8

Classes and Objects: A Deeper Look

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:

```
1  // Exercise 8.4 Solution: Rectangle.java
2  // Definition of class Rectangle
3
4  public class Rectangle
5  {
6     private double length; // the length of the rectangle
7     private double width; // the width of the rectangle
8
9     // constructor without parameters
10    public Rectangle()
11    {
12       setLength( 1.0 );
13       setWidth( 1.0 );
14    } // end Rectangle no-argument constructor
15
```

```
16    // constructor with length and width supplied
17    public Rectangle( double theLength, double theWidth )
18    {
19       setLength( theLength );
20       setWidth( theWidth );
21    } // end Rectangle two-argument constructor
22
23    // validate and set length
24    public void setLength( double theLength )
25    {
26       length = ( theLength > 0.0 && theLength < 20.0 ? theLength : 1.0 );
27    } // end method setLength
28
29    // validate and set width
30    public void setWidth( double theWidth )
31    {
32       width = ( theWidth > 0 && theWidth < 20.0 ? theWidth : 1.0 );
33    } // end method setWidth
34
35    // get value of length
36    public double getLength()
37    {
38       return length;
39    } // end method getLength
40
41    // get value of width
42    public double getWidth()
43    {
44       return width;
45    } // end method getWidth
46
47    // calculate rectangle's perimeter
48    public double perimeter()
49    {
50       return 2 * length + 2 * width;
51    } // end method perimeter
52
53    // calculate rectangle's area
54    public double area()
55    {
56       return length * width;
57    } // end method area
58
59    // convert to String
60    public String toString()
61    {
62       return String.format( "%s: %f\n%s: %f\n%s: %f\n%s: %f",
63          "Length", length, "Width", width,
64          "Perimeter", perimeter(), "Area", area() );
65    } // end method toRectangleString
66 } // end class Rectangle
```

```
1  // Exercise 8.4 Solution: RectangleTest.java
2  // Program tests class Rectangle.
3  import java.util.Scanner;
4
5  public class RectangleTest
6  {
7     public static void main( String args[] )
8     {
9        Scanner input = new Scanner( System.in );
10
11       Rectangle rectangle = new Rectangle();
12
13       int choice = getMenuChoice();
14
15       while ( choice != 3 )
16       {
17          switch ( choice )
18          {
19             case 1:
20                System.out.print( "Enter length: " );
21                rectangle.setLength( input.nextDouble() );
22                break;
23
24             case 2:
25                System.out.print ( "Enter width: " );
26                rectangle.setWidth( input.nextDouble() );
27                break;
28          } // end switch
29
30          System.out.println ( rectangle.toString() );
31
32          choice = getMenuChoice();
33       } // end while
34    } // end main
35
36    // prints a menu and returns a value coressponding to the menu choice
37    private static int getMenuChoice()
38    {
39       Scanner input = new Scanner( System.in );
40
41       System.out.println( "1. Set Length" );
42       System.out.println( "2. Set Width" );
43       System.out.println( "3. Exit" );
44       System.out.print( "Choice: " );
45
46       return input.nextInt();
47    } // end method getMenuChoice
48 } // 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:

```
1  // Exercise 8.5 Solution: Time2.java
2  // Time2 class definition maintains the time in 24-hour format.
3
4  public class Time2
5  {
6     private int totalSeconds;
7
8     // no-argument constructor initializes totalSeconds to zero;
9     public Time2()
10    {
11       totalSeconds = 0;
12    } // end no-argument constructor
13
14    // Time2 constructor: hour supplied, minute and second defaulted to 0
```

```
15      public Time2( int h )
16      {
17         setTime( h, 0, 0 );
18      } // end hour constructor
19
20      // Time2 constructor: hour and minute supplied, second defaulted to 0
21      public Time2( int h, int m )
22      {
23         setTime( h, m, 0 );
24      } // end hour and minute constructor
25
26      // Time2 constructor: hour, minute and second supplied
27      public Time2( int h, int m, int s )
28      {
29         setTime( h, m, s );
30      } // end hour, minute and second constructor
31
32      // Time2 constructor: another Time2 object supplied
33      public Time2( Time2 time )
34      {
35         setTime( time.getHour(), time.getMinute(), time.getSecond() );
36      } // end Time2 constructor
37
38      // set a new time value using total seconds; perform
39      // validity checks on data; set invalid values to zero
40      public void setTime( int h, int m, int s )
41      {
42         setHour( h );
43         setMinute( m );
44         setSecond( s );
45      } // end method setTime
46
47      // set hour value
48      public void setHour( int h )
49      {
50         int hours = ( h >= 0 && h < 24 ) ? h : 0;
51
52         totalSeconds = ( hours * 3600 ) + ( getMinute() * 60 ) +
53            getSecond();
54      } // end method setHour
55
56      // set minute value
57      public void setMinute( int m )
58      {
59         int minutes = ( m >= 0 && m < 60 ) ? m : 0;
60
61         totalSeconds = ( getHour() * 3600 ) + ( minutes * 60 ) +
62            getSecond();
63      } // end method setMinute
64
65      // set second value
66      public void setSecond( int s )
67      {
68         int seconds = ( s >= 0 && s < 60 ) ? s : 0;
```

```
69
70          totalSeconds = ( getHour() * 3600 ) + ( getMinute() *60 ) + seconds;
71       } // end method setSecond
72
73       // get hour value
74       public int getHour()
75       {
76          return ( totalSeconds / 3600 );
77       } // end method getHour
78
79       // get minute value
80       public int getMinute()
81       {
82          return ( ( totalSeconds % 3600 ) / 60 );
83       } // end method getMinute
84
85       // get second value
86       public int getSecond()
87       {
88          return ( ( totalSeconds % 3600 ) % 60 );
89       } // end method getSecond
90
91       // convert to String in universal-time format (HH:MM:SS)
92       public String toUniversalString()
93       {
94          return String.format(
95             "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
96       } // end method toUniversalString
97
98       // convert to String in standard-time format (H:MM:SS AM or PM)
99       public String toString()
100      {
101         return String.format( "%d:%02d:%02d %s",
102            ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
103            getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
104      } // end method toStandardString
105   } // end class Time2
```

```
1  // Exercise 8.5 Solution: Time2Test.java
2  // Class TimeTest3 to test class Time2
3
4  public class Time2Test
5  {
6     public static void main( String args[] )
7     {
8        Time2 t = new Time2();
9
10       System.out.print( "The initial universal time is: " );
11       System.out.println( t.toUniversalString() );
12       System.out.print( "The initial standard time is: " );
13       System.out.println( t.toString() );
14
15       t.setTime( 13, 27, 6 );
```

```
16         System.out.print( "\nUniversal time after setTime is: " );
17         System.out.println( t.toUniversalString() );
18         System.out.print( "Standard time after setTime is: " );
19         System.out.println( t.toString() );
20
21         t.setTime( 99, 99, 99 );
22         System.out.println( "\nAfter attempting invalid settings:" );
23         System.out.print( "The initial universal time is: " );
24         System.out.println( t.toUniversalString() );
25         System.out.print( "The initial standard time is: " );
26         System.out.println( t.toString() );
27      } // end main
28   } // 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:

```
1  // Exercise 8.6 Solution: SavingAccount
2  // SavingAccount class definition
3
4  public class SavingAccount
5  {
6     // interest rate for all accounts
7     private static double annualInterestRate = 0;
8
9     private double savingsBalance; // balance for currrent account
10
11    // constructor, creates a new account with the specified balance
12    public SavingAccount( double balance )
13    {
14       savingsBalance = balance;
```

```
15     } // end constructor
16
17     // get monthly interest
18     public void calculateMonthlyInterest()
19     {
20        savingsBalance += savingsBalance * ( annualInterestRate / 12.0 );
21     } // end method calculateMonthlyInterest
22
23     // modify interest rate
24     public static void modifyInterestRate( double newRate )
25     {
26        annualInterestRate =
27           ( newRate >= 0 && newRate <= 1.0 ) ? newRate : 0.04;
28     } // end method modifyInterestRate
29
30     // get string representation of SavingAccount
31     public String toString()
32     {
33        return String.format( "$%.2f", savingsBalance );
34     } // end method toSavingAccountString
35  } // end class SavingAccount
```

```
1  // Exercise 8.6 Solution: SavingAccountTest.java
2  // Program that tests SavingAccount class
3
4  public class SavingAccountTest
5  {
6     public static void main( String args[] )
7     {
8        SavingAccount saver1 = new SavingAccount( 2000 );
9        SavingAccount saver2 = new SavingAccount( 3000 );
10       SavingAccount.modifyInterestRate( 0.04 );
11
12       System.out.println( "Monthly balances for one year at .04" );
13       System.out.println( "Balances:" );
14
15       System.out.printf( "%20s%10s\n", "Saver 1", "Saver 2" );
16       System.out.printf( "%-10s%10s%10s\n", "Base",
17          saver1.toString(), saver2.toString() );
18
19       for ( int month = 1; month <= 12; month++ )
20       {
21          String monthLabel = String.format( "Month %d:", month );
22          saver1.calculateMonthlyInterest();
23          saver2.calculateMonthlyInterest();
24
25          System.out.printf( "%-10s%10s%10s\n", monthLabel,
26             saver1.toString(), saver2.toString() );
27       } // end for
28
29       SavingAccount.modifyInterestRate( .05 );
30       saver1.calculateMonthlyInterest();
31       saver2.calculateMonthlyInterest();
```

```
32
33        System.out.println( "\nAfter setting interest rate to .05" );
34        System.out.println( "Balances:" );
35        System.out.printf( "%-10s%10s\n", "Saver 1", "Saver 2" );
36        System.out.printf( "%-10s%10s\n",
37           saver1.toString(),  saver2.toString() );
38     } // end main
39  } // 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:

```
1  // Exercise 8.7 Solution: Time2.java
2  // Time2 class definition with methods tick,
3  // incrementMinute and incrementHour.
4
5  public class Time2
6  {
7     private int hour; // 0 - 23
8     private int minute; // 0 - 59
9     private int second; // 0 - 59
10
```

```
11    // Time2 no-argument constructor: initializes each instance variable
12    // to zero; ensures that Time2 objects start in a consistent state
13    public Time2()
14    {
15       this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
16    } // end Time2 no-argument constructor
17
18    // Time2 constructor: hour supplied, minute and second defaulted to 0
19    public Time2( int h )
20    {
21       this( h, 0, 0 ); // invoke Time2 constructor with three arguments
22    } // end Time2 one-argument constructor
23
24    // Time2 constructor: hour and minute supplied, second defaulted to 0
25    public Time2( int h, int m )
26    {
27       this( h, m, 0 ); // invoke Time2 constructor with three arguments
28    } // end Time2 two-argument constructor
29
30    // Time2 constructor: hour, minute and second supplied
31    public Time2( int h, int m, int s )
32    {
33       setTime( h, m, s ); // invoke setTime to validate time
34    } // end Time2 three-argument constructor
35
36    // Time2 constructor: another Time2 object supplied
37    public Time2( Time2 time )
38    {
39       // invoke Time2 constructor with three arguments
40       this( time.getHour(), time.getMinute(), time.getSecond() );
41    } // end Time2 constructor with Time2 argument
42
43    // Set Methods
44    // set a new time value using universal time; perform
45    // validity checks on data; set invalid values to zero
46    public void setTime( int h, int m, int s )
47    {
48       setHour( h ); // set the hour
49       setMinute( m ); // set the minute
50       setSecond( s ); // set the second
51    } // end method setTime
52
53    // validate and set hour
54    public void setHour( int h )
55    {
56       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
57    } // end method setHour
58
59    // validate and set minute
60    public void setMinute( int m )
61    {
62       minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
63    } // end method setMinute
64
```

```
65      // validate and set second
66      public void setSecond( int s )
67      {
68         second = ( ( s >= 0 && s < 60 ) ? s : 0 );
69      } // end method setSecond
70
71      // Get Methods
72      // get hour value
73      public int getHour()
74      {
75         return hour;
76      } // end method getHour
77
78      // get minute value
79      public int getMinute()
80      {
81         return minute;
82      } // end method getMinute
83
84      // get second value
85      public int getSecond()
86      {
87         return second;
88      } // end method getSecond
89
90      // Tick the time by one second
91      public void tick()
92      {
93         setSecond( second + 1 );
94
95         if ( second == 0 )
96            incrementMinute();
97      } // end method tick
98
99      // Increment the minute
100     public void incrementMinute()
101     {
102        setMinute( minute + 1 );
103
104        if ( minute == 0 )
105           incrementHour();
106     } // end method incrementMinute
107
108     // Increment the hour
109     public void incrementHour()
110     {
111        setHour( hour + 1 );
112     } // end method incrementHour
113
114     // convert to String in universal-time format (HH:MM:SS)
115     public String toUniversalString()
116     {
117        return String.format(
118           "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
```

```
119    } // end method toUniversalString
120
121    // convert to String in standard-time format (H:MM:SS AM or PM)
122    public String toString()
123    {
124       return String.format( "%d:%02d:%02d %s",
125          ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
126          getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
127    } // end method toStandardString
128 } // end class Time2
```

```
1  // Exercise 8.7 Solution: Time2Test.java
2  // Demonstrating the Time2 class set and get methods
3  import java.util.Scanner;
4
5  public class Time2Test
6  {
7     public static void main( String args[] )
8     {
9        Scanner input = new Scanner( System.in );
10
11       Time2 time = new Time2();
12
13       System.out.println( "Enter the time" );
14       System.out.print( "Hours: " );
15       time.setHour( input.nextInt() );
16       System.out.print( "Minutes: " );
17       time.setMinute( input.nextInt() );
18       System.out.print( "Seconds: " );
19       time.setSecond( input.nextInt() );
20
21       int choice = getMenuChoice();
22
23       while ( choice != 5 )
24       {
25          switch ( choice )
26          {
27             case 1: // add 1 second
28                time.tick();
29                break;
30
31             case 2: // add 1 minute
32                time.incrementMinute();
33                break;
34
35             case 3: // and 1 hour
36                time.incrementHour();
37                break;
38
39             case 4: // add arbitrary seconds
40                System.out.print( "Enter seconds to tick: " );
41                int ticks = input.nextInt();
42
```

```
43                for ( int i = 0; i < ticks; i++ )
44                   time.tick();
45
46                break;
47             } // end switch
48
49             System.out.printf( "Hour: %d  Minute: %d  Second: %d\n",
50                time.getHour(), time.getMinute(), time.getSecond() );
51             System.out.printf( "Universal time: %s   Standard time: %s\n",
52                time.toUniversalString(), time.toString() );
53
54
55             choice = getMenuChoice();
56          } // end while
57       } // end main
58
59       // prints a menu and returns a value corresponding to the menu choice
60       private static int getMenuChoice()
61       {
62          Scanner input = new Scanner( System.in );
63
64          System.out.println( "1. Add 1 second" );
65          System.out.println( "2. Add 1 minute" );
66          System.out.println( "3. Add 1 hour" );
67          System.out.println( "4. Add seconds" );
68          System.out.println( "5. Exit" );
69          System.out.print( "Choice: " );
70
71          return input.nextInt();
72       } // end method getMenuChoice
73    } // 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 };

```

```
54         // check if day in range for month
55         if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
56            return testDay;
57
58         // check for leap year
59         if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
60              ( year % 4 == 0 && year % 100 != 0 ) ) )
61            return testDay;
62
63         System.out.printf( "Invalid day (%d) set to 1.\n", testDay );
64
65         return 1; // maintain object in consistent state
66      } // end method checkDay
67
68      // increment the day and check if doing so will change the month
69      public void nextDay()
70      {
71         int testDay = day + 1;
72
73         if ( checkDay( testDay ) == testDay )
74            day = testDay;
75         else
76         {
77            day = 1;
78            nextMonth();
79         } // end else
80      } // end method nextDay
81
82      // increment the month and check if doing so will change the year
83      public void nextMonth()
84      {
85         if ( 12 == month )
86           year++;
87
88         month = month % 12 + 1;
89      } // end method nextMonth
90
91      // return a String of the form month/day/year
92      public String toString()
93      {
94         return String.format( "%d/%d/%d", month, day, year );
95      } // end method toDateString
96   } // end class Date
```

```
1   // Exercise 8.8 Solution: DateTest
2   // Program tests Date class.
3
4   public class DateTest
5   {
6      // method main begins execution of Java application
7      public static void main( String args[] )
8      {
9         System.out.println( "Checking increment" );
10        Date testDate = new Date( 11, 27, 1988 );
```

```
11
12        // test incrementing of day, month and year
13        for ( int counter = 0; counter < 40; counter++ )
14        {
15           testDate.nextDay();
16           System.out.printf( "Incremented Date: %s\n",
17              testDate.toString() );
18        } // end for
19     } // end main
20  } // 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:

```
1  // Exercise 8.9 Solution: Time2.java
2  // Time2 class definition with methods tick,
3  // incrementMinute and incrementHour.
4
5  public class Time2
6  {
7     private int hour; // 0 - 23
8     private int minute; // 0 - 59
9     private int second; // 0 - 59
10
11    // Time2 no-argument constructor: initializes each instance variable
12    // to zero; ensures that Time2 objects start in a consistent state
13    public Time2()
14    {
15       this( 0, 0, 0 ); // invoke Time2 constructor with three arguments
16    } // end Time2 no-argument constructor
17
18    // Time2 constructor: hour supplied, minute and second defaulted to 0
19    public Time2( int h )
20    {
21       this( h, 0, 0 ); // invoke Time2 constructor with three arguments
22    } // end Time2 one-argument constructor
23
24    // Time2 constructor: hour and minute supplied, second defaulted to 0
25    public Time2( int h, int m )
26    {
27       this( h, m, 0 ); // invoke Time2 constructor with three arguments
28    } // end Time2 two-argument constructor
29
30    // Time2 constructor: hour, minute and second supplied
31    public Time2( int h, int m, int s )
32    {
33       setTime( h, m, s ); // invoke setTime to validate time
34    } // end Time2 three-argument constructor
35
36    // Time2 constructor: another Time2 object supplied
37    public Time2( Time2 time )
38    {
39       // invoke Time2 constructor with three arguments
40       this( time.getHour(), time.getMinute(), time.getSecond() );
41    } // end Time2 constructor with Time2 argument
42
43    // Set Methods
44    // set a new time value using universal time; perform
45    // validity checks on data; set invalid values to zero
46    public boolean setTime( int h, int m, int s )
47    {
```

```
48          boolean hourValid = setHour( h ); // set the hour
49          boolean minuteValid = setMinute( m ); // set the minute
50          boolean secondValid = setSecond( s ); // set the second
51
52          return ( hourValid && minuteValid && secondValid );
53       } // end method setTime
54
55       // validate and set hour
56       public boolean setHour( int h )
57       {
58           if ( h >= 0 && h < 24 )
59           {
60             hour = h;
61             return true;
62          } // end if
63          else
64          {
65             hour = 0;
66             return false;
67          } // end else
68       } // end method setHour
69
70       // validate and set minute
71       public boolean setMinute( int m )
72       {
73          if ( m >= 0 && m < 60 )
74          {
75             minute = m;
76             return true;
77          } // end if
78          else
79          {
80             minute = 0;
81             return false;
82          } // end else
83       } // end method setMinute
84
85       // validate and set second
86       public boolean setSecond( int s )
87       {
88          if ( s >= 0 && s < 60 )
89          {
90             second = s;
91             return true;
92          } // end if
93          else
94          {
95             second = 0;
96             return false;
97          } // end else
98       } // end method setSecond
99
100      // Get Methods
101      // get hour value
102      public int getHour()
```

```
103    {
104       return hour;
105    } // end method getHour
106
107    // get minute value
108    public int getMinute()
109    {
110       return minute;
111    } // end method getMinute
112
113    // get second value
114    public int getSecond()
115    {
116       return second;
117    } // end method getSecond
118
119    // Tick the time by one second
120    public void tick()
121    {
122       setSecond( second + 1 );
123
124       if ( second == 0 )
125          incrementMinute();
126    } // end method tick
127
128    // Increment the minute
129    public void incrementMinute()
130    {
131       setMinute( minute + 1 );
132
133       if ( minute == 0 )
134          incrementHour();
135    } // end method incrementMinute
136
137    // Increment the hour
138    public void incrementHour()
139    {
140       setHour( hour + 1 );
141    } // end method incrementHour
142
143    // convert to String in universal-time format (HH:MM:SS)
144    public String toUniversalString()
145    {
146       return String.format(
147          "%02d:%02d:%02d", getHour(), getMinute(), getSecond() );
148    } // end method toUniversalString
149
150    // convert to String in standard-time format (H:MM:SS AM or PM)
151    public String toString()
152    {
153       return String.format( "%d:%02d:%02d %s",
154          ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
155          getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
156    } // end method toStandardString
157 } // end class Time2
```

```
1  // Exercise 8.9 Solution: Time2Test.java
2  // Program adds validation to Fig. 8.7 example
3  import java.util.Scanner;
4
5  public class Time2Test
6  {
7     public static void main( String args[] )
8     {
9        Scanner input = new Scanner( System.in );
10
11       Time2 time = new Time2(); // the Time2 object
12
13       int choice = getMenuChoice();
14
15       while ( choice != 5 )
16       {
17          switch ( choice )
18          {
19             case 1: // set hour
20                System.out.print( "Enter Hours: " );
21                int hours = input.nextInt();
22
23                if ( !time.setHour( hours ) )
24                   System.out.println( "Invalid hours." );
25
26                break;
27
28             case 2: // set minute
29                System.out.print( "Enter Minutes: " );
30                int minutes = input.nextInt();
31
32                if ( !time.setMinute( minutes ) )
33                   System.out.println( "Invalid minutes." );
34
35                break;
36
37             case 3: // set seconds
38                System.out.print( "Enter Seconds: " );
39                int seconds = input.nextInt();
40
41                if ( !time.setSecond( seconds ) )
42                   System.out.println( "Invalid seconds." );
43
44                break;
45
46             case 4: // add 1 second
47                time.tick();
48                break;
49          } // end switch
50
51          System.out.printf( "Hour: %d  Minute: %d  Second: %d\n",
52             time.getHour(), time.getMinute(), time.getSecond() );
53          System.out.printf( "Universal time: %s   Standard time: %s\n",
54             time.toUniversalString(), time.toString() );
55
```

```
56            choice = getMenuChoice();
57         } // end while
58      } // end main
59
60      // prints a menu and returns a value corresponding to the menu choice
61      private static int getMenuChoice()
62      {
63         Scanner input = new Scanner( System.in );
64
65         System.out.println( "1. Set Hour" );
66         System.out.println( "2. Set Minute" );
67         System.out.println( "3. Set Second" );
68         System.out.println( "4. Add 1 second" );
69         System.out.println( "5. Exit" );
70         System.out.print( "Choice: " );
71
72         return input.nextInt();
73      } // end method getMenuChoice
74   } // 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:

```
1  // Exercise 8.10 Solution: StaticImportTest.java
2  // Using static import to import static methods of class Math.
3  import static java.lang.Math.E;
4  import static java.lang.Math.sqrt;
5  import static java.lang.Math.ceil;
6  import static java.lang.Math.log;
```

```
7  import static java.lang.Math.cos;
8
9  public class StaticImportTest
10 {
11    public static void main( String args[] )
12    {
13       System.out.printf( "sqrt( 900.0 ) = %.1f\n", sqrt( 900.0 ) );
14       System.out.printf( "ceil( -9.8 ) = %.1f\n", ceil( -9.8 ) );
15       System.out.printf( "log( E ) = %.1f\n", log( E ) );
16       System.out.printf( "cos( 0.0 ) = %.1f\n", cos( 0.0 ) );
17    } // end main
18 } // 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:

```
1  // Exercise 8.11 Solution: TrafficLight.java
2  // Declare an enum type with constructor and explicit instance fields
3  // and accessors for these fields
4
5  public enum TrafficLight
6  {
7     // declare constants of enum type
8     RED( 50 ), // light is red for 50 seconds
9     GREEN( 40 ), // light is green for 40 seconds
10    YELLOW( 5 ); // light is yellow for 5 seconds
11
12    // instance fields
13    private final int duration; // duration of the light
14
15    // enum type constructor
16    TrafficLight( int durationSeconds )
17    {
18       duration = durationSeconds;
19    } // end enum constructor TrafficLight
20
21    // accessor for duration
22    public int getDuration()
23    {
24       return duration;
25    } // end method getDuration
26 } // end enum TrafficLight
```

```
1  // Exercise 8.11 Solution: EnumTest.java
2  // Testing enum type TrafficLight.
3
4  public class EnumTest
5  {
6     public static void main( String args[] )
7     {
8        System.out.println( "Light\tDuration\n" );
9
10       // print all traffic lights and their duration
11       for ( TrafficLight light : TrafficLight.values() )
12          System.out.printf( "%s\t%d\n", light, light.getDuration() );
13    } // end main
14 } // 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:

```
1  // Exercise 8.12 Solution: Complex.java
2  // Definition of class Complex
3
4  public class Complex
5  {
6     private double real;
7     private double imaginary;
8
```

```
9      // Initialize both parts to 0
10     public Complex()
11     {
12        this( 0.0, 0.0 );
13     } // end Complex no-argument constructor
14
15     // Initialize real part to r and imaginary part to i
16     public Complex( double r, double i )
17     {
18        real = r;
19        imaginary = i;
20     } // end Complex two-argument constructor
21
22     // Add two Complex numbers
23     public Complex add( Complex right )
24     {
25        return new Complex( real + right.real,
26           imaginary + right.imaginary );
27     } // end method add
28
29     // Subtract two Complex numbers
30     public Complex subtract( Complex right )
31     {
32        return new Complex( real - right.real,
33           imaginary - right.imaginary );
34     } // end method subtract
35
36     // Return String representation of a Complex number
37     public String toString()
38     {
39        return String.format( "(%.1f, %.1f)", real, imaginary );
40     } // end method toComplexString;
41  } // end class Complex
```

```
1   // Exercise 8.12: ComplexTest.java
2   // Test the Complex number class
3
4   public class ComplexTest
5   {
6      public static void main( String args[] )
7      {
8         // initialize two numbers
9         Complex a = new Complex( 9.5, 7.7 );
10        Complex b = new Complex( 1.2, 3.1 );
11
12        System.out.printf( "a = %s\n", a.toString() );
13        System.out.printf( "b = %s\n", b.toString() );
14        System.out.printf( "a + b = %s\n", a.add( b ).toString() );
15        System.out.printf( "a - b = %s\n", a.subtract( b ).toString() );
16     } // end main
17  } // 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:

```
1  // Exercise 8.13 Solution: DateAndTime.java
2  // DateAndTime class definition.
3
4  public class DateAndTime
5  {
6     private int month; // 1-12
7     private int day; // 1-31 based on month
8     private int year; // > 0
9     private int hour; // 0 - 23
10    private int minute; // 0 - 59
11    private int second; // 0 - 59
12
13    // no argument constructor
14    public DateAndTime()
15    {
16       setDate( 1, 1, 2000 );
17       setTime( 0, 0, 0 );
18    } // end DateAndTime no-argument constructor
19
20    // constructor
21    public DateAndTime( int theMonth, int theDay, int theYear,
22       int theHour, int theMinute, int theSecond )
23    {
24       setDate( theMonth, theDay, theYear );
25       setTime( theHour, theMinute, theSecond );
26    } // end DateAndTime six-argument constructor
27
28    // Set a new date value. Perform
29    // validity checks on data. Set invalid values to one.
30    public void setDate( int theMonth, int theDay, int theYear )
31    {
32       month = checkMonth( theMonth ); // validate month
33       year = checkYear( theYear ); // validate year
34       day = checkDay( theDay ); // validate day
35    } // end method setDate
36
37    // Set a new time value using universal time. Perform
38    // validity checks on data. Set invalid values to zero.
39    public void setTime( int h, int m, int s )
40    {
```

```
41          setHour( h ); // set the hour
42          setMinute( m ); // set the minute
43          setSecond( s ); // set the second
44       } // end method setTime
45
46       // validate and set hour
47       public void setHour( int h )
48       {
49          hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
50       } // end method setHour
51
52       // validate and set minute
53       public void setMinute( int m )
54       {
55          minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
56       } // end method setMinute
57
58       // validate and set second
59       public void setSecond( int s )
60       {
61          second = ( ( s >= 0 && s < 60 ) ? s : 0 );
62       } // end method setSecond
63
64       // Get Methods
65       // get hour value
66       public int getHour()
67       {
68          return hour;
69       } // end method getHour
70
71       // get minute value
72       public int getMinute()
73       {
74          return minute;
75       } // end method getMinute
76
77       // get second value
78       public int getSecond()
79       {
80          return second;
81       } // end method getSecond
82
83       // Tick the time by one second
84       public void tick()
85       {
86          setSecond( second + 1 );
87
88          if ( second == 0 )
89             incrementMinute();
90       } // end method tick
91
92       // Increment the minute
93       public void incrementMinute()
94       {
```

```
95          setMinute( minute + 1 );
96
97          if ( minute == 0 )
98             incrementHour();
99       } // end method incrementMinute
100
101      // Increment the hour
102      public void incrementHour()
103      {
104         setHour( hour + 1 );
105
106         if ( hour == 0 )
107            nextDay();
108      } // end method incrementHour
109
110      // utility method to confirm proper year value
111      private int checkYear( int testYear )
112      {
113         if ( testYear > 0 ) // validate year
114            return testYear;
115         else // day is invalid
116         {
117            System.out.printf( "Invalid year (%d) set to 1.\n", testYear );
118            return 1;
119         } // end else
120      } // end method checkYear
121
122      // utility method to confirm proper month value
123      private int checkMonth( int testMonth )
124      {
125         if ( testMonth > 0 && testMonth <= 12 ) // validate month
126            return testMonth;
127         else // month is invalid
128         {
129            System.out.printf( "Invalid month (%d) set to 1.\n", testMonth );
130            return 1; // maintain object in consistent state
131         } // end else
132      } // end method checkMonth
133
134      // utility method to confirm proper day value
135      // based on month and year.
136      public int checkDay( int testDay )
137      {
138         int daysPerMonth[] =
139            { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
140
141         // check if day in range for month
142         if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
143            return testDay;
144
145         // check for leap year
146         if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
147            ( year % 4 == 0 && year % 100 != 0 ) ) )
148            return testDay;
```

```
149
150        return 1; // leave object in consistent state
151     } // end method checkDay
152
153     // increment the day and check if doing so will change the month
154     public void nextDay()
155     {
156        int testDay = day + 1;
157
158        if ( checkDay( testDay ) == testDay )
159           day = testDay;
160        else
161        {
162           day = 1;
163           nextMonth();
164        } // end else
165     } // end method nextDay
166
167     // increment the month and check if doing so will change the year
168     public void nextMonth()
169     {
170        if ( 12 == month )
171          year++;
172
173        month = month % 12 + 1;
174     } // end method nextMonth
175
176     // convert to String in universal-time format
177     public String toUniversalString()
178     {
179        return String.format( "%d/%d/%d: %02d:%02d:%02d",
180           month, day, year, getHour(), getMinute(), getSecond() );
181     } // end method toUniversalString
182
183     // convert to String in standard-time format
184     public String toString()
185     {
186        return String.format( "%d/%d/%d: %d:%02d:%02d %s",
187           month, day, year,
188           ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
189           getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );
190     } // end method toStandardString
191  } // end class DateAndTime
```

```
1  // Exercise 8.13 Solution: DateAndTimeTest.java
2  // Demonstrating the DateAndTime class methods
3  import java.util.Scanner;
4
5  public class DateAndTimeTest
6  {
7     public static void main( String args[] )
8     {
9        Scanner input = new Scanner( System.in );
```

```
10
11          System.out.println( "Enter the date and time" );
12          System.out.print( "Month: " );
13          int month = input.nextInt();
14          System.out.print( "Day: " );
15          int day = input.nextInt();
16          System.out.print( "Year: " );
17          int year = input.nextInt();
18
19          System.out.print( "Hours: " );
20          int hour = input.nextInt();
21          System.out.print( "Minutes: " );
22          int minute = input.nextInt();
23          System.out.print( "Seconds: " );
24          int seconds = input.nextInt();
25
26          DateAndTime dateTime = new DateAndTime(
27             month, day, year, hour, minute, seconds );
28
29          int choice = getMenuChoice();
30
31          while ( choice != 7 )
32          {
33             switch ( choice )
34             {
35                case 1: // add 1 second
36                   dateTime.tick();
37                   break;
38
39                case 2: // add 1 minute
40                   dateTime.incrementMinute();
41                   break;
42
43                case 3: // and 1 hour
44                   dateTime.incrementHour();
45                   break;
46
47                case 4: // add 1 day
48                   dateTime.nextDay();
49                   break;
50
51                case 5: // add 1 month
52                   dateTime.nextMonth();
53                   break;
54
55                case 6: // add arbitrary seconds
56                   System.out.print( "Enter seconds to tick: " );
57                   int ticks = input.nextInt();
58
59                   for ( int i = 0; i < ticks; i++ )
60                      dateTime.tick();
61
62                   break;
63             } // end switch
64
```

```
65            System.out.printf( "Universal date and time: %s\n",
66               dateTime.toUniversalString() );
67            System.out.printf( "Standard date and time: %s\n",
68               dateTime.toString() );
69
70            choice = getMenuChoice();
71         } // end while
72      } // end main
73
74      // prints a menu and returns a value corresponding to the menu choice
75      private static int getMenuChoice()
76      {
77         Scanner input = new Scanner( System.in );
78
79         System.out.println( "1. Add 1 second" );
80         System.out.println( "2. Add 1 Minute" );
81         System.out.println( "3. Add 1 Hour" );
82         System.out.println( "4. Add 1 Day" );
83         System.out.println( "5. Add 1 Month" );
84         System.out.println( "6. Add seconds" );
85         System.out.println( "7. Exit" );
86         System.out.print( "Choice: " );
87
88         return input.nextInt();
89      } // end method getMenuChoice
90   } // 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:

```
1  // Exercise 8.14 Solution: Rectangle.java
2  // Definition of class Rectangle
3
4  public class Rectangle
5  {
6     // coordinates of the vertices.
7     private double x1, y1;
8     private double x2, y2;
9     private double x3, y3;
10    private double x4, y4;
11
12    // no-argument constructor
13    public Rectangle()
14    {
15       setCoordinates( 1, 1, 1, 1, 1, 1, 1, 1 );
16    } // end Rectangle no-argument constructor
17
18    // constructor
19    public Rectangle( double x1, double y1, double x2,
20       double y2, double x3, double y3, double x4, double y4 )
21    {
22       setCoordinates( x1, y1, x2, y2, x3, y3, x4, y4 );
23    } // end standard Rectangle constructor
24
25    // check if coordinates are valid
26    public void setCoordinates( double xInput1, double yInput1,
27       double xInput2, double yInput2, double xInput3,
28       double yInput3, double xInput4, double yInput4 )
29    {
30       x1 = ( xInput1 >= 0.0 && xInput1 <= 20.0 ? xInput1 : 1 );
31       x2 = ( xInput2 >= 0.0 && xInput2 <= 20.0 ? xInput2 : 1 );
32       x3 = ( xInput3 >= 0.0 && xInput3 <= 20.0 ? xInput3 : 1 );
33       x4 = ( xInput4 >= 0.0 && xInput4 <= 20.0 ? xInput4 : 1 );
34       y1 = ( yInput1 >= 0.0 && yInput1 <= 20.0 ? yInput1 : 1 );
35       y2 = ( yInput2 >= 0.0 && yInput2 <= 20.0 ? yInput2 : 1 );
36       y3 = ( yInput3 >= 0.0 && yInput3 <= 20.0 ? yInput3 : 1 );
37       y4 = ( yInput4 >= 0.0 && yInput4 <= 20.0 ? yInput4 : 1 );
38
39       if ( !isRectangle() )
40          System.out.println( "This is not a rectangle" );
41    } // end method setCoordinates
42
43    // calculate distance between two points
```

```
44    public double distance( double x1, double y1, double x2, double y2 )
45    {
46       return Math.sqrt( ( Math.pow( x1 - x2, 2 )
47          + Math.pow( y1 - y2, 2 ) ) );
48    } // end method distance
49
50    // check if coordinates specify a rectangle by determining if the
51    // two diagonals are of the same length.
52    public boolean isRectangle()
53    {
54       double side1 = distance( x1, y1, x2, y2 );
55       double side2 = distance( x2, y2, x3, y3 );
56       double side3 = distance( x3, y3, x4, y4 );
57
58       if ( side1 * side1 + side2 * side2 ==
59          side2 * side2 + side3 * side3 )
60          return true;
61       else
62          return false;
63    } // end method isRectangle
64
65    // check if rectangle is a square
66    public boolean isSquare()
67    {
68       return ( getLength() == getWidth() );
69    } // end method isSquare
70
71    // get length of rectangle
72    public double getLength()
73    {
74       double side1 = distance( x1, y1, x2, y2 );
75       double side2 = distance( x2, y2, x3, y3 );
76
77       return ( side1 > side2 ? side1 : side2 );
78    } // end method getLength
79
80    // get width of rectangle
81    public double getWidth()
82    {
83       double side1 = distance( x1, y1, x2, y2 );
84       double side2 = distance( x2, y2, x3, y3 );
85
86       return ( side1 < side2 ? side1 : side2 );
87    } // end method getWidth
88
89    // calculate perimeter
90    public double perimeter()
91    {
92       return 2 * getLength() + 2 * getWidth();
93    } // end method perimeter
94
95    // calculate area
96    public double area()
97    {
```

```
98          return getLength() * getWidth();
99       } // end method area
100
101      // convert to String
102      public String toString()
103      {
104         return String.format( "%s: %f\n%s: %f\n%s: %f\n%s: %f",
105            "Length", getLength(), "Width", getWidth(),
106            "Perimeter", perimeter(), "Area", area() );
107      } // end method toRectangleString
108   } // end class Rectangle
```

```
1   // Exercise 8.14 Solution: RectangleTest.java
2   // Program tests class Rectangle.
3   import java.util.Scanner;
4
5   public class RectangleTest
6   {
7      public static void main( String args[] )
8      {
9         Scanner input = new Scanner( System.in );
10
11        System.out.println( "Enter rectangle's coordinates" );
12        System.out.print( "x1: " );
13        double x1 = input.nextInt();
14        System.out.print( "y1: " );
15        double y1 = input.nextInt();
16        System.out.print( "x2: " );
17        double x2 = input.nextInt();
18        System.out.print( "y2: " );
19        double y2 = input.nextInt();
20        System.out.print( "x3: " );
21        double x3 = input.nextInt();
22        System.out.print( "y3: " );
23        double y3 = input.nextInt();
24        System.out.print( "x4: " );
25        double x4 = input.nextInt();
26        System.out.print( "y4: " );
27        double y4 = input.nextInt();
28
29        Rectangle rectangle =
30           new Rectangle( x1, y1, x2, y2, x3, y3, x4, y4 );
31
32        if ( rectangle.isRectangle() )
33           System.out.println( rectangle.toString() );
34
35        if ( rectangle.isSquare() )
36           System.out.println( "This is a square" );
37     } // end main
38  } // 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:

```
1  // Exercise 8.15 Solution: IntegerSet.java
2  // IntegerSet class definition
3
4  public class IntegerSet
5  {
6     private final int SETSIZE = 101;
7     private boolean[] set;
8
9     // no-argument constructor, creates an empty set
10    public IntegerSet()
11    {
12       set = new boolean[ SETSIZE ];
13    } // end no-argument constructor
14
15    // constructor creates a set from array of integers
16    public IntegerSet( int array[] )
17    {
18       set = new boolean[ SETSIZE ];
```

```
19
20        for( int i = 0; i < array.length; i++ )
21           insertElement( array[ i ] );
22     } // end constructor
23
24     // return string representation of set
25     public String toString()
26     {
27        int x = 1;
28        boolean empty = true; // assume set is empty
29        String setString = "{ ";
30
31        // get set elements
32        for ( int count = 0; count < 101; count++ )
33        {
34           if ( set[ count ] )
35           {
36              setString += count + " ";
37              empty = false; // set is not empty
38              x++;
39           } // end if
40        } // end for
41
42        // empty set
43        if ( empty )
44           setString += "---"; // display an empty set
45
46        setString += " }";
47
48        return setString;
49     } // end method toString
50
51     // returns the union of two sets
52     public IntegerSet union( IntegerSet integerSet )
53     {
54        IntegerSet temp = new IntegerSet();
55
56        for ( int count = 0; count < 101; count++ )
57           temp.set[ count ] = ( set[ count ] || integerSet.set[ count ] );
58
59        return temp;
60     } // end method union
61
62     // returns the intersection of two sets
63     public IntegerSet intersection( IntegerSet integerSet )
64     {
65        IntegerSet temp = new IntegerSet();
66
67        for ( int count = 0; count < 101; count++ )
68           temp.set[ count ] =
69              ( set[ count ] && integerSet.set[ count ] );
70
71        return temp;
72     } // end method intersetcion
```

```
73
74       // insert a new integer into this set
75       public void insertElement( int insertInteger )
76       {
77          if ( validEntry( insertInteger ) )
78             set[ insertInteger ] = true;
79       } // end method insertElement
80
81       // remove an integer from this set
82       public void deleteElement( int deleteInteger )
83       {
84          if ( validEntry( deleteInteger ) )
85             set[ deleteInteger ] = false;
86       } // end method deleteElement
87
88       // determine if two sets are equal
89       public boolean isEqualTo( IntegerSet integerSet )
90       {
91          for ( int count = 0; count < 101; count++ )
92          {
93             if ( set[ count ] != integerSet.set[ count ] )
94                return false; // sets are not-equal
95          } // end for
96
97          return true; // sets are equal
98       } // end method isEqualTo
99
100      // determin if input is valid
101      public boolean validEntry( int input )
102      {
103         return input >= 0 && input <= 100;
104      } // end method validEntry
105   } // end class IntegerSet
```

```
1   // Exercise 8.15 Solution: IntegerSetTest.java
2   // Program that tests IntegerSet
3   import java.util.Scanner;
4
5   public class IntegerSetTest
6   {
7      public static void main( String args[] )
8      {
9         // initialize two sets
10        System.out.println( "Input Set A" );
11        IntegerSet set1 = inputSet();
12        System.out.println( "Input Set B" );
13        IntegerSet set2 = inputSet();
14
15        IntegerSet union = set1.union( set2 );
16        IntegerSet intersection = set1.intersection( set2 );
17
18        // prepare output
19        System.out.println( "Set A contains elements:" );
```

```
20          System.out.println( set1.toString() );
21          System.out.println( "Set B contains elements:" );
22          System.out.println( set2.toString() );
23          System.out.println(
24             "Union of Set A and Set B contains elements:" );
25          System.out.println( union.toString() );
26          System.out.println(
27             "Intersection of Set A and Set B contains elements:" );
28          System.out.println( intersection.toString() );
29
30          // test whether two sets are equal
31          if ( set1.isEqualTo( set2 ) )
32             System.out.println( "Set A is equal to set B" );
33          else
34             System.out.println( "Set A is not equal to set B" );
35
36          // test insert and delete
37          System.out.println( "Inserting 77 into set A..." );
38          set1.insertElement( 77 );
39          System.out.println( "Set A now contains elements:" );
40          System.out.println( set1.toString() );
41
42          System.out.println( "Deleting 77 from set A..." );
43          set1.deleteElement( 77 );
44          System.out.println( "Set A now contains elements:" );
45          System.out.println( set1.toString() );
46
47          // test constructor
48          int intArray[] = { 25, 67, 2, 9, 99, 105, 45, -5, 100, 1 };
49          IntegerSet set5 = new IntegerSet( intArray );
50
51          System.out.println( "New Set contains elements:" );
52          System.out.println( set5.toString() );
53       } // end main
54
55       // creates a new set by reading numbers from the user
56       private static IntegerSet inputSet()
57       {
58          Scanner input = new Scanner( System.in );
59          IntegerSet temp = new IntegerSet();
60
61          System.out.print( "Enter number (-1 to end): " );
62          int number = input.nextInt();
63
64          while ( number != -1 )
65          {
66             temp.insertElement( number );
67
68             System.out.print( "Enter number (-1 to end): " );
69             number = input.nextInt();
70          } // end while
71
72          return temp;
73       } // end method inputSet
74    } // 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:

```
// 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
```

```
53      public void setMonth( int mm )
54      {
55         month = ( mm >= 1 && mm <= 12 ) ? mm : 1;
56      } // end method setMonth
57
58      // Set the year
59      public void setYear( int yyyy )
60      {
61         year = ( yyyy >= 1900 && yyyy <= 2100 ) ? yyyy : 1900;
62      } // end method setYear
63
64      // return Date in format: mm/dd/yyyy
65      public String toString()
66      {
67         return String.format( "%d/%d/%d", month, day, year );
68      } // end method toSlashDateString
69
70      // return Date in format: MonthName dd, yyyy
71      public String toMonthNameDateString()
72      {
73         return String.format(
74            "%s %d, %d", monthNames[ month - 1 ], day, year );
75      } // end method toMonthNameDateString
76
77      // return Date in format DDD YYYY
78      public String toDayDateString()
79      {
80         return String.format( "%d %d", convertToDayOfYear(), year );
81      } // end method toDayDateString
82
83      // Return the number of days in the month
84      private int daysInMonth()
85      {
86         return leapYear() && month == 2 ? 29 : monthDays[ month - 1 ];
87      } // end method daysOfMonth
88
89      // test for a leap year
90      private boolean leapYear()
91      {
92         if ( year % 400 == 0 || ( year % 4 == 0 && year % 100 != 0 ) )
93            return true;
94         else
95            return false;
96      } // end method leapYear
97
98      // sets the day and month to the proper values based on ddd
99      private void convertFromDayOfYear( int ddd )
100     {
101        int dayTotal = 0;
102
103        if ( ddd < 1 || ddd > 366 ) // check for invalid day
104           ddd = 1;
105
106        setMonth( 1 );
```

```
107
108        for ( int m = 1;
109           m < 13 && ( dayTotal + daysInMonth() ) < ddd; m++ )
110        {
111           dayTotal += daysInMonth();
112           setMonth( m + 1 );
113        } // end for
114
115        setDay( ddd - dayTotal );
116     } // end convertFromDayOfYear
117
118     // convert mm and dd to ddd
119     private int convertToDayOfYear()
120     {
121        int ddd = 0;
122
123        for ( int m = 1; m < month; m++ )
124        {
125           if ( leapYear() && m == 2 )
126              ddd += 29;
127           else
128              ddd += monthDays[ m -1 ];
129        } // end for
130
131        ddd += day;
132        return ddd;
133     } // end method convertToDayOfYear
134
135     // convert from month name to month number
136     private void convertFromMonthName( String monthName )
137     {
138        for ( int subscript = 0; subscript < 12; subscript++ )
139        {
140           if ( monthName.equals( monthNames[ subscript ] ) )
141           {
142              setMonth( subscript + 1 );
143              return; // stop checking for month
144           } // end if
145        } // end for
146
147        setMonth( 1 ); // invalid month default is january
148     } // end convertFromMonthName
149  } // end class Date
```

```
1  // Exercise 8.16 Solution: DateTest.java
2  // Program that tests Date class
3  import java.util.Scanner;
4
5  public class DateTest
6  {
7     public static void main( String args[] )
8     {
9        Scanner input = new Scanner( System.in );
10
```

```
11         int choice = getMenuChoice();
12
13         while ( choice != 4 )
14         {
15            int month; // month of year
16            int day; // day of month or day of year
17            int year; // year
18            String monthName; // name of month
19            Date date = new Date(); // the date object
20
21            switch ( choice )
22            {
23               case 1:
24                  // format: MM/DD/YYYY
25                  System.out.print( "Enter Month (1-12): " );
26                  month = input.nextInt();
27                  System.out.print( "Enter Day of Month: " );
28                  day = input.nextInt();
29                  System.out.print( "Enter Year: " );
30                  year = input.nextInt();
31
32                  date = new Date( month, day, year );
33                  break;
34
35               case 2:
36                  // format: Month DD, YYYY
37                  System.out.print( "Enter Month Name: " );
38                  monthName = input.next();
39                  System.out.print( "Enter Day of Month: " );
40                  day = input.nextInt();
41                  System.out.print( "Enter Year: " );
42                  year = input.nextInt();
43
44                  date = new Date( monthName, day, year );
45                  break;
46
47               case 3:
48                  // format: DDD YYYY
49                  System.out.print( "Enter Day of Year: " );
50                  day = input.nextInt();
51                  System.out.print( "Enter Year: " );
52                  year = input.nextInt();
53
54                  date = new Date( day, year );
55                  break;
56            } // end switch
57
58            System.out.printf( "\n%s: %s\n%s: %s\n%s: %s\n",
59               "MM/DD/YYYY", date.toString(),
60               "Month DD, YYYY", date.toMonthNameDateString(),
61               "DDD YYYY", date.toDayDateString() );
62
63            choice = getMenuChoice();
64         } // end while
65      } // end main
```

```
66
67      // get user choice
68      private static int  getMenuChoice()
69      {
70         Scanner input = new Scanner( System.in );
71         System.out.println( "Enter 1 for format: MM/DD/YYYY" );
72         System.out.println( "Enter 2 for format: Month DD, YYYY" );
73         System.out.println( "Enter 3 for format: DDD YYYY" );
74         System.out.println( "Enter 4 to exit" );
75         System.out.print( "Choice: " );
76         return input.nextInt();
77      } // end method getMenuChoice
78   } // 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:

```
1  // Exercise 8.17 Solution: Rational.java
2  // Rational class definition.
3
4  public class Rational
5  {
6     private int numerator; // numerator of the fraction
7     private int denominator; // denominator of the fraction
8
9     // no-argument constructor, initializes this Rational to 1
10    public Rational()
11    {
12       numerator = 1;
13       denominator = 1;
14    } // end Rational no-argument constructor
15
16    // initialize numerator part to n and denominator part to d
17    public Rational( int theNumerator, int theDenominator )
18    {
19       numerator = theNumerator;
20       denominator = theDenominator;
21       reduce();
22    } // end two-argument constructor
23
24    // add two Rational numbers
25    public Rational sum( Rational right )
26    {
```

```
27          int resultDenominator = denominator * right.denominator;
28          int resultNumerator = numerator * right.denominator +
29             right.numerator * denominator;
30
31          return new Rational( resultNumerator, resultDenominator );
32       } // end method sum
33
34       // subtract two Rational numbers
35       public Rational subtract( Rational right )
36       {
37          int resultDenominator = denominator * right.denominator;
38          int resultNumerator = numerator * right.denominator -
39             right.numerator * denominator;
40
41          return new Rational( resultNumerator, resultDenominator );
42       } // end method subtract
43
44       // multiply two Rational numbers
45       public Rational multiply( Rational right )
46       {
47          return new Rational( numerator * right.numerator,
48             denominator * right.denominator );
49       } // end method multiply
50
51       // divide two Rational numbers
52       public Rational divide( Rational right )
53       {
54          return new Rational( numerator * right.denominator,
55             denominator * right.numerator );
56       } // end method divide
57
58       // reduce the fraction
59       private void reduce()
60       {
61          int gcd = 0;
62          int smaller;
63
64          // find the greatest common denominator of the two numbers
65          if ( numerator < denominator )
66             smaller = numerator;
67          else
68             smaller = denominator;
69
70          for ( int divisor = smaller; divisor >= 2; divisor-- )
71          {
72             if ( numerator % divisor == 0 && denominator % divisor == 0 )
73             {
74                gcd = divisor;
75                break;
76             } // end if
77          } // end for
78
79          // divide both the numerator and denominator by the gcd
80          if ( gcd != 0 )
```

```
81          {
82             numerator /= gcd;
83             denominator /= gcd;
84          } // end if
85       } // end for
86
87       // return String representation of a Rational number
88       public String toString()
89       {
90         return numerator + "/" + denominator;
91       } // end method toRationalString
92
93       // return floating-point String representation of
94       // a Rational number
95       public String toFloatString( int digits )
96       {
97          double value = ( double ) numerator / denominator;
98          // builds a formatting string that specifies the precision
99          // based on the digits parameter
100         return String.format( "%." + digits + "f", value );
101      } // end method toFloatString
102   } // end class Rational
```

```
1   // Exercise 8.17 Solution: RationalTest.java
2   // Program tests class Rational.
3   import java.util.Scanner;
4
5   public class RationalTest
6   {
7      public static void main( String args[] )
8      {
9         Scanner input = new Scanner( System.in );
10
11        int numerator; // the numerator of a fraction
12        int denominator; // the denominator of a fraction
13        int digits; // digits to display in floating point format
14        Rational rational1; // the first rational number
15        Rational rational2; // second rational number
16        Rational result; // result of performing an operation
17
18        // read first fraction
19        System.out.print( "Enter numerator 1: " );
20        numerator = input.nextInt();
21        System.out.print( "Enter denominator 1: " );
22        denominator = input.nextInt();
23        rational1 = new Rational( numerator, denominator );
24
25        // read second fraction
26        System.out.print( "Enter numerator 2: " );
27        numerator = input.nextInt();
28        System.out.print( "Enter denominator 2: " );
29        denominator = input.nextInt();
30        rational2 = new Rational( numerator, denominator );
```

```
31
32          System.out.print( "Enter precision: " );
33          digits = input.nextInt();
34
35          int choice = getMenuChoice(); // user's choice in the menu
36
37          while ( choice != 5 )
38          {
39             switch ( choice )
40             {
41                case 1:
42                   result = rational1.sum( rational2 );
43                   System.out.printf( "a + b = %s = %s\n",
44                      result.toString(),
45                      result.toFloatString( digits ) );
46                   break;
47
48                case 2:
49                   result = rational1.subtract( rational2 );
50                   System.out.printf( "a - b = %s = %s\n",
51                      result.toString(),
52                      result.toFloatString( digits ) );
53                   break;
54
55                case 3:
56                   result = rational1.multiply( rational2 );
57                   System.out.printf( "a * b = %s = %s\n",
58                      result.toString(),
59                      result.toFloatString( digits ) );
60                   break;
61
62                case 4:
63                   result = rational1.divide( rational2 );
64                   System.out.printf( "a / b = %s = %s\n",
65                      result.toString(),
66                      result.toFloatString( digits ) );
67                   break;
68             } // end switch
69
70             choice = getMenuChoice();
71          } // end while
72       } // end main
73
74       // prints a menu and returns a value corresponding to the menu choice
75       private static int getMenuChoice()
76       {
77          Scanner input = new Scanner( System.in );
78
79          System.out.println( "1. Add" );
80          System.out.println( "2. Subtract" );
81          System.out.println( "3. Multiply" );
82          System.out.println( "4. Divide" );
83          System.out.println( "5. Exit" );
84          System.out.print( "Choice: " );
85
```

```
86         return input.nextInt();
87      } // end method getMenuChoice
88   } // 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
         {
```

```
53              temp = addValue.subtractPositives( this );
54              temp.positive = false;
55           } // end else
56
57           addValue.positive = false; // reset sign for addValue
58        } // end else if
59        // this is negative
60        else if ( !positive && addValue.positive )
61        {
62           addValue.positive = false;
63
64           if ( isGreaterThan( addValue ) )
65              temp = addValue.subtractPositives( this );
66           else
67           {
68              temp = subtractPositives( addValue );
69              temp.positive = false;
70           } // end else
71
72           addValue.positive = true; // reset sign for addValue
73        } // end else if
74
75        return temp; // return the sum
76     } // end method add
77
78     // add two positive HugeIntegers
79     public HugeInteger addPositives( HugeInteger addValue )
80     {
81        HugeInteger temp = new HugeInteger();
82        int carry = 0;
83
84        // iterate through HugeIntegers
85        for ( int i = 39; i >= 0; i-- )
86        {
87           temp.integer[ i ] =
88              integer[ i ] + addValue.integer[ i ] + carry;
89
90           // determine whether to carry a 1
91           if ( temp.integer[ i ] > 9 )
92           {
93              temp.integer[ i ] %= 10;  // reduce to 0-9
94              carry = 1;
95           } // end if
96           else
97              carry = 0;
98        } // end for
99
100       // if both are negative, set the result to negative
101       if ( !positive )
102          temp.positive = false;
103
104       return temp;
105    } // end method addPositives
106
107    // subtract two HugeIntegers
```

```
108    public HugeInteger subtract( HugeInteger subtractValue )
109    {
110       HugeInteger temp = new HugeInteger();   // temporary result
111
112       // subtractValue is negative
113       if ( positive && ( !subtractValue.positive ) )
114          temp = addPositives( subtractValue );
115       // this HugeInteger is negative
116       else if ( !positive && subtractValue.positive )
117          temp = addPositives( subtractValue );
118       // at this point, both HugeIntegers have the same sign
119       else
120       {
121          boolean isPositive = positive; // original sign
122          boolean resultPositive = positive; // sign of the result
123
124          // set both to positive so we can compare absolute values
125          positive = true;
126          subtractValue.positive = true;
127
128          if ( this.isGreaterThan( subtractValue ) )
129             temp = this.subtractPositives( subtractValue );
130          else
131          {
132             temp = subtractValue.subtractPositives( this );
133             resultPositive = !isPositive; // flip the sign
134          } // end else
135
136          positive = isPositive;
137          subtractValue.positive = isPositive;
138          temp.positive = resultPositive;
139       } // end else
140
141       return temp;
142    } // end method subtract
143
144    // subtract two positive HugeIntegers
145    public HugeInteger subtractPositives( HugeInteger subtractValue )
146    {
147       HugeInteger temp = new HugeInteger();
148
149       // iterate through HugeInteger
150       for ( int i = 39; i >= 0; i-- )
151       {
152          // borrow if needed
153          if ( integer[i] < subtractValue.integer[i] )
154          {
155             integer[ i ] += 10;
156             subtractValue.integer[ i - 1 ]--;
157          } // end if
158
159       temp.integer[ i ] = integer[ i ] - subtractValue.integer[ i ];
160       } // end for
161
162       return temp; // return difference of two HugeIntegers
163    } // end method subtractPositives
```

```
164
165    // find first non-zero position of two HugeIntegers
166    public int findFirstNonZeroPosition()
167    {
168       int position = 39;
169
170       // find first non-zero position for first HugeInteger
171       for ( int i = 0; i < DIGITS; i++ )
172       {
173          if ( integer[ i ] > 0 )
174             return i;
175       } // end for
176
177       return 39;
178    } // end method findFirstNonZeroPosition
179
180    // get string representation of HugeInteger
181    public String toString()
182    {
183       String output = "";
184
185       if ( !positive )
186          output = "-";
187
188       // get HugeInteger values without leading zeros
189       for ( int i = findFirstNonZeroPosition(); i < DIGITS; i++ )
190          output += integer[ i ];
191
192       return output;
193    } // end method toHugeIntegerString
194
195    // test if two HugeIntegers are equal
196    public boolean isEqualTo( HugeInteger compareValue )
197    {
198       // compare the sign
199       if ( positive != compareValue.positive )
200          return false;
201
202       // compare each digit
203       for ( int i = 0; i < DIGITS; i++ )
204       {
205          if ( integer[ i ] != compareValue.integer[ i ] )
206             return false;
207       } // end for
208
209       return true;
210    } // end method isEqualTo
211
212    // test if two HugeIntegers are not equal
213    public boolean isNotEqualTo( HugeInteger compareValue )
214    {
215       return !isEqualTo( compareValue );
216    } // end method isNotEqualTo
217
218    // test if one HugeInteger is greater than another
```

```
219    public boolean isGreaterThan( HugeInteger compareValue )
220    {
221       // different signs
222       if ( positive && ( !compareValue.positive ) )
223          return true;
224       else if ( !positive && compareValue.positive )
225          return false;
226
227       // same sign
228       else
229       {
230          // first number's length is less than second number's length
231          if ( findFirstNonZeroPosition() >
232             compareValue.findFirstNonZeroPosition() )
233          {
234             return !positive;
235          } // end if
236
237          // first number's length is larger than that of second number
238          else if ( findFirstNonZeroPosition() <
239             compareValue.findFirstNonZeroPosition() )
240          {
241             return positive;
242          } // end else if
243
244          // two numbers have same length
245          else
246          {
247             for ( int i = 0; i < DIGITS; i++ )
248             {
249                if ( integer[ i ] > compareValue.integer[ i ] )
250                   return positive;
251                else if ( integer[ i ] < compareValue.integer[ i ] )
252                   return !positive;
253             } // end for
254          } // end else
255       } // end outer if-elseif-else
256
257       return false;
258    } // end method isGreatThan
259
260    // test if one HugeInteger is less than another
261    public boolean isLessThan( HugeInteger compareValue )
262    {
263       return !( isGreaterThan( compareValue ) ||
264          isEqualTo( compareValue ) );
265    } // end method isLessThan
266
267    // test if one HugeInteger is great than or equal to another
268    public boolean isGreaterThanOrEqualTo( HugeInteger compareValue )
269    {
270       return !isLessThan( compareValue );
271    } // end method isGreaterThanOrEqualTo
272
273    // test if one HugeInteger is less than or equal to another
274    public boolean isLessThanOrEqualTo( HugeInteger compareValue )
```

```
275    {
276       return !isGreaterThan( compareValue );
277    } // end method isLessThanOrEqualTo
278
279    // test if one HugeInteger is zero
280    public boolean isZero()
281    {
282       // compare each digit
283       for ( int i = 0; i < DIGITS; i++ )
284       {
285          if ( integer[ i ] != 0 )
286             return false;
287       } // end for
288
289       return true;
290    } // end method isZero
291 } // end class HugeInteger
```

```
1  // Exercise 8.18 Solution: HugeIntegerTest.java
2  // Test class HugeInteger
3  import java.util.Scanner;
4
5  public class HugeIntegerTest
6  {
7     public static void main( String args[] )
8     {
9        Scanner input = new Scanner( System.in );
10
11       HugeInteger integer1 = new HugeInteger();
12       HugeInteger integer2 = new HugeInteger();
13
14       System.out.print( "Enter first HugeInteger: " );
15       integer1.input( input.next() );
16
17       System.out.print( "Enter second HugeInteger: " );
18       integer2.input( input.next() );
19
20       System.out.printf( "HugeInteger 1: %s\n", integer1.toString() );
21       System.out.printf( "HugeInteger 2: %s\n", integer2.toString() );
22
23       HugeInteger result;
24
25       // add two HugeIntegers
26       result = integer1.add( integer2 );
27       System.out.printf( "Add result: %s\n", result.toString() );
28
29       // subtract two HugeIntegers
30       result = integer1.subtract( integer2 );
31       System.out.printf( "Subtract result: %s\n", result.toString() );
32
33       // compare two HugeIntegers
34       System.out.printf(
35          "HugeInteger 1 is zero: %b\n", integer1.isZero() );
```

```
36         System.out.printf(
37            "HugeInteger 2 is zero: %b\n", integer2.isZero() );
38         System.out.printf(
39            "HugeInteger 1 is equal to HugeInteger 2: %b\n",
40            integer1.isEqualTo( integer2 ) );
41         System.out.printf(
42            "HugeInteger 1 is not equal to HugeInteger 2: %b\n",
43            integer1.isNotEqualTo( integer2 ) );
44         System.out.printf(
45            "HugeInteger 1 is greater than HugeInteger 2: %b\n",
46            integer1.isGreaterThan( integer2 ) );
47         System.out.printf(
48            "HugeInteger 1 is less than HugeInteger 2: %b\n",
49            integer1.isLessThan( integer2 ) );
50         System.out.printf(
51            "HugeInteger 1 is greater than or equal to HugeInteger 2: %b\n",
52            integer1.isGreaterThanOrEqualTo( integer2 ) );
53         System.out.printf(
54            "HugeInteger 1 is less than or equal to HugeInteger 2: %b\n",
55            integer1.isLessThanOrEqualTo( integer2 ) );
56      } // end main
57   } // end class HugeIntegerTest
```

```
Enter first HugeInteger: 123456789123456789
Enter second HugeInteger: 987654321987654321
HugeInteger 1: 123456789123456789
HugeInteger 2: 987654321987654321
Add result: 1111111111111111110
Subtract result: -864219752864219752
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:

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

```
4
5  public class TicTacToe
6  {
7     private final int BOARDSIZE = 3; // size of the board
8     private enum Status { WIN, DRAW, CONTINUE }; // game states
9     private int board[][]; // board representation
10    private boolean firstPlayer; // whether it's player 1's move
11    private boolean gameOver; // whether
12
13    // Constructor
14    public TicTacToe()
15    {
16       board = new int[ BOARDSIZE ][ BOARDSIZE ];
17       firstPlayer = true;
18       gameOver = false;
19    } // end Constructor
20
21    // start game
22    public void play()
23    {
24       Scanner input = new Scanner( System.in );
25       int row; // row for next move
26       int column; // column for next move
27
28       System.out.println( "Player 1's turn." );
29
30       while ( !gameOver )
31       {
32          int player = ( firstPlayer ? 1 : 2 );
33          // first player's turn
34
35          do
36          {
37             System.out.printf(
38                "Player %d: Enter row ( 0 <= row < 3 ): ", player );
39             row = input.nextInt();
40             System.out.printf(
41                "Player %d: Enter column ( 0 <= row < 3 ): ", player );
42             column = input.nextInt();
43          } while ( !validMove( row, column ) );
44
45          board[ row ][ column ] = player;
46
47          firstPlayer = !firstPlayer;
48
49          printBoard();
50          printStatus( player );
51       } // end while
52    } // end method makeMove
53
54    // show game status in status bar
55    public void printStatus( int player )
56    {
57       Status status = gameStatus();
58
59       // check game status
```

```
60          switch ( status )
61          {
62             case WIN:
63                System.out.println( "Player " + player + " wins." );
64                gameOver = true;
65                break;
66
67             case DRAW:
68                System.out.println( "Game is a draw." );
69                gameOver = true;
70                break;
71
72             case CONTINUE:
73                if ( player == 1 )
74                   System.out.println( "Player 2's turn." );
75                else
76                   System.out.println( "Player 1's turn." );
77                break;
78          } // end switch
79       } // end method printStatus
80
81       // get game status
82       public Status gameStatus()
83       {
84          int a;
85
86          // check for a win on diagonals
87          if ( board[ 0 ][ 0 ] != 0 && board[ 0 ][ 0 ] == board[ 1 ][ 1 ] &&
88             board[ 0 ][ 0 ] == board[ 2 ][ 2 ] )
89             return Status.WIN;
90          else if ( board[ 2 ][ 0 ] != 0 && board[ 2 ][ 0 ] ==
91             board[ 1 ][ 1 ] && board[ 2 ][ 0 ] == board[ 0 ][ 2 ] )
92             return Status.WIN;
93
94          // check for win in rows
95          for ( a = 0; a < 3; a++ )
96             if ( board[ a ][ 0 ] != 0 && board[ a ][ 0 ] ==
97                  board[ a ][ 1 ] && board[ a ][ 0 ] == board[ a ][ 2 ] )
98                return Status.WIN;
99
100         // check for win in columns
101         for ( a = 0; a < 3; a++ )
102            if ( board[ 0 ][ a ] != 0 && board[ 0 ][ a ] ==
103                 board[ 1 ][ a ] && board[ 0 ][ a ] == board[ 2 ][ a ] )
104               return Status.WIN;
105
106         // check for a completed game
107         for ( int r = 0; r < 3; r++ )
108            for ( int c = 0; c < 3; c++ )
109               if ( board[ r ][ c ] == 0 )
110                  return Status.CONTINUE; // game is not finished
111
112         return Status.DRAW; // game is a draw
113      } // end method gameStatus
114
```

```
115    // display board
116    public void printBoard()
117    {
118       System.out.println( " _____________________ " );
119
120       for ( int row = 0; row < BOARDSIZE; row++ )
121       {
122          System.out.println( "|       |       |       |" );
123
124          for ( int column = 0; column < BOARDSIZE; column++ )
125             printSymbol( column, board[ row ][ column ] );
126
127          System.out.println( "|_______|_______|_______|" );
128       } // end for
129    } // end method printBoard
130
131    // print moves
132    public void printSymbol( int column, int player )
133    {
134       switch ( player )
135       {
136          case 0:
137             System.out.print( "|       " );
138             break;
139
140          case 1:
141             System.out.print( "|   1   " );
142             break;
143
144          case 2:
145             System.out.print( "|   2   " );
146             break;
147       } // end switch
148
149       if ( column == 2 )
150          System.out.println( "|" );
151    } // end method printSymbol
152
153    // validate move
154    public boolean validMove( int row, int column )
155    {
156       return row >= 0 && row < 3 && column >= 0 && column < 3 &&
157          board[ row ][ column ] == 0;
158    } // end method validMove
159 } // end class TicTacToe
```

```
1  // Exercise 8.19 Solution: TicTacToeTest.java
2  // play a game of Tic Tac Toe
3  public class TicTacToeTest
4  {
5     public static void main( String args[] )
6     {
7        TicTacToe game = new TicTacToe();
8        game.printBoard();
```

```
 9         game.play();
10      } // end main
11   } // 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:

```
1  // GCS Exercise 8.1 Solution: MyLine.java
2  // Declaration of class MyLine
3  import java.awt.Color;
4  import java.awt.Graphics;
5
6  public class MyLine
7  {
8     private int x1; // x coordinate of first endpoint
9     private int y1; // y coordinate of first endpoint
10    private int x2; // x coordinate of second endpoint
11    private int y2; // y coordinate of second endpoint
12    private Color myColor; // color of this shape
13
14    // constructor initializes private vars with default values
15    public MyLine()
16    {
17       this( 0, 0, 0, 0, Color.BLACK ); // call constructor to set values
18    } // end MyLine no-argument constructor
19
20    // constructor with input values
21    public MyLine( int x1, int y1, int x2, int y2, Color color )
22    {
23       setX1( x1 ); // set x coordinate of first endpoint
24       setY1( y1 ); // set y coordinate of first endpoint
25       setX2( x2 ); // set x coordinate of second endpoint
```

```
26          setY2( y2 ); // set y coordinate of second endpoint
27          setColor( color ); // set the color
28       } // end MyLine constructor
29
30       // set the x-coordinate of the first point
31       public void setX1( int x1 )
32       {
33          this.x1 = ( x1 >= 0 ? x1 : 0 );
34       } // end method setX1
35
36       // get the x-coordinate of the first point
37       public int getX1()
38       {
39          return x1;
40       } // end method getX1
41
42       // set the x-coordinate of the second point
43       public void setX2( int x2 )
44       {
45          this.x2 = ( x2 >= 0 ? x2 : 0 );
46       } // end method setX2
47
48       // get the x-coordinate of the second point
49       public int getX2()
50       {
51          return x2;
52       } // end method getX2
53
54       // set the y-coordinate of the first point
55       public void setY1( int y1 )
56       {
57          this.y1 = ( y1 >= 0 ? y1 : 0 );
58       } // end method setY1
59
60       // get the y-coordinate of the first point
61       public int getY1()
62       {
63          return y1;
64       } // end method getY1
65
66       // set the y-coordinate of the second point
67       public void setY2( int y2 )
68       {
69          this.y2 = ( y2 >= 0 ? y2 : 0 );
70       } // end method setY2
71
72       // get the y-coordinate of the second point
73       public int getY2()
74       {
75          return y2;
76       } // end method getY2
77
78       // set the color
79       public void setColor( Color color )
```

```
80      {
81         myColor = color;
82      } // end method setColor
83
84      // get the color
85      public Color getColor()
86      {
87         return myColor;
88      } // end method getColor
89
90      // draw the line in the specified color
91      public void draw( Graphics g )
92      {
93         g.setColor( getColor() );
94         g.drawLine( getX1(), getY1(), getX2(), getY2() );
95      } // end method draw
96   } // end class MyLine
```

```
1   // GCS Exercise 8.1 Solution: MyOval.java
2   // Declaration of class MyOval
3   import java.awt.Color;
4   import java.awt.Graphics;
5
6   public class MyOval
7   {
8      private int x1; // x coordinate of first endpoint
9      private int y1; // y coordinate of first endpoint
10     private int x2; // x coordinate of second endpoint
11     private int y2; // y coordinate of second endpoint
12     private Color myColor; // Color of this oval
13     private boolean filled; // whether this shape is filled
14
15     // constructor initializes private vars with default values
16     public MyOval()
17     {
18        this( 0, 0, 0, 0, Color.BLACK, false ); // call constructor
19     } // end MyOval no-argument constructor
20
21     // constructor with input values
22     public MyOval( int x1, int y1, int x2, int y2,
23        Color color, boolean isFilled )
24     {
25        setX1( x1 ); // set x coordinate of first endpoint
26        setY1( y1 ); // set y coordinate of first endpoint
27        setX2( x2 ); // set x coordinate of second endpoint
28        setY2( y2 ); // set y coordinate of second endpoint
29        setColor( color ); // set the color
30        setFilled( isFilled );
31     } // end MyOval constructor
32
33     // set the x-coordinate of the first point
34     public void setX1( int x1 )
35     {
```

```
36        this.x1 = ( x1 >= 0 ? x1 : 0 );
37     } // end method setX1
38
39     // get the x-coordinate of the first point
40     public int getX1()
41     {
42        return x1;
43     } // end method getX1
44
45     // set the x-coordinate of the second point
46     public void setX2( int x2 )
47     {
48        this.x2 = ( x2 >= 0 ? x2 : 0 );
49     } // end method setX2
50
51     // get the x-coordinate of the second point
52     public int getX2()
53     {
54        return x2;
55     } // end method getX2
56
57     // set the y-coordinate of the first point
58     public void setY1( int y1 )
59     {
60        this.y1 = ( y1 >= 0 ? y1 : 0 );
61     } // end method setY1
62
63     // get the y-coordinate of the first point
64     public int getY1()
65     {
66        return y1;
67     } // end method getY1
68
69     // set the y-coordinate of the second point
70     public void setY2( int y2 )
71     {
72        this.y2 = ( y2 >= 0 ? y2 : 0 );
73     } // end method setY2
74
75     // get the y-coordinate of the second point
76     public int getY2()
77     {
78        return y2;
79     } // end method getY2
80
81     // set the color
82     public void setColor( Color color )
83     {
84        myColor = color;
85     } // end method setColor
86
87     // get the color
88     public Color getColor()
89     {
```

```
90          return myColor;
91       } // end method getColor
92
93       // get upper left x coordinate
94       public int getUpperLeftX()
95       {
96          return Math.min( getX1(), getX2() );
97       } // end method getUpperLeftX
98
99       // get upper left y coordinate
100      public int getUpperLeftY()
101      {
102         return Math.min( getY1(), getY2() );
103      } // end method getUpperLeftY
104
105      // get shape width
106      public int getWidth()
107      {
108         return Math.abs( getX2() - getX1() );
109      } // end method getWidth
110
111      // get shape height
112      public int getHeight()
113      {
114         return Math.abs( getY2() - getY1() );
115      } // end method getHeight
116
117      // determines whether this shape is filled
118      public boolean isFilled()
119      {
120         return filled;
121      } // end method is filled
122
123      // sets whether this shape is filled
124      public void setFilled( boolean isFilled )
125      {
126         filled = isFilled;
127      } // end method setFilled
128
129      // draws an oval in the specified color
130      public void draw( Graphics g )
131      {
132         g.setColor( getColor() );
133
134         if ( isFilled() )
135            g.fillOval( getUpperLeftX(), getUpperLeftY(),
136               getWidth(), getHeight() );
137         else
138            g.drawOval( getUpperLeftX(), getUpperLeftY(),
139               getWidth(), getHeight() );
140      } // end method draw
141   } // end class MyOval
```

```
1  // GCS Exercise 8.1 Solution: MyRect.java
2  // Declaration of class MyRect
3  import java.awt.Color;
4  import java.awt.Graphics;
5
6  public class MyRect
7  {
8     private int x1; // x coordinate of first endpoint
9     private int y1; // y coordinate of first endpoint
10    private int x2; // x coordinate of second endpoint
11    private int y2; // y coordinate of second endpoint
12    private Color myColor; // Color of this rectangle
13    private boolean filled; // whether this shape is filled
14
15    // constructor initializes private vars with default values
16    public MyRect()
17    {
18       this( 0, 0, 0, 0, Color.BLACK, false ); // call constructor
19    } // end MyRect no-argument constructor
20
21    // constructor with input values
22    public MyRect( int x1, int y1, int x2, int y2,
23       Color color, boolean isFilled )
24    {
25       setX1( x1 ); // set x coordinate of first endpoint
26       setY1( y1 ); // set y coordinate of first endpoint
27       setX2( x2 ); // set x coordinate of second endpoint
28       setY2( y2 ); // set y coordinate of second endpoint
29       setColor( color ); // set the color
30       setFilled( isFilled );
31    } // end MyRect constructor
32
33    // set the x-coordinate of the first point
34    public void setX1( int x1 )
35    {
36      this.x1 = ( x1 >= 0 ? x1 : 0 );
37    } // end method setX1
38
39    // get the x-coordinate of the first point
40    public int getX1()
41    {
42      return x1;
43    } // end method getX1
44
45    // set the x-coordinate of the second point
46    public void setX2( int x2 )
47    {
48      this.x2 = ( x2 >= 0 ? x2 : 0 );
49    } // end method setX2
50
51    // get the x-coordinate of the second point
52    public int getX2()
53    {
54      return x2;
```

```
55      } // end method getX2
56
57      // set the y-coordinate of the first point
58      public void setY1( int y1 )
59      {
60        this.y1 = ( y1 >= 0 ? y1 : 0 );
61      } // end method setY1
62
63      // get the y-coordinate of the first point
64      public int getY1()
65      {
66        return y1;
67      } // end method getY1
68
69      // set the y-coordinate of the second point
70      public void setY2( int y2 )
71      {
72        this.y2 = ( y2 >= 0 ? y2 : 0 );
73      } // end method setY2
74
75      // get the y-coordinate of the second point
76      public int getY2()
77      {
78        return y2;
79      } // end method getY2
80
81      // set the color
82      public void setColor( Color color )
83      {
84         myColor = color;
85      } // end method setColor
86
87      // get the color
88      public Color getColor()
89      {
90         return myColor;
91      } // end method getColor
92
93      // get upper left x coordinate
94      public int getUpperLeftX()
95      {
96         return Math.min( getX1(), getX2() );
97      } // end method getUpperLeftX
98
99      // get upper left y coordinate
100     public int getUpperLeftY()
101     {
102        return Math.min( getY1(), getY2() );
103     } // end method getUpperLeftY
104
105     // get shape width
106     public int getWidth()
107     {
108        return Math.abs( getX2() - getX1() );
```

```
109     } // end method getWidth
110
111     // get shape height
112     public int getHeight()
113     {
114        return Math.abs( getY2() - getY1() );
115     } // end method getHeight
116
117     // determines whether this shape is filled
118     public boolean isFilled()
119     {
120        return filled;
121     } // end method is filled
122
123     // sets whether this shape is filled
124     public void setFilled( boolean isFilled )
125     {
126        filled = isFilled;
127     } // end method setFilled
128
129     // draws a rectangle in the specified color
130     public void draw( Graphics g )
131     {
132        g.setColor( getColor() );
133
134        if ( isFilled() )
135           g.fillRect( getUpperLeftX(), getUpperLeftY(),
136              getWidth(), getHeight() );
137        else
138           g.drawRect( getUpperLeftX(), getUpperLeftY(),
139              getWidth(), getHeight() );
140     } // end method draw
141  } // end class MyRect
```

```
 1  // GCS Exercise 8.1 Solution: DrawPanel.java
 2  // Program that uses classes MyLine, MyOval and MyRect to draw
 3  // random shapes
 4  import java.awt.Color;
 5  import java.awt.Graphics;
 6  import java.util.Random;
 7  import javax.swing.JPanel;
 8
 9  public class DrawPanel extends JPanel
10  {
11     private Random randomNumbers = new Random();
12
13     private MyLine lines[]; // array on lines
14     private MyOval ovals[]; // array of ovals
15     private MyRect rectangles[]; // array of rectangles
16
17     // constructor, creates a panel with random shapes
18     public DrawPanel()
19     {
```

```
20          setBackground( Color.WHITE );
21
22          lines = new MyLine[ 1 + randomNumbers.nextInt( 5 ) ];
23          ovals = new MyOval[ 1 + randomNumbers.nextInt( 5 ) ];
24          rectangles = new MyRect[ 1 + randomNumbers.nextInt( 5 ) ];
25
26          // create lines
27          for ( int count = 0; count < lines.length; count++ )
28          {
29             // generate random coordinates
30             int x1 = randomNumbers.nextInt( 450 );
31             int y1 = randomNumbers.nextInt( 450 );
32             int x2 = randomNumbers.nextInt( 450 );
33             int y2 = randomNumbers.nextInt( 450 );
34
35             // generate a random color
36             Color color = new Color( randomNumbers.nextInt( 256 ),
37                randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );
38
39             // add the line to the list of lines to be displayed
40             lines[ count ] = new MyLine( x1, y1, x2, y2, color );
41          } // end for
42
43          // create ovals
44          for ( int count = 0; count < ovals.length; count++ )
45          {
46             // generate random coordinates
47             int x1 = randomNumbers.nextInt( 450 );
48             int y1 = randomNumbers.nextInt( 450 );
49             int x2 = randomNumbers.nextInt( 450 );
50             int y2 = randomNumbers.nextInt( 450 );
51
52             // generate a random color
53             Color color = new Color( randomNumbers.nextInt( 256 ),
54                randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );
55
56             // get filled property
57             boolean filled = randomNumbers.nextBoolean();
58
59             // add the line to the oval of ovals to be displayed
60             ovals[ count ] = new MyOval( x1, y1, x2, y2, color, filled );
61          } // end for
62
63          // create rectangles
64          for ( int count = 0; count < rectangles.length; count++ )
65          {
66             // generate random coordinates
67             int x1 = randomNumbers.nextInt( 450 );
68             int y1 = randomNumbers.nextInt( 450 );
69             int x2 = randomNumbers.nextInt( 450 );
70             int y2 = randomNumbers.nextInt( 450 );
71
72             // generate a random color
73             Color color = new Color( randomNumbers.nextInt( 256 ),
```

```
74              randomNumbers.nextInt( 256 ), randomNumbers.nextInt( 256 ) );
75
76           // get filled property
77           boolean filled = randomNumbers.nextBoolean();
78
79           // add the rectangle to the list of rectangles to be displayed
80           rectangles[ count ] =
81              new MyRect( x1, y1, x2, y2, color, filled );
82        } // end for
83     } // end DrawPanel constructor
84
85     // for each shape array, draw the individual shapes
86     public void paintComponent( Graphics g )
87     {
88        super.paintComponent( g );
89
90        // draw the lines
91        for ( MyLine line : lines )
92           line.draw( g );
93
94        // draw the ovals
95        for ( MyOval oval: ovals )
96           oval.draw( g );
97
98        // drat the rectangles
99        for ( MyRect rectangle : rectangles )
100          rectangle.draw( g );
101    } // end method paintComponent
102 } // end class DrawPanel
```

```
1  // GCS Exercise 8.1 Solution: TestDraw.java
2  // Test application to display a DrawPanel
3  import javax.swing.JFrame;
4
5  public class TestDraw
6  {
7     public static void main( String args[] )
8     {
9        DrawPanel panel = new DrawPanel();
10       JFrame application = new JFrame();
11
12       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13       application.add( panel );
14       application.setSize( 500, 500 );
15       application.setVisible( true );
16    } // end main
17 } // end class TestDraw
```