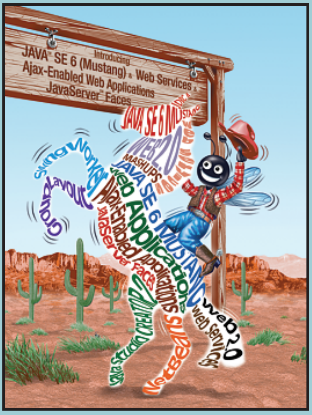
8

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.

Self-Review Exercises

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

a) When compiling a class in a package, the javac command-line option ________ specifies where to store the package and causes the compiler to create the package's directories if they do not exist.

ANS: -d.

b) String class static method ________ is similar to method System.out.printf, but returns a formatted String rather than displaying a String in a command window.

ANS: format.

c) If a method contains a local variable with the same name as one of its class's fields, the local variable ________ the field in that method's scope.

ANS: shadows.

d) The ________ method is called by the garbage collector just before it reclaims an object's memory.

ANS: finalize.

e) A(n) ________ declaration specifies one class to import.

ANS: single-type-import.

f) If a class declares constructors, the compiler will not create a(n) ________.

ANS: default constructor

g) An object's ________ method is called implicitly when an object appears in code where a String is needed.

ANS: toString.

h) *Get* methods are commonly called ________ or________.

ANS: accessor methods, query methods.

i) A(n) ________ method tests whether a condition is true or false.

ANS: predicate.

j) For every enum, the compiler generates a static method called ________ that returns an array of the enum's constants in the order in which they were declared.

ANS: values.

k) Composition is sometimes referred to as a ________ relationship.

ANS: *has-a.*

l) A(n) ________ declaration contains a comma-separated list of constants.

ANS: enum.

m) A(n) ________ variable represents classwide information that is shared by all the objects of the class.

ANS: static.

n) A(n) ________ declaration imports one static member.

ANS: single static import.

o) The ________ states that code should be granted only the amount of privilege and access that the code needs to accomplish its designated task.

ANS: principle of least privilege.

p) Keyword ________ specifies that a variable is not modifiable.

ANS: final.

q) A(n) ________ consists of a data representation and the operations that can be performed on the data.

ANS: abstract data type (ADT) .

r) There can be only one ________ in a Java source-code file, and it must precede all other declarations and statements in the file.

ANS: package declaration.

s) A(n) ________ declaration imports only the classes that the program uses from a particular package.

ANS: type-import-on-demand.

t) The compiler uses a(n) ________ to locate the classes it needs in the classpath.

ANS: class loader.

u) The classpath for the compiler and JVM can be specified with the ________ option to the `javac` or `java` command, or by setting the ________ environment variable.

ANS: `-classpath`, `CLASSPATH`.

v) *Set* methods are sometimes called ________ because they typically change a value.

ANS: mutator methods.

w) A(n) ________ imports all `static` members of a class.

ANS: `static` import on demand.

x) The `public` methods of a class are also known as the class's ________ or ________.

ANS: `public` services, `public` interface.

y) `System` class `static` method ________ indicates that the garbage collector should make a best-effort attempt to reclaim objects that are eligible for garbage collection.

ANS: `gc`.

z) An object that contains ________ has data values that are always kept in range.

ANS: consistent data.

Exercises

8.2 Explain the notion of package access in Java. Explain the negative aspects of package access.

ANS: Package access allows a class, method, or variable to be accessible within the same package. Package access does not promote good OOP when applied to an instance variable because it destroys the notion of information hiding.

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
```

8.5 *(Modifying the Internal Data Representation of a Class)* It would be perfectly reasonable for the `Time2` class of Fig. 8.5 to represent the time internally as the number of seconds since midnight rather than the three integer values `hour`, `minute` and `second`. Clients could use the same `public` methods and get the same results. Modify the `Time2` class of Fig. 8.5 to implement the `Time2` as the number of seconds since midnight and show that no change is visible to the clients of the class.

ANS:

```java
// Exercise 8.5 Solution: Time2.java
// Time2 class definition maintains the time in 24-hour format.

public class Time2
{
   private int totalSeconds;

   // no-argument constructor initializes totalSeconds to zero;
   public Time2()
   {
      totalSeconds = 0;
   } // end no-argument constructor

   // Time2 constructor: hour supplied, minute and second defaulted to 0
   public Time2( int h )
   {
      setTime( h, 0, 0 );
   } // end hour constructor

   // Time2 constructor: hour and minute supplied, second defaulted to 0
   public Time2( int h, int m )
   {
      setTime( h, m, 0 );
   } // end hour and minute constructor

   // Time2 constructor: hour, minute and second supplied
   public Time2( int h, int m, int s )
   {
      setTime( h, m, s );
   } // end hour, minute and second constructor

   // Time2 constructor: another Time2 object supplied
   public Time2( Time2 time )
   {
      setTime( time.getHour(), time.getMinute(), time.getSecond() );
   } // end Time2 constructor

   // set a new time value using total seconds; perform
   // validity checks on data; set invalid values to zero
   public void setTime( int h, int m, int s )
   {
      setHour( h );
      setMinute( m );
      setSecond( s );
   } // end method setTime

   // set hour value
   public void setHour( int h )
   {
      int hours = ( h >= 0 && h < 24 ) ? h : 0;

      totalSeconds = ( hours * 3600 ) + ( getMinute() * 60 ) +
         getSecond();
   } // end method setHour

   // set minute value
   public void setMinute( int m )
   {
      int minutes = ( m >= 0 && m < 60 ) ? m : 0;

      totalSeconds = ( getHour() * 3600 ) + ( minutes * 60 ) +
         getSecond();
   } // end method setMinute

   // set second value
   public void setSecond( int s )
   {
      int seconds = ( s >= 0 && s < 60 ) ? s : 0;
```

```java

      totalSeconds = ( getHour() * 3600 ) + ( getMinute() *60 ) + seconds;
   } // end method setSecond

   // get hour value
   public int getHour()
   {
      return ( totalSeconds / 3600 );
   } // end method getHour

   // get minute value
   public int getMinute()
   {
      return ( ( totalSeconds % 3600 ) / 60 );
   } // end method getMinute

   // get second value
   public int getSecond()
   {
      return ( ( totalSeconds % 3600 ) % 60 );
   } // end method getSecond

   // 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 8.5 Solution: Time2Test.java
// Class TimeTest3 to test class Time2

public class Time2Test
{
   public static void main( String args[] )
   {
      Time2 t = new Time2();

      System.out.print( "The initial universal time is: " );
      System.out.println( t.toUniversalString() );
      System.out.print( "The initial standard time is: " );
      System.out.println( t.toString() );

      t.setTime( 13, 27, 6 );
      System.out.print( "\nUniversal time after setTime is: " );
      System.out.println( t.toUniversalString() );
      System.out.print( "Standard time after setTime is: " );
      System.out.println( t.toString() );

      t.setTime( 99, 99, 99 );
      System.out.println( "\nAfter attempting invalid settings:" );
      System.out.print( "The initial universal time is: " );
      System.out.println( t.toUniversalString() );
      System.out.print( "The initial standard time is: " );
      System.out.println( t.toString() );
   } // end main
} // end class Time2Test
```

```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM

Universal time after setTime is: 13:27:06
Standard time after setTime is: 1:27:06 PM

After attempting invalid settings:
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM
```

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
```

8.7 *(Enhancing Class `Time2`)* Modify class `Time2` of Fig. 8.5 to include a `tick` method that increments the time stored in a `Time2` object by one second. Provide method `incrementMinute` to increment the minute and method `incrementHour` to increment the hour. The `Time2` object should always remain in a consistent state. Write a program that tests the `tick` method, the `incrementMinute` method and the `incrementHour` method to ensure that they work correctly. Be sure to test the following cases:

a) incrementing into the next minute,
b) incrementing into the next hour and
c) incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

ANS:

```java
// Exercise 8.7 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 void setTime( int h, int m, int s )
   {
      setHour( h ); // set the hour
      setMinute( m ); // set the minute
      setSecond( s ); // set the second
   } // end method setTime

   // validate and set hour
   public void setHour( int h )
   {
      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
   } // end method setHour

   // validate and set minute
   public void setMinute( int m )
   {
      minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
   } // end method setMinute

   // validate and set second
   public void setSecond( int s )
   {
      second = ( ( s >= 0 && s < 60 ) ? s : 0 );
   } // 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
```

```
// Exercise 8.7 Solution: Time2Test.java
// Demonstrating the Time2 class set and get methods
import java.util.Scanner;

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

      Time2 time = new Time2();

      System.out.println( "Enter the time" );
      System.out.print( "Hours: " );
      time.setHour( input.nextInt() );
      System.out.print( "Minutes: " );
      time.setMinute( input.nextInt() );
      System.out.print( "Seconds: " );
      time.setSecond( input.nextInt() );

      int choice = getMenuChoice();

      while ( choice != 5 )
      {
         switch ( choice )
         {
            case 1: // add 1 second
               time.tick();
               break;

            case 2: // add 1 minute
               time.incrementMinute();
               break;

            case 3: // and 1 hour
               time.incrementHour();
               break;

            case 4: // add arbitrary seconds
               System.out.print( "Enter seconds to tick: " );
               int ticks = input.nextInt();

               for ( int i = 0; i < ticks; i++ )
                  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. Add 1 second" );
      System.out.println( "2. Add 1 minute" );
      System.out.println( "3. Add 1 hour" );
      System.out.println( "4. Add seconds" );
      System.out.println( "5. Exit" );
      System.out.print( "Choice: " );

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

```
Enter the time
Hours: 23
Minutes: 59
Seconds: 59
1. Add 1 second
2. Add 1 minute
3. Add 1 hour
4. Add seconds
5. Exit
Choice: 1
Hour: 0  Minute: 0  Second: 0
Universal time: 00:00:00   Standard time: 12:00:00 AM
1. Add 1 second
2. Add 1 minute
3. Add 1 hour
4. Add seconds
5. Exit
Choice: 2
Hour: 0  Minute: 1  Second: 0
Universal time: 00:01:00   Standard time: 12:01:00 AM
1. Add 1 second
2. Add 1 minute
3. Add 1 hour
4. Add seconds
5. Exit
Choice: 3
Hour: 1  Minute: 1  Second: 0
Universal time: 01:01:00   Standard time: 1:01:00 AM
1. Add 1 second
2. Add 1 minute
3. Add 1 hour
4. Add seconds
5. Exit
Choice: 4
Enter seconds to tick: 60
Hour: 1  Minute: 2  Second: 0
Universal time: 01:02:00   Standard time: 1:02:00 AM
1. Add 1 second
2. Add 1 minute
3. Add 1 hour
4. Add seconds
5. Exit
Choice: 5
```

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:

```
// 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 );
```

```java

      // 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()
```

```java
   {
      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
```

8.10 Rewrite Fig. 8.14 to use a separate `import` declaration for each `static` member of class `Math` that is used in the example.

ANS:

```
// Exercise 8.10 Solution: StaticImportTest.java
// Using static import to import static methods of class Math.
import static java.lang.Math.E;
import static java.lang.Math.sqrt;
import static java.lang.Math.ceil;
import static java.lang.Math.log;
```

```java
import static java.lang.Math.cos;

public class StaticImportTest
{
   public static void main( String args[] )
   {
      System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt( 900.0 ) );
      System.out.printf( "ceil( -9.8 ) = %.1f\n", ceil( -9.8 ) );
      System.out.printf( "log( E ) = %.1f\n", log( E ) );
      System.out.printf( "cos( 0.0 ) = %.1f\n", cos( 0.0 ) );
   } // end main
} // end class StaticImportTest
```

```
sqrt( 900.0 ) = 30.0
ceil( -9.8 ) = -9.0
log( E ) = 1.0
cos( 0.0 ) = 1.0
```

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
```

8.12 *(Complex Numbers)* Create a class called `Complex` for performing arithmetic with complex numbers. Complex numbers have the form

realPart + *imaginaryPart* * *i*

where *i* is

$$\sqrt{-1}$$

Write a program to test your class. Use floating-point variables to represent the `private` data of the class. Provide a constructor that enables an object of this class to be initialized when it is declared. Provide a no-argument constructor with default values in case no initializers are provided. Provide `public` methods that perform the following operations:

a) Add two `Complex` numbers: The real parts are added together and the imaginary parts are added together.

b) Subtract two `Complex` numbers: The real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.

c) Print `Complex` numbers in the form `(a, b)`, where `a` is the real part and `b` is the imaginary part.

ANS:

```java
// Exercise 8.12 Solution: Complex.java
// Definition of class Complex

public class Complex
{
   private double real;
   private double imaginary;

   // Initialize both parts to 0
   public Complex()
   {
      this( 0.0, 0.0 );
   } // end Complex no-argument constructor

   // Initialize real part to r and imaginary part to i
   public Complex( double r, double i )
   {
      real = r;
      imaginary = i;
   } // end Complex two-argument constructor

   // Add two Complex numbers
   public Complex add( Complex right )
   {
      return new Complex( real + right.real,
         imaginary + right.imaginary );
   } // end method add

   // Subtract two Complex numbers
   public Complex subtract( Complex right )
   {
      return new Complex( real - right.real,
         imaginary - right.imaginary );
   } // end method subtract

   // Return String representation of a Complex number
   public String toString()
   {
      return String.format( "(%.1f, %.1f)", real, imaginary );
   } // end method toComplexString;
} // end class Complex
```

```
// Exercise 8.12: ComplexTest.java
// Test the Complex number class

public class ComplexTest
{
   public static void main( String args[] )
   {
      // initialize two numbers
      Complex a = new Complex( 9.5, 7.7 );
      Complex b = new Complex( 1.2, 3.1 );

      System.out.printf( "a = %s\n", a.toString() );
      System.out.printf( "b = %s\n", b.toString() );
      System.out.printf( "a + b = %s\n", a.add( b ).toString() );
      System.out.printf( "a - b = %s\n", a.subtract( b ).toString() );
   } // end main
} // end class ComplexTest
```

```
a = (9.5, 7.7)
b = (1.2, 3.1)
a + b = (10.7, 10.8)
a - b = (8.3, 4.6)
```

8.13 *(Date and Time Class)* Create class `DateAndTime` that combines the modified `Time2` class of Exercise 8.7 and the modified `Date` class of Exercise 8.8. Modify method `incrementHour` to call method `nextDay` if the time is incremented into the next day. Modify methods `toStandardString` and `toUniversalString` to output the date in addition to the time. Write a program to test the new class `DateAndTime`. Specifically, test incrementing the time to the next day.

ANS:

```java
// Exercise 8.13 Solution: DateAndTime.java
// DateAndTime class definition.

public class DateAndTime
{
   private int month; // 1-12
   private int day; // 1-31 based on month
   private int year; // > 0
   private int hour; // 0 - 23
   private int minute; // 0 - 59
   private int second; // 0 - 59

   // no argument constructor
   public DateAndTime()
   {
      setDate( 1, 1, 2000 );
      setTime( 0, 0, 0 );
   } // end DateAndTime no-argument constructor

   // constructor
   public DateAndTime( int theMonth, int theDay, int theYear,
      int theHour, int theMinute, int theSecond )
   {
      setDate( theMonth, theDay, theYear );
      setTime( theHour, theMinute, theSecond );
   } // end DateAndTime six-argument constructor

   // Set a new date value. Perform
   // validity checks on data. Set invalid values to one.
   public void setDate( int theMonth, int theDay, int theYear )
   {
      month = checkMonth( theMonth ); // validate month
      year = checkYear( theYear ); // validate year
      day = checkDay( theDay ); // validate day
   } // end method setDate

   // Set a new time value using universal time. Perform
   // validity checks on data. Set invalid values to zero.
   public void setTime( int h, int m, int s )
   {
      setHour( h ); // set the hour
      setMinute( m ); // set the minute
      setSecond( s ); // set the second
   } // end method setTime

   // validate and set hour
   public void setHour( int h )
   {
      hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
   } // end method setHour

   // validate and set minute
   public void setMinute( int m )
   {
      minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
   } // end method setMinute

   // validate and set second
   public void setSecond( int s )
   {
      second = ( ( s >= 0 && s < 60 ) ? s : 0 );
   } // 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 );

      if ( hour == 0 )
         nextDay();
   } // end method incrementHour

   // 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.
   public 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;
```

```java

      return 1; // leave 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

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

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

```java
// Exercise 8.13 Solution: DateAndTimeTest.java
// Demonstrating the DateAndTime class methods
import java.util.Scanner;

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

      System.out.println( "Enter the date and time" );
      System.out.print( "Month: " );
      int month = input.nextInt();
      System.out.print( "Day: " );
      int day = input.nextInt();
      System.out.print( "Year: " );
      int year = input.nextInt();

      System.out.print( "Hours: " );
      int hour = input.nextInt();
      System.out.print( "Minutes: " );
      int minute = input.nextInt();
      System.out.print( "Seconds: " );
      int seconds = input.nextInt();

      DateAndTime dateTime = new DateAndTime(
         month, day, year, hour, minute, seconds );

      int choice = getMenuChoice();

      while ( choice != 7 )
      {
         switch ( choice )
         {
            case 1: // add 1 second
               dateTime.tick();
               break;

            case 2: // add 1 minute
               dateTime.incrementMinute();
               break;

            case 3: // and 1 hour
               dateTime.incrementHour();
               break;

            case 4: // add 1 day
               dateTime.nextDay();
               break;

            case 5: // add 1 month
               dateTime.nextMonth();
               break;

            case 6: // add arbitrary seconds
               System.out.print( "Enter seconds to tick: " );
               int ticks = input.nextInt();

               for ( int i = 0; i < ticks; i++ )
                  dateTime.tick();

               break;
         } // end switch

         System.out.printf( "Universal date and time: %s\n",
            dateTime.toUniversalString() );
         System.out.printf( "Standard date and time: %s\n",
            dateTime.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. Add 1 second" );
      System.out.println( "2. Add 1 Minute" );
      System.out.println( "3. Add 1 Hour" );
      System.out.println( "4. Add 1 Day" );
      System.out.println( "5. Add 1 Month" );
      System.out.println( "6. Add seconds" );
      System.out.println( "7. Exit" );
      System.out.print( "Choice: " );

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

```
Enter the date and time
Month: 12
Day: 31
Year: 1999
Hours: 23
Minutes: 59
Seconds: 59
1. Add 1 second
2. Add 1 Minute
3. Add 1 Hour
4. Add 1 Day
5. Add 1 Month
6. Add seconds
7. Exit
Choice: 1
Universal date and time: 1/1/2000: 00:00:00
Standard date and time: 1/1/2000: 12:00:00 AM
1. Add 1 second
2. Add 1 Minute
3. Add 1 Hour
4. Add 1 Day
5. Add 1 Month
6. Add seconds
7. Exit
Choice: 7
```

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
```

8.15 *(Set of Integers)* Create class `IntegerSet`. Each `IntegerSet` object can hold integers in the range 0–100. The set is represented by an array of `boolean`s. Array element `a[i]` is `true` if integer *i* is in the set. Array element `a[j]` is `false` if integer *j* is not in the set. The no-argument constructor initializes the Java array to the “empty set” (i.e., a set whose array representation contains all `false` values).

Provide the following methods: Method `union` creates a third set that is the set-theoretic union of two existing sets (i.e., an element of the third set’s array is set to `true` if that element is `true` in either or both of the existing sets—otherwise, the element of the third set is set to `false`). Method `intersection` creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set’s array is set to `false` if that element is `false` in either or both of the existing sets—otherwise, the element of the third set is set to `true`). Method `insertElement` inserts a new integer *k* into a set (by setting `a[k]` to `true`). Method `deleteElement` deletes integer *m* (by setting `a[m]` to `false`). Method `toSetString` returns a string containing a set as a list of numbers separated by spaces. Include only those elements that are present in the set. Use --- to represent an empty set. Method `isEqualTo` determines whether two sets are equal. Write a program to test class `IntegerSet`. Instantiate several `IntegerSet` objects. Test that all your methods work properly.

ANS:

```java
// Exercise 8.15 Solution: IntegerSet.java
// IntegerSet class definition

public class IntegerSet
{
   private final int SETSIZE = 101;
   private boolean[] set;

   // no-argument constructor, creates an empty set
   public IntegerSet()
   {
      set = new boolean[ SETSIZE ];
   } // end no-argument constructor

   // constructor creates a set from array of integers
   public IntegerSet( int array[] )
   {
      set = new boolean[ SETSIZE ];

      for( int i = 0; i < array.length; i++ )
         insertElement( array[ i ] );
   } // end constructor

   // return string representation of set
   public String toString()
   {
      int x = 1;
      boolean empty = true; // assume set is empty
      String setString = "{ ";

      // get set elements
      for ( int count = 0; count < 101; count++ )
      {
         if ( set[ count ] )
         {
            setString += count + " ";
            empty = false; // set is not empty
            x++;
         } // end if
      } // end for

      // empty set
      if ( empty )
         setString += "---"; // display an empty set

      setString += " }";

      return setString;
   } // end method toString

   // returns the union of two sets
   public IntegerSet union( IntegerSet integerSet )
   {
      IntegerSet temp = new IntegerSet();

      for ( int count = 0; count < 101; count++ )
         temp.set[ count ] = ( set[ count ] || integerSet.set[ count ] );

      return temp;
   } // end method union

   // returns the intersection of two sets
   public IntegerSet intersection( IntegerSet integerSet )
   {
      IntegerSet temp = new IntegerSet();

      for ( int count = 0; count < 101; count++ )
         temp.set[ count ] =
            ( set[ count ] && integerSet.set[ count ] );

      return temp;
   } // end method intersetcion

   // insert a new integer into this set
   public void insertElement( int insertInteger )
   {
      if ( validEntry( insertInteger ) )
         set[ insertInteger ] = true;
   } // end method insertElement

   // remove an integer from this set
   public void deleteElement( int deleteInteger )
   {
      if ( validEntry( deleteInteger ) )
         set[ deleteInteger ] = false;
   } // end method deleteElement

   // determine if two sets are equal
   public boolean isEqualTo( IntegerSet integerSet )
   {
      for ( int count = 0; count < 101; count++ )
      {
         if ( set[ count ] != integerSet.set[ count ] )
            return false; // sets are not-equal
      } // end for

      return true; // sets are equal
   } // end method isEqualTo

   // determin if input is valid
   public boolean validEntry( int input )
   {
      return input >= 0 && input <= 100;
   } // end method validEntry
} // end class IntegerSet
```

```
// Exercise 8.15 Solution: IntegerSetTest.java
// Program that tests IntegerSet
import java.util.Scanner;

public class IntegerSetTest
{
   public static void main( String args[] )
   {
      // initialize two sets
      System.out.println( "Input Set A" );
      IntegerSet set1 = inputSet();
      System.out.println( "Input Set B" );
      IntegerSet set2 = inputSet();

      IntegerSet union = set1.union( set2 );
      IntegerSet intersection = set1.intersection( set2 );

      // prepare output
      System.out.println( "Set A contains elements:" );
      System.out.println( set1.toString() );
      System.out.println( "Set B contains elements:" );
      System.out.println( set2.toString() );
      System.out.println(
         "Union of Set A and Set B contains elements:" );
      System.out.println( union.toString() );
      System.out.println(
         "Intersection of Set A and Set B contains elements:" );
      System.out.println( intersection.toString() );

      // test whether two sets are equal
      if ( set1.isEqualTo( set2 ) )
         System.out.println( "Set A is equal to set B" );
      else
         System.out.println( "Set A is not equal to set B" );

      // test insert and delete
      System.out.println( "Inserting 77 into set A..." );
      set1.insertElement( 77 );
      System.out.println( "Set A now contains elements:" );
      System.out.println( set1.toString() );

      System.out.println( "Deleting 77 from set A..." );
      set1.deleteElement( 77 );
      System.out.println( "Set A now contains elements:" );
      System.out.println( set1.toString() );

      // test constructor
      int intArray[] = { 25, 67, 2, 9, 99, 105, 45, -5, 100, 1 };
      IntegerSet set5 = new IntegerSet( intArray );

      System.out.println( "New Set contains elements:" );
      System.out.println( set5.toString() );
   } // end main

   // creates a new set by reading numbers from the user
   private static IntegerSet inputSet()
   {
      Scanner input = new Scanner( System.in );
      IntegerSet temp = new IntegerSet();

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

      while ( number != -1 )
      {
         temp.insertElement( number );

         System.out.print( "Enter number (-1 to end): " );
         number = input.nextInt();
      } // end while

      return temp;
   } // end method inputSet
} // end class IntegerSetTest
```

```
Input Set A
Enter number (-1 to end): 1
Enter number (-1 to end): 2
Enter number (-1 to end): 12
Enter number (-1 to end): 34
Enter number (-1 to end): 45
Enter number (-1 to end): 67
Enter number (-1 to end): 89
Enter number (-1 to end): 99
Enter number (-1 to end): -1
Input Set B
Enter number (-1 to end): 5
Enter number (-1 to end): 11
Enter number (-1 to end): 22
Enter number (-1 to end): 33
Enter number (-1 to end): 44
Enter number (-1 to end): 55
Enter number (-1 to end): 67
Enter number (-1 to end): 88
Enter number (-1 to end): 99
Enter number (-1 to end): 100
Enter number (-1 to end): -1
Set A contains elements:
{ 1 2 12 34 45 67 89 99  }
Set B contains elements:
{ 5 11 22 33 44 55 67 88 99 100  }
Union of Set A and Set B contains elements:
{ 1 2 5 11 12 22 33 34 44 45 55 67 88 89 99 100  }
Intersection of Set A and Set B contains elements:
{ 67 99  }
Set A is not equal to set B
Inserting 77 into set A...
Set A now contains elements:
{ 1 2 12 34 45 67 77 89 99  }
Deleting 77 from set A...
Set A now contains elements:
{ 1 2 12 34 45 67 89 99  }
New Set contains elements:
{ 1 2 9 25 45 67 99 100  }
```

8.16 *(Date Class)* Create class `Date` with the following capabilities:

a) Output the date in multiple formats, such as

```
MM/DD/YYYY
June 14, 1992
DDD YYYY
```

b) Use overloaded constructors to create `Date` objects initialized with dates of the formats in part (a). In the first case the constructor should receive three integer values. In the second case it should receive a `String` and two integer values. In the third case it should receive two integer values, the first of which represents the day number in the year. [*Hint:* To convert the string representation of the month to a numeric value, compare strings using the `equals` method. For example, if `s1` and `s2` are strings, the method call `s1.equals( s2 )` returns `true` if the strings are identical and otherwise returns `false`.]

ANS:

```java
// Exercise 8.16 Solution: Date.java
// Date class definition

public class Date
{
   private int day; // day of the month
   private int month; // month in the year
   private int year; // year
   private final String monthNames[] = { "January", "February",
      "March", "April", "May", "June", "July", "August",
      "September", "October", "November", "December" };
   private final int monthDays[] = { 31, 28, 31, 30, 31, 30,
      31, 31, 30, 31, 30, 31 };

   // no-argument constructor
   public Date()
   {
      day = 1;
      month = 1;
      year = 2000;
   } // end no-argument constructor

   // constructor for format MM/DD/YYYY
   public Date( int mm, int dd, int yyyy )
   {
      setYear( yyyy );
      setMonth( mm );
      setDay( dd );
   } // end constructor for format MM/DD/YYYY

   // constructor for format MonthName dd, yyyy
   public Date( String mm, int dd, int yyyy )
   {
      setYear( yyyy );
      convertFromMonthName( mm );
      setDay( dd );
   } // end constructor for format MonthName dd, yyyy

   // constructor for format DDD YYYY
   public Date( int ddd, int yyyy )
   {
      setYear( yyyy );
      convertFromDayOfYear( ddd );
   } // end constructor for format DDD YYYY

   // Set the day
   public void setDay( int dd )
   {
      day = ( dd >= 1 && dd <= daysInMonth() ) ? dd : 1;
   } // end method setDay

   // Set the month
   public void setMonth( int mm )
   {
      month = ( mm >= 1 && mm <= 12 ) ? mm : 1;
   } // end method setMonth

   // Set the year
   public void setYear( int yyyy )
   {
      year = ( yyyy >= 1900 && yyyy <= 2100 ) ? yyyy : 1900;
   } // end method setYear

   // return Date in format: mm/dd/yyyy
   public String toString()
   {
      return String.format( "%d/%d/%d", month, day, year );
   } // end method toSlashDateString

   // return Date in format: MonthName dd, yyyy
   public String toMonthNameDateString()
   {
      return String.format(
         "%s %d, %d", monthNames[ month - 1 ], day, year );
   } // end method toMonthNameDateString

   // return Date in format DDD YYYY
   public String toDayDateString()
   {
      return String.format( "%d %d", convertToDayOfYear(), year );
   } // end method toDayDateString

   // Return the number of days in the month
   private int daysInMonth()
   {
      return leapYear() && month == 2 ? 29 : monthDays[ month - 1 ];
   } // end method daysOfMonth

   // test for a leap year
   private boolean leapYear()
   {
      if ( year % 400 == 0 || ( year % 4 == 0 && year % 100 != 0 ) )
         return true;
      else
         return false;
   } // end method leapYear

   // sets the day and month to the proper values based on ddd
   private void convertFromDayOfYear( int ddd )
   {
      int dayTotal = 0;

      if ( ddd < 1 || ddd > 366 ) // check for invalid day
         ddd = 1;

      setMonth( 1 );

      for ( int m = 1;
         m < 13 && ( dayTotal + daysInMonth() ) < ddd; m++ )
      {
         dayTotal += daysInMonth();
         setMonth( m + 1 );
      } // end for

      setDay( ddd - dayTotal );
   } // end convertFromDayOfYear

   // convert mm and dd to ddd
   private int convertToDayOfYear()
   {
      int ddd = 0;

      for ( int m = 1; m < month; m++ )
      {
         if ( leapYear() && m == 2 )
            ddd += 29;
         else
            ddd += monthDays[ m -1 ];
      } // end for

      ddd += day;
      return ddd;
   } // end method convertToDayOfYear

   // convert from month name to month number
   private void convertFromMonthName( String monthName )
   {
      for ( int subscript = 0; subscript < 12; subscript++ )
      {
         if ( monthName.equals( monthNames[ subscript ] ) )
         {
            setMonth( subscript + 1 );
            return; // stop checking for month
         } // end if
      } // end for

      setMonth( 1 ); // invalid month default is january
   } // end convertFromMonthName
} // end class Date
```

```
// Exercise 8.16 Solution: DateTest.java
// Program that tests Date class
import java.util.Scanner;

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

      int choice = getMenuChoice();

      while ( choice != 4 )
      {
         int month; // month of year
         int day; // day of month or day of year
         int year; // year
         String monthName; // name of month
         Date date = new Date(); // the date object

         switch ( choice )
         {
            case 1:
               // format: MM/DD/YYYY
               System.out.print( "Enter Month (1-12): " );
               month = input.nextInt();
               System.out.print( "Enter Day of Month: " );
               day = input.nextInt();
               System.out.print( "Enter Year: " );
               year = input.nextInt();

               date = new Date( month, day, year );
               break;

            case 2:
               // format: Month DD, YYYY
               System.out.print( "Enter Month Name: " );
               monthName = input.next();
               System.out.print( "Enter Day of Month: " );
               day = input.nextInt();
               System.out.print( "Enter Year: " );
               year = input.nextInt();

               date = new Date( monthName, day, year );
               break;

            case 3:
               // format: DDD YYYY
               System.out.print( "Enter Day of Year: " );
               day = input.nextInt();
               System.out.print( "Enter Year: " );
               year = input.nextInt();

               date = new Date( day, year );
               break;
         } // end switch

         System.out.printf( "\n%s: %s\n%s: %s\n%s: %s\n",
            "MM/DD/YYYY", date.toString(),
            "Month DD, YYYY", date.toMonthNameDateString(),
            "DDD YYYY", date.toDayDateString() );

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

```java

   // get user choice
   private static int  getMenuChoice()
   {
      Scanner input = new Scanner( System.in );
      System.out.println( "Enter 1 for format: MM/DD/YYYY" );
      System.out.println( "Enter 2 for format: Month DD, YYYY" );
      System.out.println( "Enter 3 for format: DDD YYYY" );
      System.out.println( "Enter 4 to exit" );
      System.out.print( "Choice: " );
      return input.nextInt();
   } // end method getMenuChoice
} // end class DateTest
```

```
Enter 1 for format: MM/DD/YYYY
Enter 2 for format: Month DD, YYYY
Enter 3 for format: DDD YYYY
Enter 4 to exit
Choice: 1
Enter Month (1-12): 5
Enter Day of Month: 27
Enter Year: 1985

MM/DD/YYYY: 5/27/1985
Month DD, YYYY: May 27, 1985
DDD YYYY: 147 1985
Enter 1 for format: MM/DD/YYYY
Enter 2 for format: Month DD, YYYY
Enter 3 for format: DDD YYYY
Enter 4 to exit
Choice: 2
Enter Month Name: May
Enter Day of Month: 27
Enter Year: 1985

MM/DD/YYYY: 5/27/1985
Month DD, YYYY: May 27, 1985
DDD YYYY: 147 1985
Enter 1 for format: MM/DD/YYYY
Enter 2 for format: Month DD, YYYY
Enter 3 for format: DDD YYYY
Enter 4 to exit
Choice: 3
Enter Day of Year: 147
Enter Year: 1985

MM/DD/YYYY: 5/27/1985
Month DD, YYYY: May 27, 1985
DDD YYYY: 147 1985
Enter 1 for format: MM/DD/YYYY
Enter 2 for format: Month DD, YYYY
Enter 3 for format: DDD YYYY
Enter 4 to exit
Choice: 4
```

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 )
```

```java
      {
         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
```

```java
// 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
```

8.18 *(Huge Integer Class)* Create a class `HugeInteger` which uses a 40-element array of digits to store integers as large as 40 digits each. Provide methods `input`, `output`, `add` and `subtract`. For comparing `HugeInteger` objects, provide the following methods: `isEqualTo`, `isNotEqualTo`, `isGreaterThan`, `isLessThan`, `isGreaterThanOrEqualTo` and `isLessThanOrEqualTo`. Each of these is a predicate method that returns `true` if the relationship holds between the two `HugeInteger` objects and returns `false` if the relationship does not hold. Provide a predicate method `isZero`. If you feel ambitious, also provide methods `multiply`, `divide` and `remainder`. [*Note:* Primitive boolean values can be output as the word “true” or the word “false” with format specifier `%b`.]

ANS:

```
// Exercise 8.18 Solution: HugeInteger.java
// HugeInteger class definition

public class HugeInteger
{
   private final int DIGITS = 40;
   private int[] integer; // array containing the integer
   private boolean positive; // whether the integer is positive

   // no-argument constructor
   public HugeInteger()
   {
      integer = new int[ DIGITS ];
      positive = true;
   } // end HugeInteger no-argument constructor

   // convert a String to HugeInteger
   public void input( String inputString )
   {
      char[] integerChar = inputString.toCharArray();

      // check if input is a negative number
      if ( integerChar[ 0 ] == '-' )
         positive = false;

      if ( positive )
         integer[ DIGITS - integerChar.length ] =
            integerChar[ 0 ] - '0';

      // convert string to integer array
      for ( int i = 1; i < integerChar.length; i++ )
         integer[ DIGITS - integerChar.length + i ] =
            integerChar[ i ] - '0';
   } // end method input

   // add two HugeIntegers
   public HugeInteger add( HugeInteger addValue )
   {
      HugeInteger temp = new HugeInteger(); // temporary result

      // both HugeIntegers are positive or negative
      if ( positive == addValue.positive )
         temp = addPositives( addValue );
      // addValue is negative
      else if ( positive && ( !addValue.positive ) )
      {
         addValue.positive = true;

         if ( isGreaterThan( addValue ) )
            temp = subtractPositives( addValue );
         else
         {
            temp = addValue.subtractPositives( this );
            temp.positive = false;
         } // end else

         addValue.positive = false; // reset sign for addValue
      } // end else if
      // this is negative
      else if ( !positive && addValue.positive )
      {
         addValue.positive = false;

         if ( isGreaterThan( addValue ) )
            temp = addValue.subtractPositives( this );
         else
         {
            temp = subtractPositives( addValue );
            temp.positive = false;
         } // end else

         addValue.positive = true; // reset sign for addValue
      } // end else if

      return temp; // return the sum
   } // end method add

   // add two positive HugeIntegers
   public HugeInteger addPositives( HugeInteger addValue )
   {
      HugeInteger temp = new HugeInteger();
      int carry = 0;

      // iterate through HugeIntegers
      for ( int i = 39; i >= 0; i-- )
      {
         temp.integer[ i ] =
            integer[ i ] + addValue.integer[ i ] + carry;

         // determine whether to carry a 1
         if ( temp.integer[ i ] > 9 )
         {
            temp.integer[ i ] %= 10;  // reduce to 0-9
            carry = 1;
         } // end if
         else
            carry = 0;
      } // end for

      // if both are negative, set the result to negative
      if ( !positive )
         temp.positive = false;

      return temp;
   } // end method addPositives

   // subtract two HugeIntegers
   public HugeInteger subtract( HugeInteger subtractValue )
   {
      HugeInteger temp = new HugeInteger();   // temporary result

      // subtractValue is negative
      if ( positive && ( !subtractValue.positive ) )
         temp = addPositives( subtractValue );
      // this HugeInteger is negative
      else if ( !positive && subtractValue.positive )
         temp = addPositives( subtractValue );
      // at this point, both HugeIntegers have the same sign
      else
      {
         boolean isPositive = positive; // original sign
         boolean resultPositive = positive; // sign of the result

         // set both to positive so we can compare absolute values
         positive = true;
         subtractValue.positive = true;

         if ( this.isGreaterThan( subtractValue ) )
            temp = this.subtractPositives( subtractValue );
         else
         {
            temp = subtractValue.subtractPositives( this );
            resultPositive = !isPositive; // flip the sign
         } // end else

         positive = isPositive;
         subtractValue.positive = isPositive;
         temp.positive = resultPositive;
      } // end else

      return temp;
   } // end method subtract

   // subtract two positive HugeIntegers
   public HugeInteger subtractPositives( HugeInteger subtractValue )
   {
      HugeInteger temp = new HugeInteger();

      // iterate through HugeInteger
      for ( int i = 39; i >= 0; i-- )
      {
         // borrow if needed
         if ( integer[i] < subtractValue.integer[i] )
         {
            integer[ i ] += 10;
            subtractValue.integer[ i - 1 ]--;
         } // end if

      temp.integer[ i ] = integer[ i ] - subtractValue.integer[ i ];
      } // end for

      return temp; // return difference of two HugeIntegers
   } // end method subtractPositives

   // find first non-zero position of two HugeIntegers
   public int findFirstNonZeroPosition()
   {
      int position = 39;

      // find first non-zero position for first HugeInteger
      for ( int i = 0; i < DIGITS; i++ )
      {
         if ( integer[ i ] > 0 )
            return i;
      } // end for

      return 39;
   } // end method findFirstNonZeroPosition

   // get string representation of HugeInteger
   public String toString()
   {
      String output = "";

      if ( !positive )
         output = "-";

      // get HugeInteger values without leading zeros
      for ( int i = findFirstNonZeroPosition(); i < DIGITS; i++ )
         output += integer[ i ];

      return output;
   } // end method toHugeIntegerString

   // test if two HugeIntegers are equal
   public boolean isEqualTo( HugeInteger compareValue )
   {
      // compare the sign
      if ( positive != compareValue.positive )
         return false;

      // compare each digit
      for ( int i = 0; i < DIGITS; i++ )
      {
         if ( integer[ i ] != compareValue.integer[ i ] )
            return false;
      } // end for

      return true;
   } // end method isEqualTo

   // test if two HugeIntegers are not equal
   public boolean isNotEqualTo( HugeInteger compareValue )
   {
      return !isEqualTo( compareValue );
   } // end method isNotEqualTo

   // test if one HugeInteger is greater than another
   public boolean isGreaterThan( HugeInteger compareValue )
   {
      // different signs
      if ( positive && ( !compareValue.positive ) )
         return true;
      else if ( !positive && compareValue.positive )
         return false;

      // same sign
      else
      {
         // first number's length is less than second number's length
         if ( findFirstNonZeroPosition() >
            compareValue.findFirstNonZeroPosition() )
         {
            return !positive;
         } // end if

         // first number's length is larger than that of second number
         else if ( findFirstNonZeroPosition() <
            compareValue.findFirstNonZeroPosition() )
         {
            return positive;
         } // end else if

         // two numbers have same length
         else
         {
            for ( int i = 0; i < DIGITS; i++ )
            {
               if ( integer[ i ] > compareValue.integer[ i ] )
                  return positive;
               else if ( integer[ i ] < compareValue.integer[ i ] )
                  return !positive;
            } // end for
         } // end else
      } // end outer if-elseif-else

      return false;
   } // end method isGreatThan

   // test if one HugeInteger is less than another
   public boolean isLessThan( HugeInteger compareValue )
   {
      return !( isGreaterThan( compareValue ) ||
         isEqualTo( compareValue ) );
   } // end method isLessThan

   // test if one HugeInteger is great than or equal to another
   public boolean isGreaterThanOrEqualTo( HugeInteger compareValue )
   {
      return !isLessThan( compareValue );
   } // end method isGreaterThanOrEqualTo

   // test if one HugeInteger is less than or equal to another
   public boolean isLessThanOrEqualTo( HugeInteger compareValue )
```

```java
   {
      return !isGreaterThan( compareValue );
   } // end method isLessThanOrEqualTo

   // test if one HugeInteger is zero
   public boolean isZero()
   {
      // compare each digit
      for ( int i = 0; i < DIGITS; i++ )
      {
         if ( integer[ i ] != 0 )
            return false;
      } // end for

      return true;
   } // end method isZero
} // end class HugeInteger
```

```java
// Exercise 8.18 Solution: HugeIntegerTest.java
// Test class HugeInteger
import java.util.Scanner;

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

      HugeInteger integer1 = new HugeInteger();
      HugeInteger integer2 = new HugeInteger();

      System.out.print( "Enter first HugeInteger: " );
      integer1.input( input.next() );

      System.out.print( "Enter second HugeInteger: " );
      integer2.input( input.next() );

      System.out.printf( "HugeInteger 1: %s\n", integer1.toString() );
      System.out.printf( "HugeInteger 2: %s\n", integer2.toString() );

      HugeInteger result;

      // add two HugeIntegers
      result = integer1.add( integer2 );
      System.out.printf( "Add result: %s\n", result.toString() );

      // subtract two HugeIntegers
      result = integer1.subtract( integer2 );
      System.out.printf( "Subtract result: %s\n", result.toString() );

      // compare two HugeIntegers
      System.out.printf(
         "HugeInteger 1 is zero: %b\n", integer1.isZero() );
      System.out.printf(
         "HugeInteger 2 is zero: %b\n", integer2.isZero() );
      System.out.printf(
         "HugeInteger 1 is equal to HugeInteger 2: %b\n",
         integer1.isEqualTo( integer2 ) );
      System.out.printf(
         "HugeInteger 1 is not equal to HugeInteger 2: %b\n",
         integer1.isNotEqualTo( integer2 ) );
      System.out.printf(
         "HugeInteger 1 is greater than HugeInteger 2: %b\n",
         integer1.isGreaterThan( integer2 ) );
      System.out.printf(
         "HugeInteger 1 is less than HugeInteger 2: %b\n",
         integer1.isLessThan( integer2 ) );
      System.out.printf(
         "HugeInteger 1 is greater than or equal to HugeInteger 2: %b\n",
         integer1.isGreaterThanOrEqualTo( integer2 ) );
      System.out.printf(
         "HugeInteger 1 is less than or equal to HugeInteger 2: %b\n",
         integer1.isLessThanOrEqualTo( integer2 ) );
   } // end main
} // end class HugeIntegerTest
```

```
Enter first HugeInteger: 123456789123456789
Enter second HugeInteger: 987654321987654321
HugeInteger 1: 123456789123456789
HugeInteger 2: 987654321987654321
Add result: 1111111111111111110
Subtract result: -864197532864197532
HugeInteger 1 is zero: false
HugeInteger 2 is zero: false
HugeInteger 1 is equal to HugeInteger 2: false
HugeInteger 1 is not equal to HugeInteger 2: true
HugeInteger 1 is greater than HugeInteger 2: false
HugeInteger 1 is less than HugeInteger 2: true
HugeInteger 1 is greater than or equal to HugeInteger 2: false
HugeInteger 1 is less than or equal to HugeInteger 2: true
```

8.19 *(Tic-Tac-Toe)* Create a class `TicTacToe` that will enable you to write a complete program to play the game of Tic-Tac-Toe. The class contains a private 3-by-3 two-dimensional array of integers. The constructor should initialize the empty board to all zeros. Allow two human players. Wherever the first player moves, place a 1 in the specified square, and place a 2 wherever the second player moves. Each move must be to an empty square. After each move determine whether the game has been won and whether it is a draw. If you feel ambitious, modify your program so that the computer makes the moves for one of the players. Also, allow the player to specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop a program that will play three-dimensional Tic-Tac-Toe on a 4-by-4-by-4 board [*Note:* This is a challenging project that could take many weeks of effort!].

ANS:

```java
// Exercise 8.19 Solution: TicTacToe.java
// Program that plays the game of tic-tac-toe.
import java.util.Scanner;
```

```java

public class TicTacToe
{
   private final int BOARDSIZE = 3; // size of the board
   private enum Status { WIN, DRAW, CONTINUE }; // game states
   private int board[][]; // board representation
   private boolean firstPlayer; // whether it's player 1's move
   private boolean gameOver; // whether

   // Constructor
   public TicTacToe()
   {
      board = new int[ BOARDSIZE ][ BOARDSIZE ];
      firstPlayer = true;
      gameOver = false;
   } // end Constructor

   // start game
   public void play()
   {
      Scanner input = new Scanner( System.in );
      int row; // row for next move
      int column; // column for next move

      System.out.println( "Player 1's turn." );

      while ( !gameOver )
      {
         int player = ( firstPlayer ? 1 : 2 );
         // first player's turn

         do
         {
            System.out.printf(
               "Player %d: Enter row ( 0 <= row < 3 ): ", player );
            row = input.nextInt();
            System.out.printf(
               "Player %d: Enter column ( 0 <= row < 3 ): ", player );
            column = input.nextInt();
         } while ( !validMove( row, column ) );

         board[ row ][ column ] = player;

         firstPlayer = !firstPlayer;

         printBoard();
         printStatus( player );
      } // end while
   } // end method makeMove

   // show game status in status bar
   public void printStatus( int player )
   {
      Status status = gameStatus();

      // check game status
      switch ( status )
      {
         case WIN:
            System.out.println( "Player " + player + " wins." );
            gameOver = true;
            break;

         case DRAW:
            System.out.println( "Game is a draw." );
            gameOver = true;
            break;

         case CONTINUE:
            if ( player == 1 )
               System.out.println( "Player 2's turn." );
            else
               System.out.println( "Player 1's turn." );
            break;
      } // end switch
   } // end method printStatus

   // get game status
   public Status gameStatus()
   {
      int a;

      // check for a win on diagonals
      if ( board[ 0 ][ 0 ] != 0 && board[ 0 ][ 0 ] == board[ 1 ][ 1 ] &&
         board[ 0 ][ 0 ] == board[ 2 ][ 2 ] )
         return Status.WIN;
      else if ( board[ 2 ][ 0 ] != 0 && board[ 2 ][ 0 ] ==
         board[ 1 ][ 1 ] && board[ 2 ][ 0 ] == board[ 0 ][ 2 ] )
         return Status.WIN;

      // check for win in rows
      for ( a = 0; a < 3; a++ )
         if ( board[ a ][ 0 ] != 0 && board[ a ][ 0 ] ==
              board[ a ][ 1 ] && board[ a ][ 0 ] == board[ a ][ 2 ] )
            return Status.WIN;

      // check for win in columns
      for ( a = 0; a < 3; a++ )
         if ( board[ 0 ][ a ] != 0 && board[ 0 ][ a ] ==
              board[ 1 ][ a ] && board[ 0 ][ a ] == board[ 2 ][ a ] )
            return Status.WIN;

      // check for a completed game
      for ( int r = 0; r < 3; r++ )
         for ( int c = 0; c < 3; c++ )
            if ( board[ r ][ c ] == 0 )
               return Status.CONTINUE; // game is not finished

      return Status.DRAW; // game is a draw
   } // end method gameStatus

```

```java
   // display board
   public void printBoard()
   {
      System.out.println( " _______________________ " );

      for ( int row = 0; row < BOARDSIZE; row++ )
      {
         System.out.println( "|       |       |       |" );

         for ( int column = 0; column < BOARDSIZE; column++ )
            printSymbol( column, board[ row ][ column ] );

         System.out.println( "|_______|_______|_______|" );
      } // end for
   } // end method printBoard

   // print moves
   public void printSymbol( int column, int player )
   {
      switch ( player )
      {
         case 0:
            System.out.print( "|       " );
            break;

         case 1:
            System.out.print( "|   1   " );
            break;

         case 2:
            System.out.print( "|   2   " );
            break;
      } // end switch

      if ( column == 2 )
         System.out.println( "|" );
   } // end method printSymbol

   // validate move
   public boolean validMove( int row, int column )
   {
      return row >= 0 && row < 3 && column >= 0 && column < 3 &&
         board[ row ][ column ] == 0;
   } // end method validMove
} // end class TicTacToe
```

```java
// Exercise 8.19 Solution: TicTacToeTest.java
// play a game of Tic Tac Toe
public class TicTacToeTest
{
   public static void main( String args[] )
   {
      TicTacToe game = new TicTacToe();
      game.printBoard();
```

```
        game.play();
    } // end main
} // end class TicTacToeTest
```

```
 _______________________
|       |       |       |
|       |       |       |
|_______|_______|_______|
|       |       |       |
|       |       |       |
|_______|_______|_______|
|       |       |       |
|       |       |       |
|_______|_______|_______|
Player 1's turn.
Player 1: Enter row ( 0 <= row < 3 ): 0
Player 1: Enter column ( 0 <= row < 3 ): 0
 _______________________
|       |       |       |
|   1   |       |       |
|_______|_______|_______|
|       |       |       |
|       |       |       |
|_______|_______|_______|
|       |       |       |
|       |       |       |
|_______|_______|_______|
Player 2's turn.
Player 2: Enter row ( 0 <= row < 3 ): 0
Player 2: Enter column ( 0 <= row < 3 ): 1
 _______________________
|       |       |       |
|   1   |   2   |       |
|_______|_______|_______|
|       |       |       |
|       |       |       |
|_______|_______|_______|
|       |       |       |
|       |       |       |
|_______|_______|_______|
Player 1's turn.
Player 1: Enter row ( 0 <= row < 3 ): 1
Player 1: Enter column ( 0 <= row < 3 ): 1
 _______________________
|       |       |       |
|   1   |   2   |       |
|_______|_______|_______|
|       |       |       |
|       |   1   |       |
|_______|_______|_______|
|       |       |       |
|       |       |       |
|_______|_______|_______|
Player 2's turn.
Player 2: Enter row ( 0 <= row < 3 ): 2
Player 2: Enter column ( 0 <= row < 3 ): 2
```

```
 _______________________
|       |       |       |
|   1   |   2   |       |
|_______|_______|_______|
|       |       |       |
|       |   1   |       |
|_______|_______|_______|
|       |       |       |
|       |       |   2   |
|_______|_______|_______|
Player 1's turn.
Player 1: Enter row ( 0 <= row < 3 ): 2
Player 1: Enter column ( 0 <= row < 3 ): 0
 _______________________
|       |       |       |
|   1   |   2   |       |
|_______|_______|_______|
|       |       |       |
|       |   1   |       |
|_______|_______|_______|
|       |       |       |
|   1   |       |   2   |
|_______|_______|_______|
Player 2's turn.
Player 2: Enter row ( 0 <= row < 3 ): 0
Player 2: Enter column ( 0 <= row < 3 ): 2
 _______________________
|       |       |       |
|   1   |   2   |   2   |
|_______|_______|_______|
|       |       |       |
|       |   1   |       |
|_______|_______|_______|
|       |       |       |
|   1   |       |   2   |
|_______|_______|_______|
Player 1's turn.
Player 1: Enter row ( 0 <= row < 3 ): 1
Player 1: Enter column ( 0 <= row < 3 ): 0
 _______________________
|       |       |       |
|   1   |   2   |   2   |
|_______|_______|_______|
|       |       |       |
|   1   |   1   |       |
|_______|_______|_______|
|       |       |       |
|   1   |       |   2   |
|_______|_______|_______|
Player 1 wins.
```

(Optional) GUI and Graphics Case Study

8.1 Extend the program in Figs. 8.21–8.23 to randomly draw rectangles and ovals. Create classes `MyRectangle` and `MyOval`. Both of these classes should include *x1*, *y1*, *x2*, *y2* coordinates, a color and a `boolean` flag to determine whether the shape is a filled shape. Declare a constructor in each class with arguments for initializing all the instance variables. To help draw rectangles and ovals, each class should provide methods `getUpperLeftX`, `getUpperLeftY`, `getWidth` and `getHeight` that calculate the upper-left *x*-coordinate, upper-left *y*-coordinate, width and height, respectively. The upper-left *x*-coordinate is the smaller of the two *x*-coordinate values, the upper-left *y*-coordinate is the smaller of the two *y*-coordinate values, the width is the absolute value of the difference between the two *x*-coordinate values, and the height is the absolute value of the difference between the two *y*-coordinate values.

Class `DrawPanel`, which extends `JPanel` and handles the creation of the shapes, should declare three arrays, one for each shape type. The length of each array should be a random number between 1 and 5. The constructor of class `DrawPanel` will fill each of the arrays with shapes of random position, size, color and fill.

In addition, modify all three shape classes to include the following:

a) A constructor with no arguments that sets all the coordinates of the shape to `0`, the color of the shape to `Color.BLACK`, and the filled property to `false` (`MyRect` and `MyOval` only).

b) *Set* methods for the instance variables in each class. The methods that set a coordinate value should verify that the argument is greater than or equal to zero before setting the coordinate—if it is not, they should set the coordinate to zero. The constructor should call the *set* methods rather than initialize the local variables directly.

c) *Get* methods for the instance variables in each class. Method `draw` should reference the coordinates by the *get* methods rather than access them directly.

ANS:

```java
// GCS Exercise 8.1 Solution: MyLine.java
// Declaration of class MyLine
import java.awt.Color;
import java.awt.Graphics;

public class MyLine
{
   private int x1; // x coordinate of first endpoint
   private int y1; // y coordinate of first endpoint
   private int x2; // x coordinate of second endpoint
   private int y2; // y coordinate of second endpoint
   private Color myColor; // color of this shape

   // constructor initializes private vars with default values
   public MyLine()
   {
      this( 0, 0, 0, 0, Color.BLACK ); // call constructor to set values
   } // end MyLine no-argument constructor

   // constructor with input values
   public MyLine( int x1, int y1, int x2, int y2, Color color )
   {
      setX1( x1 ); // set x coordinate of first endpoint
      setY1( y1 ); // set y coordinate of first endpoint
      setX2( x2 ); // set x coordinate of second endpoint
      setY2( y2 ); // set y coordinate of second endpoint
      setColor( color ); // set the color
   } // end MyLine constructor

   // set the x-coordinate of the first point
   public void setX1( int x1 )
   {
     this.x1 = ( x1 >= 0 ? x1 : 0 );
   } // end method setX1

   // get the x-coordinate of the first point
   public int getX1()
   {
     return x1;
   } // end method getX1

   // set the x-coordinate of the second point
   public void setX2( int x2 )
   {
     this.x2 = ( x2 >= 0 ? x2 : 0 );
   } // end method setX2

   // get the x-coordinate of the second point
   public int getX2()
   {
     return x2;
   } // end method getX2

   // set the y-coordinate of the first point
   public void setY1( int y1 )
   {
     this.y1 = ( y1 >= 0 ? y1 : 0 );
   } // end method setY1

   // get the y-coordinate of the first point
   public int getY1()
   {
     return y1;
   } // end method getY1

   // set the y-coordinate of the second point
   public void setY2( int y2 )
   {
     this.y2 = ( y2 >= 0 ? y2 : 0 );
   } // end method setY2

   // get the y-coordinate of the second point
   public int getY2()
   {
     return y2;
   } // end method getY2

   // set the color
   public void setColor( Color color )
   {
      myColor = color;
   } // end method setColor

   // get the color
   public Color getColor()
   {
      return myColor;
   } // end method getColor

   // draw the line in the specified color
   public void draw( Graphics g )
   {
      g.setColor( getColor() );
      g.drawLine( getX1(), getY1(), getX2(), getY2() );
   } // end method draw
} // end class MyLine
```

```
// GCS Exercise 8.1 Solution: MyOval.java
// Declaration of class MyOval
import java.awt.Color;
import java.awt.Graphics;

public class MyOval
{
   private int x1; // x coordinate of first endpoint
   private int y1; // y coordinate of first endpoint
   private int x2; // x coordinate of second endpoint
   private int y2; // y coordinate of second endpoint
   private Color myColor; // Color of this oval
   private boolean filled; // whether this shape is filled

   // constructor initializes private vars with default values
   public MyOval()
   {
      this( 0, 0, 0, 0, Color.BLACK, false ); // call constructor
   } // end MyOval no-argument constructor

   // constructor with input values
   public MyOval( int x1, int y1, int x2, int y2,
      Color color, boolean isFilled )
   {
      setX1( x1 ); // set x coordinate of first endpoint
      setY1( y1 ); // set y coordinate of first endpoint
      setX2( x2 ); // set x coordinate of second endpoint
      setY2( y2 ); // set y coordinate of second endpoint
      setColor( color ); // set the color
      setFilled( isFilled );
   } // end MyOval constructor

   // set the x-coordinate of the first point
   public void setX1( int x1 )
   {
     this.x1 = ( x1 >= 0 ? x1 : 0 );
   } // end method setX1

   // get the x-coordinate of the first point
   public int getX1()
   {
     return x1;
   } // end method getX1

   // set the x-coordinate of the second point
   public void setX2( int x2 )
   {
     this.x2 = ( x2 >= 0 ? x2 : 0 );
   } // end method setX2

   // get the x-coordinate of the second point
   public int getX2()
   {
     return x2;
   } // end method getX2

   // set the y-coordinate of the first point
   public void setY1( int y1 )
   {
     this.y1 = ( y1 >= 0 ? y1 : 0 );
   } // end method setY1

   // get the y-coordinate of the first point
   public int getY1()
   {
     return y1;
   } // end method getY1

   // set the y-coordinate of the second point
   public void setY2( int y2 )
   {
     this.y2 = ( y2 >= 0 ? y2 : 0 );
   } // end method setY2

   // get the y-coordinate of the second point
   public int getY2()
   {
     return y2;
   } // end method getY2

   // set the color
   public void setColor( Color color )
   {
      myColor = color;
   } // end method setColor

   // get the color
   public Color getColor()
   {
      return myColor;
   } // end method getColor

   // get upper left x coordinate
   public int getUpperLeftX()
   {
      return Math.min( getX1(), getX2() );
   } // end method getUpperLeftX

   // get upper left y coordinate
   public int getUpperLeftY()
   {
      return Math.min( getY1(), getY2() );
   } // end method getUpperLeftY

   // get shape width
   public int getWidth()
   {
      return Math.abs( getX2() - getX1() );
   } // end method getWidth

   // get shape height
   public int getHeight()
   {
      return Math.abs( getY2() - getY1() );
   } // end method getHeight

   // determines whether this shape is filled
   public boolean isFilled()
   {
      return filled;
   } // end method is filled

   // sets whether this shape is filled
   public void setFilled( boolean isFilled )
   {
      filled = isFilled;
   } // end method setFilled

   // draws an oval in the specified color
   public void draw( Graphics g )
   {
      g.setColor( getColor() );

      if ( isFilled() )
         g.fillOval( getUpperLeftX(), getUpperLeftY(),
            getWidth(), getHeight() );
      else
         g.drawOval( getUpperLeftX(), getUpperLeftY(),
            getWidth(), getHeight() );
   } // end method draw
} // end class MyOval
```

```java
// GCS Exercise 8.1 Solution: MyRect.java
// Declaration of class MyRect
import java.awt.Color;
import java.awt.Graphics;

public class MyRect
{
   private int x1; // x coordinate of first endpoint
   private int y1; // y coordinate of first endpoint
   private int x2; // x coordinate of second endpoint
   private int y2; // y coordinate of second endpoint
   private Color myColor; // Color of this rectangle
   private boolean filled; // whether this shape is filled

   // constructor initializes private vars with default values
   public MyRect()
   {
      this( 0, 0, 0, 0, Color.BLACK, false ); // call constructor
   } // end MyRect no-argument constructor

   // constructor with input values
   public MyRect( int x1, int y1, int x2, int y2,
      Color color, boolean isFilled )
   {
      setX1( x1 ); // set x coordinate of first endpoint
      setY1( y1 ); // set y coordinate of first endpoint
      setX2( x2 ); // set x coordinate of second endpoint
      setY2( y2 ); // set y coordinate of second endpoint
      setColor( color ); // set the color
      setFilled( isFilled );
   } // end MyRect constructor

   // set the x-coordinate of the first point
   public void setX1( int x1 )
   {
     this.x1 = ( x1 >= 0 ? x1 : 0 );
   } // end method setX1

   // get the x-coordinate of the first point
   public int getX1()
   {
     return x1;
   } // end method getX1

   // set the x-coordinate of the second point
   public void setX2( int x2 )
   {
     this.x2 = ( x2 >= 0 ? x2 : 0 );
   } // end method setX2

   // get the x-coordinate of the second point
   public int getX2()
   {
     return x2;
   } // end method getX2

   // set the y-coordinate of the first point
   public void setY1( int y1 )
   {
     this.y1 = ( y1 >= 0 ? y1 : 0 );
   } // end method setY1

   // get the y-coordinate of the first point
   public int getY1()
   {
     return y1;
   } // end method getY1

   // set the y-coordinate of the second point
   public void setY2( int y2 )
   {
     this.y2 = ( y2 >= 0 ? y2 : 0 );
   } // end method setY2

   // get the y-coordinate of the second point
   public int getY2()
   {
     return y2;
   } // end method getY2

   // set the color
   public void setColor( Color color )
   {
      myColor = color;
   } // end method setColor

   // get the color
   public Color getColor()
   {
      return myColor;
   } // end method getColor

   // get upper left x coordinate
   public int getUpperLeftX()
   {
      return Math.min( getX1(), getX2() );
   } // end method getUpperLeftX

   // get upper left y coordinate
   public int getUpperLeftY()
   {
      return Math.min( getY1(), getY2() );
   } // end method getUpperLeftY

   // get shape width
   public int getWidth()
   {
      return Math.abs( getX2() - getX1() );
   } // end method getWidth

   // get shape height
   public int getHeight()
   {
      return Math.abs( getY2() - getY1() );
   } // end method getHeight

   // determines whether this shape is filled
   public boolean isFilled()
   {
      return filled;
   } // end method is filled

   // sets whether this shape is filled
   public void setFilled( boolean isFilled )
   {
      filled = isFilled;
   } // end method setFilled

   // draws a rectangle in the specified color
   public void draw( Graphics g )
   {
      g.setColor( getColor() );

      if ( isFilled() )
         g.fillRect( getUpperLeftX(), getUpperLeftY(),
            getWidth(), getHeight() );
      else
         g.drawRect( getUpperLeftX(), getUpperLeftY(),
            getWidth(), getHeight() );
   } // end method draw
} // end class MyRect
```

```
// GCS Exercise 8.1 Solution: DrawPanel.java
// Program that uses classes MyLine, MyOval and MyRect to draw
// random shapes
import java.awt.Color;
import java.awt.Graphics;
import java.util.Random;
import javax.swing.JPanel;

public class DrawPanel extends JPanel
{
   private Random randomNumbers = new Random();

   private MyLine lines[]; // array on lines
   private MyOval ovals[]; // array of ovals
   private MyRect rectangles[]; // array of rectangles

   // constructor, creates a panel with random shapes
   public DrawPanel()
   {
```

```java
      setBackground( Color.WHITE );

      lines = new MyLine[ 1 + randomNumbers.nextInt( 5 ) ];
      ovals = new MyOval[ 1 + randomNumbers.nextInt( 5 ) ];
      rectangles = new MyRect[ 1 + randomNumbers.nextInt( 5 ) ];

      // create lines
      for ( int count = 0; count < lines.length; count++ )
      {
         // generate random coordinates
         int x1 = randomNumbers.nextInt( 450 );
         int y1 = randomNumbers.nextInt( 450 );
         int x2 = randomNumbers.nextInt( 450 );
         int y2 = randomNumbers.nextInt( 450 );

         // generate a random color
         Color color = new Color( randomNumbers.nextInt( 256 ),
            randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );

         // add the line to the list of lines to be displayed
         lines[ count ] = new MyLine( x1, y1, x2, y2, color );
      } // end for

      // create ovals
      for ( int count = 0; count < ovals.length; count++ )
      {
         // generate random coordinates
         int x1 = randomNumbers.nextInt( 450 );
         int y1 = randomNumbers.nextInt( 450 );
         int x2 = randomNumbers.nextInt( 450 );
         int y2 = randomNumbers.nextInt( 450 );

         // generate a random color
         Color color = new Color( randomNumbers.nextInt( 256 ),
            randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );

         // get filled property
         boolean filled = randomNumbers.nextBoolean();

         // add the line to the oval of ovals to be displayed
         ovals[ count ] = new MyOval( x1, y1, x2, y2, color, filled );
      } // end for

      // create rectangles
      for ( int count = 0; count < rectangles.length; count++ )
      {
         // generate random coordinates
         int x1 = randomNumbers.nextInt( 450 );
         int y1 = randomNumbers.nextInt( 450 );
         int x2 = randomNumbers.nextInt( 450 );
         int y2 = randomNumbers.nextInt( 450 );

         // generate a random color
         Color color = new Color( randomNumbers.nextInt( 256 ),
            randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );

         // get filled property
         boolean filled = randomNumbers.nextBoolean();

         // add the rectangle to the list of rectangles to be displayed
         rectangles[ count ] =
            new MyRect( x1, y1, x2, y2, color, filled );
      } // end for
   } // end DrawPanel constructor

   // for each shape array, draw the individual shapes
   public void paintComponent( Graphics g )
   {
      super.paintComponent( g );

      // draw the lines
      for ( MyLine line : lines )
         line.draw( g );

      // draw the ovals
      for ( MyOval oval: ovals )
         oval.draw( g );

      // drat the rectangles
      for ( MyRect rectangle : rectangles )
         rectangle.draw( g );
   } // end method paintComponent
} // end class DrawPanel
```

```
// GCS Exercise 8.1 Solution: TestDraw.java
// Test application to display a DrawPanel
import javax.swing.JFrame;

public class TestDraw
{
   public static void main( String args[] )
   {
      DrawPanel panel = new DrawPanel();
      JFrame application = new JFrame();

      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
      application.add( panel );
      application.setSize( 500, 500 );
      application.setVisible( true );
   } // end main
} // end class TestDraw
```