5

Control Statements: Part 2

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

OBJECTIVES

In this chapter you will 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 are used 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) or `&` (boolean logical 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 must 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:

```
1  // Exercise 5.3a Solution: PartA.java
2  public class PartA
3  {
4     public static void main( String args[] )
5     {
6        int sum;
7        int count;
8
9        sum = 0;
10       for ( count = 1; count <= 99; count += 2 )
11          sum += count;
12
13       System.out.printf( "sum = %d", sum );
14    } // end main
15 } // end class PartA
```

```
sum = 2500
```

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

ANS:

```
1  // Exercise 5.3b Solution: PartB.java
2  public class PartB
3  {
4     public static void main( String args[] )
5     {
6        double result = Math.pow( 2.5, 3 );
7
8        System.out.printf( "2.5 raised to the power of 3 is %.3f", result );
9     } // end main
10 } // end class PartB
```

```
2.5 raised to the power of 3 is 15.625
```

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:

```
1  // Exercise 5.3c Solution: PartC.java
2  public class PartC
```

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

      i = 1;

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

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

         ++i;
      }
   } // end main
} // end class PartC
```

```
1       2       3       4       5
6       7       8       9       10
11      12      13      14      15
16      17      18      19      20
```

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

ANS:

```
// Exercise 5.3d Solution: PartD.java
public class PartD
{
   public static void main( String args[] )
   {
      int i;

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

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

```
1       2       3       4       5
6       7       8       9       10
11      12      13      14      15
16      17      18      19      20
```

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 `}`.

```
1  // Exercise 5.4a: PartA.java
2  public class PartA
3  {
4     public static void main( String args[] )
5     {
6        int i;
7
8        i = 1;
9
10       while ( i <= 10 );
11          i++;
12       }
13    } // end main
14 } // end class PartA
```

```
PartA.java:14: 'class' or 'interface' expected
} // end class PartA
^
PartA.java:15: 'class' or 'interface' expected
^
2 errors
```

```
1  // Exercise 5.4a Solution: PartACorrected.java
2  public class PartACorrected
3  {
4     public static void main( String args[] )
5     {
6        int i;
7
8        i = 1;
9
10       while ( i <= 10 )
11          i++;
12    } // end main
13 } // end class PartACorrected
```

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.

```
1  // Exercise 5.4b: PartB.java
2  public class PartB
3  {
4     public static void main( String args[] )
5     {
6       double k;
7
8       for ( k = 0.1; k != 1.0; k += 0.1 )
9          System.out.println( k );
10    } // end main
11 } // end class PartB
```

```
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
1.0999999999999999
1.2
1.3
1.4000000000000001
1.5000000000000002
1.6000000000000003
1.7000000000000004
1.8000000000000005
1.9000000000000006
.
.
.
```

```
1  // Exercise 5.4b Solution: PartBCorrected.java
2  public class PartBCorrected
3  {
4     public static void main( String args[] )
5     {
6        int k;
7
8        for ( k = 1; k != 10; k++ )
9           System.out.println( (double) k / 10 );
```

```
10      } // end main
11   } // end class PartBCorrected
```

```
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
```

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

```
1  // Exercise 5.4c: PartC.java
2  public class PartC
3  {
4     public static void main( String args[] )
5     {
6        int n = 1;
7
8        switch ( n )
9        {
10          case 1:
11             System.out.println( "The number is 1" );
12          case 2:
13             System.out.println( "The number is 2" );
14             break;
15          default:
16             System.out.println( "The number is not 1 or 2" );
17             break;
18       }
19    } // end main
20 } // end class PartC
```

```
The number is 1
The number is 2
```

```
1  // Exercise 5.4c: PartCCorrected.java
2  public class PartCCorrected
3  {
4     public static void main( String args[] )
5     {
6        int n = 1;
7
8        switch ( n )
9        {
10          case 1:
11             System.out.println( "The number is 1" );
12             break;
13          case 2:
14             System.out.println( "The number is 2" );
15             break;
16          default:
17             System.out.println( "The number is not 1 or 2" );
18             break;
19       }
20    } // end main
21 } // end class PartCCorrected
```

```
The number is 1
```

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` condition.
Correction: Use `<=` rather than `<`, or change `10` to `11`.

```
1  // Exercise 5.4d: PartD.java
2  public class PartD
3  {
4     public static void main( String args[] )
5     {
6        int n;
7
8        n = 1;
9
10       while ( n < 10 )
11          System.out.println( n++ );
12    } // end main
13 } // end class PartD
```

```
1
2
3
4
5
6
7
8
9
```

```
1  // Exercise 5.4d: PartDCorrected.java
2  public class PartDCorrected
3  {
4     public static void main( String args[] )
5     {
6        int n;
7
8        n = 1;
9
10       while ( n <= 10 )
11          System.out.println( n++ );
12    } // end main
13 } // end class PartDCorrected
```

```
1
2
3
4
5
6
7
8
9
10
```

Exercises

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 wa 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:9: ';' expected
         System.out.println( 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        {
10          case 0:
11             System.out.println( "Even integer" );
12             break;
13          case 1:
14             System.out.println( "Odd integer" );
15       }
16    } // end main
17 } // 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 -=.

```
1  // Exercise 5.9c: PartC.java
2  public class PartC
3  {
4     public static void main( String args[] )
5     {
6        int i;
7
8        for ( i = 19; i >= 1; i += 2 )
9           System.out.println( i );
10    } // end main
11 } // end class PartC
```

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

```
1  // Exercise 5.9c Solution: PartCCorrected.java
2  public class PartCCorrected
3  {
4     public static void main( String args[] )
5     {
6        int i;
7
8        for ( i = 19; i >= 1; i -= 2 )
9           System.out.println( i );
10    } // end main
11 } // 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  public class Printing
2  {
3     public static void main( String args[] )
4     {
5        for ( int i = 1; i <= 10; i++ )
6        {
7           for ( int j = 1; j <= 5; j++ )
8              System.out.print( '@' );
9
10          System.out.println();
11       } // end outer for
12    } // end main
13 } // end class Printing
```

ANS:

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

5.11 Write an application that finds the smallest of several integers. Assume that the first value read specifies the number of values to input from the user.

ANS:

```
1  // Exercise 5.11 Solution: Small.java
2  // Program finds the smallest of several integers.
3  import java.util.Scanner;
4
5  public class Small
6  {
7     // finds the smallest integer
8     public void findSmallest()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int smallest = 0; // smallest number
13       int number = 0; // number entered by user
14       int integers; // number of integers
15
16       System.out.print( "Enter number of integers: " );
17       integers = input.nextInt();
```

```
18
19        for ( int counter = 1; counter <= integers; counter++ )
20        {
21           System.out.print( "Enter integer: " );
22           number = input.nextInt();
23
24           if ( counter == 1 )
25              smallest = number;
26           else if ( number < smallest )
27              smallest = number;
28        } // end for loop
29
30        System.out.printf( "Smallest Integer is: %d\n", smallest );
31     } // end method findSmallest
32  } // end of class Small
```

```
1  // Exercise 5.11 Solution: SmallTest.java
2  // Test application for class Small
3  public class SmallTest
4  {
5     public static void main(String args[])
6     {
7        Small application = new Small();
8        application.findSmallest();
9     } // end main
10 } // end class SmallTest
```

```
Enter number of integers: 3
Enter integer: -5
Enter integer: 46
Enter integer: 0
Smallest Integer is: -5
```

5.12 Write an application that calculates the product of the odd integers from 1 to 15.
ANS:

```
1  // Exercise 5.12 Solution: Odd.java
2  // Program prints the product of the odd integers from 1 to 15
3  public class Odd
4  {
5     public static void main( String args[] )
6     {
7        int product = 1; // the product of all the odd numbers
8
9        // loop through all odd numbers from 3 to 15
10       for ( int x = 3; x <= 15; x += 2 )
11          product *= x;
12
13       // show results
14       System.out.printf( "Product is %d\n", product );
```

```
15      } // end main
16   } // end class Odd
```

```
Product is 2027025
```

5.13 *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 evaluates the factorials of the integers from 1 to 5. Display the results in tabular format. What difficulty might prevent you from calculating the factorial of 20?

ANS: Calculating the factorial of 20 might be difficult because the value of 20! exceeds the maximum value that can be stored in an `int`.

```
1   // Exercise 5.13 Solution: Factorial.java
2   // Program calculates factorials.
3   public class Factorial
4   {
5      public static void main( String args[] )
6      {
7         System.out.println( "n\tn!\n" );
8
9         for ( int number = 1; number <= 5; number++ )
10        {
11           int factorial = 1;
12
13           for ( int smaller = 1; smaller <= number; smaller++ )
14              factorial *= smaller;
15
16           System.out.printf( "%d\t%d\n", number, factorial );
17        } // end for loop
18     } // end main
19  } // end class Factorial
```

```
n       n!

1       1
2       2
3       6
4       24
5       120
```

5.14 Modify the compound-interest application of Fig. 5.6 to repeat its steps for interest rates of 5%, 6%, 7%, 8%, 9% and 10%. Use a `for` loop to vary the interest rate.

ANS:

```
1   // Exercise 5.14 Solution: Interest.java
2   // Calculating compound interest
3   public class Interest
4   {
5      public static void main( String args[] )
```

```
6      {
7         //initial deposit
8         double principal = 1000.0;
9
10        // print out statistics for each rate
11        for ( int interestRate = 5; interestRate <= 10; interestRate++ )
12        {
13           double rate = interestRate / 100.0;
14           System.out.printf( "\nInterest Rate: %d%%\n", interestRate );
15           System.out.println( "Year\tAmount on deposit" );
16
17           // for each rate, print a ten-year forecast
18           for ( int year = 1; year <= 10; year++ )
19           {
20              double amount = principal * ( 1.0 + rate );
21
22              // raise the amount to the power of the year
23              for ( int power = 2; power <= year; power++ )
24                 amount *= ( 1.0 + rate );
25
26              System.out.printf( "%d\t%.2f\n", year, amount );
27           } // end yearly for loop
28        } // end interest for loop
29     } // end main
30  } // end class Interest
```

```
Interest Rate: 5%
Year    Amount on deposit
1       1050.00
2       1102.50
3       1157.63
4       1215.51
5       1276.28
6       1340.10
7       1407.10
8       1477.46
9       1551.33
10      1628.89

Interest Rate: 6%
Year    Amount on deposit
1       1060.00
2       1123.60
3       1191.02
4       1262.48
5       1338.23
6       1418.52
7       1503.63
8       1593.85
9       1689.48
10      1790.85
```

```
Interest Rate: 7%
Year    Amount on deposit
1       1070.00
2       1144.90
3       1225.04
4       1310.80
5       1402.55
6       1500.73
7       1605.78
8       1718.19
9       1838.46
10      1967.15

Interest Rate: 8%
Year    Amount on deposit
1       1080.00
2       1166.40
3       1259.71
4       1360.49
5       1469.33
6       1586.87
7       1713.82
8       1850.93
9       1999.00
10      2158.92


Interest Rate: 9%
Year    Amount on deposit
1       1090.00
2       1188.10
3       1295.03
4       1411.58
5       1538.62
6       1677.10
7       1828.04
8       1992.56
9       2171.89
10      2367.36

Interest Rate: 10%
Year    Amount on deposit
1       1100.00
2       1210.00
3       1331.00
4       1464.10
5       1610.51
6       1771.56
7       1948.72
8       2143.59
9       2357.95
10      2593.74
```

5.15 Write an application that displays the following patterns separately, one below the other. Use `for` loops to generate the patterns. All asterisks (`*`) should be printed by a single statement of the form `System.out.print( '*' );` which causes the asterisks to print side by side. A statement of

the form `System.out.println();` can be used to move to the next line. A statement of the form `System.out.print( ' ' );` can be used to display a space for the last two patterns. There should be no other output statements in the program. [*Hint:* The last two patterns require that each line begin with an appropriate number of blank spaces.]

(a)

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

(b)

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

(c)

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

(d)

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

ANS:

```
1  // Exercise 5.15 Solution: Triangles.java
2  // Program prints four triangles, one below the other
3  public class Triangles
4  {
5     // draw four triangles
6     public void drawTriangles()
7     {
8        int row; // the row position
9        int column; // the column position
10       int space; // number of spaces to print
11
12       // first triangle
13       for ( row = 1; row <= 10; row++ )
14       {
15          for ( column = 1; column <= row; column++ )
16             System.out.print( "*" );
17
18          System.out.println();
19       } // end for loop
20
21       System.out.println();
22
23       // second triangle
24       for ( row = 10; row >= 1; row-- )
25       {
26          for ( column = 1; column <= row; column++ )
27             System.out.print( "*" );
28
29          System.out.println();
30       } // end for loop
31
32       System.out.println();
33
34       // third triangle
35       for ( row = 10; row >= 1; row-- )
```

```
36         {
37            for ( space = 10; space > row; space-- )
38               System.out.print( " " );
39
40            for ( column = 1; column <= row; column++ )
41               System.out.print( "*" );
42
43            System.out.println();
44         } // end for loop
45
46         System.out.println();
47
48         // fourth triangle
49         for ( row = 10; row >= 1; row-- ) {
50
51            for ( space = 1; space < row; space++ )
52               System.out.print( " " );
53
54            for ( column = 10; column >= row; column-- )
55               System.out.print( "*" );
56
57            System.out.println();
58         } // end for loop
59      } // end method drawTriangles
60   } // end class Triangles
```

```
1   // Exercise 5.15 Solution: TrianglesTest.java
2   // Test application for class Triangles
3   public class TrianglesTest
4   {
5      public static void main( String args[] )
6      {
7         Triangles application = new Triangles();
8         application.drawTriangles();
9      } // end main
10  } // end class TrianglesTest
```

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

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

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

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

5.16 One interesting application of computers is to display graphs and bar charts. Write an application that reads five numbers between 1 and 30. For each number that is read, your program should display the same number of adjacent asterisks. For example, if your program reads the number 7, it should display *******.

ANS:

```
1  // Exercise 5.16 Solution: Graphs.java
2  // Program prints 5 histograms with lengths determined by user.
3  import java.util.Scanner;
4
```

```
5  public class Graphs
6  {
7     // draws 5 histograms
8     public void drawHistograms()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int number1 = 0; // first number
13       int number2 = 0; // second number
14       int number3 = 0; // third number
15       int number4 = 0; // fourth number
16       int number5 = 0; // fifth number
17
18       int inputNumber; // number entered by user
19       int value = 0; // number of stars to print
20       int counter = 1; // counter for current number
21
22       while ( counter <= 5 )
23       {
24          System.out.print( "Enter number: " );
25          inputNumber = input.nextInt();
26
27          // define appropriate num if input is between 1-30
28          if ( inputNumber >= 1 && inputNumber <= 30 )
29          {
30             switch ( counter )
31             {
32                case 1:
33                   number1 = inputNumber;
34                   break; // done processing case
35
36                case 2:
37                   number2 = inputNumber;
38                   break; // done processing case
39
40                case 3:
41                   number3 = inputNumber;
42                   break; // done processing case
43
44                case 4:
45                   number4 = inputNumber;
46                   break; // done processing case
47
48                case 5:
49                   number5 = inputNumber;
50                   break; // done processing case
51             } // end switch
52
53             counter++;
54          } // end if
55          else
56             System.out.println(
57                "Invalid Input\nNumber should be between 1 and 30" );
58       } // end while
```

```
59
60          // print histograms
61          for ( counter = 1; counter <= 5; counter++ )
62          {
63             switch ( counter )
64             {
65                case 1:
66                   value = number1;
67                   break; // done processing case
68
69                case 2:
70                   value = number2;
71                   break; // done processing case
72
73                case 3:
74                   value = number3;
75                   break; // done processing case
76
77                case 4:
78                   value = number4;
79                   break; // done processing case
80
81                case 5:
82                   value = number5;
83                   break; // done processing case
84             }
85
86             for ( int j = 1; j <= value; j++ )
87                System.out.print( "*" );
88
89             System.out.println();
90          } // end for loop
91       } // end method drawHistograms
92    } // end class Graphs
```

```
1  // Exercise 5.16 Solution: GraphsTest.java
2  // Test application for class Graphs
3  public class GraphsTest
4  {
5     public static void main( String args[] )
6     {
7        Graphs application = new Graphs();
8        application.drawHistograms();
9     } // end main
10 } // end class GraphsTest
```

```
Enter number: 20
Enter number: 6
Enter number: 15
Enter number: 28
Enter number: 5
********************
******
***************
****************************
*****
```

5.17 A mail-order house sells five products whose retail prices are as follows: product 1, $2.98; product 2, $4.50; product 3, $9.98; product 4, $4.49 and product 5, $6.87. Write an application that reads a series of pairs of numbers as follows:

a) product number
b) quantity sold

Your program should use a `switch` statement to determine the retail price for each product. It should calculate and display the total retail value of all products sold. Use a sentinel-controlled loop to determine when the program should stop looping and display the final results.

ANS:

```
1  // Exercise 5.17 Solution: Sales.java
2  // Program calculates sales, based on an input of product
3  // number and quantity sold
4  import java.util.Scanner;
5
6  public class Sales
7  {
8     // calculates sales for 5 products
9     public void calculateSales()
10    {
11       Scanner input = new Scanner( System.in );
12
13       double product1 = 0; // amount sold of first product
14       double product2 = 0; // amount sold of second product
15       double product3 = 0; // amount sold of third product
16       double product4 = 0; // amount sold of fourth product
17       double product5 = 0; // amount sold of fifth product
18
19       int productId = 1; // current product id number
20
21       // ask user for product number until flag value entered
22       while ( productId != 0 )
23       {
24          // determine the product chosen
25          System.out.print(
26             "Enter product number (1-5) (0 to stop): " );
27          productId = input.nextInt();
28
29          if ( productId >= 1 && productId <= 5 )
30          {
```

```
            // determine the number sold of the item
            System.out.print( "Enter quantity sold: " );
            int quantity = input.nextInt();

            // increment the total for the item by the
            // price times the quantity sold
            switch ( productId )
            {
               case 1:
                  product1 += quantity * 2.98;
                  break;

               case 2:
                  product2 += quantity * 4.50;
                  break;

               case 3:
                  product3 += quantity * 9.98;
                  break;

               case 4:
                  product4 += quantity * 4.49;
                  break;

               case 5:
                  product5 += quantity * 6.87;
                  break;
            } // end switch
         } // end if
         else if ( productId != 0 )
            System.out.println(
               "Product number must be between 1 and 5 or 0 to stop" );
      } // end while loop

      // print summary
      System.out.println();
      System.out.printf( "Product 1: $%.2f\n", product1 );
      System.out.printf( "Product 2: $%.2f\n", product2 );
      System.out.printf( "Product 3: $%.2f\n", product3 );
      System.out.printf( "Product 4: $%.2f\n", product4 );
      System.out.printf( "Product 5: $%.2f\n", product5 );
   } // end method calculateSales
} // end class Sales
```

```
// Exercise 5.17 Solution: SalesTest.java
// Test application for class Sales
public class SalesTest
{
   public static void main( String args[] )
   {
      Sales application = new Sales();
      application.calculateSales();
```

```
      } // end main
   } // end class SalesTest
```

```
Enter product number (1-5) (0 to stop): 1
Enter quantity sold: 5
Enter product number (1-5) (0 to stop): 5
Enter quantity sold: 10
Enter product number (1-5) (0 to stop): 0

Product 1: $14.90
Product 2: $0.00
Product 3: $0.00
Product 4: $0.00
Product 5: $68.70
```

5.18 Modify the application in Fig. 5.6 to use only integers to calculate the compound interest. [*Hint*: Treat all monetary amounts as integral numbers of pennies. Then break the result into its dollars and cents portions by using the division and remainder operations, respectively. Insert a period between the dollars and the cents portions.]

ANS:

```
// Exercise 5.18: Interest.java
// Calculating compound interest.
public class Interest
{
   public static void main( String args[] )
   {
      int amount; // amount on deposit at end of each year
      int principal = 100000; // initial amount (number of pennies)
      double rate = 0.05; // interest rate

      // print the headers
      System.out.printf( "%s%20s\n", "Year", "Amount on deposit" );

      // calculate amount on deposit for each of ten years
      for ( int year = 1; year <= 10; year++ )
      {
         // calculate new amount for specified year
         amount =
            ( int ) ( principal * Math.pow( 1.0 + rate, year ) );

         int dollars = amount / 100; // calculate dollars portion
         int cents = amount % 100; // calculate cents portion

         // print the year, the dollars portion and a decimal point
         System.out.printf( "%4d%,17d.", year, dollars );

         // print the cents portion (with a leading zero if cents < 10 )
         if ( cents < 10 )
            System.out.printf( "0%d\n", cents );
         else
            System.out.printf( "%d\n", cents );
```

```
      } // end for loop
   } // end main
} // end class Interest
```

```
Year   Amount on deposit
   1            1,050.00
   2            1,102.50
   3            1,157.62
   4            1,215.50
   5            1,276.28
   6            1,340.09
   7            1,407.10
   8            1,477.45
   9            1,551.32
  10            1,628.89
```

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.

```
// Exercise 5.19 Solution: Mystery.java
// Printing conditional expressions outputs 'true' or 'false'.
public class Mystery
{
   public static void main( String args[] )
   {
      int i = 1;
      int j = 2;
      int k = 3;
      int m = 2;

      // part a
      System.out.println( i == 1 );

      // part b
      System.out.println( j == 3 );

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

```
20
21          // part d
22          System.out.println( ( m <= 99 ) & ( k < m ) );
23
24          // part e
25          System.out.println( ( j >= i ) || ( k == m ) );
26
27          // part f
28          System.out.println( ( k + m < j ) | ( 3 - j >= k ) );
29
30          // part g
31          System.out.println( !( k > m ) );
32       } // end main
33    } // end class Mystery
```

```
true
false
true
false
true
false
false
```

5.20 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 one term of this series, by two terms, by three terms, and so on. How many terms of this series do you have to use before you first get 3.14? 3.141? 3.1415? 3.14159?

ANS: [*Note:* The program starts to converge around 3.14 at 119, but does not really approach 3.141 until 1688, well beyond the program's range.]

```
1  // Exercise 5.20 Solution: Pi.java
2  // Program calculates Pi from the infinite series.
3
4  public class Pi
5  {
6     public static void main( String args[] )
7     {
8        double piValue = 0; // current approximation of pi
9        double number = 4.0; // numerator of fraction
10       double denominator = 1.0; // denominator
11       int accuracy = 400; // maximum number of terms in series
12
13       System.out.printf( "Accuracy: %d\n\n", accuracy );
14       System.out.println( "Term\t\tPi" );
15
16       for ( int term = 1; term <= accuracy; term++ )
17       {
18          if ( term % 2 != 0 )
19             piValue += number / denominator;
```

```
            else
               piValue -= number / denominator;

            System.out.printf( "%d\t\t%.16f\n", term, piValue);
            denominator += 2.0;
         } // end for loop
      } // end main
   } // end class Pi
```

```
Accuracy: 400

Term            Pi
1               4.0000000000000000
2               2.6666666666666670
3               3.4666666666666670
4               2.8952380952380956
5               3.3396825396825403

.
.
.

395             3.1441242951029738
396             3.1390674050903313
397             3.1441115412820086
398             3.1390800947411280
399             3.1440989153182923
400             3.1390926574960143
```

5.21 (*Pythagorean Triples*) A right triangle can have sides whose lengths are all integers. The set of three integer values for the lengths of the sides of a right triangle is called a Pythagorean triple. The lengths of the three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Write an application to find all Pythagorean triples for `side1`, `side2` and the `hypotenuse`, all no larger than 500. Use a triple-nested `for` loop that tries all possibilities. This method is an example of "brute-force" computing. You will learn in more advanced computer science courses that for many interesting problems there is no known algorithmic approach other than using sheer brute force.

ANS:

```
// Exercise 5.21 Solution: Triples.java
// Program calculates Pythagorean triples
public class Triples
{
   public static void main( String args[] )
   {
      // declare the three sides of a triangle
      int side1;
      int side2;
      int hypotenuse;

      for ( side1 = 1; side1 <= 500; side1++ )
         for ( side2 = 1; side2 <= 500; side2++ )
```

```
            for ( hypotenuse = 1; hypotenuse <= 500; hypotenuse++ )
               // use Pythagorean Theorem to print right triangles
               if ( ( side1 * side1 ) + ( side2 * side2 ) ==
                  ( hypotenuse * hypotenuse ) )
                  System.out.printf( "s1: %d, s2: %d, h: %d\n",
                     side1, side2, hypotenuse );
      } // end main
   } // end class Triples
```

```
s1: 3, s2: 4, h: 5
s1: 4, s2: 3, h: 5
s1: 5, s2: 12, h: 13
s1: 6, s2: 8, h: 10
s1: 7, s2: 24, h: 25

.
.
.

s1: 480, s2: 31, h: 481
s1: 480, s2: 88, h: 488
s1: 480, s2: 108, h: 492
s1: 480, s2: 140, h: 500
s1: 483, s2: 44, h: 485
```

5.22 Modify Exercise 5.15 to combine your code from the four separate triangles of asterisks such that all four patterns print side by side. Make clever use of nested `for` loops.

ANS:

```
// Exercise 5.22 Solution: Triangles.java
// Program prints four triangles side by side
public class Triangles
{
   // draws four triangles
   public void drawTriangles()
   {
      int row; // current row
      int column; // current column
      int space; // number of spaces printed

      // print one row at a time, tabbing between triangles
      for ( row = 1; row <= 10; row++ )
      {
         // triangle one
         for ( column = 1; column <= row; column++ )
            System.out.print( "*" );

         for ( space = 1; space <= 10 - row; space++ )
            System.out.print( " " );

         System.out.print( "\t" );

```

```
            // triangle two
            for ( column = 10; column >= row; column-- )
               System.out.print( "*" );

            for ( space = 1; space < row; space++ )
               System.out.print( " " );

            System.out.print( "\t" );

            // triangle three
            for ( space = 1; space < row; space++ )
               System.out.print( " " );

            for ( column = 10; column >= row; column-- )
               System.out.print( "*" );

            System.out.print( "\t" );

            // triangle four
            for ( space = 10; space > row; space-- )
               System.out.print( " " );

            for ( column = 1; column <= row; column++ )
               System.out.print( "*" );

            System.out.println();
         } // end for loop
      } // end method drawTriangles
   } // end class Triangles
```

```
// Exercise 5.22 Solution: TrianglesTest.java
// Test application for class Triangles
public class TrianglesTest
{
   public static void main( String args[] )
   {
      Triangles application = new Triangles();
      application.drawTriangles();
   } // end main
} // end class TrianglesTest
```

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

5.23 (*De Morgan's Laws*) In this chapter, we have discussed the logical operators &&, &, ||, |, ^ and !. De Morgan's Laws can sometimes make it more convenient for us to express a logical expression. These laws state that the expression !(*condition1* && *condition2*) is logically equivalent to the expression (!*condition1* || !*condition2*). Also, the expression !(*condition1* || *condition2*) is logically equivalent to the expression (!*condition1* && !*condition2*). Use De Morgan's Laws to write equivalent expressions for each of the following, then write an application to show that both the original expression and the new expression in each case produce the same value:

a) `!( x < 5 ) && !( y >= 7 )`
b) `!( a == b ) || !( g != 5 )`
c) `!( ( x <= 8 ) && ( y > 4 ) )`
d) `!( ( i > 4 ) || ( j <= 6 ) )`

ANS:

```
1  // Exercise 5.23 Solution: DeMorgan.java
2  // Program tests DeMorgan's laws.
3  public class DeMorgan
4  {
5     public static void main( String args[] )
6     {
7        int x = 6;
8        int y = 0;
9
10       // part a
11       if ( !( x < 5 ) && !( y >= 7 ) )
12          System.out.println( "!( x < 5 ) && !( y >= 7 )" );
13
14       if ( !( ( x < 5 ) || ( y >= 7 ) ) )
15          System.out.println( "!( ( x < 5 ) || ( y >= 7 )" );
16
17       int a = 8;
18       int b = 22;
19       int g = 88;
20
21       // part b
22       if ( !( a == b ) || !( g != 5 ) )
23          System.out.println( "!( a == b ) || !( g != 5 )" );
24
25       if ( !( ( a == b ) && ( g != 5 ) ) )
26          System.out.println( "!( ( a == b ) && ( g != 5 ) )" );
27
28       x = 8;
29       y = 2;
30
31       // part c
32       if ( !( ( x <= 8 ) && ( y > 4 ) ) )
33          System.out.println( "!( ( x <= 8 ) && ( y > 4 ) )" );
34
35       if ( !( x <= 8 ) || !( y > 4 ) )
36          System.out.println( "!( x <= 8 ) || !( y > 4 )" );
37
38       int i = 0;
39       int j = 7;
40
```

```
41         // part d
42         if ( !( ( i > 4 ) || ( j <= 6 ) ) )
43            System.out.println( "!( ( i > 4 ) || ( j <= 6 ) )" );
44
45         if ( !( i > 4 ) && !( j <= 6 ) )
46            System.out.println( "!( i > 4 ) && !( j <= 6 )" );
47      } // end main
48   } // end class DeMorgan
```

```
!( x < 5 ) && !( y >= 7 )
!( ( x < 5 ) || ( y >= 7 ) )
!( a == b ) || !( g != 5 )
!( ( a == b ) && ( g != 5 ) )
!( ( x <= 8 ) && ( y > 4 ) )
!( x <= 8 ) || !( y > 4 )
!( ( i > 4 ) || ( j <= 6 ) )
!( i > 4 ) && !( j <= 6 )
```

5.24 Write an application that prints the following diamond shape. You may use output statements that print a single asterisk (*), a single space or a single newline character. Maximize your use of repetition (with nested `for` statements), and minimize the number of output statements.

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

ANS:

```
1  // Exercise 5.24 Solution: Diamond.java
2  // Program prints a diamond with minimal output statements
3  public class Diamond
4  {
5     // draws a diamond of asterisks
6     public void drawDiamond()
7     {
8        int row; // the current row
9        int stars; // the number of stars
10       int spaces; // the number of spaces
11
12       // top half (1st five lines)
13       for ( row = 1; row <= 5; row++ )
14       {
15          for ( spaces = 5; spaces > row; spaces-- )
16             System.out.print( " " );
```

```
17
18          for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
19             System.out.print( "*" );
20
21          System.out.println();
22       } // end for statement
23
24       // bottom half (last four rows)
25       for ( row = 4; row >= 1; row-- )
26       {
27          for ( spaces = 5; spaces > row; spaces-- )
28             System.out.print( " " );
29
30          for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
31             System.out.print( "*" );
32
33          System.out.println();
34       } // end for statement
35    } // end method drawDiamond
36 } // end class Diamond
```

```
1  // Exercise 5.24 Solution: DiamondTest.java
2  // Test application for class Diamond
3  public class DiamondTest
4  {
5     public static void main( String args[] )
6     {
7        Diamond application = new Diamond();
8        application.drawDiamond();
9     } // end main
10 } // end class DiamondTest
```

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

5.25 Modify the application you wrote in Exercise 5.24 to read an odd number in the range 1 to 19 to specify the number of rows in the diamond. Your program should then display a diamond of the appropriate size.

ANS:

```
1  // Exercise 5.25 Solution: Diamond.java
2  // Program prints a diamond of a user-specified size
3  import java.util.Scanner;
```

```
4
5  public class Diamond
6  {
7     // draws a diamond of asterisks
8     public void drawDiamond()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int row; // current row
13       int stars; // number of stars
14       int spaces; // number of spaces
15       int numRows; // number of rows
16
17       // ask user for an entry until within range
18       do
19       {
20          System.out.print(
21             "Enter number of rows (odd number 1 to 19): " );
22          numRows = input.nextInt();
23       } while ( ( numRows > 19 ) || ( numRows < 0 ) ||
24          ( numRows % 2 == 0 ) );
25
26       // top half
27       for ( row = 1; row < ( numRows / 2 ) + 1; row++ )
28       {
29          for ( spaces = numRows / 2; spaces >= row; spaces-- )
30             System.out.print( " " );
31
32          for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
33             System.out.print( "*" );
34
35          System.out.println();
36       } // end for loop
37
38       // bottom half, note that the first clause of the for
39       // loop isn't needed; row is already defined
40       for ( ; row >= 1; row-- )
41       {
42          for ( spaces = numRows / 2; spaces >= row; spaces-- )
43             System.out.print( " " );
44
45          for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
46             System.out.print( "*" );
47
48          System.out.println();
49       } // end for loop
50    } // end method drawDiamond
51 } // end class Diamond
```

```
1  // Exercise 5.25 Solution: DiamondTest.java
2  // Test application for class Diamond
3  public class DiamondTest
4  {
```

```
5      public static void main( String args[] )
6      {
7         Diamond application = new Diamond();
8         application.drawDiamond();
9      } // end main
10  } // end class DiamondTest
```

```
Enter number of rows (odd number 1 to 19): 13
      *
     ***
    *****
   *******
  *********
 ***********
*************
 ***********
  *********
   *******
    *****
     ***
      *
```

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 would remove any `break` statement from a loop in a program and replace that statement 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.

```
1  // Exercise. 5.26: WithoutBreak.java
2  // Terminating a loop without break.
3  public class WithoutBreak
4  {
5     public static void main( String args[] )
6     {
7        int count; // control variable
8
9        for ( count = 1; count <= 10; count++ ) // loop 10 times
10       {
11          System.out.printf( "%d ", count );
12
13          if ( count == 4 ) // if count is 4,
14             count = 10; // reset count to terminate loop
15       } // end for
16
```

```
17         System.out.println( "\nBroke out of loop by resetting count" );
18      } // end main
19   } // end class WithoutBreak
```

```
1 2 3 4
Broke out of loop by resetting count
```

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:

```
1  // Exercise 5.27 Solution: Mystery.java
2  // Prints 5 groups of 3 lines, each containing 4 asterisks.
3  public class Mystery
4  {
5     public static void main( String args[] )
6     {
7        int i;
8        int j;
9        int k;
10
11       for ( i = 1; i <= 5; i++ )
12       {
13          for ( j = 1; j <= 3; j++ )
14          {
15             for ( k = 1; k <= 4; k++ )
16                System.out.print( '*' );
17
18             System.out.println();
19          } // end inner for
20
21          System.out.println();
22       } // end outer for
23    } // end main
24 } // end class Mystery
```

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

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

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

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

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

5.28 Describe in general how you would 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.

```
1  // Exercise 5.28 Solution: ContinueTest.java
2  // Alternative to the continue statement in a for statement
3  public class ContinueTest
4  {
5     public static void main( String args[] )
6     {
7        for ( int count = 1; count <= 10; count++ ) // loop 10 times
8           if ( count != 5 ) // if count is not 5
9              System.out.printf( "%d ", count ); // print
10
11       System.out.println( "\nRemoved continue to skip printing 5" );
12    } // end main
13 } // end class ContinueTest
```

```
1 2 3 4 6 7 8 9 10
Removed continue to skip printing 5
```

5.29 *("The Twelve Days of Christmas" Song)* Write an application that uses repetition and `switch` statements to print the song "The Twelve Days of Christmas." One `switch` statement should be used to print the day ("first," "second," and so on). A separate `switch` statement should be used to print the remainder of each verse. Visit the website `en.wikipedia.org/wiki/Twelvetide` for the lyrics of the song.

ANS:

```
1  // Exercise 5.29 Solution: Twelve.java
2  // Program prints the 12 days of Christmas song.
3  public class Twelve
4  {
5     // print the 12 days of Christmas song
6     public void printSong()
7     {
8        for ( int day = 1; day <= 12; day++ )
9        {
10          System.out.print( "On the " );
11
12          // add correct day to String
13          switch ( day )
14          {
15             case 1:
16                System.out.print( "first" );
17                break;
18
19             case 2:
20                System.out.print( "second" );
21                break;
22
23             case 3:
24                System.out.print( "third" );
25                break;
26
27             case 4:
28                System.out.print( "fourth" );
29                break;
30
31             case 5:
32                System.out.print( "fifth" );
33                break;
34
35             case 6:
36                System.out.print( "sixth" );
37                break;
38
39             case 7:
40                System.out.print( "seventh" );
41                break;
42
43             case 8:
44                System.out.print( "eighth" );
45                break;
46
```

```
47             case 9:
48                System.out.print( "ninth" );
49                break;
50
51             case 10:
52                System.out.print( "tenth" );
53                break;
54
55             case 11:
56                System.out.print( "eleventh" );
57                break;
58
59             case 12:
60                System.out.print( "twelfth" );
61                break;
62          } // end switch
63
64          System.out.println(
65             " day of Christmas, my true love gave to me:" );
66
67          // add remainder of verse to String
68          switch ( day )
69          {
70             case 12:
71                System.out.println( "Twelve lords-a-leaping," );
72
73             case 11:
74                System.out.println( "Eleven pipers piping," );
75
76             case 10:
77                System.out.println( "Ten drummers drumming," );
78
79             case 9:
80                System.out.println( "Nine ladies dancing," );
81
82             case 8:
83                System.out.println( "Eight maids-a-milking," );
84
85             case 7:
86                System.out.println( "Seven swans-a-swimming," );
87
88             case 6:
89                System.out.println( "Six geese-a-laying," );
90
91             case 5:
92                System.out.println( "Five golden rings." );
93
94             case 4:
95                System.out.println( "Four calling birds," );
96
97             case 3:
98                System.out.println( "Three French hens," );
99
100            case 2:
```

```
                System.out.println( "Two turtle doves, and" );

            case 1:
               System.out.println( "a Partridge in a pear tree." );
            } // end switch

            System.out.println();
         } // end for
      } // end method printSong
   } // end class Twelve
```

```
// Exercise 5.29 Solution: TwelveTest.java
// Test application for class Twelve
public class TwelveTest
{
   public static void main( String args[] )
   {
      Twelve application = new Twelve();
      application.printSong();
   } // end main
} // end class TwelveTest
```

```
On the first day of Christmas, my true love gave to me:
a Partridge in a pear tree.

On the second day of Christmas, my true love gave to me:
Two turtle doves, and
a Partridge in a pear tree.

On the third day of Christmas, my true love gave to me:
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.

On the fourth day of Christmas, my true love gave to me:
Four calling birds,
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.

On the fifth day of Christmas, my true love gave to me:
Five golden rings.
Four calling birds,
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.

On the sixth day of Christmas, my true love gave to me:
Six geese-a-laying,
Five golden rings.
Four calling birds,
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.
```

```
On the seventh day of Christmas, my true love gave to me:
Seven swans-a-swimming,
Six geese-a-laying,
Five golden rings.
Four calling birds,
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.

On the eighth day of Christmas, my true love gave to me:
Eight maids-a-milking,
Seven swans-a-swimming,
Six geese-a-laying,
Five golden rings.
Four calling birds,
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.

On the ninth day of Christmas, my true love gave to me:
Nine ladies dancing,
Eight maids-a-milking,
Seven swans-a-swimming,
Six geese-a-laying,
Five golden rings.
Four calling birds,
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.

On the tenth day of Christmas, my true love gave to me:
Ten drummers drumming,
Nine ladies dancing,
Eight maids-a-milking,
Seven swans-a-swimming,
Six geese-a-laying,
Five golden rings.
Four calling birds,
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.

On the eleventh day of Christmas, my true love gave to me:
Eleven pipers piping,
Ten drummers drumming,
Nine ladies dancing,
Eight maids-a-milking,
Seven swans-a-swimming,
Six geese-a-laying,
Five golden rings.
Four calling birds,
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.
```

```
On the twelfth day of Christmas, my true love gave to me:
Twelve lords-a-leaping,
Eleven pipers piping,
Ten drummers drumming,
Nine ladies dancing,
Eight maids-a-milking,
Seven swans-a-swimming,
Six geese-a-laying,
Five golden rings.
Four calling birds,
Three French hens,
Two turtle doves, and
a Partridge in a pear tree.
```