6

Methods: A Deeper Look

The greatest invention of the nineteenth century was the invention of the method of invention.
—Alfred North Whitehead

Call me Ishmael.
—Herman Melville

When you call me that, smile!
—Owen Wister

Answer me in one word.
—William Shakespeare

O! call back yesterday, bid time return.
—William Shakespeare

There is a point at which methods devour themselves.
—Frantz Fanon

OBJECTIVES

In this chapter you will learn:

- How `static` methods and fields are associated with an entire class rather than specific instances of the class.
- To use common `Math` methods available in the Java API.
- To understand the mechanisms for passing information between methods.
- How the method call/return mechanism is supported by the method-call stack and activation records.
- How packages group related classes.
- How to use random-number generation to implement game-playing applications.
- How the visibility of declarations is limited to specific regions of programs.
- What method overloading is and how to create overloaded methods.

Student Solution Exercises

6.7 What is the value of x after each of the following statements is executed?

a) `x = Math.abs( 7.5 );`
ANS: 7.5
b) `x = Math.floor( 7.5 );`
ANS: 7.0
c) `x = Math.abs( 0.0 );`
ANS: 0.0
d) `x = Math.ceil( 0.0 );`
ANS: 0.0
e) `x = Math.abs( -6.4 );`
ANS: 6.4
f) `x = Math.ceil( -6.4 );`
ANS: -6.0
g) `x = Math.ceil( -Math.abs( -8 + Math.floor( -5.5 ) ) );`
ANS: -14.0

6.9 An application of method `Math.floor` is rounding a value to the nearest integer. The statement

```
y = Math.floor( x + 0.5 );
```

will round the number x to the nearest integer and assign the result to y. Write an application that reads `double` values and uses the preceding statement to round each of the numbers to the nearest integer. For each number processed, display both the original number and the rounded number.
ANS:

```
// Exercise 6.9 Solution: RoundingTest.java
// Program tests Math.floor.
import java.util.Scanner;

public class RoundingTest
{
   public static void main( String args[] )
   {
      Scanner input = new Scanner( System.in );

      System.out.printf( "%s\n%s\n   %s\n   %s\n",
         "Enter decimal numbers.",
         "Type the end-of-file indicator to terminate input:",
         "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
         "On Windows type <ctrl> z then press Enter" );

      while ( input.hasNext() )
      {
         double x = input.nextDouble();

         System.out.printf( "Number: %f\tMath.floor( x + .5 ): %f\n",
            x, Math.floor( x + .5 ) );
      } // end while loop
   } // end method main
} // end class RoundingTest
```

```
Enter decimal numbers.
Type the end-of-file indicator to terminate input:
   On UNIX/Linux/Mac OS X type <ctrl> d then press Enter
   On Windows type <ctrl> z then press Enter
5.49
Number: 5.490000        Math.floor( x + .5 ): 5.000000
6.28
Number: 6.280000        Math.floor( x + .5 ): 6.000000
6.86
Number: 6.860000        Math.floor( x + .5 ): 7.000000
^Z
```

6.11 Answer each of the following questions:

a) What does it mean to choose numbers "at random?"

ANS: Every number has an equal chance of being chosen at any time.

b) Why is the `nextInt` method of class `Random` useful for simulating games of chance?

ANS: Because it produces a series of random numbers.

c) Why is it often necessary to scale or shift the values produced by a `Random` object?

ANS: To produce random numbers in a specific range.

d) Why is computerized simulation of real-world situations a useful technique?

ANS: It enables more accurate predictions of random events, such as cars arriving at toll booths and people arriving in lines at a supermarket. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at specified times.

6.12 Write statements that assign random integers to the variable *n* in the following ranges:

a) $1 \leq n \leq 2$

ANS: `n = 1 + randomNumbers.nextInt( 2 );`

b) $1 \leq n \leq 100$

ANS: `n = 1 + randomNumbers.nextInt( 100 );`

c) $0 \leq n \leq 9$

ANS: `n = randomNumbers.nextInt( 10 );`

d) $1000 \leq n \leq 1112$

ANS: `n = 1000 + randomNumbers.nextInt( 113 );`

e) $-1 \leq n \leq 1$

ANS: `n = -1 + randomNumbers.nextInt( 3 );`

f) $-3 \leq n \leq 11$

ANS: `n = -3 + randomNumbers.nextInt( 15 );`

```java
// Exercise 6.12 Solution: RandomRange.java
import java.util.Random;

public class RandomRange
{
   public static void main( String args[] )
   {
      Random randomNumbers = new Random();

      // a)
      System.out.println( 1 + randomNumbers.nextInt( 2 ) );

      // b)
      System.out.println( 1 + randomNumbers.nextInt( 100 ) );

      // c)
      System.out.println( randomNumbers.nextInt( 10 ) );

      // d)
      System.out.println( 1000 + randomNumbers.nextInt( 113 ) );

      // e)
      System.out.println( -1 + randomNumbers.nextInt( 3 ) );

      // f)
      System.out.println( -3 + randomNumbers.nextInt( 15 ) );
   } // end main
} // end class RandomRange
```

```
2
18
0
1061
1
7
```

```
2
63
5
1071
1
-2
```

6.15 Define a method `hypotenuse` that calculates the length of the hypotenuse of a right triangle when the lengths of the other two sides are given. (Use the sample data in Fig. 6.26.) The method should take two arguments of type `double` and return the hypotenuse as a `double`. Incorporate this method into an application that reads values for `side1` and `side2` and performs the calculation with the `hypotenuse` method. Determine the length of the hypotenuse for each of the triangles in Fig. 6.26.

Triangle	Side 1	Side 2
1	3.0	4.0
2	5.0	12.0
3	8.0	15.0

Fig. 6.26 | Values for the sides of triangles in Exercise 6.15.

ANS:

```java
// Exercise 6.15 Solution: Triangle.java
// Program calculates the hypotenuse of a right triangle.
import java.util.Scanner;

public class Triangle
{
   // reads in two sides and prints the hypotenuse
   public void calculateHypotenuse()
   {
      Scanner input = new Scanner( System.in );

      double side1; // first side of triangle
      double side2; // second side of triangle

      System.out.print( "Enter side 1 (negative to quit): " );
      side1 = input.nextDouble();

      while ( side1 > 0 )
      {
         System.out.print( "Enter side 2: " );
         side2 = input.nextDouble();

         System.out.printf( "Hypotenuse is: %f\n",
            hypotenuse( side1, side2 ) );

         System.out.print( "Enter side 1 (negative to quit): " );
         side1 = input.nextDouble();
      } // end while
   } // end method calculateHypotenuse

   // calculate hypotenuse given lengths of two sides
   public double hypotenuse( double side1, double side2 )
   {
      double hypotenuseSquared = Math.pow( side1, 2 ) +
         Math.pow( side2, 2 );

      return Math.sqrt( hypotenuseSquared );
   } // end method hypotenuse
 } // end class Triangle
```

```java
// Exercise 6.15 Solution: TriangleTest.java
// Test application for class Triangle
public class TriangleTest
{
   public static void main( String args[] )
   {
      Triangle application = new Triangle();
      application.calculateHypotenuse();
   } // end main
} // end class TriangleTest
```

```
Enter side 1 (negative to quit): 8
Enter side 2: 15
Hypotenuse is: 17.000000
Enter side 1 (negative to quit): 5
Enter side 2: 12
Hypotenuse is: 13.000000
Enter side 1 (negative to quit): 3
Enter side 2: 4
Hypotenuse is: 5.000000
Enter side 1 (negative to quit): -1
```

6.17 Write a method `isEven` that uses the remainder operator (%) to determine whether an integer is even. The method should take an integer argument and return `true` if the integer is even and `false` otherwise. Incorporate this method into an application that inputs a sequence of integers (one at a time) and determines whether each is even or odd.

ANS:

```java
// Exercise 6.17 Solution: EvenOdd.java
// Program determines if a number is odd or even.
import java.util.Scanner;

public class EvenOdd
{
   // determines whether numbers are even or odd
   public void checkEvenOdd()
   {
      Scanner input = new Scanner( System.in );

      System.out.printf( "%s\n%s\n   %s\n   %s\n",
         "Enter numbers to determine if they are even or odd.",
         "Type the end-of-file indicator to terminate input:",
         "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
         "On Windows type <ctrl> z then press Enter" );

      while ( input.hasNext() )
      {
         int number = input.nextInt();

         if ( isEven( number ) )
            System.out.printf( "%d is even\n", number );
         else
            System.out.printf( "%d is odd\n", number );
      } // end while loop
   } // end method checkEvenOdd

   // return true if number is even
   public boolean isEven( int number )
   {
      return number % 2 == 0;
   } // end method isEven
} // end class EvenOdd
```

```java
// Exercise 6.17 Solution: EvenOddTest.java
// Test application for class EvenOdd
public class EvenOddTest
{
   public static void main( String args[] )
   {
      EvenOdd application = new EvenOdd();
      application.checkEvenOdd();
   } // end main
} // end class EvenOddTest
```

```
Enter numbers to determine if they are even or odd.
Type the end-of-file indicator to terminate input:
   On UNIX/Linux/Mac OS X type <ctrl> d then press Enter
   On Windows type <ctrl> z then press Enter
11
11 is odd
6
6 is even
13
13 is odd
5
5 is odd
^Z
```

6.18 Write a method `squareOfAsterisks` that displays a solid square (the same number of rows and columns) of asterisks whose side is specified in integer parameter `side`. For example, if `side` is `4`, the method should display

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

Incorporate this method into an application that reads an integer value for `side` from the user and outputs the asterisks with the `squareOfAsterisks` method.

ANS:

```java
// Exercise 6.18 Solution: Square.java
// Program draws a square of asterisks.
import java.util.Scanner;

public class Square
{
   // obtain value from user
   public void drawSquare()
   {
      Scanner input = new Scanner( System.in );

      System.out.print( "Enter square size: " );
      int size = input.nextInt();

      squareOfAsterisks( size );
   } // end method drawSquare

   // draw a square of asteriks
   public void squareOfAsterisks( int side )
   {
      for ( int count = 1; count <= side * side; count++ ) {
         System.out.print( "*" );

         if ( count % side == 0 )
            System.out.println();
      } // end for loop
   }  // end method squareOfAsterisks
} // end class Square
```

```java
// Exercise 6.18 Solution: SquareTest.java
// Test application for class Square
public class SquareTest
{
   public static void main( String args[] )
   {
      Square application = new Square();
      application.drawSquare();
   } // end main
} // end class SquareTest
```

```
Enter square size: 8
********
********
********
********
********
********
********
********
```

```
Enter square size: 12
************
************
************
************
************
************
************
************
************
************
************
************
```

6.21 Write program segments that accomplish each of the following tasks:

a) Calculate the integer part of the quotient when integer `a` is divided by integer `b`.

b) Calculate the integer remainder when integer `a` is divided by integer `b`.

c) Use the program pieces developed in parts (a) and (b) to write a method `displayDigits` that receives an integer between `1` and `99999` and displays it as a sequence of digits, separating each pair of digits by two spaces. For example, the integer `4562` should appear as

```
4  5  6  2
```

d) Incorporate the method developed in part (c) into an application that inputs an integer and calls `displayDigits` by passing the method the integer entered. Display the results.

ANS:

```java
// Exercise 6.21 Solution: Digits.java
// Program separates a five-digit number
// into its individual digits.
import java.util.Scanner;

public class Digits
{
   // displays the individual digits of a number
   public void separateDigits()
   {
      Scanner input = new Scanner( System.in );

      System.out.print( "Enter the integer (0 to exit): " );
      int number = input.nextInt();

      while ( number != 0 )
      {
         if ( number <= 99999 && number >= 1 )
           displayDigits( number );
         else
            System.out.println( "number must be between 1 and 99999" );

         System.out.print( "Enter the integer (0 to exit): " );
         number = input.nextInt();
      } // end while loop
   } // end method separateDigits

   // part A
   public int quotient( int a, int b )
   {
      return a / b;
   } // end method quotient

   // part B
   public int remainder( int a, int b )
   {
      return a % b;
   } // end method remainder

   // part C
   public void displayDigits( int number )
   {
      int divisor = 1, digit;
      String result = "";

      // Loop for highest divisor
      for ( int i = 1; i < number; i *= 10 )
         divisor = i;

      while ( divisor >= 1 )
      {
         digit = quotient( number, divisor );

         result += digit + "  ";

         number = remainder( number, divisor );
         divisor = quotient( divisor, 10 );
      } // end while loop

      System.out.println( result );
   } // end method displayDigits
} // end class Digits
```

```java
// Exercise 6.21 Solution: DigitsTest.java
// Test application for class Digits
public class DigitsTest
{
   public static void main( String args[] )
   {
      Digits application = new Digits();
      application.separateDigits();
   } // end main
} // end class DigitsTest
```

```
Enter the integer (0 to exit): 60450
6  0  4  5  0
Enter the integer (0 to exit): 0
```

6.22 Implement the following integer methods:

a) Method `celsius` returns the Celsius equivalent of a Fahrenheit temperature, using the calculation

```
celsius = 5.0 / 9.0 * ( fahrenheit - 32 );
```

b) Method `fahrenheit` returns the Fahrenheit equivalent of a Celsius temperature, using the calculation

```
fahrenheit = 9.0 / 5.0 * celsius + 32;
```

c) Use the methods from parts (a) and (b) to write an application that enables the user either to enter a Fahrenheit temperature and display the Celsius equivalent or to enter a Celsius temperature and display the Fahrenheit equivalent.

ANS:

```java
// Exercise 6.22 Solution: Convert.java
// Program converts Fahrenheit to Celsius and vice versa.
import java.util.Scanner;

public class Convert
{
   // convert temperatures
   public void convertTemperatures()
   {
      Scanner input = new Scanner( System.in );

      int choice; // the user's choice in the menu

      do
      {
         // print the menu
         System.out.println( "1. Fahrenheit to Celsius" );
         System.out.println( "2. Celsius to Fahrenheit" );
         System.out.println( "3. Exit" );
         System.out.print( "Choice: " );
         choice = input.nextInt();

         if ( choice != 3 )
         {
            System.out.print( "Enter temperature: " );
            int oldTemperature = input.nextInt();

            // convert the temperature appropriately
            switch ( choice )
            {
               case 1:
                  System.out.printf( "%d Fahrenheit is %d Celsius\n",
                     oldTemperature, celsius( oldTemperature ) );
                  break;

               case 2:
                  System.out.printf( "%d Celsius is %d Fahrenheit\n",
                    oldTemperature, fahrenheit( oldTemperature ) );
                  break;
            } // end switch
         } // end if
      } while ( choice != 3 );
   } // end method convertTemperatures

   // return Celsius equivalent of Fahrenheit temperature
   public int celsius( int fahrenheitTemperature )
   {
      return ( (int) ( 5.0 / 9.0 * ( fahrenheitTemperature - 32 ) ) );
   } // end method celsius

   // return Fahrenheit equivalent of Celsius temperature
   public int fahrenheit( int celsiusTemperature )
   {
      return ( (int) ( 9.0 / 5.0 * celsiusTemperature + 32 ) );
   } // end method fahrenheit
} // end class Convert
```

```
// Exercise 6.22 Solution: ConvertTest.java
// Test application for class Convert
public class ConvertTest
{
   public static void main( String args[] )
   {
      Convert application = new Convert();
      application.convertTemperatures();
   } // end main
} // end class ConvertTest
```

```
1. Fahrenheit to Celsius
2. Celsius to Fahrenheit
3. Exit
Choice: 1
Enter temperature: 32
32 Fahrenheit is 0 Celsius
1. Fahrenheit to Celsius
2. Celsius to Fahrenheit
3. Exit
Choice: 2
Enter temperature: 100
100 Celsius is 212 Fahrenheit
1. Fahrenheit to Celsius
2. Celsius to Fahrenheit
3. Exit
Choice: 3
```

6.24 An integer number is said to be a *perfect number* if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number, because 6 = 1 + 2 + 3. Write a method `perfect` that determines whether parameter `number` is a perfect number. Use this method in an application that determines and displays all the perfect numbers between 1 and 1000. Display the factors of each perfect number to confirm that the number is indeed perfect. Challenge the computing power of your computer by testing numbers much larger than 1000. Display the results.

ANS:

```
// Exercise 6.24 Solution: PerfectNumber.java
// Program displays all perfect numbers between 1 and 1000.

public class PerfectNumber
{
   // finds all the perfect numbers from 2 to 1000
   public void findPerfects()
   {
      for ( int number = 2; number <= 1000; number++ )
      {
         String result = perfect( number );

         if ( result != null )
            System.out.printf ( "%d is perfect.\n\tFactors: %s\n",
               number, result );
      } // end for
   } // end main

   // returns a string of factors if parameter is a
   // perfect number, or null if it isn't.
   public String perfect( int value )
   {
      int factorSum = 1;
      String factors = "1 ";

      for ( int test = 2; test <= value / 2; test++ )
      {
         if ( value % test == 0 )
         {
            factorSum += test;
            factors += test + " ";
         } // end if
      } // end for

      if ( factorSum == value )
         return factors;

      return null;
   } // end method perfect
} // end class PerfectNumber
```

```
// Exercise 6.24 Solution: PerfectNumberTest.java
// Test application for class PerfectNumber
public class PerfectNumberTest
{
   public static void main( String args[] )
   {
      PerfectNumber application = new PerfectNumber();
      application.findPerfects();
   } // end main
} // end class PerfectNumberTest
```

```
6 is perfect.
        Factors: 1 2 3
28 is perfect.
        Factors: 1 2 4 7 14
496 is perfect.
        Factors: 1 2 4 8 16 31 62 124 248
```

6.26 Write a method that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the method should return 1367. Incorporate the method into an application that reads a value from the user and displays the result.

ANS:

```java
// Exercise 6.26 Solution: Reverse.java
// Program takes a number and prints it out
// with its digits reversed.
import java.util.Scanner;

public class Reverse
{
   // reverses an Integer
   public void reverseInteger()
   {
      Scanner input = new Scanner( System.in );

      System.out.print( "Enter an integer (-1 to exit): " );
      int number = input.nextInt();

      while ( number != -1 )
      {
         System.out.printf( "%d reversed is %d\n",
         number, reverseDigits( number ) );

         System.out.print( "Enter an integer (-1 to exit): " );
         number = input.nextInt();
      } // end while loop
   } // end method reverseInteger

   // print parameter number with digits reversed
   public int reverseDigits( int number )
   {
      int reverseNumber = 0; // the number in reverse order
      int placeValue; // the value at the current place

      while ( number > 0 )
      {
         placeValue = number % 10;
         number = number / 10;
         reverseNumber = reverseNumber * 10 + placeValue;
      } // end while loop

      return reverseNumber;
   } // end method reverseDigits
} // end class Reverse
```

```java
// Exercise 6.26 Solution: ReverseTest.java
// Test application for class Reverse
public class ReverseTest
{
   public static void main( String args[] )
   {
      Reverse application = new Reverse();
      application.reverseInteger();
   } // end main
} // end class ReverseTest
```

```
Enter an integer (-1 to exit): 54321
54321 reversed is 12345
Enter an integer (-1 to exit): -1
```

6.28 Write a method `qualityPoints` that inputs a student's average and returns `4` if the student's average is 90–100, `3` if the average is 80–89, `2` if the average is 70–79, `1` if the average is 60–69 and `0` if the average is lower than 60. Incorporate the method into an application that reads a value from the user and displays the result.

ANS:

```java
// Exercise 6.28 Solution: Average.java
// Program displays a number
// representing the student's average.
import java.util.Scanner;

public class Average
{
   public void computePoints()
   {
      Scanner input = new Scanner( System.in );

      System.out.print( "Enter average (-1 to quit): " );
      int inputNumber = input.nextInt();

      while ( inputNumber != -1 )
      {
         if ( inputNumber >= 0 && inputNumber <= 100 )
            System.out.printf( "Point is: %d\n",
               qualityPoints( inputNumber ) );
         else
            System.out.println( "Invalid input." );

         System.out.print( "Enter average (-1 to quit): " );
         inputNumber = input.nextInt();
      } // end while loop
   } // end method actionPerformed

   // return single-digit value of grade
   public int qualityPoints( int grade )
   {
      if ( grade >= 90 )
         return 4;
      else if ( grade >= 80 )
         return 3;
      else if ( grade >= 70 )
         return 2;
      else if ( grade >= 60 )
         return 1;
      else
         return 0;
   } // end method qualityPoints
} // end class Average
```

```java
// Exercise 6.28 Solution: AverageTest.java
// Test application for class Average
public class AverageTest
{
   public static void main( String args[] )
   {
      Average application = new Average();
      application.computePoints();
   } // end main
} // end class AverageTest
```

```
Enter average (-1 to quit): 90
Point is: 4
Enter average (-1 to quit): 80
Point is: 3
Enter average (-1 to quit): -1
```

6.29 Write an application that simulates coin tossing. Let the program toss a coin each time the user chooses the "`Toss Coin`" menu option. Count the number of times each side of the coin appears. Display the results. The program should call a separate method `flip` that takes no arguments and returns `false` for tails and `true` for heads. [*Note*: If the program realistically simulates coin tossing, each side of the coin should appear approximately half the time.]

ANS:

```java
// Exercise 6.29 Solution: Coin.java
// Program simulates tossing a coin.
import java.util.*;

public class Coin
{
   private Random randomNumbers = new Random();

   // flips a coin many times
   public void flipCoins()
   {
      Scanner input = new Scanner( System.in );

      int heads = 0; // the number of times heads shows up
      int tails = 0; // the number of times tails shows up
      int choice; // the user's choice

      do
      {
         // display a menu
         System.out.println( "1. Toss Coin" );
         System.out.println( "2. Exit" );
         System.out.print( "Choice: " );
         choice = input.nextInt();

```

```
            if ( choice == 1 )
            {
               if ( flip() )
                  heads++;
               else
                  tails++;

               System.out.printf( "Heads: %d, Tails: %d\n", heads, tails );
            } // end if

         } while ( choice != 2 );
      } // end method flipCoins

      // simulate flipping
      public boolean flip()
      {
         return randomNumbers.nextInt( 2 ) == 1;
      } // end method flip
   } // end class Coin
```

```
// Exercise 6.29 Solution: CoinTest.java
// Test application for class Coin
public class CoinTest
{
   public static void main( String args[] )
   {
      Coin application = new Coin();
      application.flipCoins();
   } // end main
} // end class CoinTest
```

```
1. Toss Coin
2. Exit
Choice: 1
Heads: 0, Tails: 1
1. Toss Coin
2. Exit
Choice: 1
Heads: 0, Tails: 2

.
.
.

1. Toss Coin
2. Exit
Choice: 1
Heads: 23, Tails: 24
1. Toss Coin
2. Exit
Choice: 1
Heads: 24, Tails: 24
1. Toss Coin
2. Exit
Choice: 2
```

6.36 Write method `distance` to calculate the distance between two points (*x1*, *y1*) and (*x2*, *y2*). All numbers and return values should be of type `double`. Incorporate this method into an application that enables the user to enter the coordinates of the points.

ANS:

```java
// Exercise 6.36 Solution: Points.java
// Program calculates the distance between two points.
import java.util.Scanner;

public class Points
{
   // calculates the distance between two points
   public void calculateDistance()
   {
      Scanner input = new Scanner( System.in );

      System.out.printf( "%s\n   %s\n   %s\n",
         "Type the end-of-file indicator to terminate",
         "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
         "On Windows type <ctrl> z then press Enter" );
      System.out.print( "Or Enter X1: " );

      // continually read in inputs until the user terminates
      while ( input.hasNext() )
      {
         double x1 = input.nextDouble();
         System.out.print( "Enter Y1: " );
         double y1 = input.nextDouble();
         System.out.print( "Enter X2: " );
```

```
            double x2 = input.nextDouble();
            System.out.print( "Enter Y2: " );
            double y2 = input.nextDouble();

            double distance = distance( x1, y1, x2, y2 );
            System.out.printf( "Distance is %f\n\n", distance );

            System.out.printf( "%s\n   %s\n   %s\n",
               "Type the end-of-file indicator to terminate",
               "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
               "On Windows type <ctrl> z then press Enter" );
            System.out.print( "Or Enter X1: " );
         } // end while
      } // end method calculateDistance

      // calculate distance between two points
      public double distance( double x1, double y1, double x2, double y2 )
      {
         return Math.sqrt( Math.pow( ( x1 - x2 ), 2 ) +
            Math.pow( ( y1 - y2 ), 2 ) );
      } // end method distance
   } // end class Points
```

```
// Exercise 6.36 Solution: PointsTest.java
// Test application for class Points
public class PointsTest
{
   public static void main( String args[] )
   {
      Points application = new Points();
      application.calculateDistance();
   } // end main
} // end class PointsTest
```

```
Type the end-of-file indicator to terminate
   On UNIX/Linux/Mac OS X type <ctrl> d then press Enter
   On Windows type <ctrl> z then press Enter
Or Enter X1: 1
Enter Y1: 1
Enter X2: 4
Enter Y2: 5
Distance is 5.000000

Type the end-of-file indicator to terminate
   On UNIX/Linux/Mac OS X type <ctrl> d then press Enter
   On Windows type <ctrl> z then press Enter
Or Enter X1: ^Z
```