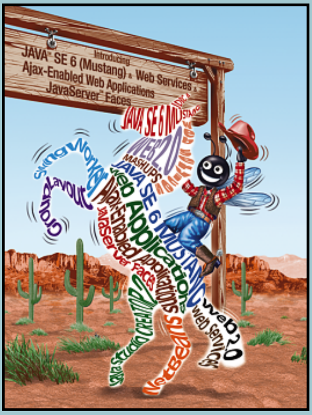
8

Classes and Objects: A Deeper Look


Introducing
JAVA SE 6 (Mustang) • Web Services •
Ajax-Enabled Web Applications •
JavaServer Faces

Instead of this absurd division into sexes, they ought to class people as static and dynamic.

—Evelyn Waugh

Is it a world to hide virtues in?

—William Shakespeare

But what, to serve
our private ends,
Forbids the cheating
of our friends?

—Charles Churchill

This above all: to thine own self be true.

—William Shakespeare

Don't be "consistent," but be simply true.

—Oliver Wendell Holmes, Jr.

OBJECTIVES

In this chapter you will learn:

- Encapsulation and data hiding.
- The notions of data abstraction and abstract data types (ADTs).
- To use keyword `this`.
- To use `static` variables and methods.
- To import `static` members of a class.
- To use the **`enum`** type to create sets of constants with unique identifiers.
- To declare **`enum`** constants with parameters.
- To organize classes in packages to promote reuse.

Student Solution Exercises

8.3 What happens when a return type, even void, is specified for a constructor?

ANS: It is treated as a method and is not considered to be a constructor.

8.4 *(Rectangle Class)* Create a class Rectangle. The class has attributes length and width, each of which defaults to 1. It has methods that calculate the perimeter and the area of the rectangle. It has *set* and *get* methods for both length and width. The *set* methods should verify that length and width are each floating-point numbers larger than 0.0 and less than 20.0. Write a program to test class Rectangle.

ANS:

```java
// Exercise 8.4 Solution: Rectangle.java
// Definition of class Rectangle

public class Rectangle
{
   private double length; // the length of the rectangle
   private double width; // the width of the rectangle

   // constructor without parameters
   public Rectangle()
   {
      setLength( 1.0 );
      setWidth( 1.0 );
   } // end Rectangle no-argument constructor

   // constructor with length and width supplied
   public Rectangle( double theLength, double theWidth )
   {
      setLength( theLength );
      setWidth( theWidth );
   } // end Rectangle two-argument constructor

   // validate and set length
   public void setLength( double theLength )
   {
      length = ( theLength > 0.0 && theLength < 20.0 ? theLength : 1.0 );
   } // end method setLength

   // validate and set width
   public void setWidth( double theWidth )
   {
      width = ( theWidth > 0 && theWidth < 20.0 ? theWidth : 1.0 );
   } // end method setWidth

   // get value of length
   public double getLength()
   {
      return length;
   } // end method getLength

   // get value of width
   public double getWidth()
   {
      return width;
   } // end method getWidth

   // calculate rectangle's perimeter
   public double perimeter()
   {
      return 2 * length + 2 * width;
   } // end method perimeter

   // calculate rectangle's area
   public double area()
   {
      return length * width;
   } // end method area

   // convert to String
   public String toString()
   {
      return String.format( "%s: %f\n%s: %f\n%s: %f\n%s: %f",
         "Length", length, "Width", width,
         "Perimeter", perimeter(), "Area", area() );
   } // end method toRectangleString
} // end class Rectangle
```

```java
// Exercise 8.4 Solution: RectangleTest.java
// Program tests class Rectangle.
import java.util.Scanner;

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

      Rectangle rectangle = new Rectangle();

      int choice = getMenuChoice();

      while ( choice != 3 )
      {
         switch ( choice )
         {
            case 1:
               System.out.print( "Enter length: " );
               rectangle.setLength( input.nextDouble() );
               break;

            case 2:
               System.out.print ( "Enter width: " );
               rectangle.setWidth( input.nextDouble() );
               break;
         } // end switch

         System.out.println ( rectangle.toString() );

         choice = getMenuChoice();
      } // end while
   } // end main

   // prints a menu and returns a value coressponding to the menu choice
   private static int getMenuChoice()
   {
      Scanner input = new Scanner( System.in );

      System.out.println( "1. Set Length" );
      System.out.println( "2. Set Width" );
      System.out.println( "3. Exit" );
      System.out.print( "Choice: " );

      return input.nextInt();
   } // end method getMenuChoice
} // end class RectangleTest
```

```
1. Set Length
2. Set Width
3. Exit
Choice: 1
Enter length: 10
Length: 10.000000
Width: 1.000000
Perimeter: 22.000000
Area: 10.000000
1. Set Length
2. Set Width
3. Exit
Choice: 2
Enter width: 15
Length: 10.000000
Width: 15.000000
Perimeter: 50.000000
Area: 150.000000
1. Set Length
2. Set Width
3. Exit
Choice: 1
Enter length: 99
Length: 1.000000
Width: 15.000000
Perimeter: 32.000000
Area: 15.000000
1. Set Length
2. Set Width
3. Exit
Choice: 3
```

8.6 *(Savings Account Class)* Create class `SavingsAccount`. Use a `static` variable `annualInterestRate` to store the annual interest rate for all account holders. Each object of the class contains a `private` instance variable `savingsBalance` indicating the amount the saver currently has on deposit. Provide method `calculateMonthlyInterest` to calculate the monthly interest by multiplying the `savingsBalance` by `annualInterestRate` divided by 12—this interest should be added to `savingsBalance`. Provide a `static` method `modifyInterestRate` that sets the `annualInterestRate` to a new value. Write a program to test class `SavingsAccount`. Instantiate two `savingsAccount` objects, `saver1` and `saver2`, with balances of $2000.00 and $3000.00, respectively. Set `annualInterestRate` to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the `annualInterestRate` to 5%, calculate the next month's interest and print the new balances for both savers.

ANS:

```java
// Exercise 8.6 Solution: SavingAccount
// SavingAccount class definition

public class SavingAccount
{
   // interest rate for all accounts
   private static double annualInterestRate = 0;

   private double savingsBalance; // balance for currrent account

   // constructor, creates a new account with the specified balance
   public SavingAccount( double balance )
   {
      savingsBalance = balance;
   } // end constructor

   // get monthly interest
   public void calculateMonthlyInterest()
   {
      savingsBalance += savingsBalance * ( annualInterestRate / 12.0 );
   } // end method calculateMonthlyInterest

   // modify interest rate
   public static void modifyInterestRate( double newRate )
   {
      annualInterestRate =
         ( newRate >= 0 && newRate <= 1.0 ) ? newRate : 0.04;
   } // end method modifyInterestRate

   // get string representation of SavingAccount
   public String toString()
   {
      return String.format( "$%.2f", savingsBalance );
   } // end method toSavingAccountString
} // end class SavingAccount
```

```java
// Exercise 8.6 Solution: SavingAccountTest.java
// Program that tests SavingAccount class

public class SavingAccountTest
{
   public static void main( String args[] )
   {
      SavingAccount saver1 = new SavingAccount( 2000 );
      SavingAccount saver2 = new SavingAccount( 3000 );
      SavingAccount.modifyInterestRate( 0.04 );

      System.out.println( "Monthly balances for one year at .04" );
      System.out.println( "Balances:" );

      System.out.printf( "%20s%10s\n", "Saver 1", "Saver 2" );
      System.out.printf( "%-10s%10s%10s\n", "Base",
         saver1.toString(), saver2.toString() );

      for ( int month = 1; month <= 12; month++ )
      {
         String monthLabel = String.format( "Month %d:", month );
         saver1.calculateMonthlyInterest();
         saver2.calculateMonthlyInterest();

         System.out.printf( "%-10s%10s%10s\n", monthLabel,
            saver1.toString(), saver2.toString() );
      } // end for

      SavingAccount.modifyInterestRate( .05 );
      saver1.calculateMonthlyInterest();
      saver2.calculateMonthlyInterest();

      System.out.println( "\nAfter setting interest rate to .05" );
      System.out.println( "Balances:" );
      System.out.printf( "%-10s%10s\n", "Saver 1", "Saver 2" );
      System.out.printf( "%-10s%10s\n",
         saver1.toString(),  saver2.toString() );
   } // end main
} // end class SavingAccountTest
```

```
Monthly balances for one year at .04
Balances:
             Saver 1   Saver 2
Base        $2000.00  $3000.00
Month 1:    $2006.67  $3010.00
Month 2:    $2013.36  $3020.03
Month 3:    $2020.07  $3030.10
Month 4:    $2026.80  $3040.20
Month 5:    $2033.56  $3050.33
Month 6:    $2040.33  $3060.50
Month 7:    $2047.14  $3070.70
Month 8:    $2053.96  $3080.94
Month 9:    $2060.81  $3091.21
Month 10:   $2067.68  $3101.51
Month 11:   $2074.57  $3111.85
Month 12:   $2081.48  $3122.22

After setting interest rate to .05
Balances:
Saver 1      Saver 2
$2090.16    $3135.23
```

8.8 *(Enhancing Class Date)* Modify class `Date` of Fig. 8.7 to perform error checking on the initializer values for instance variables `month`, `day` and `year` (currently it validates only the month and day). Provide a method `nextDay` to increment the day by one. The `Date` object should always remain in a consistent state. Write a program that tests the `nextDay` method in a loop that prints the date during each iteration of the loop to illustrate that the `nextDay` method works correctly. Test the following cases:

a) incrementing into the next month and
b) incrementing into the next year.

ANS:

```java
// Exercise 8.8 Solution: Date.java
// Date class declaration.

public class Date
{
   private int month; // 1-12
   private int day;   // 1-31 based on month
   private int year;  // > 0

   // constructor: call checkMonth to confirm proper value for month;
   // call checkDay to confirm proper value for day
   public Date( int theMonth, int theDay, int theYear )
   {
      month = checkMonth( theMonth ); // validate month
      year = checkYear( theYear ); // validate year
      day = checkDay( theDay ); // validate day

      System.out.printf(
         "Date object constructor for date %s\n", toString() );
   } // end Date constructor

   // utility method to confirm proper year value
   private int checkYear( int testYear )
   {
      if ( testYear > 0 ) // validate year
         return testYear;
      else // day is invalid
      {
         System.out.printf(
            "Invalid year (%d) set to 1.\n", testYear );
         return 1;
      } // end else
   } // end method checkYear

   // utility method to confirm proper month value
   private int checkMonth( int testMonth )
   {
      if ( testMonth > 0 && testMonth <= 12 ) // validate month
         return testMonth;
      else // month is invalid
      {
         System.out.printf(
            "Invalid month (%d) set to 1.\n", testMonth );
         return 1; // maintain object in consistent state
      } // end else
   } // end method checkMonth

   // utility method to confirm proper day value based on month and year
   private int checkDay( int testDay )
   {
      int daysPerMonth[] =
         { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

      // check if day in range for month
      if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
         return testDay;

      // check for leap year
      if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
           ( year % 4 == 0 && year % 100 != 0 ) ) )
         return testDay;

      System.out.printf( "Invalid day (%d) set to 1.\n", testDay );

      return 1; // maintain object in consistent state
   } // end method checkDay

   // increment the day and check if doing so will change the month
   public void nextDay()
   {
      int testDay = day + 1;

      if ( checkDay( testDay ) == testDay )
         day = testDay;
      else
      {
         day = 1;
         nextMonth();
      } // end else
   } // end method nextDay

   // increment the month and check if doing so will change the year
   public void nextMonth()
   {
      if ( 12 == month )
        year++;

      month = month % 12 + 1;
   } // end method nextMonth

   // return a String of the form month/day/year
   public String toString()
   {
      return String.format( "%d/%d/%d", month, day, year );
   } // end method toDateString
} // end class Date
```

```
// Exercise 8.8 Solution: DateTest
// Program tests Date class.

public class DateTest
{
   // method main begins execution of Java application
   public static void main( String args[] )
   {
      System.out.println( "Checking increment" );
      Date testDate = new Date( 11, 27, 1988 );

      // test incrementing of day, month and year
      for ( int counter = 0; counter < 40; counter++ )
      {
         testDate.nextDay();
         System.out.printf( "Incremented Date: %s\n",
            testDate.toString() );
      } // end for
   } // end main
} // end class DateTest
```

```
Checking increment
Date object constructor for date 11/27/1988
Incremented Date: 11/28/1988
Incremented Date: 11/29/1988
Incremented Date: 11/30/1988
Invalid day (31) set to 1.
Incremented Date: 12/1/1988
Incremented Date: 12/2/1988
Incremented Date: 12/3/1988
Incremented Date: 12/4/1988
Incremented Date: 12/5/1988
Incremented Date: 12/6/1988
Incremented Date: 12/7/1988
Incremented Date: 12/8/1988
Incremented Date: 12/9/1988
Incremented Date: 12/10/1988
Incremented Date: 12/11/1988
Incremented Date: 12/12/1988
Incremented Date: 12/13/1988
Incremented Date: 12/14/1988
Incremented Date: 12/15/1988
Incremented Date: 12/16/1988
Incremented Date: 12/17/1988
Incremented Date: 12/18/1988
Incremented Date: 12/19/1988
Incremented Date: 12/20/1988
Incremented Date: 12/21/1988
Incremented Date: 12/22/1988
Incremented Date: 12/23/1988
Incremented Date: 12/24/1988
Incremented Date: 12/25/1988
Incremented Date: 12/26/1988
Incremented Date: 12/27/1988
Incremented Date: 12/28/1988
Incremented Date: 12/29/1988
Incremented Date: 12/30/1988
Incremented Date: 12/31/1988
Invalid day (32) set to 1.
Incremented Date: 1/1/1989
Incremented Date: 1/2/1989
Incremented Date: 1/3/1989
Incremented Date: 1/4/1989
Incremented Date: 1/5/1989
Incremented Date: 1/6/1989
```

8.9 *(Returning Error Indicators from Methods)* Modify the *set* methods in class `Time2` of Fig. 8.5 to return appropriate error values if an attempt is made to set one of the instance variables `hour`, `minute` or `second` of an object of class `Time` to an invalid value. [*Hint:* Use `boolean` return types on each method.] Write a program that tests these new *set* methods and outputs error messages when incorrect values are supplied.

ANS:

```java
// Exercise 8.9 Solution: Time2.java
// Time2 class definition with methods tick,
// incrementMinute and incrementHour.

public class Time2
{
   private int hour; // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // Time2 no-argument constructor: initializes each instance variable
   // to zero; ensures that Time2 objects start in a consistent state
   public Time2()
   {
      this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 no-argument constructor

   // Time2 constructor: hour supplied, minute and second defaulted to 0
   public Time2( int h )
   {
      this( h, 0, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 one-argument constructor

   // Time2 constructor: hour and minute supplied, second defaulted to 0
   public Time2( int h, int m )
   {
      this( h, m, 0 ); // invoke Time2 constructor with three arguments
   } // end Time2 two-argument constructor

   // Time2 constructor: hour, minute and second supplied
   public Time2( int h, int m, int s )
   {
      setTime( h, m, s ); // invoke setTime to validate time
   } // end Time2 three-argument constructor

   // Time2 constructor: another Time2 object supplied
   public Time2( Time2 time )
   {
      // invoke Time2 constructor with three arguments
      this( time.getHour(), time.getMinute(), time.getSecond() );
   } // end Time2 constructor with Time2 argument

   // Set Methods
   // set a new time value using universal time; perform
   // validity checks on data; set invalid values to zero
   public boolean setTime( int h, int m, int s )
   {
      boolean hourValid = setHour( h ); // set the hour
      boolean minuteValid = setMinute( m ); // set the minute
      boolean secondValid = setSecond( s ); // set the second

      return ( hourValid && minuteValid && secondValid );
   } // end method setTime

   // validate and set hour
   public boolean setHour( int h )
   {
       if ( h >= 0 && h < 24 )
       {
         hour = h;
         return true;
      } // end if
      else
      {
         hour = 0;
         return false;
      } // end else
   } // end method setHour

   // validate and set minute
   public boolean setMinute( int m )
   {
      if ( m >= 0 && m < 60 )
      {
         minute = m;
         return true;
      } // end if
      else
      {
         minute = 0;
         return false;
      } // end else
   } // end method setMinute

   // validate and set second
   public boolean setSecond( int s )
   {
      if ( s >= 0 && s < 60 )
      {
         second = s;
         return true;
      } // end if
      else
      {
         second = 0;
         return false;
      } // end else
   } // end method setSecond

   // Get Methods
   // get hour value
   public int getHour()
   {
      return hour;
   } // end method getHour

   // get minute value
   public int getMinute()
   {
      return minute;
   } // end method getMinute

   // get second value
   public int getSecond()
   {
      return second;
   } // end method getSecond

   // Tick the time by one second
   public void tick()
   {
      setSecond( second + 1 );

      if ( second == 0 )
         incrementMinute();
   } // end method tick

   // Increment the minute
   public void incrementMinute()
   {
      setMinute( minute + 1 );

      if ( minute == 0 )
         incrementHour();
   } // end method incrementMinute

   // Increment the hour
   public void incrementHour()
   {
      setHour( hour + 1 );
   } // end method incrementHour

   // convert to String in universal-time format (HH:MM:SS)
   public String toUniversalString()
   {
      return String.format(
         "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
   } // end method toUniversalString

   // convert to String in standard-time format (H:MM:SS AM or PM)
   public String toString()
   {
      return String.format( "%d:%02d:%02d %s",
         ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
         getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
   } // end method toStandardString
} // end class Time2
```

```java
// Exercise 8.9 Solution: Time2Test.java
// Program adds validation to Fig. 8.7 example
import java.util.Scanner;

public class Time2Test
{
   public static void main( String args[] )
   {
      Scanner input = new Scanner( System.in );

      Time2 time = new Time2(); // the Time2 object

      int choice = getMenuChoice();

      while ( choice != 5 )
      {
         switch ( choice )
         {
            case 1: // set hour
               System.out.print( "Enter Hours: " );
               int hours = input.nextInt();

               if ( !time.setHour( hours ) )
                  System.out.println( "Invalid hours." );

               break;

            case 2: // set minute
               System.out.print( "Enter Minutes: " );
               int minutes = input.nextInt();

               if ( !time.setMinute( minutes ) )
                  System.out.println( "Invalid minutes." );

               break;

            case 3: // set seconds
               System.out.print( "Enter Seconds: " );
               int seconds = input.nextInt();

               if ( !time.setSecond( seconds ) )
                  System.out.println( "Invalid seconds." );

               break;

            case 4: // add 1 second
               time.tick();
               break;
         } // end switch

         System.out.printf( "Hour: %d  Minute: %d  Second: %d\n",
            time.getHour(), time.getMinute(), time.getSecond() );
         System.out.printf( "Universal time: %s   Standard time: %s\n",
            time.toUniversalString(), time.toString() );

         choice = getMenuChoice();
      } // end while
   } // end main

   // prints a menu and returns a value corresponding to the menu choice
   private static int getMenuChoice()
   {
      Scanner input = new Scanner( System.in );

      System.out.println( "1. Set Hour" );
      System.out.println( "2. Set Minute" );
      System.out.println( "3. Set Second" );
      System.out.println( "4. Add 1 second" );
      System.out.println( "5. Exit" );
      System.out.print( "Choice: " );

      return input.nextInt();
   } // end method getMenuChoice
} // end class Time2Test
```

```
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 1
Enter Hours: 10
Hour: 10  Minute: 0  Second: 0
Universal time: 10:00:00   Standard time: 10:00:00 AM
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 2
Enter Minutes: 10
Hour: 10  Minute: 10  Second: 0
Universal time: 10:10:00   Standard time: 10:10:00 AM
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 3
Enter Seconds: 10
Hour: 10  Minute: 10  Second: 10
Universal time: 10:10:10   Standard time: 10:10:10 AM
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 3
Enter Seconds: 99
Invalid seconds.
Hour: 10  Minute: 10  Second: 0
Universal time: 10:10:00   Standard time: 10:10:00 AM
1. Set Hour
2. Set Minute
3. Set Second
4. Add 1 second
5. Exit
Choice: 5
```

8.11 Write an `enum` type `TrafficLight`, whose constants (`RED`, `GREEN`, `YELLOW`) take one parameter—the duration of the light. Write a program to test the `TrafficLight` `enum` so that it displays the `enum` constants and their durations.

ANS:

```java
// Exercise 8.11 Solution: TrafficLight.java
// Declare an enum type with constructor and explicit instance fields
// and accessors for these fields

public enum TrafficLight
{
   // declare constants of enum type
   RED( 50 ), // light is red for 50 seconds
   GREEN( 40 ), // light is green for 40 seconds
   YELLOW( 5 ); // light is yellow for 5 seconds

   // instance fields
   private final int duration; // duration of the light

   // enum type constructor
   TrafficLight( int durationSeconds )
   {
      duration = durationSeconds;
   } // end enum constructor TrafficLight

   // accessor for duration
   public int getDuration()
   {
      return duration;
   } // end method getDuration
} // end enum TrafficLight
```

```java
// Exercise 8.11 Solution: EnumTest.java
// Testing enum type TrafficLight.

public class EnumTest
{
   public static void main( String args[] )
   {
      System.out.println( "Light\tDuration\n" );

      // print all traffic lights and their duration
      for ( TrafficLight light : TrafficLight.values() )
         System.out.printf( "%s\t%d\n", light, light.getDuration() );
   } // end main
} // end class EnumTest
```

```
Light   Duration

RED     50
GREEN   40
YELLOW  5
```

8.14 *(Enhanced Rectangle Class)* Create a more sophisticated Rectangle class than the one you created in Exercise 8.4. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a *set* method that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single *x*- or *y*-coordinate larger than 20.0. The *set* method also verifies that the supplied coordinates specify a rectangle. Provide methods to calculate the length, width, perimeter and area. The length is the larger of the two dimensions. Include a predicate method isSquare which determines whether the rectangle is a square. Write a program to test class Rectangle.

ANS:

```java
// Exercise 8.14 Solution: Rectangle.java
// Definition of class Rectangle

public class Rectangle
{
   // coordinates of the vertices.
   private double x1, y1;
   private double x2, y2;
   private double x3, y3;
   private double x4, y4;

   // no-argument constructor
   public Rectangle()
   {
      setCoordinates( 1, 1, 1, 1, 1, 1, 1, 1 );
   } // end Rectangle no-argument constructor

   // constructor
   public Rectangle( double x1, double y1, double x2,
      double y2, double x3, double y3, double x4, double y4 )
   {
      setCoordinates( x1, y1, x2, y2, x3, y3, x4, y4 );
   } // end standard Rectangle constructor

   // check if coordinates are valid
   public void setCoordinates( double xInput1, double yInput1,
      double xInput2, double yInput2, double xInput3,
      double yInput3, double xInput4, double yInput4 )
   {
      x1 = ( xInput1 >= 0.0 && xInput1 <= 20.0 ? xInput1 : 1 );
      x2 = ( xInput2 >= 0.0 && xInput2 <= 20.0 ? xInput2 : 1 );
      x3 = ( xInput3 >= 0.0 && xInput3 <= 20.0 ? xInput3 : 1 );
      x4 = ( xInput4 >= 0.0 && xInput4 <= 20.0 ? xInput4 : 1 );
      y1 = ( yInput1 >= 0.0 && yInput1 <= 20.0 ? yInput1 : 1 );
      y2 = ( yInput2 >= 0.0 && yInput2 <= 20.0 ? yInput2 : 1 );
      y3 = ( yInput3 >= 0.0 && yInput3 <= 20.0 ? yInput3 : 1 );
      y4 = ( yInput4 >= 0.0 && yInput4 <= 20.0 ? yInput4 : 1 );

      if ( !isRectangle() )
         System.out.println( "This is not a rectangle" );
   } // end method setCoordinates

   // 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

   // check if coordinates specify a rectangle by determining if the
   // two diagonals are of the same length.
   public boolean isRectangle()
   {
      double side1 = distance( x1, y1, x2, y2 );
      double side2 = distance( x2, y2, x3, y3 );
      double side3 = distance( x3, y3, x4, y4 );

      if ( side1 * side1 + side2 * side2 ==
         side2 * side2 + side3 * side3 )
         return true;
      else
         return false;
   } // end method isRectangle

   // check if rectangle is a square
   public boolean isSquare()
   {
      return ( getLength() == getWidth() );
   } // end method isSquare

   // get length of rectangle
   public double getLength()
   {
      double side1 = distance( x1, y1, x2, y2 );
      double side2 = distance( x2, y2, x3, y3 );

      return ( side1 > side2 ? side1 : side2 );
   } // end method getLength

   // get width of rectangle
   public double getWidth()
   {
      double side1 = distance( x1, y1, x2, y2 );
      double side2 = distance( x2, y2, x3, y3 );

      return ( side1 < side2 ? side1 : side2 );
   } // end method getWidth

   // calculate perimeter
   public double perimeter()
   {
      return 2 * getLength() + 2 * getWidth();
   } // end method perimeter

   // calculate area
   public double area()
   {
      return getLength() * getWidth();
   } // end method area

   // convert to String
   public String toString()
   {
      return String.format( "%s: %f\n%s: %f\n%s: %f\n%s: %f",
         "Length", getLength(), "Width", getWidth(),
         "Perimeter", perimeter(), "Area", area() );
   } // end method toRectangleString
} // end class Rectangle
```

```
// Exercise 8.14 Solution: RectangleTest.java
// Program tests class Rectangle.
import java.util.Scanner;

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

      System.out.println( "Enter rectangle's coordinates" );
      System.out.print( "x1: " );
      double x1 = input.nextInt();
      System.out.print( "y1: " );
      double y1 = input.nextInt();
      System.out.print( "x2: " );
      double x2 = input.nextInt();
      System.out.print( "y2: " );
      double y2 = input.nextInt();
      System.out.print( "x3: " );
      double x3 = input.nextInt();
      System.out.print( "y3: " );
      double y3 = input.nextInt();
      System.out.print( "x4: " );
      double x4 = input.nextInt();
      System.out.print( "y4: " );
      double y4 = input.nextInt();

      Rectangle rectangle =
         new Rectangle( x1, y1, x2, y2, x3, y3, x4, y4 );

      if ( rectangle.isRectangle() )
         System.out.println( rectangle.toString() );

      if ( rectangle.isSquare() )
         System.out.println( "This is a square" );
   } // end main
} // end class RectangleTest
```

```
Enter rectangle's coordinates
x1: 10
y1: 8
x2: 10
y2: 1
x3: 1
y3: 1
x4: 1
y4: 8
Length: 9.000000
Width: 7.000000
Perimeter: 32.000000
Area: 63.000000
```

8.17 *(Rational Numbers)* Create a class called `Rational` for performing arithmetic with fractions. Write a program to test your class. Use integer variables to represent the `private` instance variables of the class—the `numerator` and the `denominator`. Provide a constructor that enables an object of this class to be initialized when it is declared. The constructor should store the fraction in reduced form—the fraction

```
2/4
```

is equivalent to `1/2` and would be stored in the object as 1 in the `numerator` and 2 in the `denominator`. Provide a no-argument constructor with default values in case no initializers are provided. Provide `public` methods that perform each of the following operations:

a) Add two `Rational` numbers: The result of the addition should be stored in reduced form.

b) Subtract two `Rational` numbers: The result of the subtraction should be stored in reduced form.

c) Multiply two `Rational` numbers: The result of the multiplication should be stored in reduced form.

d) Divide two `Rational` numbers: The result of the division should be stored in reduced form.

e) Print `Rational` numbers in the form `a/b`, where `a` is the `numerator` and `b` is the `denominator`.

f) Print `Rational` numbers in floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)

ANS:

```java
// Exercise 8.17 Solution: Rational.java
// Rational class definition.

public class Rational
{
   private int numerator; // numerator of the fraction
   private int denominator; // denominator of the fraction

   // no-argument constructor, initializes this Rational to 1
   public Rational()
   {
      numerator = 1;
      denominator = 1;
   } // end Rational no-argument constructor

   // initialize numerator part to n and denominator part to d
   public Rational( int theNumerator, int theDenominator )
   {
      numerator = theNumerator;
      denominator = theDenominator;
      reduce();
   } // end two-argument constructor

   // add two Rational numbers
   public Rational sum( Rational right )
   {
      int resultDenominator = denominator * right.denominator;
      int resultNumerator = numerator * right.denominator +
         right.numerator * denominator;

      return new Rational( resultNumerator, resultDenominator );
   } // end method sum

   // subtract two Rational numbers
   public Rational subtract( Rational right )
   {
      int resultDenominator = denominator * right.denominator;
      int resultNumerator = numerator * right.denominator -
         right.numerator * denominator;

      return new Rational( resultNumerator, resultDenominator );
   } // end method subtract

   // multiply two Rational numbers
   public Rational multiply( Rational right )
   {
      return new Rational( numerator * right.numerator,
         denominator * right.denominator );
   } // end method multiply

   // divide two Rational numbers
   public Rational divide( Rational right )
   {
      return new Rational( numerator * right.denominator,
         denominator * right.numerator );
   } // end method divide

   // reduce the fraction
   private void reduce()
   {
      int gcd = 0;
      int smaller;

      // find the greatest common denominator of the two numbers
      if ( numerator < denominator )
         smaller = numerator;
      else
```

```
         smaller = denominator;

      for ( int divisor = smaller; divisor >= 2; divisor-- )
      {
         if ( numerator % divisor == 0 && denominator % divisor == 0 )
         {
            gcd = divisor;
            break;
         } // end if
      } // end for

      // divide both the numerator and denominator by the gcd
      if ( gcd != 0 )
      {
         numerator /= gcd;
         denominator /= gcd;
      } // end if
   } // end for

   // return String representation of a Rational number
   public String toString()
   {
     return numerator + "/" + denominator;
   } // end method toRationalString

   // return floating-point String representation of
   // a Rational number
   public String toFloatString( int digits )
   {
      double value = ( double ) numerator / denominator;
      // builds a formatting string that specifies the precision
      // based on the digits parameter
      return String.format( "%." + digits + "f", value );
   } // end method toFloatString
} // end class Rational
```

```
// Exercise 8.17 Solution: RationalTest.java
// Program tests class Rational.
import java.util.Scanner;

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

      int numerator; // the numerator of a fraction
      int denominator; // the denominator of a fraction
      int digits; // digits to display in floating point format
      Rational rational1; // the first rational number
      Rational rational2; // second rational number
      Rational result; // result of performing an operation

      // read first fraction
      System.out.print( "Enter numerator 1: " );
      numerator = input.nextInt();
      System.out.print( "Enter denominator 1: " );
      denominator = input.nextInt();
      rational1 = new Rational( numerator, denominator );

      // read second fraction
      System.out.print( "Enter numerator 2: " );
      numerator = input.nextInt();
      System.out.print( "Enter denominator 2: " );
      denominator = input.nextInt();
      rational2 = new Rational( numerator, denominator );

      System.out.print( "Enter precision: " );
      digits = input.nextInt();

      int choice = getMenuChoice(); // user's choice in the menu

      while ( choice != 5 )
      {
         switch ( choice )
         {
            case 1:
               result = rational1.sum( rational2 );
               System.out.printf( "a + b = %s = %s\n",
                  result.toString(),
                  result.toFloatString( digits ) );
               break;

            case 2:
               result = rational1.subtract( rational2 );
               System.out.printf( "a - b = %s = %s\n",
                  result.toString(),
                  result.toFloatString( digits ) );
               break;

            case 3:
               result = rational1.multiply( rational2 );
               System.out.printf( "a * b = %s = %s\n",
                  result.toString(),
                  result.toFloatString( digits ) );
               break;

            case 4:
               result = rational1.divide( rational2 );
               System.out.printf( "a / b = %s = %s\n",
                  result.toString(),
                  result.toFloatString( digits ) );
               break;
         } // end switch

         choice = getMenuChoice();
      } // end while
   } // end main

   // prints a menu and returns a value corresponding to the menu choice
   private static int getMenuChoice()
   {
      Scanner input = new Scanner( System.in );

      System.out.println( "1. Add" );
      System.out.println( "2. Subtract" );
      System.out.println( "3. Multiply" );
      System.out.println( "4. Divide" );
      System.out.println( "5. Exit" );
      System.out.print( "Choice: " );

      return input.nextInt();
   } // end method getMenuChoice
} // end class RationalTest
```

```
Enter numerator 1: 12
Enter denominator 1: 3
Enter numerator 2: 34
Enter denominator 2: 5
Enter precision: 5
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Choice: 1
a + b = 54/5 = 10.80000
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Choice: 2
a - b = -14/5 = -2.80000
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Choice: 3
a * b = 136/5 = 27.20000
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Choice: 4
a / b = 10/17 = 0.58823
1. Add
2. Subtract
3. Multiply
4. Divide
5. Exit
Choice: 5
```