4

Control Statements: Part 1

Let's all move one place on.
—Lewis Carroll

The wheel is come full circle.
—William Shakespeare

How many apples fell on Newton's head before he took the hint!
—Robert Frost

All the evolution we know of proceeds from the vague to the definite.
—Charles Sanders Peirce

OBJECTIVES

In this chapter you will learn:

- To use basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement using pseudocode.
- To use the `if` and `if...else` selection statements to choose among alternative actions.
- To use the `while` repetition statement to execute statements in a program repeatedly.
- To use counter-controlled repetition and sentinel-controlled repetition.
- To use the compound assignment, increment and decrement operators.
- The primitive data types.

Self-Review Exercises

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

a) All programs can be written in terms of three types of control structures: _____, _____ and _____.

ANS: sequence, selection, repetition.

b) The _____ statement is used to execute one action when a condition is true and another when that condition is false.

ANS: `if...else`.

c) Repeating a set of instructions a specific number of times is called _____ repetition.

ANS: counter-controlled (or definite).

d) When it is not known in advance how many times a set of statements will be repeated, a(n) _____ value can be used to terminate the repetition.

ANS: sentinel, signal, flag or dummy.

e) The _____ structure is built into Java—by default, statements execute in the order they appear.

ANS: sequence.

f) Instance variables of types `char`, `byte`, `short`, `int`, `long`, `float` and `double` are all given the value _____ by default.

ANS: `0` (zero).

g) Java is a _____ language—it requires all variables to have a type.

ANS: strongly typed.

h) If the increment operator is _____ to a variable, the variable is incremented by 1 first, then its new value is used in the expression.

ANS: prefixed.

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

a) An algorithm is a procedure for solving a problem in terms of the actions to execute and the order in which they execute.

ANS: True.

b) A set of statements contained within a pair of parentheses is called a block.

ANS: False. A set of statements contained within a pair of braces (`{` and `}`) is called a block.

c) A selection statement specifies that an action is to be repeated while some condition remains true.

ANS: False. A repetition statement specifies that an action is to be repeated while some condition remains true.

d) A nested control statement appears in the body of another control statement.

ANS: True.

e) Java provides the arithmetic compound assignment operators `+=`, `-=`, `*=`, `/=` and `%=` for abbreviating assignment expressions.

ANS: True.

f) The primitive types (`boolean`, `char`, `byte`, `short`, `int`, `long`, `float` and `double`) are portable across only Windows platforms.

ANS: False. The primitive types (`boolean`, `char`, `byte`, `short`, `int`, `long`, `float` and `double`) are portable across all computer platforms that support Java.

g) Specifying the order in which statements (actions) execute in a program is called program control.

ANS: True.

h) The unary cast operator `(double)` creates a temporary integer copy of its operand.

ANS: False. The unary cast operator `(double)` creates a temporary floating-point copy of its operand.

i) Instance variables of type `boolean` are given the value `true` by default.

ANS: False. Instance variables of type `boolean` are given the value `false` by default.

j) Pseudocode helps a programmer think out a program before attempting to write it in a programming language.

ANS: True.

4.3 Write four different Java statements that each add 1 to integer variable `x`.

ANS:
```
x = x + 1;
x += 1;
++x;
x++;
```

4.4 Write Java statements to accomplish each of the following tasks:

a) Assign the sum of `x` and `y` to `z`, and increment `x` by 1 after the calculation. Use only one statement.

ANS:
```
z = x++ + y;
```

b) Test whether variable `count` is greater than 10. If it is, print `"Count is greater than 10"`.

ANS:
```
if ( count > 10 )
   System.out.println( "Count is greater than 10" );
```

c) Decrement the variable `x` by 1, then subtract it from the variable `total`. Use only one statement.

ANS:
```
total -= --x;
```

d) Calculate the remainder after `q` is divided by `divisor`, and assign the result to `q`. Write this statement in two different ways.

ANS:
```
q %= divisor;
q = q % divisor;
```

4.5 Write a Java statement to accomplish each of the following tasks:

a) Declare variables `sum` and `x` to be of type `int`.

ANS:
```
int sum, x;
```

b) Assign 1 to variable `x`.

ANS:
```
x = 1;
```

c) Assign 0 to variable `sum`.

ANS:
```
sum = 0;
```

d) Add variable `x` to variable `sum`, and assign the result to variable `sum`.

ANS:
```
sum += x; or sum = sum + x;
```

e) Print `"The sum is: "`, followed by the value of variable `sum`.

ANS:
```
System.out.printf( "The sum is: %d\n", sum );
```

4.6 Combine the statements that you wrote in Exercise 4.5 into a Java application that calculates and prints the sum of the integers from 1 to 10. Use a `while` statement to loop through the calculation and increment statements. The loop should terminate when the value of `x` becomes 11.

ANS: The program is as follows:

```
1  // Exercise 4.6 Solution: Calculate.java
2  // Calculate the sum of the integers from 1 to 10
3  public class Calculate
4  {
5     public static void main( String args[] )
6     {
7        int sum;
8        int x;
9
```

```
10         x = 1;   // initialize x to 1 for counting
11         sum = 0; // initialize sum to 0 for totaling
12
13         while ( x <= 10 ) // while x is less than or equal to 10
14         {
15            sum += x; // add x to sum
16            ++x; // increment x
17         } // end while
18
19         System.out.printf( "The sum is: %d\n", sum );
20      } // end main
21   } // end class Calculate
```

```
The sum is: 55
```

4.7 Determine the value of the variables in the following statement after the calculation is performed. Assume that when the statement begins executing, all variables are type `int` and have the value 5.

```
product *= x++;
```

ANS: `product = 25, x = 6`

4.8 Identify and correct the errors in each of the following sets of code:

a)
```
while ( c <= 5 )
{
   product *= c;
   ++c;
```

b)
```
if ( gender == 1 )
   System.out.println( "Woman" );
else;
   System.out.println( "Man" );
```

ANS: Error: The closing right brace of the `while` statement's body is missing.
Correction: Add a closing right brace after the statement `++c;`.

4.9 What is wrong with the following `while` statement?

```
while ( z >= 0 )
   sum += z;
```

ANS: The value of the variable `z` is never changed in the `while` statement. Therefore, if the loop-continuation condition `( z >= 0 )` is true, an infinite loop is created. To prevent an infinite loop from occurring, `z` must be decremented so that it eventually becomes less than 0.

Exercises

4.10 Compare and contrast the `if` single-selection statement and the `while` repetition statement. How are these two statements similar? How are they different?

ANS: The `if` single-selection statement and the `while` repetition statement both perform an action (or set of actions) based on whether a condition is true or false. However, if the condition is true, the `if` single-selection statement performs the action(s) once, whereas the `while` repetition statement repeatedly performs the action(s) until the condition becomes false.

4.11 Explain what happens when a Java program attempts to divide one integer by another. What happens to the fractional part of the calculation? How can a programmer avoid that outcome?

ANS: Dividing two integers results in integer division—any fractional part of the calculation is lost (i.e., truncated). For example, 7 ÷ 4, which yields 1.75 in conventional arithmetic, truncates to 1 in integer arithmetic, rather than rounding to 2. To obtain a floating-point result from dividing integer values, a programmer must temporarily treat these values as floating-point numbers in the calculation by using the unary cast operator `(double)`. As long as the `(double)` cast operator is applied to any variable in the calculation, the calculation will yield a `double` result which can be assigned to a `double` variable.

4.12 Describe the two ways in which control statements can be combined.

ANS: Control statements can be "attached" (that is, stacked) to one another by connecting the exit point of one to the entry point of the next. Control statements also may be nested by placing one control statement inside another.

4.13 What type of repetition would be appropriate for calculating the sum of the first 100 positive integers? What type of repetition would be appropriate for calculating the sum of an arbitrary number of positive integers? Briefly describe how each of these tasks could be performed.

ANS: Counter-controlled repetition would be appropriate for calculating the sum of the first 100 positive integers because the number of repetitions is known in advance. The program that performs this task could use a `while` repetition statement with a counter variable that takes on the values 1 to 100. The program could then add the current counter value to the total variable in each repetition of the loop. Sentinel-controlled repetition would be appropriate for calculating the sum of an arbitrary number of positive integers. The program that performs this task could use a sentinel value of –1 to mark the end of data entry. The program could use a `while` repetition statement that totals positive integers from the user until the user enters the sentinel value.

4.14 What is the difference between preincrementing and postincrementing a variable?

ANS: Preincrementing a variable causes it to be incremented by 1, and then the new value of the variable is used in the expression in which it appears. Postincrementing a variable causes the current value of the variable to be used in the expression in which it appears, and then the variable's value is incremented by 1. Preincrementing and postincrementing a variable have the same effect when the incrementing operation appears in a statement by itself.

4.15 Identify and correct the errors in each of the following pieces of code. [*Note:* There may be more than one error in each piece of code.]

a)
```
if ( age >= 65 );
   System.out.println( "Age greater than or equal to 65" );
else
   System.out.println( "Age is less than 65 )";
```
ANS: The semicolon at the end of the `if` condition should be removed. The closing double quote of the second `System.out.println` should be inside the closing parenthesis.

b)
```
int x = 1, total;
while ( x <= 10 )
{
   total += x;
   ++x;
}
```
ANS: The variable `total` should be initialized to zero.

c)
```
while ( x <= 100 )
   total += x;
   ++x;
```
ANS: The two statements should be enclosed in curly braces to properly group them into the body of the `while`; otherwise the loop will be an infinite loop.

d)
```
while ( y > 0 )
{
   System.out.println( y );
   ++y;
```
ANS: The `++` operator should be changed to `--`; otherwise the loop will be an infinite loop. The closing curly brace for the `while` loop is missing.

4.16 What does the following program print?

```
1  public class Mystery
2  {
3     public static void main( String args[] )
4     {
5        int y;
6        int x = 1;
7        int total = 0;
8
9        while ( x <= 10 )
10       {
11          y = x * x;
12          System.out.println( y );
13          total += y;
14          ++x;
15       } // end while
16
17       System.out.printf( "Total is %d\n", total );
18    } // end main
19
20 } // end class Mystery
```

ANS:

```
1
4
9
16
25
36
49
64
81
100
Total is 385
```

For Exercise 4.17 through Exercise 4.20, perform each of the following steps:

a) Read the problem statement.
b) Formulate the algorithm using pseudocode and top-down, stepwise refinement.
c) Write a Java program.
d) Test, debug and execute the Java program.
e) Process three complete sets of data.

4.17 Drivers are concerned with the mileage their automobiles get. One driver has kept track of several tankfuls of gasoline by recording the miles driven and gallons used for each tankful. Develop a Java application that will input the miles driven and gallons used (both as integers) for each tankful. The program should calculate and display the miles per gallon obtained for each tankful and print the combined miles per gallon obtained for all tankfuls up to this point. All averaging calculations should produce floating-point results. Use class `Scanner` and sentinel-controlled repetition to obtain the data from the user.

ANS:

```
1  // Exercise 4.17 Solution: Gas.java
2  // Program calculates average mpg
3  import java.util.Scanner;
4
5  public class Gas
6  {
7     // perform miles-per-gallon calculations
8     public void calculateMPG()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int miles; // miles for one tankful
13       int gallons; // gallons for one tankful
14       int totalMiles = 0; // total mailes for trip
15       int totalGallons = 0; // total gallons for trip
16
17       double milesPerGallon; // miles per gallon for tankful
18       double totalMilesPerGallon; // miles per gallon for trip
19
20       // prompt user for miles and obtain the input from user
21       System.out.print( "Enter miles (-1 to quit): " );
```

```
22          miles = input.nextInt();
23
24          // exit if the input is -1; otherwise, proceed with the program
25          while ( miles != -1 )
26          {
27             // prompt user for gallons and obtain the input from user
28             System.out.print( "Enter gallons:" );
29             gallons = input.nextInt();
30
31             // add gallons and miles for this tank to total
32             totalMiles += miles;
33             totalGallons += gallons;
34
35             // calculate miles per gallon for the current tank
36             if ( gallons != 0 )
37             {
38                milesPerGallon = (double) miles / gallons;
39                System.out.printf( "MPG this tankful: %.2f\n",
40                   milesPerGallon );
41             } // end if statement
42
43             // calculate miles per gallon for the total trip
44             if ( totalGallons != 0 )
45             {
46                totalMilesPerGallon = (double) totalMiles / totalGallons;
47                System.out.printf( "Total MPG: %.2f\n",
48                   totalMilesPerGallon );
49             } // end if statement
50
51             // prompt user for new value for miles
52             System.out.print( "Enter miles (-1 to quit): " );
53             miles = input.nextInt();
54          } // end while loop
55       } // end method calculateMPG
56    } // end class Gas
```

```
1  // Exercise 4.17 Solution: GasTest.java
2  // Test application for class Gas
3  public class GasTest
4  {
5     public static void main( String args[] )
6     {
7        Gas application = new Gas();
8        application.calculateMPG();
9     } // end main
10 } // end class GasTest
```

```
Enter miles (-1 to quit): 350
Enter gallons: 18
MPG this tankful: 19.44
Total MPG: 19.44
Enter miles (-1 to quit): 475
Enter gallons: 16
MPG this tankful: 29.69
Total MPG: 24.26
Enter miles (-1 to quit): 400
Enter gallons: 17
MPG this tankful: 23.53
Total MPG: 24.02
Enter miles (-1 to quit): -1
```

4.18 Develop a Java application that will determine whether any of several department-store customers has exceeded the credit limit on a charge account. For each customer, the following facts are available:

a) account number
b) balance at the beginning of the month
c) total of all items charged by the customer this month
d) total of all credits applied to the customer's account this month
e) allowed credit limit.

The program should input all these facts as integers, calculate the new balance (= *beginning balance* + *charges* – *credits*), display the new balance and determine whether the new balance exceeds the customer's credit limit. For those customers whose credit limit is exceeded, the program should display the message `"Credit limit exceeded"`.

ANS:

```
1  // Exercise 4.18 Solution: Credit.java
2  // Program monitors accounts.
3  import java.util.Scanner;
4
5  public class Credit
6  {
7     // calculates the balance on several credit accounts
8     public void calculateBalance()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int account; // account number
13       int oldBalance; // starting balance
14       int charges; // total charges
15       int credits; // total credits
16       int creditLimit; // allowed credit limit
17       int newBalance; // new balance
18
19       System.out.print( "Enter Account Number (or -1 to quit): " );
20       account = input.nextInt(); // read in account number
21
22       // exit if the input is -1; otherwise, proceed with the program
23       while ( account != -1 )
```

```
24         {
25            System.out.print( "Enter Balance: " );
26            oldBalance = input.nextInt(); // read in original balance
27
28            System.out.print( "Enter Charges: " );
29            charges = input.nextInt(); // read in charges
30
31            System.out.print( "Enter Credits: " );
32            credits = input.nextInt(); // read in credits
33
34            System.out.print( "Enter Credit Limit: " );
35            creditLimit = input.nextInt(); // read in credit limit
36
37            // calculate and display new balance
38            newBalance = oldBalance + charges - credits;
39            System.out.printf( "New balance is %d\n", newBalance );
40
41            // display a warning if the user has exceed the credit limit
42            if ( newBalance > creditLimit )
43               System.out.println( "Credit limit exceeded" );
44
45            System.out.print( "\nEnter Account Number (or -1 to quit): " );
46            account = input.nextInt(); // read in next account number
47         } // end while
48      } // end method calculateBalance
49   } // end class Credit
```

```
1  // Exercise 4.18 Solution: CreditTest.java
2  // Test application for class Credit
3  public class CreditTest
4  {
5     public static void main(String args[])
6     {
7        Credit application = new Credit();
8        application.calculateBalance();
9     } // end main
10 } // end class CreditTest
```

```
Enter Account Number (or -1 to quit): 1
Enter Balance: 100
Enter Charges: 80
Enter Credits: 25
Enter Credit Limit: 200
New balance is 155

Enter Account Number (or -1 to quit): 2
Enter Balance: 450
Enter Charges: 240
Enter Credits: 300
Enter Credit Limit: 600
New balance is 390

Enter Account Number (or -1 to quit): 3
Enter Balance: 500
Enter Charges: 300
Enter Credits: 125
Enter Credit Limit: 400
New balance is 675
Credit limit exceeded

Enter Account Number (or -1 to quit): -1
```

4.19 A large company pays its salespeople on a commission basis. The salespeople receive $200 per week plus 9% of their gross sales for that week. For example, a salesperson who sells $5000 worth of merchandise in a week receives $200 plus 9% of $5000, or a total of $650. You have been supplied with a list of the items sold by each salesperson. The values of these items are as follows:

Item	Value
1	239.99
2	129.75
3	99.95
4	350.89

Develop a Java application that inputs one salesperson's items sold for last week and calculates and displays that salesperson's earnings. There is no limit to the number of items that can be sold by a salesperson.

ANS:

```
1  // Exercise 4.19 Solution: Sales.java
2  // Program calculates commissions based on sales.
3  import java.util.Scanner;
4
5  public class Sales
6  {
7     // calculate sales for individual products
8     public void calculateSales()
9     {
10       Scanner input = new Scanner( System.in );
11
12       double gross = 0.0; // total gross sales
13       double earnings; // earnings made from sales
```

```
14         int product = 0; // the product number
15         int numberSold; // number sold of a given product
16
17         while ( product < 4 )
18         {
19            product++;
20
21            // prompt and read number of the product sold from user
22            System.out.printf( "Enter number sold of product #%d: ",
23               product );
24            numberSold = input.nextInt();
25
26            // determine gross of each individual product and add to total
27            if ( product == 1 )
28               gross += numberSold * 239.99;
29            else if ( product == 2 )
30               gross += numberSold * 129.75;
31            else if ( product == 3 )
32               gross += numberSold * 99.95;
33            else if ( product == 4 )
34               gross += numberSold * 350.89;
35         } // end while loop
36
37         earnings = 0.09 * gross + 200; // calculate earnings
38         System.out.printf( "Earnings this week: $%.2f\n", earnings );
39      } // end method calculateSales
40   } // end class Sales
```

```
1  // Exercise 4.19 Solution: SalesTest.java
2  // Test application for class Sales
3  public class SalesTest
4  {
5     public static void main( String args[] )
6     {
7        Sales application = new Sales();
8        application.calculateSales();
9     } // end main
10 } // end class SalesTest
```

```
Enter number sold of product #1: 100
Enter number sold of product #2: 65
Enter number sold of product #3: 854
Enter number sold of product #4: 193
Earnings this week: $16896.06
```

4.20 Develop a Java application that will determine the gross pay for each of three employees. The company pays straight time for the first 40 hours worked by each employee and time and a half for all hours worked in excess of 40 hours. You are given a list of the employees of the company, the number of hours each employee worked last week and the hourly rate of each employee. Your program should input this information for each employee and should determine and display the employee's gross pay. Use class `Scanner` to input the data.

ANS:

```
// Exercise 4.20 Solution: Wages.java
// Program calculates wages.
import java.util.Scanner;

public class Wages
{
   // calculates wages for 3 employees
   public void calculateWages()
   {
      Scanner input = new Scanner( System.in );

      double pay; // gross pay
      double hours; // hours worked
      double rate; // hourly rate
      int count = 1; // number of employees

      // repeat calculation for 3 employees
      while ( count <= 3 )
      {
         System.out.print( "Enter hourly rate: " );
         rate = input.nextDouble(); // read the hourly rate

         System.out.printf( "Enter hours worked: " );
         hours = input.nextDouble(); // read hours worked

         // calculate wages
         if ( hours > 40 ) // with overtime
            pay = ( 40.0 * rate ) + ( hours - 40 ) * ( rate * 1.5 );
         else // straight time
            pay = hours * rate;

         // print the pay for the current employee
         System.out.printf( "Pay for Employee %d is $%.2f\n",
            count, pay );

         count++;
      } // end while loop
   } // end method calculateWages
} // end class Wages
```

```
// Exercise 4.20 Solution: WagesTest.java
// Test application for class Wages
public class WagesTest
{
   public static void main( String args[] )
   {
      Wages application = new Wages();
      application.calculateWages();
   } // end main
} // end class WagesTest
```

```
Enter hourly rate: 12.50
Enter hours worked: 50
Pay for Employee 1 is $687.50
Enter hourly rate: 27
Enter hours worked: 40
Pay for Employee 2 is $1080.00
Enter hourly rate: 8.25
Enter hours worked: 30.5
Pay for Employee 3 is $251.63
```

4.21 The process of finding the largest value (i.e., the maximum of a group of values) is used frequently in computer applications. For example, a program that determines the winner of a sales contest would input the number of units sold by each salesperson. The salesperson who sells the most units wins the contest. Write a pseudocode program and then a Java application that inputs a series of 10 integers and determines and prints the largest integer. Your program should use at least the following three variables:

a) `counter`: A counter to count to 10 (i.e., to keep track of how many numbers have been input and to determine when all 10 numbers have been processed).
b) `number`: The integer most recently input by the user.
c) `largest`: The largest number found so far.

ANS:

```
1  // Exercise 4.21 Solution: Largest.java
2  // Program determines and prints the largest of 10 numbers.
3  import java.util.Scanner;
4
5  public class Largest
6  {
7     // determine the largest of 10 numbers
8     public void determineLargest()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int largest; // largest number
13       int number; // user input
14       int counter; // number of values entered
15
16       // get first number and assign it to variable largest
17       System.out.print( "Enter number: " );
18       largest = input.nextInt();
19
20       counter = 1;
21
22       // get rest of the numbers and find the largest
23       while ( counter < 10 )
24       {
25          System.out.print( "Enter number: " );
26          number = input.nextInt();
27
28          if ( number > largest )
29             largest = number;
```

```
30
31          counter++;
32       } // end while loop
33
34       System.out.printf( "Largest number is %d\n", largest );
35    } // end method determineLargest
36 } // end class Largest
```

```
1  // Exercise 4.21 Solution: LargestTest.java
2  // Test application for class Largest
3  public class LargestTest
4  {
5     public static void main( String args[] )
6     {
7        Largest application = new Largest();
8        application.determineLargest();
9     } // end main
10 } // end class LargestTest
```

```
Enter number: 56
Enter number: -10
Enter number: 200
Enter number: 25
Enter number: 8
Enter number: 500
Enter number: -20
Enter number: 678
Enter number: 345
Enter number: 45
Largest number is 678
```

4.22 Write a Java application that uses looping to print the following table of values:

```
N       10*N    100*N   1000*N

1       10      100     1000
2       20      200     2000
3       30      300     3000
4       40      400     4000
5       50      500     5000
```

ANS:

```
1  // Exercise 4.22 Solution: Table.java
2  // Program prints a table of values using a while loop.
3
4  public class Table
5  {
6     public static void main( String args[] )
```

```
   {
      int n = 1;

      System.out.println( "N\t10*N\t100*N\t1000*N\n" );

      while ( n <= 5 )
      {
         System.out.printf( "%d\t%d\t%d\t%d\n",
            n,  ( 10 * n ), ( 100 * n ), ( 1000 * n ) );
         n++;
      } // end while loop
   } // end main
} // end class Table
```

```
N       10*N    100*N   1000*N

1       10      100     1000
2       20      200     2000
3       30      300     3000
4       40      400     4000
5       50      500     5000
```

4.23 Using an approach similar to that for Exercise 4.21, find the *two* largest values of the 10 values entered. [*Note*: You may input each number only once.]

ANS:

```
// Exercise 4.23 Solution: TwoLargest.java
// Program determines and prints the two largest of 10 numbers.
import java.util.Scanner;

public class TwoLargest
{
   // determine the two largest of 10 integers
   public void determineTwoLargest()
   {
      Scanner input = new Scanner( System.in );

      int largest; // largest number
      int nextLargest; // second largest number
      int number; // user input
      int counter; // number of values entered

      // get first number and assign it to variable largest
      System.out.print( "Enter number: " );
      largest = input.nextInt();

      // get second number and compare it with first number
      System.out.print( "Enter number: " );
      number = input.nextInt();

      if ( number > largest )
      {
```

```
            nextLargest = largest;
            largest = number;
         } // end if
         else
            nextLargest = number;

         counter = 2;

         // get rest of the numbers and find the largest and nextLargest
         while ( counter < 10 )
         {
            System.out.print( "Enter number: " );
            number = input.nextInt();

            if ( number > largest ) {
               nextLargest = largest;
               largest = number;
            } // end if
            else if ( number > nextLargest )
               nextLargest = number;

            counter++;
         } // end while loop

         System.out.printf( "Largest is %d\nSecond largest is %d\n",
            largest, nextLargest );
      } // end method determineTwoLargest
   } // end class TwoLargest
```

```
1  // Exercise 4.23 Solution: TwoLargestTest.java
2  // Test application for class TwoLargest
3  public class TwoLargestTest
4  {
5     public static void main( String args[] )
6     {
7        TwoLargest application = new TwoLargest();
8        application.determineTwoLargest();
9     } // end main
10 } // end class TwoLargestTest
```

```
Enter number: 42
Enter number: 67
Enter number: 32
Enter number: 98
Enter number: 87
Enter number: 56
Enter number: 3
Enter number: -35
Enter number: 123
Enter number: -56
Largest is 123
Second largest is 98
```

4.24 Modify the program in Fig. 4.12 to validate its inputs. For any input, if the value entered is other than 1 or 2, keep looping until the user enters a correct value.

ANS:

```
1  // Exercise 4.24 Solution: Analysis.java
2  // Program performs analysis of examination results.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class Analysis
6  {
7     // anaylze the results of 10 tests
8     public void processExamResults()
9     {
10       // create Scanner to obtain input from command window
11       Scanner input = new Scanner( System.in );
12
13       // initializing variables in declarations
14       int passes = 0; // number of passes
15       int failures = 0; // number of failures
16       int studentCounter = 1; // student counter
17       int result; // one exam result
18
19       // process 10 students using counter-controlled loop
20       while ( studentCounter <= 10 )
21       {
22          // prompt user for input and obtain value from user
23          System.out.print( "Enter result (1 = pass, 2 = fail): " );
24          result = input.nextInt();
25
26          // if...else nested in while
27          if ( result == 1 )
28          {
29             passes = passes + 1;
30             studentCounter = studentCounter + 1;
31          } // end if
32          else if ( result == 2 )
33          {
34             failures = failures + 1;
35             studentCounter = studentCounter + 1;
36          } // end else if
37          else
38             System.out.println( "Invalid Input" );
39       } // end while
40
41       // termination phase; prepare and display results
42       System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );
43
44       // determine whether more than 8 students passed
45       if ( passes > 8 )
46          System.out.println( "Raise Tuition" );
47    } // end method processExamResults
48 } // end class Analysis
```

```
1  // Exercise 4.24 Solution: AnalysisTest.java
2  // Test application for class Analysis
3  public class AnalysisTest
4  {
5     public static void main( String args[] )
6     {
7        Analysis application = new Analysis();
8        application.processExamResults();
9     } // end main
10 } // end class AnalysisTest
```

```
Enter result (1 = pass, 2 = fail): 3
Invalid Input
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 8
Failed: 2
```

4.25 What does the following program print?

```
1  public class Mystery2
2  {
3     public static void main( String args[] )
4     {
5        int count = 1;
6
7        while ( count <= 10 )
8        {
9           System.out.println( count % 2 == 1 ? "****" : "++++++++" );
10          ++count;
11       } // end while
12    } // end main
13
14 } // end class Mystery2
```

ANS:

```
****
++++++++
****
++++++++
****
++++++++
****
++++++++
****
++++++++
```

4.26 What does the following program print?

```
1  public class Mystery3
2  {
3     public static void main( String args[] )
4     {
5        int row = 10;
6        int column;
7
8        while ( row >= 1 )
9        {
10          column = 1;
11
12          while ( column <= 10 )
13          {
14             System.out.print( row % 2 == 1 ? "<" : ">" );
15             ++column;
16          } // end while
17
18          --row;
19          System.out.println();
20       } // end while
21    } // end main
22
23 } // end class Mystery3
```

ANS:

```
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
```

4.27 *(Dangling-`else` Problem)* Determine the output for each of the given sets of code when `x` is `9` and `y` is `11` and when `x` is `11` and `y` is `9`. Note that the compiler ignores the indentation in a Java program. Also, the Java compiler always associates an `else` with the immediately preceding `if` unless told to do otherwise by the placement of braces (`{}`). On first glance, the programmer may not be sure which `if` an `else` matches—this situation is referred to as the "dangling-`else` problem." We have eliminated the indentation from the following code to make the problem more challenging. [*Hint*: Apply the indentation conventions you have learned.]

a)
```
if ( x < 10 )
if ( y > 10 )
System.out.println( "*****" );
else
System.out.println( "#####" );
System.out.println( "$$$$$" );
```

ANS:

```
When:  x = 9, y = 11
*****
$$$$$

When:  x = 11, y = 9
$$$$$
```

b)
```
if ( x < 10 )
{
if ( y > 10 )
System.out.println( "*****"  );
}
else
{
System.out.println( "#####" );
System.out.println( "$$$$$" );
}
```

ANS:

```
When:  x = 9, y = 11
*****

When:  x = 11, y = 9
#####
$$$$$
```

4.28 *(Another Dangling-`else` Problem)* Modify the given code to produce the output shown in each part of the problem. Use proper indentation techniques. Make no changes other than inserting braces and changing the indentation of the code. The compiler ignores indentation in a Java program. We have eliminated the indentation from the given code to make the problem more challenging. [*Note*: It is possible that no modification is necessary for some of the parts.]

```
if ( y == 8 )
if ( x == 5 )
```

```
System.out.println( "@@@@@" );
else
System.out.println( "#####" );
System.out.println( "$$$$$" );
System.out.println( "&&&&&" );
```

a) Assuming that x = 5 and y = 8, the following output is produced:

```
@@@@@
$$$$$
&&&&&
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      System.out.println( "@@@@@" );
   else
      System.out.println( "#####" );
System.out.println( "$$$$$" );
System.out.println( "&&&&&" );
```

b) Assuming that x = 5 and y = 8, the following output is produced:

```
@@@@@
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      System.out.println( "@@@@@" );
   else
   {
      System.out.println( "#####" );
      System.out.println( "$$$$$" );
      System.out.println( "&&&&&" );
   }
```

c) Assuming that x = 5 and y = 8, the following output is produced:

```
@@@@@
&&&&&
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      System.out.println( "@@@@@" );
   else
   {
      System.out.println( "#####" );
      System.out.println( "$$$$$" );
   }
System.out.println( "&&&&&" );
```

d) Assuming that x = 5 and y = 7, the following output is produced. [*Note*: The last three output statements after the `else` are all part of a block.]

```
#####
$$$$$
&&&&&
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      System.out.println( "@@@@@" );
else
{
   System.out.println( "#####" );
   System.out.println( "$$$$$" );
   System.out.println( "&&&&&" );
}
```

4.29 Write an application that prompts the user to enter the size of the side of a square, then displays a hollow square of that size made of asterisks. Your program should work for squares of all side lengths between 1 and 20.

ANS:

```
1  // Exercise 4.29 Solution: Hollow.java
2  // Program prints a hollow square.
3  import java.util.Scanner;
4
5  public class Hollow
6  {
7     // draw a hollow box surrounded by stars
8     public void drawHollowBox()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int stars; // number of stars on a side
13       int column; // the current column of the square being printed
14       int row = 1; // the current row of the square being printed
15
16       // prompt and read the length of the side from the user
17       System.out.print( "Enter length of side:" );
18       stars = input.nextInt();
19
20       if ( stars < 1 )
21       {
22          stars = 1;
23          System.out.println( "Invalid Input\nUsing default value 1" );
24       } // end if
25       else if ( stars > 20 )
26       {
27          stars = 20;
28          System.out.println( "Invalid Input\nUsing default value 20" );
29       } // end else if
30
31       // repeat for as many rows as the user entered
32       while ( row <= stars )
33       {
```

```
            column = 1;

            // and for as many columns as rows
            while ( column <= stars )
            {
               if ( row == 1 )
                  System.out.print( "*" );
               else if ( row == stars )
                  System.out.print( "*" );
               else if ( column == 1 )
                  System.out.print( "*" );
               else if ( column == stars )
                  System.out.print( "*" );
               else
                  System.out.print( " " );

               column++;
            } // end inner while loop

            System.out.println();
            row++;
         } // end outer while loop
      } // end method drawHollowBox
   } // end class Hollow
```

```
// Exercise 4.29 Solution: HollowTest.java
// Test application for class Hollow
public class HollowTest
{
   public static void main( String args[] )
   {
      Hollow application = new Hollow();
      application.drawHollowBox();
   } // end main
} // end class HollowTest
```

```
Enter length of side:15
***************
*             *
*             *
*             *
*             *
*             *
*             *
*             *
*             *
*             *
*             *
*             *
*             *
*             *
***************
```

4.30 *(Palindromes)* A palindrome is a sequence of characters that reads the same backward as forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write an application that reads in a five-digit integer and determines whether it is a palindrome. If the number is not five digits long, display an error message and allow the user to enter a new value.

ANS:

```
1  // Exercise 4.30 Solution: Palindrome.java
2  // Program tests for a palindrome
3  import java.util.Scanner;
4
5  public class Palindrome
6  {
7     // checks if a 5-digit number is a palindrome
8     public void checkPalindrome()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int number; // user input number
13       int digit1; // first digit
14       int digit2; // second digit
15       int digit4; // fourth digit
16       int digit5; // fifth digit
17       int digits; // number of digits in input
18
19       number = 0;
20       digits = 0;
21
22       // Ask for a number until it is five digits
23       while ( digits != 5 )
24       {
25          System.out.print( "Enter a 5-digit number: " );
26          number = input.nextInt();
27
28          if ( number < 100000 )
29          {
30             if ( number > 9999 )
31                digits = 5;
32             else
33                System.out.println( "Number must be 5 digits" );
34          } // end if
35          else
36             System.out.println( "Number must be 5 digits" );
37       } // end while loop
38
39       // get the digits
40       digit1 = number / 10000;
41       digit2 = number % 10000 / 1000;
42       digit4 = number % 10000 % 1000 % 100 / 10;
43       digit5 = number % 10000 % 1000 % 100 % 10;
44
45       // print whether the number is a palindrome
46       System.out.print( number );
```

```
47         if ( digit1 == digit5 )
48         {
49            if ( digit2 == digit4 )
50               System.out.println( " is a palindrome!!!" );
51            else
52               System.out.println( " is not a palindrome." );
53         }
54         else
55            System.out.println( " is not a palindrome." );
56      } // end method checkPalindrome
57   } // end class Palindrome
```

```
1  // Exercise 4.30 Solution: PalindromeTest.java
2  // Test application for class Palindrome
3  public class PalindromeTest
4  {
5     public static void main( String args[] )
6     {
7        Palindrome application = new Palindrome();
8        application.checkPalindrome();
9     } // end main
10 } // end class PalindromeTest
```

```
Enter a 5-digit number: 54345
54345 is a palindrome!!!
```

4.31 Write an application that inputs an integer containing only 0s and 1s (i.e., a binary integer) and prints its decimal equivalent. [*Hint*: Use the remainder and division operators to pick off the binary number's digits one at a time, from right to left. In the decimal number system, the rightmost digit has a positional value of 1 and the next digit to the left has a positional value of 10, then 100, then 1000, and so on. The decimal number 234 can be interpreted as 4 * 1 + 3 * 10 + 2 * 100. In the binary number system, the rightmost digit has a positional value of 1, the next digit to the left has a positional value of 2, then 4, then 8, and so on. The decimal equivalent of binary 1101 is 1 * 1 + 0 * 2 + 1 * 4 + 1 * 8, or 1 + 0 + 4 + 8 or, 13.]

ANS:

```
1  // Exercise 4.31 Solution: Binary.java
2  // Program prints the decimal equivalent of a binary number.
3  import java.util.Scanner;
4
5  public class Binary
6  {
7     // converts a binary number to decimal
8     public void convertToDecimal()
9     {
10       Scanner input = new Scanner( System.in );
11       int binary; // binary value
12       int bit; // bit positional value
13       int decimal; // decimal value
14
```

```
15         bit = 1;
16         decimal = 0;
17
18         // prompt for and read in a binary number
19         System.out.print( "Enter a binary number: " );
20         binary = input.nextInt();
21
22         // convert to decimal equivalent
23         while ( binary != 0 )
24         {
25            decimal += binary % 10 * bit;
26            binary /= 10;
27            bit *= 2;
28         } // end while loop
29
30         System.out.printf( "Decimal is: %d\n", decimal );
31      } // end method convertToDecimal
32   } // end class Binary
```

```
1   // Exercise 4.31 Solution: BinaryTest.java
2   // Test application for class Binary
3   public class BinaryTest
4   {
5      public static void main( String args[] )
6      {
7         Binary application = new Binary();
8         application.convertToDecimal();
9      } // end main
10  } // end class BinaryTest
```

```
Enter a binary number: 11000000
Decimal is: 192
```

4.32 Write an application that uses only the output statements

```
System.out.print( "* " );
System.out.print( " " );
System.out.println();
```

to display the checkerboard pattern that follows. Note that a `System.out.println` method call with no arguments causes the program to output a single newline character. [*Hint*: Repetition statements are required.]

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

ANS:

```
1  // Exercise 4.32 Solution: Stars.java
2  // Program prints a checkerboard pattern.
3  public class Stars
4  {
5     public static void main( String args[] )
6     {
7        int row = 1;
8
9        while ( row <= 8 )
10       {
11          int column = 1;
12
13          if ( row % 2 == 0 )
14             System.out.print( " " );
15
16          while ( column <= 8 )
17          {
18             System.out.print( "* " );
19             column++;
20          } // end inner while loop
21
22          System.out.println();
23          row++;
24       } // end outer while loop
25    } // end main
26 } // end class Stars
```

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

4.33 Write an application that keeps displaying in the command window the multiples of the integer 2—namely, 2, 4, 8, 16, 32, 64, and so on. Your loop should not terminate (i.e., create an infinite loop). What happens when you run this program?

ANS: Eventually, the numbers become too large to represent in an `int` variable and the program displays 0s instead.

```
1  // Exercise 4.33 Solution: Infinite.java
2  // Program creates an infinite loop.
3
4  public class Infinite
5  {
6     public static void main( String args[] )
7     {
```

```
        int x = 1;

        while ( true )
        {
           x *= 2;
           System.out.println( x );
        } // end while loop
     } // end main
  } // end class Infinite
```

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
-2147483648
0
0
0
```

4.34 What is wrong with the following statement? Provide the correct statement to add one to the sum of x and y.

```
System.out.println( ++(x + y) );
```

ANS: ++ can be applied only to a variable—not to san expression with multiple terms. The correct statement is `System.out.println( x + y + 1 );`

4.35 Write an application that reads three nonzero values entered by the user and determines and prints whether they could represent the sides of a triangle.

ANS:

```
// Exercise 4.35 Solution: Triangle1.java
// Program takes three values and determines if
// they form the sides of a triangle.
import java.util.Scanner;

public class Triangle1
{
   // checks if three sides can form a triangle
   public void checkSides()
   {
      Scanner input = new Scanner( System.in );

      double side1; // length of side 1
      double side2; // length of side 2
      double side3; // length of side 3
      boolean isTriangle; // whether the sides can form a triangle

      // get values of three sides
      System.out.print( "Enter side 1: " );
      side1 = input.nextDouble();

      System.out.print( "Enter side 2: " );
      side2 = input.nextDouble();

      System.out.print( "Enter side 3: " );
      side3 = input.nextDouble();

      // triangle testing
      isTriangle = false;

      if ( side1 + side2 > side3 )
      {
         if ( side2 + side3 > side1 )
         {
            if ( side3 + side1 > side2 )
               isTriangle = true;
         } // end inner if statement
      } // end outer if statement

      if ( isTriangle )
         System.out.println( "These could be sides to a triangle " );
      else
         System.out.println( "These do not form a triangle." );
   } // end method checkSides
} // end class Triangle1
```

```
// Exercise 4.35 Solution: Triangle1Test.java
// Test application for class Triangle1
```

```
3  public class Triangle1Test
4  {
5     public static void main( String args[] )
6     {
7        Triangle1 application = new Triangle1();
8        application.checkSides();
9     } // end main
10 } // end class Triangle1Test
```

```
Enter side 1: 3
Enter side 2: 4
Enter side 3: 5
These could be sides to a triangle
```

4.36 Write an application that reads three nonzero integers and determines and prints whether they could represent the sides of a right triangle.

ANS:

```
1  // Exercise 4.36 Solution: Triangle2.java
2  // Program takes three integers and determines if they
3  // form the sides of a right triangle.
4  import java.util.Scanner;
5
6  public class Triangle2
7  {
8     // checks if three sides can form a right triangle
9     public void checkSides()
10    {
11       Scanner input = new Scanner( System.in );
12
13       int side1; // length of side 1
14       int side2; // length of side 2
15       int side3; // length of side 3
16       boolean isRightTriangle; // whether the sides can form a triangle
17
18       // get values of three sides
19       System.out.print( "Enter side 1: " );
20       side1 = input.nextInt();
21
22       System.out.print( "Enter side 2: " );
23       side2 = input.nextInt();
24
25       System.out.print( "Enter side 3: " );
26       side3 = input.nextInt();
27
28       // square the sides
29       int side1Square = side1 * side1;
30       int side2Square = side2 * side2;
31       int side3Square = side3 * side3;
32
33       // test if these form a right triangle
34       isRightTriangle = false;
```

```
35
36          if ( ( side1Square + side2Square ) == side3Square )
37            isRightTriangle = true;
38          else if ( ( side1Square + side3Square ) == side2Square )
39             isRightTriangle = true;
40          else if ( ( side2Square + side3Square ) == side1Square )
41             isRightTriangle = true;
42
43          if ( isRightTriangle )
44             System.out.println( "These are the sides of a right triangle." );
45          else
46             System.out.println( "These do not form a right triangle." );
47       } // end method checkSides
48    } // end class Triangle2
```

```
1  // Exercise 4.36 Solution: Triangle2Test.java
2  // Test application for class Triangle2
3  public class Triangle2Test
4  {
5     public static void main( String args[] )
6     {
7        Triangle2 application = new Triangle2();
8        application.checkSides();
9     } // end main
10 } // end class Triangle2Test
```

```
Enter side 1: 5
Enter side 2: 4
Enter side 3: 3
These are the sides of a right triangle
```

4.37 A company wants to transmit data over the telephone but is concerned that its phones may be tapped. It has asked you to write a program that will encrypt the data so that it may be transmitted more securely. All the data is transmitted as four-digit integers. Your application should read a four-digit integer entered by the user and encrypt it as follows: Replace each digit with the result of adding 7 to the digit and getting the remainder after dividing the new value by 10. Then swap the first digit with the third, and swap the second digit with the fourth. Then print the encrypted integer. Write a separate application that inputs an encrypted four-digit integer and decrypts it to form the original number.

ANS:

```
1  // Exercise 4.37 Part A Solution: Encrypt.java
2  // Program encrypts a four-digit number.
3  import java.util.Scanner;
4
5  public class Encrypt
6  {
7     // encrypt a four-digit number
8     public void encrypt()
9     {
```

```
10         Scanner input = new Scanner( System.in );
11
12         int number; // original number
13         int digit1; // first digit
14         int digit2; // second digit
15         int digit3; // third digit
16         int digit4; // fourth digit
17         int encryptedNumber; // encrypted number
18
19         // enter four-digit number to be encrypted
20         System.out.print( "Enter a four-digit number: " );
21         number = input.nextInt();
22
23         // encrypt
24         digit1 = ( number / 1000 + 7 ) % 10;
25         digit2 = ( number % 1000 / 100 + 7 ) % 10;
26         digit3 = ( number % 100 / 10 + 7 ) % 10;
27         digit4 = ( number % 10 + 7 ) % 10;
28
29         encryptedNumber = digit1 * 10 + digit2 +
30            digit3 * 1000 + digit4 * 100;
31
32         System.out.printf( "Encrypted number is %d\n", encryptedNumber );
33      } // end method encrypt
34   } // end class Encrypt
```

```
1   // Exercise 4.37 Part A Solution: EncryptTest.java
2   // Test application for class Encrypt
3   public class EncryptTest
4   {
5      public static void main( String args[] )
6      {
7         Encrypt application = new Encrypt();
8         application.encrypt();
9      } // end main
10  } // end class Encrypt
```

```
Enter a four-digit number: 5948
Encrypted number is 1526
```

```
1   // Exercise 4.37 Part B Solution: Decrypt.java
2   // Program decrypts a four-digit number.
3   import java.util.Scanner;
4
5   public class Decrypt
6   {
7      // decrypts a four-digit number
8      public void decrypt()
9      {
10        Scanner input = new Scanner( System.in );
```

```
11
12        int number; // original number
13        int digit1; // first digit
14        int digit2; // second digit
15        int digit3; // third digit
16        int digit4; // fourth digit
17        int decryptedNumber; // encrypted number
18
19        // enter four digit number to be decrypted
20        System.out.print( "Enter a four-digit number: " );
21        number = input.nextInt();
22
23        // decrypt
24        digit1 = ( number / 1000 + 3 ) % 10;
25        digit2 = ( number % 1000 / 100 + 3 ) % 10;
26        digit3 = ( number % 100 / 10 + 3 ) % 10;
27        digit4 = ( number % 10 + 3 ) % 10;
28
29        decryptedNumber = digit1 * 10 + digit2 +
30           digit3 * 1000 + digit4  * 100;
31
32        System.out.printf( "Decrypted number is %d\n", decryptedNumber );
33     } // end method decrypt
34  } // end class Decrypt
```

```
1  // Exercise 4.37 Part B Solution: DecryptTest.java
2  // Test application for class Decrypt
3  public class DecryptTest
4  {
5     public static void main( String args[] )
6     {
7        Decrypt application = new Decrypt();
8        application.decrypt();
9     } // end main
10 } // end class Decrypt
```

```
Enter a four-digit number: 1526
Decrypted number is 5948
```

4.38 The factorial of a nonnegative integer n is written as $n!$ (pronounced "n factorial") and is defined as follows:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1 \quad \text{(for values of } n \text{ greater than or equal to 1)}$$

and

$$n! = 1 \quad \text{(for } n = 0\text{)}$$

For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, which is 120.

a) Write an application that reads a nonnegative integer and computes and prints its factorial.

ANS:

```
1  // Exercise 4.38 Part A Solution: Factorial.java
2  // Program calculates a factorial.
3  import java.util.Scanner;
4
5  public class Factorial
6  {
7     // calculates the factorial of a number
8     public void calculateFactorial()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int number; // user input
13       int factorial; // factorial of input value
14
15       factorial = 1;
16
17       System.out.print( "Enter a positive Integer: " );
18       number = input.nextInt();
19
20       System.out.printf( "%d! is ", number);
21
22       // calculate factorial
23       while ( number > 0 )
24       {
25          factorial *= number;
26          number--;
27       } // end while loop
28
29       System.out.println( factorial );
30    } // end method calculateFactorial
31 } // end class Factorial
```

```
1  // Exercise 4.38 Part A Solution: FactorialTest.java
2  // Test application for class Factorial
3  public class FactorialTest
4  {
5     public static void main( String args[] )
6     {
7        Factorial application = new Factorial();
8        application.calculateFactorial();
9     } // end main
10 } // end class Factorial
```

```
Enter a positive Integer: 14
14! is 1278945280
```

b) Write an application that estimates the value of the mathematical constant *e* by using the formula

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

ANS:

```
1  // Exercise 4.38 Part B Solution: E.java
2  // Program calculates estimated value of e.
3  import java.util.Scanner;
4
5  public class E
6  {
7     // approximates the value of E
8     public void approximate()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int number; // counter
13       int accuracy; // accuracy of estimate
14       int factorial; // value of factorial
15       double e; // estimate value of e
16
17       number = 1;
18       factorial = 1;
19       e = 1.0;
20
21       System.out.print( "Enter desired accuracy of e: " );
22       accuracy = input.nextInt();
23
24       // calculate estimation
25       while ( number < accuracy )
26       {
27          factorial *= number;
28          e += 1.0 / factorial;
29          number++;
30       } // end while loop
31
32       System.out.print( "e is " );
33       System.out.println( e );
34    } // end method approximate
35 } // end class E
```

```
1  // Exercise 4.38 Part B Solution: ETest.java
2  // Test application for class E
3  public class ETest
4  {
5     public static void main( String args[] )
6     {
7        E application = new E();
8        application.approximate();
9     } // end main
10 } // end class E
```

```
Enter desired accuracy of e: 12
e is 2.718281826198493
```

c) Write an application that computes the value of e^x by using the formula

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

ANS:

```
1  // Exercise 4.38 Part C Solution: EtoX.java
2  // Program calculates e raised to x.
3  import java.util.Scanner;
4
5  public class EtoX
6  {
7     // approximates the value of e to the X
8     public void approximate()
9     {
10       Scanner input = new Scanner( System.in );
11
12       int number; // counter
13       int accuracy; // accuracy of estimate
14       int factorial; // value of factorial
15       int x; // x value
16       double e; // estimate value of e
17       double exponent; // exponent value
18
19       number = 1;
20       factorial = 1;
21       e = 1.0;
22       exponent = 1.0;
23
24       System.out.print( "Enter exponent: " );
25       x = input.nextInt();
26
27       System.out.print( "Enter desired accuracy of e: " );
28       accuracy = input.nextInt();
29
30       // calculate estimation
31       while ( number < accuracy )
32       {
33          exponent *= x;
34          factorial *= number;
35          e += exponent / factorial;
36          number++;
37       } // end while loop
38
39       System.out.printf( "e to the %d is ", x );
40       System.out.println( e );
41    } // end method approximate
42 } // end class EtoX
```

```
1  // Exercise 4.38 Part C Solution: EtoXTest.java
2  // Test application for class EtoX
3  public class EtoXTest
4  {
5     public static void main( String args[] )
6     {
7        EtoX application = new EtoX();
8        application.approximate();
9     } // end main
10 } // end class EtoX
```

```
Enter exponent: 2
Enter desired accuracy of e: 10
e to the 2 is 7.3887125220458545
```

(Optional) GUI and Graphics Case Study

4.1 Using loops and control statements to draw lines can lead to many interesting designs.

a) Create the design in the left screen capture of Fig. 4.1. This design draws lines from the top-left corner, fanning out the lines until they cover the upper-left half of the panel. One approach is to divide the width and height into an equal number of steps (we found 15 steps worked well). The first endpoint of a line will always be in the top-left corner (0, 0). The second endpoint can be found by starting at the bottom-left corner and s vertical step and right one horizontal step. Draw a line between the two endpoints. Continue moving up and to the right one step to find each successive endpoint. The figure should scale accordingly as you resize the window.

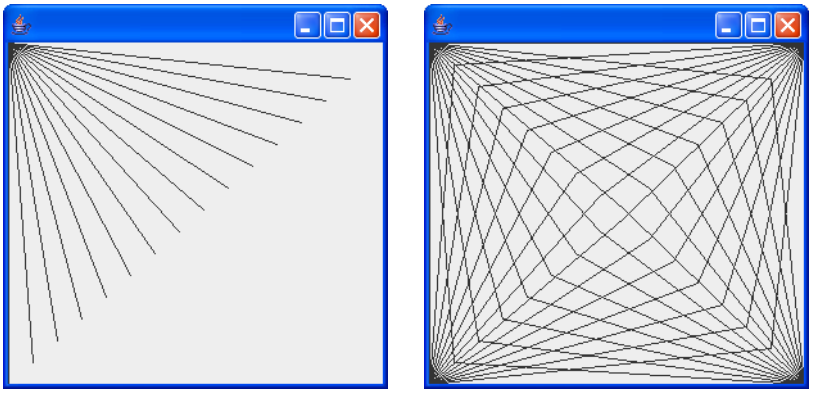


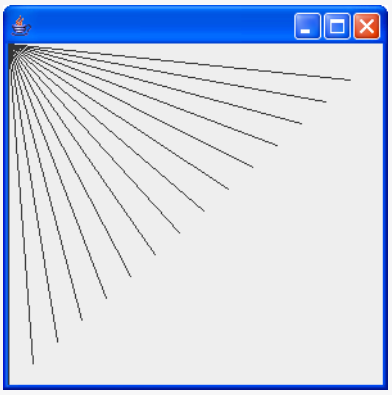
Fig. 4.1 | Lines fanning from a corner.

ANS:

```
1  GCS Exercise 4.1 Part A: Lines1.java
2  // Draws lines using a loop
3  import java.awt.Graphics;
```

```
4  import javax.swing.JPanel;
5
6  public class Lines1 extends JPanel
7  {
8     // draws lines that fan out from the corners and intersect
9     // along the center
10    public void paintComponent( Graphics g )
11    {
12       super.paintComponent( g );
13
14       int increments = 15; // number of increments each side is divided
15
16       int width = getWidth(); // total width
17       int height = getHeight(); // total height
18
19       int widthStep = width / increments; // width increment
20       int heightStep = height / increments; // height increment
21
22       int count = 0; // loop counter
23
24       while ( count < increments )
25       {
26          // lines fanning from the top-left
27          g.drawLine( 0, 0,
28             count * widthStep, height - count * heightStep );
29
30          count++;
31       } // end while
32    } // end method paintComponent
33 } // end class Lines1
```

```
1  // GCS Exercise 4.1 Part A: Lines1Test.java
2  // Application to display class Lines1
3  import javax.swing.JFrame;
4
5  public class Lines1Test
6  {
7     public static void main( String args[] )
8     {
9        Lines1 panel = new Lines1(); // create the panel with the drawing
10       JFrame application = new JFrame(); // create a new frame
11
12       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13       application.add( panel ); // add the panel to the frame
14       application.setSize( 300, 300 ); // set the size
15       application.setVisible( true ); // show the frame
16    } // end main
17 } // end class Lines1Test
```



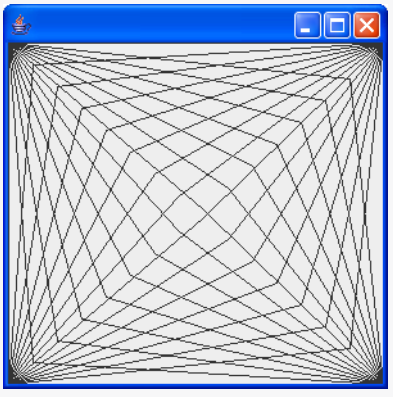
b) Modify your answer in part (a) to have lines fan out from all four corners, as shown in the right screen capture of Fig. 4.1. Lines from opposite corners should intersect along the middle.

ANS:

```
1  // GCS Exercise 4.1 Part B: Lines1.java
2  // Draws lines using a loop
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class Lines1 extends JPanel
7  {
8     // draws lines that fan out from the corners and intersect
9     // along the center
10    public void paintComponent( Graphics g )
11    {
12       super.paintComponent( g );
13
14       int increments = 15; // number of increments each side is divided
15
16       int width = getWidth(); // total width
17       int height = getHeight(); // total height
18
19       int widthStep = width / increments; // width increment
20       int heightStep = height / increments; // height increment
21
22       int count = 0; // loop counter
23
24       while ( count < increments )
25       {
26          // lines fanning from the top-left
27          g.drawLine( 0, 0,
28             count * widthStep, height - count * heightStep );
29
30          // lines fanning from the bottom-right
31          g.drawLine( width, height,
32             count * widthStep, height - count * heightStep );
```

```
33
34          // lines fanning from the bottom-left
35          g.drawLine( 0, height, count * widthStep, count * heightStep );
36
37          // lines fanning from the top-right
38          g.drawLine( width, 0, count * widthStep, count * heightStep );
39
40          count++;
41       } // end while
42    } // end method paintComponent
43 } // end class Lines1
```

```
1  // GCS Exercise 4.1 Part B: Lines1Test.java
2  // Application to display class Lines1
3  import javax.swing.JFrame;
4
5  public class Lines1Test
6  {
7     public static void main( String args[] )
8     {
9        Lines1 panel = new Lines1(); // create the panel with the drawing
10       JFrame application = new JFrame(); // create a new frame
11
12       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13       application.add( panel ); // add the panel to the frame
14       application.setSize( 300, 300 ); // set the size
15       application.setVisible( true ); // show the frame
16    } // end main
17 } // end class Lines1Test
```



4.2 Figure 4.2 displays two additional designs created using `while` loops and `drawLine`.

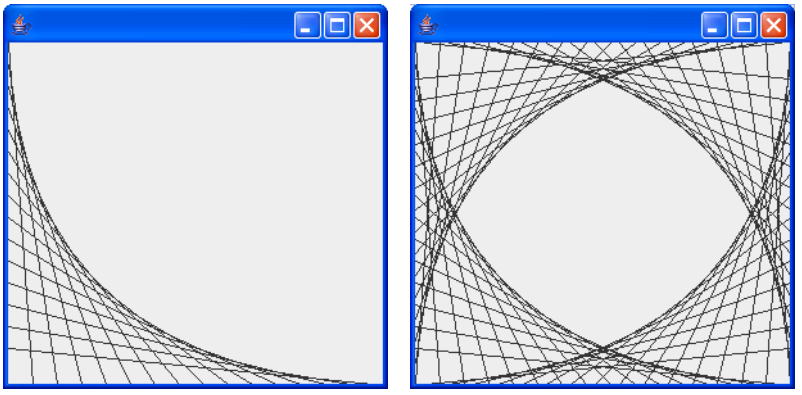


Fig. 4.2 | Line art with loops and `drawLine`.

a) Create the design in the left screen capture of Fig. 4.2. Begin by dividing each edge into an equal number of increments (we chose 15 again). The first line starts in the top-left corner and ends one step right on the bottom edge. For each successive line, move down one increment on the left edge and right one increment on the bottom edge. Continue drawing lines until you reach the bottom-right corner. The figure should scale as you resize the window so that the endpoints always touch the edges.

ANS:

```
1  // GCS Exercise 4.2 Part A: Lines2.java
2  // Draws lines using a loop
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class Lines2 extends JPanel
7  {
8     // draws lines from one edge to another
9     public void paintComponent( Graphics g )
10    {
11       super.paintComponent( g );
12
13       int increments = 15; // number of increments each side is divided
14
15       int width = getWidth(); // total width
16       int height = getHeight(); // total height
17
18       int widthStep = width / increments; // width increment
19       int heightStep = height / increments; // height increment
20
21       int count = 0; // loop counter
22
23       // draws lines in a pattern between adjacent edges
24       while ( count < increments )
25       {
```

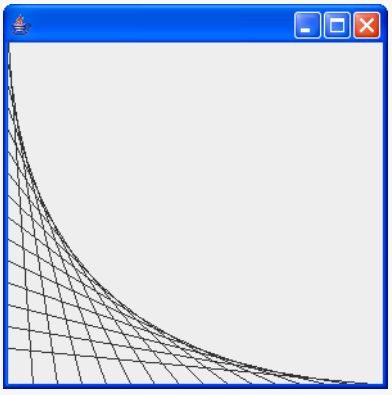
```
            // left to bottom
            g.drawLine( 0, count * heightStep,
               ( count + 1 ) * widthStep, height );

            count++;
         } // end while
      } // end method paintComponent
   } // end class Lines2
```

```
// GCS Exercise 4.2 Part A: Lines2Test.java
// Application to display class Lines1
import javax.swing.JFrame;

public class Lines2Test
{
   public static void main( String args[] )
   {
      Lines2 panel = new Lines2(); // create the panel with the drawing
      JFrame application = new JFrame(); // create a new frame

      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      application.add( panel ); // add the panel to the frame
      application.setSize( 300, 300 ); // set the size
      application.setVisible( true ); // show the frame
   } // end main
} // end class Lines2Test
```



b) Modify your answer in part (a) to mirror the design in all four corners, as shown in the right screen capture of Fig. 4.2.

ANS:

```
// GCS Exercise 4.2 Part B: Lines2.java
// Draws lines using a loop
import java.awt.Graphics;
import javax.swing.JPanel;

```

```
6  public class Lines2 extends JPanel
7  {
8     // draws lines from one edge to another
9     public void paintComponent( Graphics g )
10    {
11       super.paintComponent( g );
12
13       int increments = 15; // number of increments each side is divided
14
15       int width = getWidth(); // total width
16       int height = getHeight(); // total height
17
18       int widthStep = width / increments; // width increment
19       int heightStep = height / increments; // height increment
20
21       int count = 0; // loop counter
22
23       // draws lines in a pattern between adjacent edges
24       while ( count < increments )
25       {
26          // left to bottom
27          g.drawLine( 0, count * heightStep,
28             ( count + 1 ) * widthStep, height );
29
30          // right to top
31          g.drawLine( width, count * heightStep,
32             width - ( count + 1 ) * widthStep, height );
33
34          // right to bottom
35          g.drawLine( width, height - count * heightStep,
36             width - ( count + 1 ) * widthStep, 0 );
37
38          // left to top
39          g.drawLine( 0, height - count * heightStep,
40             ( count + 1 ) * widthStep, 0 );
41
42          count++;
43       } // end while
44    } // end method paintComponent
45 } // end class Lines2
```

```
1  // GCS Exercise 4.2 Part B: Lines2Test.java
2  // Application to display class Lines1
3  import javax.swing.JFrame;
4
5  public class Lines2Test
6  {
7     public static void main( String args[] )
8     {
9        Lines2 panel = new Lines2(); // create the panel with the drawing
10       JFrame application = new JFrame(); // create a new frame
11
12       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

```
13         application.add( panel ); // add the panel to the frame
14         application.setSize( 300, 300 ); // set the size
15         application.setVisible( true ); // show the frame
16      } // end main
17   } // end class Lines2Test
```