Lab 6 Sample Answers

Question 1

```
public class Shape {
	private double x;
	private double y;

	public Shape(double x, double y) {
		this.x = x;
		this.y = y;
	}
	public double getX() {
		return x;
	}
	public double getY() {
		return y;
	}
	public double area() {
		System.out.println("An unknown shape has an unknown area!");
		return -1.0; // We must return a double.
	}
	public static void testShape() {
		Shape s = new Shape(1.2, 3.4);
		System.out.println(s.getX() == 1.2);
		System.out.println(s.getY() == 3.4);
		System.out.println(s.area() == -1.0); // Will print an error message too.
	}
}

public class Start {
	public static void main(String[] args) {
		Shape.testShape();
	}
}
```

Question 2

```
public class Shape {
	... same as above ...
}

// The Circle class derives from the Shape class.
public class Circle extends Shape {
	private double radius;

	// x and y are given to the constructor of the base class Shape.
	public Circle(double x, double y, double radius) {
		super(x, y);
		this.radius = radius;
	}
	// Overriding the area method inherited from Shape.
	@Override
	public double area() {
		return Math.PI * radius * radius;
	}
	public static void testCircle() {
		Circle c = new Circle(1.2, 3.4, 4.0);
		// getX and getY are inherited from Shape.
```

```
            // area comes from Circle itself.
            System.out.println(c.getX() == 1.2);
            System.out.println(c.getY() == 3.4);
            System.out.println(c.area() == Math.PI * 16.0);
      }
}

public class Start {
      public static void main(String[] args) {
            Shape.testShape();
            Circle.testCircle();
      }
}
```

Question 3

```
public class Shape {
      ... same as above ...
}

public class Circle extends Shape {
      ... same as above ...
}

// The Dot class derives from the Shape class.
public class Dot extends Shape {
      // x and y are given to the constructor of the base class Shape.
      public Dot(double x, double y) {
            super(x, y);
      }
      // Overriding the area method inherited from Shape.
      @Override
      public double area() {
            return 0.0;
      }
      public static void testDot() {
            Dot d = new Dot(1.2, 3.4);
            // getX and getY are inherited from Shape.
            // area comes from Dot itself.
            System.out.println(d.getX() == 1.2);
            System.out.println(d.getY() == 3.4);
            System.out.println(d.area() == 0.0);
      }
}

public class Start {
      public static void main(String[] args) {
            Shape.testShape();
            Circle.testCircle();
            Dot.testDot();
      }
}
```

Question 4

```
public class Shape {
      ... same as above ...
}

public class Circle extends Shape {
      ... same as above ...
```

```
}

public class Dot extends Shape {
	... same as above ...
}

// The Rectangle class derives from the Shape class.
public class Rectangle extends Shape {
	private double width;
	private double length;

	// x and y are given to the constructor of the base class Shape.
	public Rectangle(double x, double y, double width, double length) {
		super(x, y);
		this.width = width;
		this.length = length;
	}
	// Overriding the area method inherited from Shape.
	@Override
	public double area() {
		return width * length;
	}
	public static void testRectangle() {
		Rectangle r = new Rectangle(1.2, 3.4, 4.0, 5.0);
		// getX and getY are inherited from Shape.
		// area comes from Rectangle itself.
		System.out.println(r.getX() == 1.2);
		System.out.println(r.getY() == 3.4);
		System.out.println(r.area() == 20.0);
	}
}

// The Square class derives from the Rectangle class.
public class Square extends Rectangle {
	// Everything is given to the constructor of the base class
	// Rectangle.  The size of the square is both its width and length.
	public Square(double x, double y, double size) {
		super(x, y, size, size);
	}
	// The area method is inherited from Rectangle.
	public static void testSquare() {
		Square s = new Square(1.2, 3.4, 5.0);
		// getX and getY are inherited from Shape.
		// area is inherited from Rectangle.
		System.out.println(s.getX() == 1.2);
		System.out.println(s.getY() == 3.4);
		System.out.println(s.area() == 25.0);
	}
}

public class Start {
	public static void main(String[] args) {
		Shape.testShape();
		Circle.testCircle();
		Dot.testDot();
		Rectangle.testRectangle();
		Square.testSquare();
	}
}
```

The **Square** class does not need its own **area** method because the **area** method inherited from **Rectangle** also works correctly for **Square**.

Question 5

```
public class Shape {
	... same as above ...
}

public class Circle extends Shape {
	... same as above ...
}

public class Dot extends Shape {
	... same as above ...
}

public class Rectangle extends Shape {
	... same as above ...
}

public class Square extends Rectangle {
	... same as above ...
}

import java.util.ArrayList;

public class ManyShapes {
	private ArrayList allShapes;

	public ManyShapes() {
		this.allShapes = new ArrayList();
	}
	public void addShape(Shape s) {
		allShapes.add(s); // Upcast from Shape to Object.
	}
	public void listAllShapes() {
		for(int i = 0; i < allShapes.size(); i++) {
			Object o = allShapes.get(i);
			Shape s = (Shape)o; // Downcast from Object to Shape.
			System.out.println("Shape has area " + s.area());
		}
		// Another solution, a little bit shorter:
		for(int i = 0; i < allShapes.size(); i++) {
			System.out.println("Shape has area " +
((Shape)allShapes.get(i)).area());
		}
		// Another solution, using an enhanced for loop:
		for(Object o: allShapes) {
			System.out.println("Shape has area " + ((Shape)o).area());
		}
	}
	public static void testManyShapes() {
		ManyShapes m = new ManyShapes();
		m.addShape(new Circle(1.2, 3.4, 4.0));          // Upcast from Circle to Shape.
		m.addShape(new Dot(1.2, 3.4));                  // Upcast from Dot to Shape.
		m.addShape(new Rectangle(1.2, 3.4, 4.0, 5.0)); // Upcast from Rectangle to
Shape.
		m.addShape(new Shape(1.2, 3.4));
		m.addShape(new Square(1.2, 3.4, 5.0));          // Upcast from Square to Shape.
		m.listAllShapes();
	}
}

public class Start {
	public static void main(String[] args) {
```

```
            Shape.testShape();
            Circle.testCircle();
            Dot.testDot();
            Rectangle.testRectangle();
            Square.testSquare();
            ManyShapes.testManyShapes();
        }
}
```

We provide above three different loops in the **listAllShapes** method.

The first loop uses the **get** method of the arraylist to get the element at index **i** in the arraylist. That element is of type **Object**, because all the elements are stored in the arraylist using the **Object** type. The type of the object is then downcasted from **Object** to **Shape**, after which we can use the **area** method of the shape to print what we want.

The second loop works exactly like the first loop, except that it does all the three steps in one single expression: call the **get** method of the arraylist, downcast the resulting object from the **Object** type to the **Shape** type, and call the **area** method of the result of the downcast.

The third loop is different: it is what is called an "enhanced **for** loop" in Java. When using an enhanced **for** loop, Java does the looping for you and there is no need to use an index variable **i** to remember where you are in the arraylist. Instead Java keeps track of where you are in the arraylist automatically for you. Every time Java goes around the loop, Java automatically assigns the variable **o** to point to the next object in the arraylist. Therefore, as Java loops over the arraylist automatically, the variable **o** points to each element of the arraylist one after the other. Every time the variable **o** points to the next element of the arraylist, Java automatically executes the code inside the **for** loop, which does a downcast of **o** from the **Object** type to the **Shape** type, and calls the **area** method of the result of the downcast. Using an enhanced **for** loop like this is much more convenient then writing a **for** loop with your own index **i**: the enhanced **for** loop is guaranteed to always loop over all the elements in the arraylist without making any mistake.

Question 6

```
public class Shape {
        ... same as above ...
}

public class Circle extends Shape {
        ... same as above ...
}

public class Dot extends Shape {
        ... same as above ...
}

public class Rectangle extends Shape {
        ... same as above ...
}

public class Square extends Rectangle {
        ... same as above ...
}

import java.util.ArrayList;

public class ManyShapes {
        private ArrayList allShapes;

        public ManyShapes() {
```

```
            this.allShapes = new ArrayList();
      }
      public void addShape(Shape s) {
            allShapes.add(s); // Upcast from Shape to Object.
      }
      public void listAllShapes() {
            for(Object o: allShapes) {
                  if(o instanceof Circle) {
                        System.out.println("Circle has area " + ((Circle)o).area());
                  } else if(o instanceof Dot) {
                        System.out.println("Dot has area " + ((Dot)o).area());
                  } else if(o instanceof Square) { // Before Rectangle!
                        System.out.println("Square has area " + ((Square)o).area());
                  } else if(o instanceof Rectangle) {
                       System.out.println("Rectangle has area " + ((Rectangle)o).area());
                  } else if(o instanceof Shape) { // Last among shapes.
                        System.out.println("Shape has area " + ((Shape)o).area());
                  } else {
                        System.out.println("ERROR"); // Must never happen.
                  }
            }
      }
      public static void testManyShapes() {
            ManyShapes m = new ManyShapes();
           m.addShape(new Circle(1.2, 3.4, 4.0));          // Upcast from Circle to Shape.
            m.addShape(new Dot(1.2, 3.4));                 // Upcast from Dot to Shape.
            m.addShape(new Rectangle(1.2, 3.4, 4.0, 5.0)); // Upcast from Rectangle to
Shape.
            m.addShape(new Shape(1.2, 3.4));
           m.addShape(new Square(1.2, 3.4, 5.0));          // Upcast from Square to Shape.
            m.listAllShapes();
      }
}

public class Start {
      ... same as above ...
}
```

Note that in the **listAllShapes** method above, we need to test **(o instanceof Square)** before we test **(o instanceof Rectangle)**, because a square can also work as a rectangle so both tests **(o instanceof Square)** and **(o instanceof Rectangle)** are true for a square. Similarly, testing **(o instanceof Shape)** must be done last because this test is true for any shape, including circles, dots, squares, and rectangles.

Question 7

```
public class Shape {
      private double x;
      private double y;

      public Shape(double x, double y) {
            this.x = x;
            this.y = y;
      }
      public double getX() {
            return x;
      }
      public double getY() {
            return y;
      }
      public double area() {
            System.out.println("An unknown shape has an unknown area!");
```

```
            return -1.0; // We must return a double.
      }
      @Override
      public String toString() {
            return "Shape area is " + this.area();
      }
      public static void testShape() {
            Shape s = new Shape(1.2, 3.4);
            System.out.println(s.getX() == 1.2);
            System.out.println(s.getY() == 3.4);
            System.out.println(s.area() == -1.0); // Will print an error message too.
            System.out.println(s.toString().equals("Shape area is -1.0"));
      }
}

// The Circle class derives from the Shape class.
public class Circle extends Shape {
      private double radius;

      // x and y are given to the constructor of the base class Shape.
      public Circle(double x, double y, double radius) {
            super(x, y);
            this.radius = radius;
      }
      // Overriding the area method inherited from Shape.
      @Override
      public double area() {
            return Math.PI * radius * radius;
      }
      @Override
      public String toString() {
            return "Circle area is " + this.area();
      }
      public static void testCircle() {
            Circle c = new Circle(1.2, 3.4, 4.0);
            // getX and getY are inherited from Shape.
            // area comes from Circle itself.
            System.out.println(c.getX() == 1.2);
            System.out.println(c.getY() == 3.4);
            System.out.println(c.area() == Math.PI * 16.0);
            System.out.println(c.toString().equals("Circle area is " + Math.PI * 16.0));
      }
}

// The Dot class derives from the Shape class.
public class Dot extends Shape {
      // x and y are given to the constructor of the base class Shape.
      public Dot(double x, double y) {
            super(x, y);
      }
      // Overriding the area method inherited from Shape.
      @Override
      public double area() {
            return 0.0;
      }
      @Override
      public String toString() {
            return "Dot area is " + this.area();
      }
      public static void testDot() {
            Dot d = new Dot(1.2, 3.4);
            // getX and getY are inherited from Shape.
            // area comes from Dot itself.
```

```
            System.out.println(d.getX() == 1.2);
            System.out.println(d.getY() == 3.4);
            System.out.println(d.area() == 0.0);
            System.out.println(d.toString().equals("Dot area is 0.0"));
      }
}

// The Rectangle class derives from the Shape class.
public class Rectangle extends Shape {
      private double width;
      private double length;

      // x and y are given to the constructor of the base class Shape.
      public Rectangle(double x, double y, double width, double length) {
            super(x, y);
            this.width = width;
            this.length = length;
      }
      // Overriding the area method inherited from Shape.
      @Override
      public double area() {
            return width * length;
      }
      @Override
      public String toString() {
            return "Rectangle area is " + this.area();
      }
      public static void testRectangle() {
            Rectangle r = new Rectangle(1.2, 3.4, 4.0, 5.0);
            // getX and getY are inherited from Shape.
            // area comes from Rectangle itself.
            System.out.println(r.getX() == 1.2);
            System.out.println(r.getY() == 3.4);
            System.out.println(r.area() == 20.0);
            System.out.println(r.toString().equals("Rectangle area is 20.0"));
      }
}

// The Square class derives from the Rectangle class.
public class Square extends Rectangle {
      // Everything is given to the constructor of the base class
      // Rectangle.  The size of the square is both its width and length.
      public Square(double x, double y, double size) {
            super(x, y, size, size);
      }
      // The area method is inherited from Rectangle.
      @Override
      public String toString() {
            return "Square area is " + this.area();
      }
      public static void testSquare() {
            Square s = new Square(1.2, 3.4, 5.0);
            // getX and getY are inherited from Shape.
            // area is inherited from Rectangle.
            System.out.println(s.getX() == 1.2);
            System.out.println(s.getY() == 3.4);
            System.out.println(s.area() == 25.0);
            System.out.println(s.toString().equals("Square area is 25.0"));
      }
}

import java.util.ArrayList;
```

```
public class ManyShapes {
	private ArrayList allShapes;

	public ManyShapes() {
		this.allShapes = new ArrayList();
	}
	public void addShape(Shape s) {
		allShapes.add(s); // Upcast from Shape to Object.
	}
	public void listAllShapes() {
		for(Object o: allShapes) {
			System.out.println(o);
		}
	}
	public static void testManyShapes() {
		ManyShapes m = new ManyShapes();
		m.addShape(new Circle(1.2, 3.4, 4.0));          // Upcast from Circle to Shape.
		m.addShape(new Dot(1.2, 3.4));                  // Upcast from Dot to Shape.
		m.addShape(new Rectangle(1.2, 3.4, 4.0, 5.0)); // Upcast from Rectangle to
Shape.
		m.addShape(new Shape(1.2, 3.4));
		m.addShape(new Square(1.2, 3.4, 5.0));          // Upcast from Square to Shape.
		m.listAllShapes();
	}
}

public class Start {
	... same as above ...
}
```