Not everything that can be counted counts, and not every thing that counts can be counted.
—Albert Einstein

Who can control his fate?
—William Shakespeare

The used key is always bright.
—Benjamin Franklin

Intelligence … is the faculty of making artificial objects, especially tools to make tools.
—Henri Bergson

Every advantage in the past is judged in the light of the final issue.
—Demosthenes

5

Control Statements: Part 2

Objectives

In this Chapter you'll learn:

- The essentials of counter-controlled repetition.
- To use the `for` and `do...while` repetition statements to execute statements in a program repeatedly.
- To understand multiple selection using the `switch` selection statement.
- To use the `break` and `continue` program control statements to alter the flow of control.
- To use the logical operators to form complex conditional expressions in control statements.

Self-Review Exercises

5.1 Fill in the blanks in each of the following statements:

a) Typically, _____ statements are used for counter-controlled repetition and _____ statements for sentinel-controlled repetition.

ANS: `for`, `while`.

b) The `do...while` statement tests the loop-continuation condition _____ executing the loop's body; therefore, the body always executes at least once.

ANS: after.

c) The _____ statement selects among multiple actions based on the possible values of an integer variable or expression.

ANS: `switch`.

d) The _____ statement, when executed in a repetition statement, skips the remaining statements in the loop body and proceeds with the next iteration of the loop.

ANS: `continue`.

e) The _____ operator can be used to ensure that two conditions are *both* true before choosing a certain path of execution.

ANS: `&&` (conditional AND).

f) If the loop-continuation condition in a `for` header is initially _____, the program does not execute the `for` statement's body.

ANS: `false`.

g) Methods that perform common tasks and do not require objects are called _____ methods.

ANS: `static`.

5.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The `default` case is required in the `switch` selection statement.

ANS: False. The `default` case is optional. If no default action is needed, then there is no need for a `default` case.

b) The `break` statement is required in the last case of a `switch` selection statement.

ANS: False. The `break` statement is used to exit the `switch` statement. The `break` statement is not required for the last case in a `switch` statement.

c) The expression `( ( x > y ) && ( a < b ) )` is true if either `x > y` is true or `a < b` is true.

ANS: False. Both of the relational expressions must be true for the entire expression to be true when using the `&&` operator.

d) An expression containing the `||` operator is true if either or both of its operands are true.

ANS: True.

e) The comma (`,`) formatting flag in a format specifier (e.g., `%,20.2f`) indicates that a value should be output with a thousands separator.

ANS: True.

f) To test for a range of values in a `switch` statement, use a hyphen (`-`) between the start and end values of the range in a `case` label.

ANS: False. The `switch` statement does not provide a mechanism for testing ranges of values, so every value that must be tested should be listed in a separate `case` label.

g) Listing cases consecutively with no statements between them enables the cases to perform the same set of statements.

ANS: True.

5.3 Write a Java statement or a set of Java statements to accomplish each of the following tasks:

a) Sum the odd integers between 1 and 99, using a `for` statement. Assume that the integer variables `sum` and `count` have been declared.

ANS:

```
sum = 0;
for ( count = 1; count <= 99; count += 2 )
   sum += count;
```

b) Calculate the value of `2.5` raised to the power of `3`, using the `pow` method.

ANS:
```
double result = Math.pow( 2.5, 3 );
```

c) Print the integers from 1 to 20, using a `while` loop and the counter variable `i`. Assume that the variable `i` has been declared, but not initialized. Print only five integers per line. [*Hint:* Use the calculation `i % 5`. When the value of this expression is 0, print a newline character; otherwise, print a tab character. Assume that this code is an application. Use the `System.out.println()` method to output the newline character, and use the `System.out.print( '\t' )` method to output the tab character.]

ANS:
```
i = 1;

while ( i <= 20 )
{
   System.out.print( i );

   if ( i % 5 == 0 )
      System.out.println();
   else
      System.out.print( '\t' );

   ++i;
}
```

d) Repeat part (c), using a `for` statement.

ANS:
```
for ( i = 1; i <= 20; i++ )
{
    System.out.print( i );

   if ( i % 5 == 0 )
      System.out.println();
   else
      System.out.print( '\t' );
}
```

5.4 Find the error in each of the following code segments, and explain how to correct it:

a)
```
i = 1;

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

ANS: Error: The semicolon after the `while` header causes an infinite loop, and there is a missing left brace.
Correction: Replace the semicolon by a `{`, or remove both the `;` and the `}`.

b)
```
for ( k = 0.1; k != 1.0; k += 0.1 )
   System.out.println( k );
```

ANS: Error: Using a floating-point number to control a `for` statement may not work, because floating-point numbers are represented only approximately by most computers.
Correction: Use an integer, and perform the proper calculation in order to get the values you desire:

```
for ( k = 1; k != 10; k++ )
   System.out.println( (double) k / 10 );
```

c)
```
switch ( n )
{
   case 1:
      System.out.println( "The number is 1" );
   case 2:
      System.out.println( "The number is 2" );
      break;
   default:
      System.out.println( "The number is not 1 or 2" );
      break;
}
```

ANS: Error: The missing code is the `break` statement in the statements for the first `case`. Correction: Add a `break` statement at the end of the statements for the first `case`. Note that this omission is not necessarily an error if you want the statement of `case 2:` to execute every time the `case 1:` statement executes.

d) The following code should print the values 1 to 10:
```
n = 1;
while ( n < 10 )
   System.out.println( n++ );
```

ANS: Error: An improper relational operator is used in the `while`'s continuation condition. Correction: Use `<=` rather than `<`, or change `10` to `11`.

Exercises

NOTE: Solutions to the programming exercises are located in the `ch05solutions` folder. Each exercise has its own folder named `ex05_##` where ## is a two-digit number representing the exercise number. For example, exercise 5.11's solution is located in the folder `ex05_11`.

5.5 Describe the four basic elements of counter-controlled repetition.

ANS: Counter-controlled repetition requires a control variable (or loop counter), an initial value of the control variable, an increment (or decrement) by which the control variable is modified each time through the loop, and a loop-continuation condition that determines whether looping should continue.

5.6 Compare and contrast the `while` and `for` repetition statements.

ANS: The `while` and `for` repetition statements repeatedly execute a statement or set of statements as long as a loop-continuation condition remains true. Both statements execute their bodies zero or more times. The `for` repetition statement specifies the counter-controlled-repetition details in its header, whereas the control variable in a `while` statement normally is initialized before the loop and incremented in the loop's body. Typically, `for` statements are used for counter-controlled repetition, and `while` statements are used for sentinel-controlled repetition. However, `while` and `for` can each be used for either repetition type.

5.7 Discuss a situation in which it would be more appropriate to use a `do...while` statement than a `while` statement. Explain why.

ANS: If you want some statement or set of statements to execute at least once, then repeat based on a condition, a `do...while` is more appropriate than a `while` (or a `for`). A `do...while` statement tests the loop-continuation condition *after* executing the loop's body; therefore, the body always executes at least once. A `while` tests the loop-continuation condition before executing the loop's body, so the program would need to include the statement(s) required to execute at least once both before the loop and in the body of the loop. Using a `do...while` avoids this duplication of code. Suppose a

program needs to obtain an integer value from the user, and the integer value entered must be positive for the program to continue. In this case, a `do...while`'s body could contain the statements required to obtain the user input, and the loop-continuation condition could determine whether the value entered is less than 0. If so, the loop would repeat and prompt the user for input again. This would continue until the user entered a value greater than or equal to zero. Once this criterion was met, the loop-continuation condition would become false, and the loop would terminate, allowing the program to continue past the loop. This process is often called validating input.

5.8 Compare and contrast the `break` and `continue` statements.

ANS: The `break` and `continue` statements alter the flow of control through a control statement. The `break` statement, when executed in one of the repetition statements, causes immediate exit from that statement. Execution continues with the first statement after the control statement. In contrast, the `continue` statement, when executed in a repetition statement, skips the remaining statements in the loop body and proceeds with the next iteration of the loop. In `while` and `do...while` statements, the program evaluates the loop-continuation test immediately after the `continue` statement executes. In a `for` statement, the increment expression executes, then the program evaluates the loop-continuation test.

5.9 Find and correct the error(s) in each of the following segments of code:

a)
```
For ( i = 100, i >= 1, i++ )
   System.out.println( i );
```

ANS: The F in `for` should be lowercase. Semicolons should be used in the `for` header instead of commas. ++ should be --.

```
1  // Exercise 5.9a: PartA.java
2  public class PartA
3  {
4     public static void main( String args[] )
5     {
6        int i;
7
8        For ( i = 100, i >= 1, i++ )
9           System.out.println( i );
10    } // end main
11 } // end class PartA
```

```
PartA.java:8: ';' expected
      For ( i = 100, i >= 1, i++ )
                                ^
1 error
```

```
1  // Exercise 5.9a Solution: PartACorrected.java
2  public class PartACorrected
3  {
4     public static void main( String args[] )
5     {
6        int i;
7
8        for ( i = 100; i >= 1; i-- )
9           System.out.println( i );
10    } // end main
11 } // end class PartACorrected
```

```
100
99
98
.
.
.
3
2
1
```

b) The following code should print whether integer `value` is odd or even:

```
switch ( value % 2 )
{
   case 0:
      System.out.println( "Even integer" );

   case 1:
      System.out.println( "Odd integer" );
}
```

ANS: A `break` statement should be placed in `case 0:`.

```
1  // Exercise 5.9b: PartB.java
2  public class PartB
3  {
4     public static void main( String args[] )
5     {
6        int value = 8;
7
8        switch ( value % 2 )
9        {
10          case 0:
11             System.out.println( "Even integer" );
12
13          case 1:
14             System.out.println( "Odd integer" );
15
16    } // end main
17 } // end class PartB
```

```
Even integer
Odd integer
```

```
1  // Exercise 5.9b Solution: PartBCorrected.java
2  public class PartBCorrected
3  {
4     public static void main( String args[] )
5     {
6        int value = 8;
7
8        switch ( value % 2 )
9        {
```

```
            case 0:
               System.out.println( "Even integer" );
               break;
            case 1:
               System.out.println( "Odd integer" );
         }
      } // end main
   } // end class PartBCorrected
```

```
Even integer
```

c) The following code should output the odd integers from 19 to 1:

```
for ( i = 19; i >= 1; i += 2 )
   System.out.println( i );
```

ANS: The += operator in the `for` header should be -=.

```
// Exercise 5.9c: PartC.java
public class PartC
{
   public static void main( String args[] )
   {
      int i;

      for ( i = 19; i >= 1; i += 2 )
         System.out.println( i );
   } // end main
} // end class PartC
```

```
19
21
23
25
27
29
.
.
.
```

```
// Exercise 5.9c Solution: PartCCorrected.java
public class PartCCorrected
{
   public static void main( String args[] )
   {
      int i;

      for ( i = 19; i >= 1; i -= 2 )
         System.out.println( i );
   } // end main
} // end class PartCCorrected
```

```
19
17
15
13
11
9
7
5
3
1
```

d) The following code should output the even integers from 2 to 100:

```
counter = 2;

do
{
   System.out.println( counter );
   counter += 2;
} While ( counter < 100 );
```

ANS: The `W` in `while` should be lowercase. < should be <=.

```
1  // Exercise 5.9d: PartD.java
2  public class PartD
3  {
4     public static void main( String args[] )
5     {
6        int counter;
7
8        counter = 2;
9
10       do
11       {
12          System.out.println( counter );
13          counter += 2;
14       } While ( counter < 100 );
15    } // end main
16 } // end class PartD
```

```
PartD.java:14: while expected
      } While ( counter < 100 );
        ^
PartD.java:14: ')' expected
      } While ( counter < 100 );
                               ^
2 errors
```

```
1  // Exercise 5.9d Solution: PartDCorrected.java
2  public class PartDCorrected
3  {
4     public static void main( String args[] )
5     {
6        int counter;
```

```
7
8           counter = 2;
9
10          do
11          {
12             System.out.println( counter );
13             counter += 2;
14          } while ( counter <= 100 );
15       } // end main
16    } // end class PartDCorrected
```

```
2
4
6
.
.
.
96
98
100
```

5.10 What does the following program do?

```
1  // Exercise 5.10: Printing.java
2  public class Printing
3  {
4     public static void main( String[] args )
5     {
6        for ( int i = 1; i <= 10; i++ )
7        {
8           for ( int j = 1; j <= 5; j++ )
9              System.out.print( '@' );
10
11          System.out.println();
12       } // end outer for
13    } // end main
14 } // end class Printing
```

ANS:

```
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
```

5.13 ***(Factorials)*** *Factorials* are used frequently in probability problems. The factorial of a positive integer n (written $n!$ and pronounced "n factorial") is equal to the product of the positive integers

from 1 to *n*. Write an application that calculates the factorials of 1 through 20. Use type `long`. Display the results in tabular format. What difficulty might prevent you from calculating the factorial of 100?

ANS: The value of 100! far exceeds the maximum value that can be stored in a `long`. For calculations involving large integer values, you can use class `java.math.BigInteger` instead. [See the code for this solution in the `ch05solutions\ex05_13` folder.]

5.19 Assume that `i = 1`, `j = 2`, `k = 3` and `m = 2`. What does each of the following statements print?

a) `System.out.println( i == 1 );`

ANS: true.

b) `System.out.println( j == 3 );`

ANS: false.

c) `System.out.println( ( i >= 1 ) && ( j < 4 ) );`

ANS: true.

d) `System.out.println( ( m <= 99 ) & ( k < m ) );`

ANS: false.

e) `System.out.println( ( j >= i ) || ( k == m ) );`

ANS: true.

f) `System.out.println( ( k + m < j ) | ( 3 - j >= k ) );`

ANS: false.

g) `System.out.println( !( k > m ) );`

ANS: false.

5.20 *(Calculating the Value of π)* Calculate the value of π from the infinite series

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

Print a table that shows the value of π approximated by computing the first 200,000 terms of this series. How many terms do you have to use before you first get a value that begins with 3.14159?

ANS: [*Note:* The program starts to converge around 3.14159 at 136121 terms.]

5.26 A criticism of the `break` statement and the `continue` statement is that each is unstructured. Actually, these statements can always be replaced by structured statements, although doing so can be awkward. Describe in general how you'd remove any `break` statement from a loop in a program and replace it with some structured equivalent. [*Hint:* The `break` statement exits a loop from the body of the loop. The other way to exit is by failing the loop-continuation test. Consider using in the loop-continuation test a second test that indicates "early exit because of a 'break' condition."] Use the technique you develop here to remove the `break` statement from the application in Fig. 5.12.

ANS: A loop can be written without a `break` by placing in the loop-continuation test a second test that indicates "early exit because of a 'break' condition." Alternatively, the `break` can be replaced by a statement that makes the original loop-continuation test immediately false, so that the loop terminates. [See the code for this solution in the `ch05solutions\ex05_26` folder.]

5.27 What does the following program segment do?

```
for ( i = 1; i <= 5; i++ )
{
   for ( j = 1; j <= 3; j++ )
   {
      for ( k = 1; k <= 4; k++ )
         System.out.print( '*' );

      System.out.println();
   } // end inner for

   System.out.println();
} // end outer for
```

ANS:

```
****
****
****

****
****
****

****
****
****

****
****
****

****
****
****
```

5.28 Describe in general how you'd remove any `continue` statement from a loop in a program and replace it with some structured equivalent. Use the technique you develop here to remove the `continue` statement from the program in Fig. 5.13.

ANS: A loop can be rewritten without a `continue` statement by moving all the code that appears in the body of the loop after the `continue` statement to an `if` statement that tests for the opposite of the `continue` condition. Thus, the code that was originally after the `continue` statement executes only when the `if` statement's conditional expression is true (i.e., the "continue" condition is false). When the "continue" condition is true, the body of the `if` does not execute and the program "continues" to the next iteration of the loop by not executing the remaining code in the loop's body. [See the code for this solution in the `ch05solutions\ex05_28` folder.]