Object-Oriented Programming: Inheritance

OBJECTIVES
In this chapter you will learn:

■ How inheritance promotes software reusability.
■ The notions of superclasses and subclasses.
■ To use keyword extends to create a class that inherits attributes and behaviors from another class.
■ To use access modifier protected to give subclass methods access to superclass members.
■ To access superclass members with super.
■ How constructors are used in inheritance hierarchies.
■ The methods of class Object, the direct or indirect superclass of all classes in Java.
# Assignment Checklist

Name: ____________________  Date: ____________________  
Section: ____________________

<table>
<thead>
<tr>
<th>Exercises</th>
<th>Assigned: Circle assignments</th>
<th>Date Due</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Prelab Activities</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matching</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Fill in the Blank</td>
<td>12, 13, 14, 15, 16, 17, 18, 19, 20, 21</td>
<td></td>
</tr>
<tr>
<td>Short Answer</td>
<td>22, 23, 24, 25, 26</td>
<td></td>
</tr>
<tr>
<td>Correct the Code</td>
<td>27, 28, 29</td>
<td></td>
</tr>
<tr>
<td><strong>Lab Exercise</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 1 — Employee Hierarchy</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>Follow-Up Question and Activity</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Postlab Activities</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coding Exercises</td>
<td>1, 2, 3</td>
<td></td>
</tr>
<tr>
<td>Programming Challenge</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Prelab Activities

Matching

Name: ___________________________ Date: ___________________________
Section: ___________________________

After reading Chapter 9 of *Java How to Program: 8/e*, answer the given questions. The questions are intended to test and reinforce your understanding of key concepts. You may answer the questions before or during the lab.

For each term in the left column, write the letter for the description from the right column that best matches the term.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>1. extends</td>
</tr>
<tr>
<td>K</td>
<td>2. object class</td>
</tr>
<tr>
<td>G</td>
<td>3. class hierarchy</td>
</tr>
<tr>
<td>J</td>
<td>4. protected</td>
</tr>
<tr>
<td>H</td>
<td>5. inheritance</td>
</tr>
<tr>
<td>C</td>
<td>6. super</td>
</tr>
<tr>
<td>A</td>
<td>7. direct superclass</td>
</tr>
<tr>
<td>T</td>
<td>8. brittle software</td>
</tr>
<tr>
<td>B</td>
<td>9. override</td>
</tr>
<tr>
<td>F</td>
<td>10. clone method</td>
</tr>
<tr>
<td>E</td>
<td>11. indirect superclass</td>
</tr>
<tr>
<td>a)</td>
<td>A subclass’s immediate superclass extends.</td>
</tr>
<tr>
<td>b)</td>
<td>Term used to describe the case when a subclass method is defined with the same signature as a superclass method.</td>
</tr>
<tr>
<td>c)</td>
<td>Keyword used to refer to an object’s superclass members.</td>
</tr>
<tr>
<td>d)</td>
<td>Indicates that a subclass will inherit from a particular superclass.</td>
</tr>
<tr>
<td>e)</td>
<td>A class that a subclass does not directly extend; however, the subclass still has an <em>is-a</em> relationship with the class.</td>
</tr>
<tr>
<td>f)</td>
<td>A protected method that takes no arguments. It returns a copy of the object on which it is called.</td>
</tr>
<tr>
<td>g)</td>
<td>A set of classes related by inheritance.</td>
</tr>
<tr>
<td>h)</td>
<td><em>Is-a</em> relationship.</td>
</tr>
<tr>
<td>i)</td>
<td>A small change in the superclass can “break” subclass implementations.</td>
</tr>
<tr>
<td>j)</td>
<td>Such members can be accessed in the class in which they are defined, and in all subclasses of that class.</td>
</tr>
<tr>
<td>k)</td>
<td>Superclass of all Java classes.</td>
</tr>
</tbody>
</table>
Fill in the Blank

Name: ___________________________ Date: ___________________________

Section: _________________________

Fill in the blanks for each of the following statements:

12. A subclass can explicitly invoke a constructor of its superclass by using the superclass constructor call syntax—keyword super followed by a set of parentheses containing the superclass constructor’s arguments.

13. Every object of a subclass is also an object of that class’s superclass. However, a superclass object is not an object of any of its subclasses.

14. In a(n) has-a relationship, an object has a reference to an object of another class as a member.

15. In a(n) is-a relationship, an object of a subclass type may also be treated as an object of the superclass type.

16. Some day, most new software likely will be constructed from standardized reusable components, just as automobiles and most computer hardware are constructed today.

17. A subclass cannot directly access private members of its superclass.

18. The direct superclass of a subclass is the superclass from which the subclass inherits.

19. With single inheritance, a class is derived from one superclass. Java does not support multiple inheritance.

20. A subclass inherits the members of its superclass that are declared with the access specifiers public and protected.

21. The toString method of class Object is normally overridden by a subclass.
Prelab Activities

Short Answer

Name: ____________________________ Date: ____________________________

Section: __________________________

In the space provided, answer each of the given questions. Your answers should be concise; aim for two or three sentences.

22. How does inheritance promote software reusability?
Inheritance enables programmers to create new classes from existing classes by absorbing their attributes and behaviors, and adding capabilities the new classes require. Inheritance also takes advantage of class relationships where objects of a certain class have the same characteristics.

23. Explain protected member access.
The protected members of a class are accessible to the class in which they are defined, all subclasses of that class and other classes that are part of the same package.

24. Explain the difference between composition (i.e., the has-a relationship) and inheritance (i.e., the is-a relationship).
With inheritance, a class inherits attributes and behaviors from another class. All objects of the new class can be treated as their own type and as their direct and indirect superclass types. With composition, a new class is composed of objects of existing classes. In this relationship, the new class normally cannot be treated as an object of any of its component classes.

25. When an object of a subclass is created, the constructor for that subclass object is invoked to initialize the subclass object. Explain the complete details of initializing an object of class BasePlusCommissionEmployee with six arguments. Assume the CommissionEmployee-BasePlusCommissionEmployee hierarchy discussed in this chapter.
When a program creates a BasePlusCommissionEmployee object, the BasePlusCommissionEmployee constructor is called. That constructor calls CommissionEmployee's constructor, which in turn calls Object's constructor. Class Object's constructor has an empty body, so it immediately returns control to CommissionEmployee's constructor, which then sets the values of the first name, last name, social security number, gross sales and commission rate. When CommissionEmployee's constructor completes execution, it returns control to BasePlusCommissionEmployee's constructor, which sets the base salary.

26. Explain how to invoke a superclass method from a subclass method for the case in which the subclass method overrides a superclass method and the case in which the subclass method does not override a superclass method.
In the case where the subclass method overrides the superclass method, it is necessary to explicitly use the keyword super, to invoke the method of the superclass. When the subclass method does not override the superclass method, the superclass method can be invoked simply by using its name and appropriate arguments.
Correct the Code

Determine if there is an error in each of the following program segments. If there is an error, specify whether it is a logic error or a compilation error, circle the error in the program and write the corrected code in the space provided after each problem. If the code does not contain an error, write “no error.” [Note: There may be more than one error in a program segment.]

For questions 27–29 assume the definitions of classes CommissionEmployee and BasePlusCommissionEmployee in Fig. L 9.1 and Fig. L 9.2.

```java
// Fig. 9.15: CommissionEmployee.java
// CommissionEmployee class represents a commission employee.

public class CommissionEmployee
{
    private String firstName;
    private String lastName;
    private String socialSecurityNumber;
    private double grossSales; // gross weekly sales
    private double commissionRate; // commission percentage

    // five-argument constructor
    public CommissionEmployee( String first, String last, String ssn,
                               double sales, double rate )
    {
        // implicit call to Object constructor occurs here
        firstName = first;
        lastName = last;
        socialSecurityNumber = ssn;
        setGrossSales(sales); // validate and store gross sales
        setCommissionRate(rate); // validate and store commission rate
        System.out.printf("%nCommissionEmployee constructor:%n%s", this);
    } // end five-argument CommissionEmployee constructor

    // set first name
    public void setFirstName( String first )
    {
        firstName = first;
    } // end method setFirstName

    // return first name
    public String getFirstName()
    {
        return firstName;
    } // end method getFirstName

Fig. L 9.1 | CommissionEmployee.java (Part 1 of 3.)
Correct the Code

```java
// set last name
public void setLastName( String last )
{
    lastName = last;
} // end method setLastName

// return last name
public String getLastName()
{
    return lastName;
} // end method getLastName

// set social security number
public void setSocialSecurityNumber( String ssn )
{
    socialSecurityNumber = ssn; // should validate
} // end method setSocialSecurityNumber

// return social security number
public String getSocialSecurityNumber()
{
    return socialSecurityNumber;
} // end method getSocialSecurityNumber

// set gross sales amount
public void setGrossSales( double sales )
{
    grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end method setGrossSales

// return gross sales amount
public double getGrossSales()
{
    return grossSales;
} // end method getGrossSales

// set commission rate
public void setCommissionRate( double rate )
{
    commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end method setCommissionRate

// return commission rate
public double getCommissionRate()
{
    return commissionRate;
} // end method getCommissionRate

// calculate earnings
public double earnings()
{
    return getCommissionRate() * getGrossSales();
} // end method earnings

// return String representation of CommissionEmployee object
public String toString()
{

Fig. L 9.1 | CommissionEmployee.java (Part 2 of 3.)
Correct the Code

```java
return String.format( "%s: %s
%s: %s %s
%s: %s
%s: %.2f
%s: %.2f", 
"commission employee", getFirstName(), getLastName(),
"social security number", getSocialSecurityNumber(),
"gross sales", getGrossSales(),
"commission rate", getCommissionRate() );
} // end method toString

Fig. L 9.1 | CommissionEmployee.java. (Part 3 of 3.)

```
Prelab Activities

Correct the Code

27. The following constructor, when inserted into class BasePlusCommissionEmployee, should invoke a CommissionEmployee constructor to initialize the CommissionEmployee part of a BasePlusCommissionEmployee object.

Your answer:

- Line 5, missing parameter to CommissionEmployee constructor. Compilation error.

28. The following toString method, when inserted into class BasePlusCommissionEmployee, should return a string consisting of CommissionEmployee's output string along with the base salary of this object.

Your answer:

- Method toString calls itself. Must use super to invoke the superclass toString method. Logic error.
Prelab Activities

Correct the Code

29. The following earnings method, when inserted into class BasePlusCommissionEmployee, should use CommissionEmployee’s earnings method to help calculate the earnings for this object.

```java
// calculate earnings
public double earnings()
{
    return getBaseSalary() + super.earnings();
} // end method earnings
```

Your answer:

```java
// calculate earnings
public double earnings()
{
    return getBaseSalary() + super.earnings();
} // end method earnings
```

- Method earnings calls itself. Must use super to invoke the superclass earnings method. Logic error.
Lab Exercise

Lab Exercise 1 — Employee Hierarchy

This problem is intended to be solved in a closed-lab session with a teaching assistant or instructor present. The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. Sample Output
4. Program Template (Fig. L 9.3)
5. Problem-Solving Tips
6. Follow-Up Question and Activity

The program template represents a complete working Java program, with one or more key lines of code replaced with comments. Read the problem description and examine the sample output; then study the template code. Using the problem-solving tips as a guide, replace the /* */ comments with Java code. Compile and execute the program. Compare your output with the sample output provided. Then answer the follow-up question. The source code for the template is available at www.pearsonhighered.com/deitel.

Lab Objectives
This lab was designed to reinforce programming concepts from Chapter 9 of Java How to Program: 8/e. In this lab, you will practice:

• Using a has-a relationship.

The follow-up question and activity also will give you practice:

• Comparing the is-a relationship to the has-a relationship.

Description of the Problem
Many programs written with inheritance could be written with composition instead, and vice versa. Rewrite class BasePlusCommissionEmployee of the CommissionEmployee—BasePlusCommissionEmployee hierarchy (Section 9.4.5) to use composition rather than inheritance.

Template

```
// Exercise 9.3 solution: BasePlusCommissionEmployee.java

public class BasePlusCommissionEmployee
{
    /* declare instance variable to satisfy the has-a relationship */
    private double baseSalary; // base salary per week

    // six-argument constructor
    public BasePlusCommissionEmployee( String first, String last,
        String ssn, double sales, double rate, double salary )
    {
        /* construct the CommissionEmployee portion of this object */
        setBaseSalary( salary ); // validate and store base salary
    }

    // end six-argument BasePlusCommissionEmployee constructor
```

Fig. L 9.3 | BasePlus CommissionEmployee.java (Part 1 of 3.)
Lab Exercise

Lab Exercise 1 — Employee Hierarchy

16    // set first name
17    public void setFirstName( String first )
18    {
19       /* set the first name of the composed CommissionEmployee object */
20    } // end method setFirstName
21
22    // return first name
23    public String getFirstName()
24    {
25       /* return the first name of the composed CommissionEmployee object */
26    } // end method getFirstName
27
28    // set last name
29    public void setLastName( String last )
30    {
31       /* set the last name of the composed CommissionEmployee object */
32    } // end method setLastName
33
34    // return last name
35    public String getLastName()
36    {
37       /* return the last name of the composed CommissionEmployee object */
38    } // end method getLastName
39
40    // set social security number
41    public void setSocialSecurityNumber( String ssn )
42    {
43       /* set the social security number of the composed CommissionEmployee object */
44    } // end method setSocialSecurityNumber
45
46    // return social security number
47    public String getSocialSecurityNumber()
48    {
49       /* return the social security number of the composed CommissionEmployee object */
50    } // end method getSocialSecurityNumber
51
52    // set commission employee's gross sales amount
53    public void setGrossSales( double sales )
54    {
55       /* set the gross sales of the composed CommissionEmployee object */
56    } // end method setGrossSales
57
58    // return commission employee's gross sales amount
59    public double getGrossSales()
60    {
61       /* return the gross sales of the composed CommissionEmployee object */
62    } // end method getGrossSales
63
64    // set commission employee's rate
65    public void setCommissionRate( double rate )
66    {
67       /* Set the commission rate of the composed CommissionEmployee object */
68    } // end method setCommissionRate
69
Lab Exercise

Lab Exercise 1 — Employee Hierarchy

Problem-Solving Tips

1. Look at the CommissionEmployee class to determine which functionality to use in each of the BasePlusCommissionEmployee class’s constructor and methods.

2. If you have any questions as you proceed, ask your lab instructor for assistance.

Solution

```java
public class BasePlusCommissionEmployee {
    private CommissionEmployee commissionEmployee; // composition
    private double baseSalary; // base salary per week

    // six-argument constructor
    public BasePlusCommissionEmployee( String first, String last, String ssn, double sales, double rate, double salary ) {
        // Exercise 9.3 solution: BasePlusCommissionEmployee.java
    }
}
```
Lab Exercise

Lab Exercise 1 — Employee Hierarchy

```java
commissionEmployee =
    new CommissionEmployee( first, last, ssn, sales, rate );
setBaseSalary( salary ); // validate and store base salary
} // end six-argument BasePlusCommissionEmployee constructor

// set first name
public void setFirstName( String first )
{
    commissionEmployee.setFirstName( first );
} // end method setFirstName

// return first name
public String getFirstName()
{
    return commissionEmployee.getFirstName();
} // end method getFirstName

// set last name
public void setLastName( String last )
{
    commissionEmployee.setLastName( last );
} // end method setLastName

// return last name
public String getLastName()
{
    return commissionEmployee.getLastName();
} // end method getLastName

// set social security number
public void setSocialSecurityNumber( String ssn )
{
    commissionEmployee.setSocialSecurityNumber( ssn );
} // end method setSocialSecurityNumber

// return social security number
public String getSocialSecurityNumber()
{
    return commissionEmployee.getSocialSecurityNumber();
} // end method getSocialSecurityNumber

// set commission employee's gross sales amount
public void setGrossSales( double sales )
{
    commissionEmployee.setGrossSales( sales );
} // end method setGrossSales

// return commission employee's gross sales amount
public double getGrossSales()
{
    return commissionEmployee.getGrossSales();
} // end method getGrossSales

// set commission employee's rate
public void setCommissionRate( double rate )
{
    commissionEmployee.setCommissionRate( rate );
} // end method setCommissionRate
```
Lab Exercise

Lab Exercise 1 — Employee Hierarchy

Follow-Up Question and Activity

1. Assess the relative merits of the two approaches for the CommissionEmployee and BasePlusCommissionEmployee problems, as well as for object-oriented programs in general. Which approach is more natural? Why?

For a relatively short program like this one, either approach is acceptable. But as programs become larger with more and more objects being instantiated, inheritance becomes preferable because it makes the program easier to modify and promotes the reuse of code. The inheritance approach is more natural because a base-salaried commission employee is a commission employee. Composition is defined by the has-a relationship, and clearly it would be strange to say that “a base-salaried commission employee has a commission employee.”
Postlab Activities

Coding Exercises

Name: ____________________  Date: ____________________
Section: ____________________

These coding exercises reinforce the lessons learned in the lab and provide additional programming experience outside the classroom and laboratory environment. They serve as a review after you have successfully completed the Prelab Activities and Lab Exercises.

For each of the following problems, write a program or a program segment that performs the specified action.

1. Declare the headers for the classes in the class diagram of Fig. L 9.4.

Fig. L 9.4  |  Inheritance hierarchy for Shapes.

```java
// Exercise 9.6 Solution: Shape.java
// Definition of class Shape.

public abstract class Shape
{
  private int x, y;  // coordinates of shape

  // constructor
  public Shape( int x, int y )
  {
    this.x = x;
    this.y = y;
  }

  // set x coordinate
  public void setX( int x )
  {
    this.x = x;
  }

  // set y coordinate
  public void setY( int y )
  {
    this.y = y;
  }
}
```
Postlab Activities

Coding Exercises

```java
// get x coordinate
public int getX()
{
    return x;
}

// get y coordinate
public int getY()
{
    return y;
}

// abstract methods
public abstract String getName();
}  // end class Shape

// Exercise 9.6 Solution: TwoDimensionalShape.java
// Definition of class TwoDimensionalShape.

public abstract class TwoDimensionalShape extends Shape
{
    private double dimension1, dimension2;

    // constructor
    public TwoDimensionalShape( int x, int y, double d1, double d2 )
    {
        super( x, y );
        dimension1 = d1;
        dimension2 = d2;
    }

    // set methods
    public void setDimension1( double d )
    {
        dimension1 = d;
    }

    public void setDimension2( double d )
    {
        dimension2 = d;
    }

    // get methods
    public double getDimension1()
    {
        return dimension1;
    }

    public double getDimension2()
    {
        return dimension2;
    }

    // abstract method
    public abstract double area();
}  // end class TwoDimensionalShape
```
Postlab Activities

Coding Exercises

// Exercise 9.6 Solution: ThreeDimensionalShape.java
// Definition of class ThreeDimensionalShape

public abstract class ThreeDimensionalShape extends Shape
{
    private double dimension1, dimension2, dimension3;

    // constructor
    public ThreeDimensionalShape( int x, int y, double d1, double d2, double d3 )
    {
        super( x, y );
        dimension1 = d1;
        dimension2 = d2;
        dimension3 = d3;
    }

    // set methods
    public void setDimension1( double d )
    {
        dimension1 = d;
    }

    public void setDimension2( double d )
    {
        dimension2 = d;
    }

    public void setDimension3( double d )
    {
        dimension3 = d;
    }

    // get methods
    public double getDimension1()
    {
        return dimension1;
    }

    public double getDimension2()
    {
        return dimension2;
    }

    public double getDimension3()
    {
        return dimension3;
    }

    // abstract methods
    public abstract double area();
    public abstract double volume();
}  // end class ThreeDimensionalShape
1 // Exercise 9.6 Solution: Circle.java
2 // Definition of class Circle.
3
4 public class Circle extends TwoDimensionalShape
5 {
6     // constructor
7     public Circle( int x, int y, double radius )
8     {
9         super( x, y, radius, radius );
10     }
11
12     // overridden methods
13     public String getName()
14     {
15         return "Circle";
16     }
17
18     public double area()
19     {
20         return Math.PI * super.getDimension1() * super.getDimension1();
21     }
22
23     // set method
24     public void setRadius( double radius )
25     {
26         super.setDimension1( radius );
27     }
28
29     // get method
30     public double getRadius()
31     {
32         return super.getDimension1();
33     }
34 }  // end class Circle

1 // Exercise 9.6 Solution: Square.java
2 // Definition of class Square.
3
4 public class Square extends TwoDimensionalShape
5 {
6     // constructor
7     public Square( int x, int y, double side )
8     {
9         super( x, y, side, side );
10     }
11
12     // overridden methods
13     public String getName()
14     {
15         return "Square";
16     }
17
18     public double area()
19     {
20         return super.getDimension1() * super.getDimension1();
21     }
22
Postlab Activities

Coding Exercises

// set method
public void setSide( double side )
{
    super.setDimension1( side );
}

// get method
public double getSide()
{
    return super.getDimension1();
}
} // end class Square

// Exercise 9.6 Solution: Sphere.java
// Definition of class Sphere.
public class Sphere extends ThreeDimensionalShape
{

    // constructor
    public Sphere( int x, int y, double radius )
    {
        super( x, y, radius, radius, radius );
    }

    // overridden methods
    public String getName()
    {
        return "Sphere";
    }

    public double area()
    {
        return 4 * Math.PI * super.getDimension1() * super.getDimension1();
    }

    public double volume()
    {
        return 4.0 / 3.0 * Math.PI * super.getDimension1() * super.getDimension1();
    }

    // set method
    public void setRadius( double radius )
    {
        super.setDimension1( radius );
    }

    // get method
    public double getRadius()
    {
        return super.getDimension1();
    }
} // end class Sphere
Postlab Activities

Coding Exercises

1. // Exercise 9.6 Solution: Cube.java
   // Definition of class Cube.

   public class Cube extends ThreeDimensionalShape
   {
      // constructor
      public Cube( int x, int y, double side )
      {
         super( x, y, side, side, side );
      }

      // overridden methods
      public String getName() {
         return "Cube";
      }

      public double area() {
         return 6 * super.getDimension1() * super.getDimension1();
      }

      public double volume() {
         return super.getDimension1() * super.getDimension1() * super.getDimension1();
      }

      // set method
      public void setSide( double side ) {
         super.setDimension1( side );
      }

      // get method
      public double getSide() {
         return super.getDimension1();
      }
   }  // end class Cube

2. Declare toString methods for all the classes in Coding Exercise 1. Class Shape's toString method should return the string "Shape". The toString method of each of the subclasses in the hierarchy should return a string containing the class's name, the string " is a " and the result of a call to the superclass's toString method.

   // Exercise 9.6 Solution: Shape.java
   // Definition of class Shape.

   public abstract class Shape
   {
      private int x, y;  // coordinates of shape

      // constructor
      public Shape( int x, int y )
      {
Postlab Activities

Coding Exercises

```java
11     this.x = x;
12     this.y = y;
13 }
14
15 // set x coordinate
16 public void setX( int x )
17 {
18     this.x = x;
19 }
20
21 // set y coordinate
22 public void setY( int y )
23 {
24     this.y = y;
25 }
26
27 // get x coordinate
28 public int getX()
29 {
30     return x;
31 }
32
33 // get y coordinate
34 public int getY()
35 {
36     return y;
37 }
38
39 // abstract methods
40 public abstract String getName();
41
42 public String toString()
43 {
44     return "Shape";
45 }
46 } // end class Shape
```

// Exercise 9.6 Solution: TwoDimensionalShape.java
// Definition of class TwoDimensionalShape.

```java
public abstract class TwoDimensionalShape extends Shape
{
    private double dimension1, dimension2;

    // constructor
    public TwoDimensionalShape( int x, int y, double d1, double d2 )
    {
        super( x, y );
        dimension1 = d1;
        dimension2 = d2;
    }

    // set methods
    public void setDimension1( double d )
    {
        dimension1 = d;
    }
```
Postlab Activities

Coding Exercises

```java
public void setDimension2( double d )
{
    dimension2 = d;
}
// get methods
public double getDimension1()
{
    return dimension1;
}
public double getDimension2()
{
    return dimension2;
}
// abstract method
public abstract double area();
public String toString()
{
    return String.format( "TwoDimensionalShape is a %s", super.toString() );
}
} // end class TwoDimensionalShape
```

```java
// Exercise 9.6 Solution: ThreeDimensionalShape.java
// Definition of class ThreeDimensionalShape
public abstract class ThreeDimensionalShape extends Shape
{
    private double dimension1, dimension2, dimension3;
    // constructor
    public ThreeDimensionalShape( int x, int y, double d1, double d2, double d3 )
    {
        super( x, y );
        dimension1 = d1;
        dimension2 = d2;
        dimension3 = d3;
    }
    // set methods
    public void setDimension1( double d )
    {
        dimension1 = d;
    }
    public void setDimension2( double d )
    {
        dimension2 = d;
    }
    public void setDimension3( double d )
    {
        dimension3 = d;
    }
```
Postlab Activities

Coding Exercises

```java
// get methods
public double getDimension1()
{
    return dimension1;
}

public double getDimension2()
{
    return dimension2;
}

public double getDimension3()
{
    return dimension3;
}

// abstract methods
public abstract double area();
public abstract double volume();

public String toString()
{
    return String.format( "ThreeDimensionalShape is a %s", super.toString() );
}
}  // end class ThreeDimensionalShape

// Exercise 9.6 Solution: Circle.java
// Definition of class Circle.

public class Circle extends TwoDimensionalShape {
    // constructor
    public Circle( int x, int y, double radius )
    {
        super( x, y, radius, radius );
    }

    // overridden methods
    public String getName()
    {
        return "Circle";
    }

    public double area()
    {
        return Math.PI * super.getDimension1() * super.getDimension1();
    }

    // set method
    public void setRadius( double radius )
    {
        super.setDimension1( radius );
    }
```
Postlab Activities

Coding Exercises

```java
// get method
public double getRadius()
{
    return super.getDimension1();
}

public String toString()
{
    return String.format( "Circle is a %s", super.toString() );
}
}  // end class Circle

// Exercise 9.6 Solution: Square.java
// Definition of class Square.
public class Square extends TwoDimensionalShape
{
    // constructor
    public Square( int x, int y, double side )
    {
        super( x, y, side, side );
    }

    // overridden methods
    public String getName()
    {
        return "Square";
    }
    public double area()
    {
        return super.getDimension1() * super.getDimension1();
    }
    public void setSide( double side )
    {
        super.setDimension1( side );
    }

    // get method
    public double getSide()
    {
        return super.getDimension1();
    }
    public String toString()
    {
        return String.format( "Square is a %s", super.toString() );
    }
} // end class Square
```
Postlab Activities

Coding Exercises

```java
1  // Exercise 9.6 Solution: Sphere.java
2  // Definition of class Sphere.
3  public class Sphere extends ThreeDimensionalShape
4  {
5      // constructor
6      public Sphere( int x, int y, double radius )
7      {
8          super( x, y, radius, radius, radius );
9      }
10     }
11     
12     // overridden methods
13     public String getName() {
14         return "Sphere";
15     }
16     public double area() {
17         return 4 * Math.PI * super.getDimension1() * super.getDimension1();
18     }
19     public double volume() {
20         return 4.0 / 3.0 * Math.PI * super.getDimension1() * super.getDimension1() * super.getDimension1();
21     }
22     
23     // set method
24     public void setRadius( double radius ) {
25         super.setDimension1( radius );
26     }
27     
28     // get method
29     public double getRadius() {
30         return super.getDimension1();
31     }
32     public String toString() {
33         return String.format( "Sphere is a %s", super.toString() );
34     }
35  } // end class Sphere
```

```java
1  // Exercise 9.6 Solution: Cube.java
2  // Definition of class Cube.
3  public class Cube extends ThreeDimensionalShape
4  {
5      // constructor
6      public Cube( int x, int y, double side )
7      {
8          super( x, y, side, side, side );
9      }
10     }
11     
```

Fig. L 9.5 | Cube.java (Part 1 of 2.)
Postlab Activities

Coding Exercises

3. Write an application that creates one object of each of the classes Circle, Square, Sphere and Cube, and invokes their toString methods. The output for each object should show the is-a relationships between that object's class and its superclasses.

```java
// Exercise 9.8 Solution: ShapeTest.java
// Program tests the Shape hierarchy.

public class ShapeTest {
    private Shape shapeArray[];

    // create shapes
    public ShapeTest() {
        shapeArray = new Shape[4];

        shapeArray[0] = new Circle(22, 88, 1.25);
        shapeArray[1] = new Square(71, 96, 2.5);
        shapeArray[2] = new Sphere(71, 96, 2.5);
        shapeArray[3] = new Cube(22, 88, 1.25);
    }
}
```

Fig. L 9.5 | Cube.java (Part 2 of 2.)
Postlab Activities

Name:

Coding Exercises

```java
shapeArray[ 2 ] = new Sphere( 8, 89, 3.75 );
shapeArray[ 3 ] = new Cube( 79, 61, 5.0 );
} // end ShapeTest constructor

// display shape info
public void displayShapeInfo()
{
    // call method toString on all shapes
    for ( int i = 0; i < shapeArray.length; i++ )
    {
        System.out.printf( "%s: %s\n", shapeArray[ i ].getName(), shapeArray[ i ] );

        if ( shapeArray[ i ] instanceof TwoDimensionalShape )
            { TwoDimensionalShape current = ( TwoDimensionalShape ) shapeArray[ i ];
              System.out.printf( "%s's area is %.2f\n", current.getName(), current.area() );
            } // end if

        if ( shapeArray[ i ] instanceof ThreeDimensionalShape )
            { ThreeDimensionalShape current = ( ThreeDimensionalShape ) shapeArray[ i ];
              System.out.printf( "%s's area is %.2f\n%s's volume is %.2f\n", current.getName(), current.area(), current.getName(), current.volume() );
            } // end if
    } // end for
} // end method displayShapeInfo

// create ShapeTest object and display info
public static void main( String args[] )
{
    ShapeTest driver = new ShapeTest();
    driver.displayShapeInfo();
} // end main
```
Postlab Activities

Programming Challenges

Name: ___________________  Date: ___________________

Section: ___________________

The Programming Challenges are more involved than the Coding Exercises and may require a significant amount of time to complete. Write a Java program for each of the problems in this section. The answers to these problems are available at www.pearsonhighered.com/deitel. Pseudocode, hints or sample outputs are provided for each problem to aid you in your programming.

1. Write an inheritance hierarchy for classes Quadrilateral, Trapezoid, Parallelogram, Rectangle and Square. Use Quadrilateral as the superclass of the hierarchy. Specify the instance variables and methods for each class. The private instance variables of Quadrilateral should be the x-y coordinate pairs for the four endpoints of the Quadrilateral. Write a program that instantiates objects of your classes and outputs each object’s area (except Quadrilateral).

Hints:

- Create and use a Point class to represent the corners of the shapes.
- Your output should appear as follows:

Coordinates of Quadrilateral are:  
( 1.1, 1.2 ), ( 6.6, 2.8 ), ( 6.2, 9.9 ), ( 2.2, 7.4 )

Coordinates of Trapezoid are:  
( 0.0, 0.0 ), ( 10.0, 0.0 ), ( 8.0, 5.0 ), ( 3.3, 5.0 )
Height is: 5.0  
Area is: 36.75

Coordinates of Parallelogram are:  
( 5.0, 5.0 ), ( 11.0, 5.0 ), ( 12.0, 20.0 ), ( 6.0, 20.0 )
Width is: 6.0  
Height is: 15.0  
Area is: 90.0

Coordinates of Rectangle are:  
( 17.0, 14.0 ), ( 30.0, 14.0 ), ( 30.0, 28.0 ), ( 17.0, 28.0 )
Width is: 13.0  
Height is: 14.0  
Area is: 182.0

Coordinates of Square are:  
( 4.0, 0.0 ), ( 8.0, 0.0 ), ( 8.0, 4.0 ), ( 4.0, 4.0 )
Side is: 4.0  
Area is: 16.0
Postlab Activities

Programming Challenges

Solution

```java
// Exercise 9.8 Solution: QuadrilateralTest.java
// Driver for Exercise 9.8

public class QuadrilateralTest
{
    public static void main( String args[] )
    {
        // NOTE: All coordinates are assumed to form the proper shapes
        // A quadrilateral is a four-sided polygon
        Quadrilateral quadrilateral = new Quadrilateral( new Point( 1.1, 1.2 ),
            new Point( 6.6, 2.8 ), new Point( 6.2, 9.9 ), new Point( 2.2, 7.4 ) );
        // A trapezoid is a quadrilateral having exactly two parallel sides
        Trapezoid trapezoid = new Trapezoid( new Point( 0.0, 0.0 ),
            new Point( 10.0, 0.0 ), new Point( 8.0, 5.0 ), new Point( 3.3, 5.0 ) );
        // A parallelogram is a quadrilateral with opposite sides parallel
        Parallelogram parallelogram = new Parallelogram( new Point( 5.0, 5.0 ),
            new Point( 11.0, 5.0 ), new Point( 12.0, 20.0 ), new Point( 6.0, 20.0 ) );
        // A rectangle is an equiangular parallelogram
        Rectangle rectangle = new Rectangle( new Point( 17.0, 14.0 ),
            new Point( 30.0, 14.0 ), new Point( 30.0, 28.0 ), new Point( 17.0, 28.0 ) );
        // A square is an equiangular and equilateral parallelogram
        Square square = new Square( new Point( 4.0, 0.0 ),
            new Point( 8.0, 0.0 ), new Point( 8.0, 4.0 ), new Point( 4.0, 4.0 ) );

        System.out.printf( "%s %s %s %s %s\n", quadrilateral, trapezoid, parallelogram,
            rectangle, square );
    } // end main
} // end class QuadrilateralTest
```

```java
// Exercise 9.8 solution: Point.java
// Class Point definition

public class Point
{
    private double x; // x coordinate
    private double y; // y coordinate

    // two-argument constructor
    public Point( double xCoordinate, double yCoordinate )
    {
        x = xCoordinate; // set x
        y = yCoordinate; // set y
    } // end two-argument Point constructor

    // return x
    public double getX()
    {
        return x;
    } // end method getX
```
Postlab Activities

Programming Challenges

```java
// return y
public double getY()
{
    return y;
} // end method getY

// return string representation of Point object
public String toString()
{
    return String.format( "( %.1f, %.1f )", getX(), getY() );
} // end method toString
```

```java
// Exercise 9.8 solution: Quadrilateral.java
// Class Quadrilateral definition

public class Quadrilateral
{
    private Point point1; // first endpoint
    private Point point2; // second endpoint
    private Point point3; // third endpoint
    private Point point4; // fourth endpoint

    // eight-argument constructor
    public Quadrilateral( Point first, Point second, Point third, Point fourth )
    {
        point1 = first;
        point2 = second;
        point3 = third;
        point4 = fourth;
    } // end eight-argument Quadrilateral constructor

    // return point1
    public Point getPoint1()
    {
        return point1;
    } // end method getPoint1

    // return point2
    public Point getPoint2()
    {
        return point2;
    } // end method getPoint2

    // return point3
    public Point getPoint3()
    {
        return point3;
    } // end method getPoint3

    // return point4
    public Point getPoint4()
    {
        return point4;
    } // end method getPoint4
```
Postlab Activities

Programming Challenges

44  // return string representation of a Quadrilateral object
45  public String toString()
46  {
47      return String.format( "%s:
48          Coordinates of Quadrilateral are", getCoordinatesAsString() );
49  } // end method toString
50
51  // return string containing coordinates as strings
52  public String getCoordinatesAsString()
53  {
54      return String.format(
55          "%s, %s, %s, %s\n", point1, point2, point3, point4 );
56  } // end method getCoordinatesAsString
57 } // end class Quadrilateral

1 // Exercise 9.8 solution: Trapezoid.java
2 // Class Trapezoid definition
3
4 public class Trapezoid extends Quadrilateral
5 {
6    private double height; // height of trapezoid
7
8    // eight-argument constructor
9    public Trapezoid( Point first, Point second, Point third, Point fourth )
10    {
11         super( first, second, third, fourth );
12    } // end of eight-argument Trapezoid constructor
13
14    // return height
15    public double getHeight()
16    {
17        if ( getPoint1().getY() == getPoint2().getY() )
18            return Math.abs( getPoint2().getY() - getPoint3().getY() );
19        else
20            return Math.abs( getPoint1().getY() - getPoint2().getY() );
21    } // end method getHeight
22
23    // return area
24    public double getArea()
25    {
26            return getSumOfTwoSides() * getHeight() / 2.0;
27    } // end method getArea
28
29    // return the sum of the trapezoid's two sides
30    public double getSumOfTwoSides()
31    {
32        if ( getPoint1().getY() == getPoint2().getY() )
33            return Math.abs( getPoint1().getX() - getPoint2().getX() ) +
34                   Math.abs( getPoint3().getX() - getPoint4().getX() );
35        else
36            return Math.abs( getPoint2().getX() - getPoint3().getX() ) +
37                   Math.abs( getPoint4().getX() - getPoint1().getX() );
38    } // end method getSumOfTwoSides
39
40    // return string representation of Trapezoid object
41    public String toString()
42    {
Postlab Activities

Programming Challenges

```java
43         return String.format("\n%s:
44             Coordinates of Trapezoid are", getCoordinatesAsString(),
45             "Height is", getHeight(), "Area is", getArea() );
46     } // end method toString
47 } // end class Trapezoid

1 // Exercise 9.8 solution: Parallelogram.java
2 // Class Parallelogram definition
3 public class Parallelogram extends Trapezoid
4 {
5     // eight-argument constructor
6     public Parallelogram( Point first, Point second, Point third, Point fourth )
7     {
8         super( first, second, third, fourth);
9     } // end eight-argument Parallelogram constructor
10    // return width of parallelogram
11    public double getWidth()
12    {
13         if ( getPoint1().getY() == getPoint2().getY() )
14             return Math.abs( getPoint1().getX() - getPoint2().getX() );
15         else
16             return Math.abs( getPoint2().getX() - getPoint3().getX() );
17     } // end method getWidth
18    // return string representation of Parallelogram object
19    public String toString()
20    {
21         return String.format("\n%s:
22             Coordinates of Parallelogram are", getCoordinatesAsString(),
23             "Width is", getWidth(), "Height is", getHeight(),
24             "Area is", getArea() );
25     } // end method toString
26 } // end class Parallelogram

1 // Exercise 9.8 solution: Rectangle.java
2 // Class Rectangle definition
3 public class Rectangle extends Parallelogram
4 {
5     // eight-argument constructor
6     public Rectangle( Point first, Point second, Point third, Point fourth )
7     {
8         super( first, second, third, fourth);
9     } // end eight-argument Rectangle constructor
10    // return string representation of Rectangle object
11    public String toString()
12    {
13         return String.format("\n%s:
14             Coordinates of Rectangle are", getCoordinatesAsString(),
15             "Width is", getWidth(), "Height is", getHeight(),
16             "Area is", getArea() );
17     } // end method toString
18 } // end class Rectangle
```
public class Square extends Parallelogram {

    // eight-argument constructor
    public Square( Point first, Point second, Point third, Point fourth )
    {
        super( first, second, third, fourth );
    } // end eight-argument Square constructor

    // return string representation of Square object
    public String toString()
    {
        return String.format( "Coordinates of Square are", getCoordinatesAsString() );
    } // end method toString

} // end class Square