COMP 249:
Object Oriented Programming II

Tutorial 3:
Polymorphism and Abstract Classes

Exercise 1

What type of polymorphism is used in this code?
in this code?

```
public class Calculator {
   int add(int a, int b) {
      return a + b; }

   double add(double a, double b) {
      return a + b;  }
}

public class Main {
   public static void main(String[] args) {
      Calculator calculator = new Calculator();

      int result1 = calculator.add(5, 7);
      double result2 = calculator.add(3.5, 2.5); }
}
```

Exercise 2

Suppose you have an abstract class Animal with an abstract method makeSound() and two concrete subclasses: Dog and Cat.

Which of the following statements about abstraction is correct?

A) You can create an object of the Animal class.

B) You can directly call the makeSound() method on an object of the Animal class.

C) The Dog and Cat subclasses must implement the makeSound() method.

D) Abstraction is not applicable in this scenario.

Exercise 3

Consider a scenario in Java where you have a class called Emotions with an abstract method express(). You also have two concrete subclasses, Sad and Mad, both of which extend Emotions and provide their own implementations of the express() method.

What concept of OOP is mainly being demonstrated here?

A) Compile-time polymorphism
B) Run-time polymorphism
C) Abstraction

Exercise 4

Consider the following Java code snippet:

```
public interface Drivable {
int numberOfWheels;
void startEngine();
void accelerate();
void brake(); }
public class Car implements Drivable { }
```

Will this code compile? Why or why not?

Exercise 5

What is the output of the code?

```
abstract class Shape {
   abstract void draw();
   abstract void display() {
      System.out.println("Displaying the shape.");
   }
}
class Circle extends Shape {
   void draw() {
      System.out.println("Drawing a circle.");
   }
}
}
public class Main {
   public static void main(String[] args) {
      Circle c= new Circle();
      c.draw();
      c.display(); } }
```

Exercise 6

What is the output of this code? 1

```
abstract class Shape {
   private int x, y;

   Shape(int x, int y) {
      this.x = x;
      this.y = y;
   }
   abstract double area();

   void displayPosition() {
      System.out.println("Position: (" + x + ", " + y + ")");
   }
}
```

2

```
class Circle extends Shape {
   private double radius;

   Circle(int x, int y, double radius) {
      super(x, y);
      this.radius = radius;
   }

   @Override
   double area() {
      return Math.PI * radius * radius;
   }

   void displayDetails() {
      System.out.println("Shape: Circle");
      displayPosition();
      System.out.println("Radius: " + radius);
      System.out.println("Area: " + area());
   }
}
```

3

```
public class Main {
   public static void main(String[] args) {
      Circle circle = new Circle(3, 4, 5.0);
      circle.displayDetails();
}
}
```

Exercise 7

List two key differences between abstract classes and interfaces?

Coding Exercise 1

- Given an abstract class Shape and followed by main class ShapeDriver:

```
package shapes;


//an abstract class Shape
abstract class Shape {
// abstract method getArea
     abstract double getArea();
}
```

Coding Exercise 1 (class ShapeDriver)

```
package shapes;
public class ShapeDriver {
 public static void main(String[] args)
 {
   Circle c1= new Circle();
   c1.setRadius(2.0);
   System.out.println("Area of c1 " +c1);
   Circle c2= new Circle();
   c2.setRadius(4.0);
   System.out.println("Area of c2 " +c2);
```

Coding Exercise 1 (class ShapeDriver Cont.)

```
    Rectangle r1 = new Rectangle();
    r1.setHeight(2.0);
    r1.setWidth(4.0);
    System.out.println("Area of r1 " +r1);
    Rectangle r2= new Rectangle();
    r2.setHeight(3.0);
    r2.setWidth(6.0);
    System.out.println("Area of r2 " +r2);
    Shape shapes[]={c1,c2,r1,r2};
   // We are using the "totalArea" method here
    System.out.println("Total Area is: " + totalArea(shapes));
  } //TODO: Define method “totalArea” here
}
```

Coding Exercise 1(Continue)

- Define classes Rectangle and Circle which extend Shape and provide the expected result below:

Expected result:

```
Area of c1 Circle: 12.56
Area of c2 Circle: 50.24
Area of r1 Rectangle: 8.0
Area of r2 Rectangle: 18.0
```

Note:

Area of Rectangle = height*width;

Area of Circle = 3.14*radius*radius;

Coding Exercise 2 (Continue)

- The capability to reference instances of Rectangle and Circle as Shape types brings the advantage of treating a set of different types of shapes as one common type. Define a method "totalArea" in the class ShapeDriver in order to get the result below:

Expected result:

```
Area of c1 Circle: 12.56
Area of c2 Circle: 50.24
Area of r1 Rectangle: 8.0
Area of r2 Rectangle: 18.0
Total Area is: 88.8
```

Coding Exercise 3

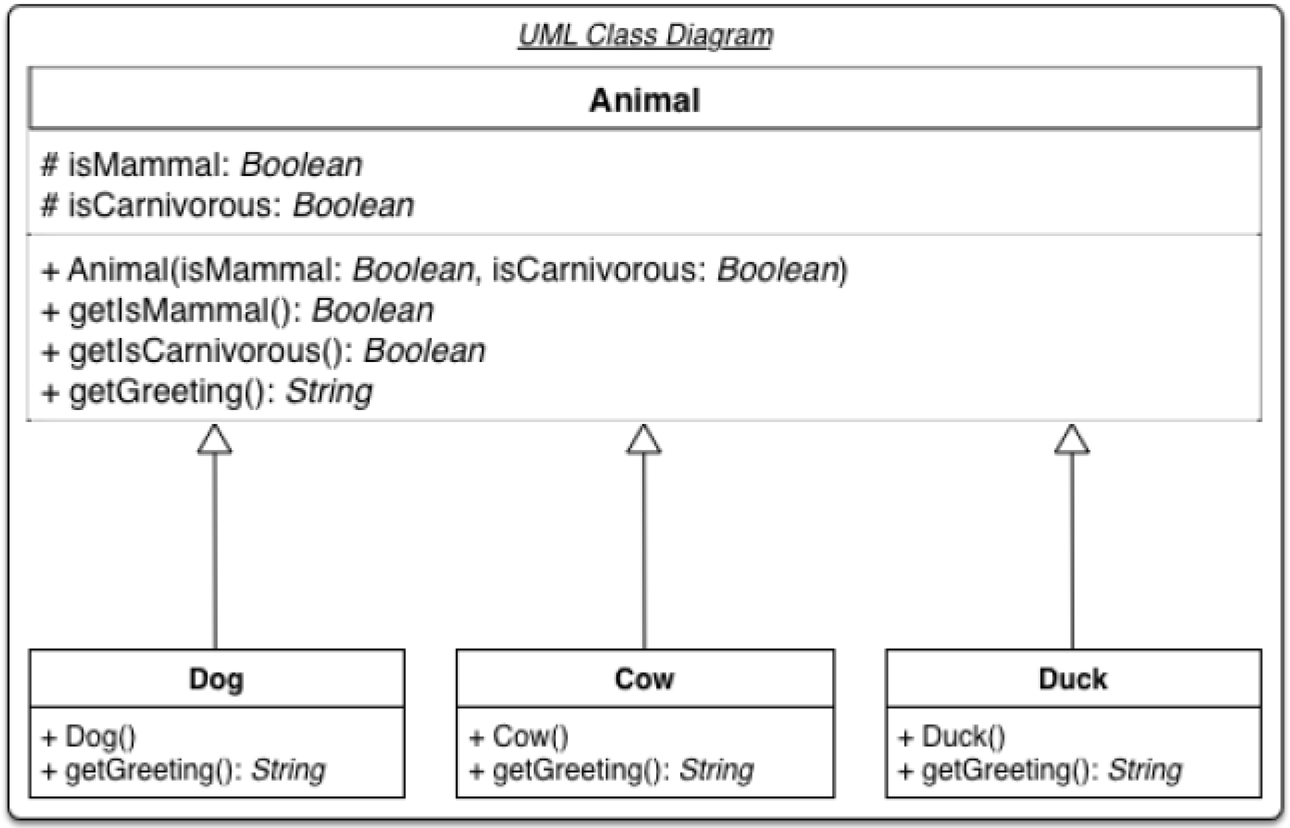
- Rewrite the class Shape without using an abstract class. This new class Shape should still match classes ShapeDriver, Rectangle and Circle (programmed in previous question) and have the same expected result as seen below.

Expected result:

```
Area of c1 Circle: 12.56
Area of c2 Circle: 50.24
Area of r1 Rectangle: 8.0
Area of r2 Rectangle: 18.0
Total Area is: 88.8
```

Coding Exercise 4

Consider the following UML diagram:



A UML diagram of Animal, Dog, Cow, and Duck classes. Recall that - denotes *private*, + denotes *public*, and # denotes *protected*.

Things to do:

1. Declare an abstract class named Animal with the implementations for getIsMammal() and getIsCarnivorous() methods, as well as an abstract method named getGreeting() .

2. Create Dog, Cow , and Duck objects.

3. Call the getIsMammal() , getIsCarnivorous() , and getGreeting() methods on each of these respective objects.

4. Three classes named Dog , Cow , and Duck that inherit from the Animal class.

5. No-argument constructors for each class that initialize the instance variables inherited from the superclass.

6. Each class must implement the getGreeting() method:
- For a Dog object, this must return the string ruff .
- For a Cow object, this must return the string moo .
- For a Duck object, this must return the string quack .

Input Format
There is no input for this challenge.

Output Format
The getGreeting() method must always return a string denoting the appropriate greeting for the implementing class.

Sample Output
```
A dog says 'ruff', is carnivorous, and is a mammal.
A cow says 'moo', is not carnivorous, and is a mammal.
A duck says 'quack', is not carnivorous, and is not a
mammal.
```