-95 mph or 64-82 kt or 119-153 km/hr

Category 2: 96-110 mph or 83-95 kt or 154-177 km/hr

Category 3: 111-130 mph or 96-113 kt or 178-209 km/hr

Category 4: 131-155 mph or 114-135 kt or 210-249 km/hr

Category 5: greater than 155 mph or 135 kt or 249 km/hr

Generating Random Numbers

Linear Congruential Method

Games, simulators, screen savers, and many other types of applications make use of random numbers. A widely used method for generating random numbers is called the *Linear Congruential Method*. This method uses a formula to generate a sequence of numbers. Although the numbers in the sequence vary and for most applications can be considered random, the sequence will at some point repeat. Therefore, random numbers in a computer application are referred to as *pseudorandom* (like random).

pseudorandom

Random class

Java includes the Random class in the java.util package for generating random numbers. This class uses the Linear Congruential Method. Some of the Random class methods include:

Class Random (java.util.Random)

Methods

`nextInt()`	returns the next random integer in the random number generator's sequence.
`nextInt(int n)`	returns the next random integer between 0 (inclusive) and `n` in the random number generator's sequence.
`nextDouble()`	returns the next random double between 0.0 and 1.0 in the random number generator's sequence.

The RandomNumberDemo class below instantiates a Random object and displays the first five numbers between 0 and 100 in its sequence:

```
public class RandomNumberDemo {

  public static void main(String[] args) {
      Random r = new Random();

      System.out.println("First number: " + r.nextInt(100));
      System.out.println("Second number: " + r.nextInt(100));
      System.out.println("Third number: " + r.nextInt(100));
      System.out.println("Fourth number: " + r.nextInt(100));
      System.out.println("Fifth number: " + r.nextInt(100));
  }
}
```

RandomNumberDemo produces output similar to the following:

```
First number: 13
Second number: 80
Third number: 93
Fourth number: 90
Fifth number: 46
```

random numbers in a range

To generate a random number in a range with a minimum value greater than 0, the following formula is used:

```
r.nextInt(highNum - lowNum + 1) + lowNum
```

A similar formula can be used to generate random doubles in a range:

```
(highNum - lowNum + 1) * r.nextDouble() + lowNum
```

seed

The Linear Congruential Method requires a *seed*, which is a starting value, for calculating a sequence of numbers. Java automatically generates this seed when a new Random object is instantiated. However, if a seed value is specified, the same sequence of "random" numbers will be generated each time the program is run. This can be useful for debugging. After the program has been tested, remove the seed so that the sequence of generated numbers vary each time the application runs. For example, the code below specifies a seed. Two runs of the application are displayed:

```
public class RandomNumberDemo {

  public static void main(String[] args) {
      Random r = new Random(10);

      System.out.println("First number: " + r.nextInt(100));
      System.out.println("Second number: " + r.nextInt(100));
      System.out.println("Third number: " + r.nextInt(100));
      System.out.println("Fourth number: " + r.nextInt(100));
      System.out.println("Fifth number: " + r.nextInt(100));

  }
}
```

first run:

```
First number: 17
Second number: 71
Third number: 98
Fourth number: 34
Fifth number: 64
```

second run:

```
First number: 17
Second number: 71
Third number: 98
Fourth number: 34
Fifth number: 64
```

Two runs of RandomNumberDemo using the same seed. Note that both runs generate the same sequence of numbers.

Review: RandomNum

Create a RandomNum application that prompts the user for two numbers. The first number is a minimum value and the second is a maximum value. RandomNum then displays an integer between the min and max values entered by the user.

Compound Boolean Expressions

&& and || logical And

Conditions with complex criteria are formed using the `&&` and `||` operators. The `&&` operator is called the *logical And*. It is used to form an expression that evaluates to true only when both operands are also true. The `||` operator is called the *logical Or*. An expression formed with this operator evaluates to true when either operand is true. For example, the following statement tests for invalid guesses:

logical Or

TIP The | key is located above the Enter key on most standard keyboards.

```
if (guess < 1 || guess > 50) {        //invalid guess
   System.out.println("Invalid guess.");
} else if (guess == SECRET_NUM) {   //correct guess
   System.out.println("You guessed it!");
}
```

When a guess is either less than 1 *or* greater than 50, "Invalid guess." is displayed. Note that the expression also evaluates to true when both operands are true. The condition in the `if` statement is called a *compound Boolean expression* because more than one Boolean expression determines whether the condition is true or false.

truth table

How a compound Boolean expression evaluates with `&&` and `||` operators can be shown with truth tables. A *truth table* shows the possible outcomes of compound Boolean expressions:

	And			Or	
Exp1	Exp2	Result	Exp1	Exp2	Result
True	True	True	True	True	True
True	False	False	True	False	True
False	True	False	False	True	True
False	False	False	False	False	False

As another example, consider an application that computes a discount depending on the item and quantity purchased:

```
if (itemNum == 873 && quantity > 50) {  //more than 50 of 873
   discount = 1;                        //$1 discount
}
```

This `if` statement executes the `discount = 1` statement if item number is 873 *and* quantity is greater than 50.

!

A third operator is `!`. The `!` operator is called the *logical Not*. An expression including `!` is evaluated according to the following truth table:

	Not
Exp	Result
True	False
False	True

For example, the following statements display a message when the item number is *not* 873:

```
if (!itemNum == 873) {        //any item EXCEPT 873
   System.out.println("No discount given.");
}
```

short circuit evaluation

Java uses short-circuit evaluation for determining the result of a compound Boolean expression that includes `&&` or `||`. In *short-circuit evaluation*, the left operand is evaluated first. If the result of the entire expression can be determined by the value of the left operand, then no other operands will be evaluated. For example, the expression `x < 0 || x > 5` evaluates to `true` if `x` is less than `0` regardless of the value of `x > 5`. Therefore, when `x` is less then `0`, the second operand will not be evaluated. As another example, the expression `x > 5 && x < 20` evaluates to `false` if `x` is less than or equal to `5` regardless of the value of `x < 20`. Therefore, when `x` is less than or equal to `5`, the second operand will not be evaluated.

order of operations

In the order of operations, `!` is evaluated before `&&`. `||` is evaluated last. For example, the expression `!5 < 6 || 2 > 4 && 3 < 6` evaluates to false because `!5 < 6` is performed first, then `2 > 4 && 3 < 6`, and then `False || False`. Use parentheses to change operator precedence and to make code more readable.

Review: Delivery

Create a Delivery application that prompts the user for the length, width, and height of a package, and then displays "Reject" if any dimension is greater than 10, and "Accept" if all the dimensions are less than or equal to 10.

The Math Class

java.lang

Java includes the Math class in the java.lang package for performing math functions such as exponentiation and square root. The Math class contains numerous methods, which include:

TIP The Math class also contains the `double` constant PI that approximates π.

Class Math (java.lang.Math)

Methods

`abs(num)` returns the absolute value of `num`, which can be an `int` or a `double` value.

`pow(double num1, double num2)` returns the `num1` raised to the `num2` power.

`sqrt(double num)` returns the square root of `num`, where `num` is a positive number.

Calling a Math method requires using the class name. For example, `Math.abs(-3)` returns 3. The application on the next page demonstrates the Math methods:

```
import java.lang.Math;

 public class TestMathMethods {

   public static void main(String[] args) {
       int posNum = 12, negNum = -12;
       int num1 = 2, num2 = 6;
       int square = 49;

       System.out.println("The absolute value of " + posNum
                          + " is " + Math.abs(posNum));
       System.out.println("The absolute value of " + negNum
                          + " is " + Math.abs(negNum));
       System.out.println(num1 + " raised to the " + num2
                       + " power is " + Math.pow(num1, num2));
       System.out.println("The square root of " + square
                          + " is " + Math.sqrt(square));
   }
}
```

The TestMathMethods produces the following output:

```
The absolute value of 12 is 12
The absolute value of -12 is 12
2 raised to the 6 power is 64.0
The square root of 49 is 7.0
```

Review: PerfectSquare

Create a PerfectSquare application that prompts the user for an integer and then displays a message indicating whether or not the number is a perfect square. This can be determined by finding the square root of a number, truncating it (by casting the `double` result), and then squaring that result.