Programação Orientada a Objetos

Dadas as classes Shape, Point, Circle, Cilinder e Test:

Classe Shape:

Object é uma classe do pacote *java.lang* do Java.

```
4  public abstract class Shape extends Object {
5
6     // return shape's area
7     public double area()
8     {
9        return 0.0;
10    }
11
12    // return shape's volume
13    public double volume()
14    {
15       return 0.0;
16    }
17
18    // abstract method must be defined by concrete subclasses
19    // to return appropriate shape name
20    public abstract String getName();
21
22 }  // end class Shape
23
```

Programação Orientada a Objetos

Classe Point:

```
4  public class Point extends Shape {
5     protected int x, y; // coordinates of the Point
6
7     // no-argument constructor
8     public Point()
9     {
10       setPoint( 0, 0 );
11    }
12
13    // constructor
14    public Point( int xCoordinate, int yCoordinate )
15    {
16       setPoint( xCoordinate, yCoordinate );
17    }
18
19    // set x and y coordinates of Point
20    public void setPoint( int xCoordinate, int yCoordinate )
21    {
22       x = xCoordinate;
23       y = yCoordinate;
24    }
25
26    // get x coordinate
27    public int getX()
28    {
29       return x;
30    }
31
32    // get y coordinate
33    public int getY()
34    {
35       return y;
36    }
37
38    // convert point into String representation
39    public String toString()
40    {
41       return "[" + x + ", " + y + "]";
42    }
43
44    // return shape name
45    public String getName()
46    {
47       return "Point";
48    }
49
50 }  // end class Point
```

Classe Circle:

```
4  public class Circle extends Point {  // inherits from Point
5     protected double radius;
6
7     // no-argument constructor
8     public Circle()
9     {
10       // implicit call to superclass constructor here
11       setRadius( 0 );
12    }
13
14    // constructor
15    public Circle( double circleRadius, int xCoordinate,
16       int yCoordinate )
17    {
18       // call superclass constructor
19       super( xCoordinate, yCoordinate );
20
21       setRadius( circleRadius );
22    }
23
24    // set radius of Circle
25    public void setRadius( double circleRadius )
26    {
27       radius = ( circleRadius >= 0 ? circleRadius : 0 );
28    }
29
30    // get radius of Circle
31    public double getRadius()
32    {
33       return radius;
34    }
35
36    // calculate area of Circle
37    public double area()
38    {
39       return Math.PI * radius * radius;
40    }
41
42    // convert Circle to a String represention
43    public String toString()
44    {
45       return "Center = " + super.toString() +
46          "; Radius = " + radius;
47    }
48
49    // return shape name
50    public String getName()
51    {
52       return "Circle";
53    }
54
55 }  // end class Circle
56
57
```

Programação Orientada a Objetos

Classe Cilinder:

```
4  public class Cylinder extends Circle {
5     protected double height;  // height of Cylinder
6
7     // no-argument constructor
8     public Cylinder()
9     {
10       // implicit call to superclass constructor here
11       setHeight( 0 );
12    }
13
14    // constructor
15    public Cylinder( double cylinderHeight,
16       double cylinderRadius, int xCoordinate,
17       int yCoordinate )
18    {
19       // call superclass constructor
20       super( cylinderRadius, xCoordinate, yCoordinate );
21
22       setHeight( cylinderHeight );
23    }
24
25    // set height of Cylinder
26    public void setHeight( double cylinderHeight )
27    {
28       height = ( cylinderHeight >= 0 ? cylinderHeight : 0 );
29    }
30
31    // get height of Cylinder
32    public double getHeight()
33    {
34       return height;
35    }
36
37    // calculate area of Cylinder (i.e., surface area)
38    public double area()
39    {
40       return 2 * super.area() + 2 * Math.PI * radius * height;
41    }
42
43    // calculate volume of Cylinder
44    public double volume()
45    {
46       return super.area() * height;
47    }
48
```

```
49    // convert Cylinder to a String representation
50    public String toString()
51    {
52       return super.toString() + "; Height = " + height;
53    }
54
55    // return shape name
56    public String getName()
57    {
58       return "Cylinder";
59    }
60
61 }  // end class Cylinder
```

Programação Orientada a Objetos

Classe Test:

```
// Java core packages
import java.text.DecimalFormat;

// Java extension packages
import javax.swing.JOptionPane;

public class Test {

   // test Shape hierarchy
   public static void main( String args[] )
   {
      // create shapes
      Point point = new Point( 7, 11 );
      Circle circle = new Circle( 3.5, 22, 8 );
      Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );

      // create Shape array
      Shape arrayOfShapes[] = new Shape[ 3 ];

      // aim arrayOfShapes[ 0 ] at subclass Point object
      arrayOfShapes[ 0 ] = point;

      // aim arrayOfShapes[ 1 ] at subclass Circle object
      arrayOfShapes[ 1 ] = circle;

      // aim arrayOfShapes[ 2 ] at subclass Cylinder object
      arrayOfShapes[ 2 ] = cylinder;

      // get name and String representation of each shape
      String output =
         point.getName() + ": " + point.toString() + "\n" +
         circle.getName() + ": " + circle.toString() + "\n" +
         cylinder.getName() + ": " + cylinder.toString();

      DecimalFormat precision2 = new DecimalFormat( "0.00" );

      // loop through arrayOfShapes and get name,
      // area and volume of each shape in arrayOfShapes
      for ( int i = 0; i < arrayOfShapes.length; i++ ) {
         output += "\n\n" + arrayOfShapes[ i ].getName() +
            ": " + arrayOfShapes[ i ].toString() +
            "\nArea = " +
            precision2.format( arrayOfShapes[ i ].area() ) +
            "\nVolume = " +
            precision2.format( arrayOfShapes[ i ].volume() );
      }

      JOptionPane.showMessageDialog( null, output,
         "Demonstrating Polymorphism",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }

}  // end class Test
```

Exercício 5: Fazer o diagrama de classes das classes que compõem a aplicação.

Exercício 6: Explicar a herança das implementações de Shape pela classe Point.

Exercício 7: Qual (is) método(s) a classe Circle herda da classe Point?

Exercício 8: A classe Cylinder tem sua própria implementação para dois métodos presentes na herança. Quais são esses métodos, e quais as diferenças na implementação?

Exercício 9: Que método é sobrescrito obrigatoriamente em cada uma das subclasses? Por que é obrigatório? Que métodos são sobrescritos nas subclasses apenas de acordo com a necessidade das mesmas?