10

Object-Oriented Programming: Polymorphism

One Ring to rule them all,
One Ring to find them,
One Ring to bring them all
and in the darkness bind them.

—John Ronald Reuel Tolkien

General propositions do not decide concrete cases.

—Oliver Wendell Holmes

A philosopher of imposing stature doesn't think in a vacuum. Even his most abstract ideas are, to some extent, conditioned by what is or is not known in the time when he lives.

—Alfred North Whitehead

Why art thou cast down, O my soul?

—Psalms 42:5

OBJECTIVES

In this chapter you will learn:

- The concept of polymorphism.
- To use overridden methods to effect polymorphism.
- To distinguish between abstract and concrete classes.
- To declare abstract methods to create abstract classes.
- How polymorphism makes systems extensible and maintainable.
- To determine an object's type at execution time.
- To declare and implement interfaces.

Student Solution Exercises

10.3 How does polymorphism enable you to program "in the general" rather than "in the specific"? Discuss the key advantages of programming "in the general."

ANS: Polymorphism enables the programmer to concentrate on the common operations that are applied to objects of all the classes in a hierarchy. The general processing capabilities can be separated from any code that is specific to each class. Those general portions of the code can accommodate new classes without modification. In some polymorphic applications, only the code that creates the objects needs to be modified to extend the system with new classes.

10.5 What are abstract methods? Describe the circumstances in which an abstract method would be appropriate.

ANS: An abstract method is one with keyword `abstract` in its declaration. Abstract methods do not provide implementations. Each concrete subclass of an abstract superclass must provide concrete implementations of the superclass's abstract methods. An abstract method is appropriate when it does not make sense to provide an implementation for a method in a superclass (i.e., some additional subclass-specific data is required to implement the method in a meaningful manner).

10.7 Discuss four ways in which you can assign superclass and subclass references to variables of superclass and subclass types.

ANS: 1) Assigning a superclass reference to a superclass variable. 2) Assigning a subclass reference to a subclass variable. 3) Assigning a subclass object's reference to a superclass variable is safe, because the subclass object *is an* object of its superclass. However, this reference can be used to refer only to superclass members. If this code refers to subclass-only members through the superclass variable, the compiler reports errors. 4) Attempting to assign a superclass object's reference to a subclass variable is a compilation error. To avoid this error, the superclass reference must be downcast to a subclass type explicitly. At execution time, if the object to which the reference refers is not a subclass object, an exception will occur. The `instanceof` operator can be used to ensure that such a cast is performed only if the object is a subclass object.

10.9 *(Payroll System Modification)* Modify the payroll system of Figs. 10.4–10.9 to include `private` instance variable `birthDate` in class `Employee`. Use class `Date` of Fig. 8.7 to represent an employee's birthday. Add *get* methods to class `Date` and replace method `toDateString` with method `toString`. Assume that payroll is processed once per month. Create an array of `Employee` variables to store references to the various employee objects. In a loop, calculate the payroll for each `Employee` (polymorphically), and add a $100.00 bonus to the person's payroll amount if the current month is the one in which the `Employee`'s birthday occurs.

ANS:

```
1  // Exercise 10.9 Solution: Employee.java
2  // Employee abstract superclass.
3
4  public abstract class Employee
5  {
6     private String firstName;
7     private String lastName;
8     private String socialSecurityNumber;
9     private Date birthDate;
10
```

```
11      // six-argument constructor
12      public Employee( String first, String last, String ssn,
13         int month, int day, int year )
14      {
15         firstName = first;
16         lastName = last;
17         socialSecurityNumber = ssn;
18         birthDate = new Date( month, day, year );
19      } // end six-argument Employee constructor
20
21      // set first name
22      public void setFirstName( String first )
23      {
24         firstName = first;
25      } // end method setFirstName
26
27      // return first name
28      public String getFirstName()
29      {
30         return firstName;
31      } // end method getFirstName
32
33      // set last name
34      public void setLastName( String last )
35      {
36         lastName = last;
37      } // end method setLastName
38
39      // return last name
40      public String getLastName()
41      {
42         return lastName;
43      } // end method getLastName
44
45      // set social security number
46      public void setSocialSecurityNumber( String ssn )
47      {
48         socialSecurityNumber = ssn;  // should validate
49      } // end method setSocialSecurityNumber
50
51      // return social security number
52      public String getSocialSecurityNumber()
53      {
54         return socialSecurityNumber;
55      } // end method getSocialSecurityNumber
56
57      // set birth date
58      public void setBirthDate( int month, int day, int year )
59      {
60         birthDate = new Date( month, day, year );
61      } // end method setBirthDate
62
63      // return birth date
64      public Date getBirthDate()
65      {
66         return birthDate;
```

```
67     } // end method getBirthDate
68
69     // return String representation of Employee object
70     public String toString()
71     {
72        return String.format( "%s %s\n%s: %s\n%s: %s",
73           getFirstName(), getLastName(),
74           "social security number", getSocialSecurityNumber(),
75           "birth date", getBirthDate() );
76     } // end method toString
77
78     // abstract method overridden by subclasses
79     public abstract double earnings();
80  } // end abstract class Employee
```

```
1  // Exercise 10.9 Solution: Date.java
2  // Date class declaration with get methods added.
3
4  public class Date
5  {
6     private int month; // 1-12
7     private int day;   // 1-31 based on month
8     private int year;  // any year
9
10    // constructor: call checkMonth to confirm proper value for month;
11    // call checkDay to confirm proper value for day
12    public Date( int theMonth, int theDay, int theYear )
13    {
14       month = checkMonth( theMonth ); // validate month
15       year = theYear; // could validate year
16       day = checkDay( theDay ); // validate day
17
18       System.out.printf(
19          "Date object constructor for date %s\n", toString() );
20    } // end Date constructor
21
22    // utility method to confirm proper month value
23    private int checkMonth( int testMonth )
24    {
25       if ( testMonth > 0 && testMonth <= 12 ) // validate month
26          return testMonth;
27       else // month is invalid
28       {
29          System.out.printf( "Invalid month (%d) set to 1.\n", testMonth );
30          return 1; // maintain object in consistent state
31       } // end else
32    } // end method checkMonth
33
34    // utility method to confirm proper day value based on month and year
35    private int checkDay( int testDay )
36    {
37       int daysPerMonth[] =
38          { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

```
39
40        // check if day in range for month
41        if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
42           return testDay;
43
44        // check for leap year
45        if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
46             ( year % 4 == 0 && year % 100 != 0 ) ) )
47           return testDay;
48
49        System.out.printf( "Invalid day (%d) set to 1.\n", testDay );
50
51        return 1;  // maintain object in consistent state
52     } // end method checkDay
53
54     // return day
55     public int getDay()
56     {
57        return day;
58     } // end method getDay
59
60     // return month
61     public int getMonth()
62     {
63        return month;
64     } // end method getMonth
65
66     // return year
67     public int getYear()
68     {
69        return year;
70     } // end method getYear
71
72     // return a String of the form month/day/year
73     public String toString()
74     {
75        return String.format( "%d/%d/%d", month, day, year );
76     } // end method toString
77  } // end class Date
```

```
1  // Exercise 10.9 Solution: SalariedEmployee.java
2  // SalariedEmployee class derived from Employee.
3
4  public class SalariedEmployee extends Employee
5  {
6     private double weeklySalary;
7
8     // seven-argument constructor
9     public SalariedEmployee( String first, String last, String ssn,
10       int month, int day, int year, double salary )
11    {
12       super( first, last, ssn, month, day, year );
13       setWeeklySalary( salary );
14    } // end seven-argument SalariedEmployee constructor
```

```
15
16    // set salary
17    public void setWeeklySalary( double salary )
18    {
19       weeklySalary = salary < 0.0 ? 0.0 : salary;
20    } // end method setWeeklySalary
21
22    // return salary
23    public double getWeeklySalary()
24    {
25       return weeklySalary;
26    } // end method getWeeklySalary
27
28    // calculate earnings; override abstract method earnings in Employee
29    public double earnings()
30    {
31       return getWeeklySalary();
32    } // end method earnings
33
34    // return String representation of SalariedEmployee object
35    public String toString()
36    {
37       return String.format( "salaried employee: %s\n%s: $%,.2f",
38          super.toString(), "weekly salary", getWeeklySalary() );
39    } // end method toString
40 } // end class SalariedEmployee
```

```
1  // Exercise 10.9 Solution: HourlyEmployee.java
2  // HourlyEmployee class derived from Employee.
3
4  public class HourlyEmployee extends Employee
5  {
6     private double wage; // wage per hour
7     private double hours; // hours worked for week
8
9     // eight-argument constructor
10    public HourlyEmployee( String first, String last, String ssn,
11       int month, int day, int year,
12       double hourlyWage, double hoursWorked )
13    {
14       super( first, last, ssn, month, day, year );
15       setWage( hourlyWage );
16       setHours( hoursWorked );
17    } // end eight-argument HourlyEmployee constructor
18
19    // set wage
20    public void setWage( double hourlyWage )
21    {
22       wage = hourlyWage < 0.0 ? 0.0 : hourlyWage;
23    } // end method setWage
24
25    // return wage
26    public double getWage()
27    {
```

```
28          return wage;
29       } // end method getWage
30
31       // set hours worked
32       public void setHours( double hoursWorked )
33       {
34          hours = ( ( hoursWorked >= 0.0 ) && ( hoursWorked <= 168.0 ) ) ?
35             hoursWorked : 0.0;
36       } // end method setHours
37
38       // return hours worked
39       public double getHours()
40       {
41          return hours;
42       } // end method getHours
43
44       // calculate earnings; override abstract method earnings in Employee
45       public double earnings()
46       {
47          if ( getHours() <= 40 ) // no overtime
48             return getWage() * getHours();
49          else
50             return 40 * getWage() + ( getHours() - 40 ) * getWage() * 1.5;
51       } // end method earnings
52
53       // return String representation of HourlyEmployee object
54       public String toString()
55       {
56          return String.format( "hourly employee: %s\n%s: $%,.2f; %s: %,.2f",
57             super.toString(), "hourly wage", getWage(),
58             "hours worked", getHours() );
59       } // end method toString
60    } // end class HourlyEmployee
```

```
1  // Exercise 10.9 Solution: CommissionEmployee.java
2  // CommissionEmployee class derived from Employee.
3
4  public class CommissionEmployee extends Employee
5  {
6     private double grossSales; // gross weekly sales
7     private double commissionRate; // commission percentage
8
9     // eight-argument constructor
10    public CommissionEmployee( String first, String last, String ssn,
11       int month, int day, int year, double sales, double rate )
12    {
13       super( first, last, ssn, month, day, year );
14       setGrossSales( sales );
15       setCommissionRate( rate );
16    } // end eight-argument CommissionEmployee constructor
17
18    // set commission rate
19    public void setCommissionRate( double rate )
20    {
21       commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
```

```
22     } // end method setCommissionRate
23
24     // return commission rate
25     public double getCommissionRate()
26     {
27        return commissionRate;
28     } // end method getCommissionRate
29
30     // set gross sales amount
31     public void setGrossSales( double sales )
32     {
33        grossSales = sales < 0.0 ? 0.0 : sales;
34     } // end method setGrossSales
35
36     // return gross sales amount
37     public double getGrossSales()
38     {
39        return grossSales;
40     } // end method getGrossSales
41
42     // calculate earnings; override abstract method earnings in Employee
43     public double earnings()
44     {
45        return getCommissionRate() * getGrossSales();
46     } // end method earnings
47
48     // return String representation of CommissionEmployee object
49     public String toString()
50     {
51        return String.format( "%s: %s\n%s: $%,.2f; %s: %.2f",
52           "commission employee", super.toString(),
53           "gross sales", getGrossSales(),
54           "commission rate", getCommissionRate() );
55     } // end method toString
56  } // end class CommissionEmployee
```

```
1  // Exercise 10.9 Solution: BasePlusCommissionEmployee.java
2  // BasePlusCommissionEmployee class derived from CommissionEmployee.
3
4  public class BasePlusCommissionEmployee extends CommissionEmployee
5  {
6     private double baseSalary; // base salary per week
7
8     // nine-argument constructor
9     public BasePlusCommissionEmployee( String first, String last,
10       String ssn, int month, int day, int year,
11       double sales, double rate, double salary )
12    {
13       super( first, last, ssn, month, day, year, sales, rate );
14       setBaseSalary( salary );
15    } // end nine-argument BasePlusCommissionEmployee constructor
16
17    // set base salary
```

```
18      public void setBaseSalary( double salary )
19      {
20         baseSalary = salary < 0.0 ? 0.0 : salary; // non-negative
21      } // end method setBaseSalary
22
23      // return base salary
24      public double getBaseSalary()
25      {
26         return baseSalary;
27      } // end method getBaseSalary
28
29      // calculate earnings; override method earnings in CommissionEmployee
30      public double earnings()
31      {
32         return getBaseSalary() + super.earnings();
33      } // end method earnings
34
35      // return String representation of BasePlusCommissionEmployee object
36      public String toString()
37      {
38         return String.format( "%s %s; %s: $%,.2f",
39            "base-salaried", super.toString(),
40            "base salary", getBaseSalary() );
41      } // end method toString
42   } // end class BasePlusCommissionEmployee
```

```
1   // Exercise 10.9 Solution: PayrollSystemTest.java
2   // Employee hierarchy test program.
3   import java.util.Scanner; // program uses Scanner to obtain user input
4
5   public class PayrollSystemTest
6   {
7      public static void main( String args[] )
8      {
9         // create subclass objects
10        SalariedEmployee salariedEmployee =
11           new SalariedEmployee(
12           "John", "Smith", "111-11-1111", 6, 15, 1944, 800.00 );
13        HourlyEmployee hourlyEmployee =
14           new HourlyEmployee(
15           "Karen", "Price", "222-22-2222", 12, 29, 1960, 16.75, 40 );
16        CommissionEmployee commissionEmployee =
17           new CommissionEmployee(
18           "Sue", "Jones", "333-33-3333", 9, 8, 1954, 10000, .06 );
19        BasePlusCommissionEmployee basePlusCommissionEmployee =
20           new BasePlusCommissionEmployee(
21           "Bob", "Lewis", "444-44-4444", 3, 2, 1965, 5000, .04, 300 );
22
23        System.out.println( "Employees processed individually:\n" );
24
25        System.out.printf( "%s\n%s: $%,.2f\n\n",
26           salariedEmployee, "earned", salariedEmployee.earnings() );
27        System.out.printf( "%s\n%s: $%,.2f\n\n",
28           hourlyEmployee, "earned", hourlyEmployee.earnings() );
29        System.out.printf( "%s\n%s: $%,.2f\n\n",
```

```
30            commissionEmployee, "earned", commissionEmployee.earnings() );
31         System.out.printf( "%s\n%s: $%,.2f\n\n",
32            basePlusCommissionEmployee,
33            "earned", basePlusCommissionEmployee.earnings() );
34
35         // create four-element Employee array
36         Employee employees[] = new Employee[ 4 ];
37
38         // initialize array with Employees
39         employees[ 0 ] = salariedEmployee;
40         employees[ 1 ] = hourlyEmployee;
41         employees[ 2 ] = commissionEmployee;
42         employees[ 3 ] = basePlusCommissionEmployee;
43
44         Scanner input = new Scanner( System.in ); // to get current month
45         int currentMonth;
46
47         // get and validate current month
48         do
49         {
50            System.out.print( "Enter the current month (1 - 12): " );
51            currentMonth = input.nextInt();
52            System.out.println();
53         } while ( ( currentMonth < 1 ) || ( currentMonth > 12 ) );
54
55         System.out.println( "Employees processed polymorphically:\n" );
56
57         // generically process each element in array employees
58         for ( Employee currentEmployee : employees )
59         {
60            System.out.println( currentEmployee ); // invokes toString
61
62            // determine whether element is a BasePlusCommissionEmployee
63            if ( currentEmployee instanceof BasePlusCommissionEmployee )
64            {
65               // downcast Employee reference to
66               // BasePlusCommissionEmployee reference
67               BasePlusCommissionEmployee employee =
68                  ( BasePlusCommissionEmployee ) currentEmployee;
69
70               double oldBaseSalary = employee.getBaseSalary();
71               employee.setBaseSalary( 1.10 * oldBaseSalary );
72               System.out.printf(
73                  "new base salary with 10%% increase is: $%,.2f\n",
74                  employee.getBaseSalary() );
75            } // end if
76
77            // if month of employee's birthday, add $100 to salary
78            if ( currentMonth == currentEmployee.getBirthDate().getMonth() )
79               System.out.printf(
80                  "earned $%,.2f %s\n\n", currentEmployee.earnings(),
81                  "plus $100.00 birthday bonus" );
82            else
```

```
83                System.out.printf(
84                   "earned $%,.2f\n\n", currentEmployee.earnings() );
85             } // end for
86
87             // get type name of each object in employees array
88             for ( int j = 0; j < employees.length; j++ )
89                System.out.printf( "Employee %d is a %s\n", j,
90                   employees[ j ].getClass().getName() );
91          } // end main
92       } // end class PayrollSystemTest
```

```
Date object constructor for date 6/15/1944
Date object constructor for date 12/29/1960
Date object constructor for date 9/8/1954
Date object constructor for date 3/2/1965
Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
birth date: 6/15/1944
weekly salary: $800.00
earned: $800.00

hourly employee: Karen Price
social security number: 222-22-2222
birth date: 12/29/1960
hourly wage: $16.75; hours worked: 40.00
earned: $670.00

commission employee: Sue Jones
social security number: 333-33-3333
birth date: 9/8/1954
gross sales: $10,000.00; commission rate: 0.06
earned: $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
birth date: 3/2/1965
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
earned: $500.00

Enter the current month (1 - 12): 3
```

```
Employees processed polymorphically:

salaried employee: John Smith
social security number: 111-11-1111
birth date: 6/15/1944
weekly salary: $800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
birth date: 12/29/1960
hourly wage: $16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
birth date: 9/8/1954
gross sales: $10,000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
birth date: 3/2/1965
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00 plus $100.00 birthday bonus

Employee 0 is a SalariedEmployee
Employee 1 is a HourlyEmployee
Employee 2 is a CommissionEmployee
Employee 3 is a BasePlusCommissionEmployee
```

10.10 *(Shape Hierarchy)* Implement the `Shape` hierarchy shown in Fig. 9.3. Each `TwoDimensionalShape` should contain method `getArea` to calculate the area of the two-dimensional shape. Each `ThreeDimensionalShape` should have methods `getArea` and `getVolume` to calculate the surface area and volume, respectively, of the three-dimensional shape. Create a program that uses an array of `Shape` references to objects of each concrete class in the hierarchy. The program should print a text description of the object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a `TwoDimensionalShape` or a `ThreeDimensionalShape`. If it is a `TwoDimensionalShape`, display its area. If it is a `ThreeDimensionalShape`, display its area and volume.

ANS:

```
1  // Exercise 10.10 Solution: Shape.java
2  // Definition of class Shape.
3
4  public abstract class Shape
5  {
6     private int x; // x coordinate
7     private int y; // y coordinate
8
9     // two-argument constructor
10    public Shape( int x, int y )
11    {
12       this.x = x;
```

```
13          this.y = y;
14       } // end two-argument Shape constructor
15
16       // set x coordinate
17       public void setX( int x )
18       {
19          this.x = x;
20       } // end method setX
21
22       // set y coordinate
23       public void setY( int y )
24       {
25          this.y = y;
26       } // end method setY
27
28       // get x coordinate
29       public int getX()
30       {
31          return x;
32       } // end method getX
33
34       // get y coordinate
35       public int getY()
36       {
37          return y;
38       } // end method getY
39
40       // return String representation of Shape object
41       public String toString()
42       {
43          return String.format( "(%d, %d)", getX(), getY() );
44       }
45
46       // abstract methods
47       public abstract String getName();
48    } // end class Shape
```

```
1  // Exercise 10.10 Solution: TwoDimensionalShape.java
2  // Definition of class TwoDimensionalShape.
3
4  public abstract class TwoDimensionalShape extends Shape
5  {
6     private int dimension1;
7     private int dimension2;
8
9     // four-argument constructor
10    public TwoDimensionalShape( int x, int y, int d1, int d2 )
11    {
12       super( x, y );
13       dimension1 = d1;
14       dimension2 = d2;
15    } // end four-argument TwoDimensionalShape constructor
16
17    // set methods
18    public void setDimension1( int d )
```

```
19      {
20         dimension1 = d;
21      } // end method setDimension1
22
23      public void setDimension2( int d )
24      {
25         dimension2 = d;
26      } // end method setDimension2
27
28      // get methods
29      public int getDimension1()
30      {
31         return dimension1;
32      } // end method getDimension1
33
34      public int getDimension2()
35      {
36         return dimension2;
37      } // end method getDimension2
38
39      // abstract method
40      public abstract int getArea();
41   } // end class TwoDimensionalShape
```

```
1   // Exercise 10.10 Solution: Circle.java
2   // Definition of class Circle.
3
4   public class Circle extends TwoDimensionalShape
5   {
6      // three-argument constructor
7      public Circle( int x, int y, int radius )
8      {
9         super( x, y, radius, radius );
10     } // end three-argument Circle constructor
11
12     // overridden methods
13     public String getName()
14     {
15        return "Circle";
16     } // end method getName
17
18     public int getArea()
19     {
20        return ( int )
21           ( Math.PI * getRadius() * getRadius() );
22     } // end method getArea
23
24     // set method
25     public void setRadius( int radius )
26     {
27        setDimension1( radius );
28        setDimension2( radius );
29     } // end method setRadius
30
```

```
31      // get method
32      public int getRadius()
33      {
34         return getDimension1();
35      } // end method getRadius
36
37      public String toString()
38      {
39         return String.format( "%s %s: %d\n",
40            super.toString(), "radius", getRadius() );
41      } // end method toString
42   } // end class Circle
```

```
1   // Exercise 10.10 Solution: Square.java
2   // Definition of class Square.
3
4   public class Square extends TwoDimensionalShape
5   {
6      // three-argument constructor
7      public Square( int x, int y, int side )
8      {
9         super( x, y, side, side );
10     } // end three-argument Square constructor
11
12     // overridden methods
13     public String getName()
14     {
15        return "Square";
16     } // end method getName
17
18     public int getArea()
19     {
20        return getSide() * getSide();
21     } // end method getArea
22
23     // set method
24     public void setSide( int side )
25     {
26        setDimension1( side );
27        setDimension2( side );
28     } // end method setSide
29
30     // get method
31     public int getSide()
32     {
33        return getDimension1();
34     } // end method getSide
35
36     public String toString()
37     {
38        return String.format( "%s %s: %d\n",
39           super.toString(), "side", getSide() );
40     } // end method toString
```

```
41 } // end class Square
```

```
1  // Exercise 10.10 Solution: ThreeDimensionalShape.java
2  // Definition of class ThreeDimensionalShape.
3
4  public abstract class ThreeDimensionalShape extends Shape
5  {
6     private int dimension1;
7     private int dimension2;
8     private int dimension3;
9
10    // five-argument constructor
11    public ThreeDimensionalShape(
12       int x, int y, int d1, int d2, int d3 )
13    {
14       super( x, y );
15       dimension1 = d1;
16       dimension2 = d2;
17       dimension3 = d3;
18    } // end five-argument ThreeDimensionalShape constructor
19
20    // set methods
21    public void setDimension1( int d )
22    {
23       dimension1 = d;
24    } // end method setDimension1
25
26    public void setDimension2( int d )
27    {
28       dimension2 = d;
29    } // end method setDimension2
30
31    public void setDimension3( int d )
32    {
33       dimension3 = d;
34    } // end method setDimension3
35
36    // get methods
37    public int getDimension1()
38    {
39       return dimension1;
40    } // end method getDimension1
41
42    public int getDimension2()
43    {
44       return dimension2;
45    } // end method getDimension2
46
47    public int getDimension3()
48    {
49       return dimension3;
50    } // end method getDimension3
51
52    // abstract methods
```

```
53    public abstract int getArea();
54    public abstract int getVolume();
55 } // end class ThreeDimensionalShape
```

```
1  // Exercise 10.10 Solution: Sphere.java
2  // Definition of class Sphere.
3
4  public class Sphere extends ThreeDimensionalShape
5  {
6     // three-argument constructor
7     public Sphere( int x, int y, int radius )
8     {
9        super( x, y, radius, radius, radius );
10    } // end three-argument Shape constructor
11
12    // overridden methods
13    public String getName()
14    {
15       return "Sphere";
16    } // end method getName
17
18    public int getArea()
19    {
20       return ( int ) ( 4 * Math.PI * getRadius() * getRadius() );
21    } // end method getArea
22
23    public int getVolume()
24    {
25       return ( int ) ( 4.0 / 3.0 * Math.PI *
26          getRadius() * getRadius() * getRadius() );
27    } // end method getVolume
28
29    // set method
30    public void setRadius( int radius )
31    {
32       setDimension1( radius );
33       setDimension2( radius );
34       setDimension3( radius );
35    } // end method setRadius
36
37    // get method
38    public int getRadius()
39    {
40       return getDimension1();
41    } // end method getRadius
42
43    public String toString()
44    {
45       return String.format( "%s %s: %d\n",
46          super.toString(), "radius", getRadius() );
47    } // end method toString
48 } // end class Sphere
```

```
1  // Exercise 10.10 Solution: Cube.java
2  // Definition of class Cube.
3
4  public class Cube extends ThreeDimensionalShape
5  {
6     // three-argument constructor
7     public Cube( int x, int y, int side )
8     {
9        super( x, y, side, side, side );
10    } // end three-argument Cube constructor
11
12    // overridden methods
13    public String getName()
14    {
15       return "Cube";
16    } // end method getName
17
18    public int getArea()
19    {
20       return ( int ) ( 6 * getSide() * getSide() );
21    } // end method getArea
22
23    public int getVolume()
24    {
25       return ( int ) ( getSide() * getSide() * getSide() );
26    } // end method getVolume
27
28    // set method
29    public void setSide( int side )
30    {
31       setDimension1( side );
32       setDimension2( side );
33       setDimension3( side );
34    } // end method setSide
35
36    // get method
37    public int getSide()
38    {
39       return getDimension1();
40    } // end method getSide
41
42    public String toString()
43    {
44       return String.format( "%s %s: %d\n",
45          super.toString(), "side", getSide() );
46    } // end method toString
47 } // end class Cube
```

```
1  // Exercise 10.10 Solution: ShapeTest.java
2  // Program tests the Shape hierarchy.
3
4  public class ShapeTest
5  {
6     // create Shape objects and display their information
7     public static void main( String args[] )
8     {
```

```
         Shape shapes[] = new Shape[ 4 ];
         shapes[ 0 ] = new Circle( 22, 88, 4 );
         shapes[ 1 ] = new Square( 71, 96, 10 );
         shapes[ 2 ] = new Sphere( 8, 89, 2 );
         shapes[ 3 ] = new Cube( 79, 61, 8 );

         // call method print on all shapes
         for ( Shape currentShape : shapes )
         {
            System.out.printf( "%s: %s",
               currentShape.getName(), currentShape );

            if ( currentShape instanceof TwoDimensionalShape )
            {
               TwoDimensionalShape twoDimensionalShape =
                  ( TwoDimensionalShape ) currentShape;

               System.out.printf( "%s's area is %s\n",
                  currentShape.getName(), twoDimensionalShape.getArea() );
            } // end if

            if ( currentShape instanceof ThreeDimensionalShape )
            {
               ThreeDimensionalShape threeDimensionalShape =
                  ( ThreeDimensionalShape ) currentShape;

               System.out.printf( "%s's area is %s\n",
                  currentShape.getName(), threeDimensionalShape.getArea() );
               System.out.printf( "%s's volume is %s\n",
                  currentShape.getName(),
                  threeDimensionalShape.getVolume() );
            } // end if

            System.out.println();
         } // end for
      } // end main
   } // end class ShapeTest
```

```
Circle: [22, 88] radius: 4
Circle's area is 50

Square: [71, 96] side: 10
Square's area is 100

Sphere: [8, 89] radius: 2
Sphere's area is 50
Sphere's volume is 33

Cube: [79, 61] side: 8
Cube's area is 384
Cube's volume is 512
```