CSA0961-JAVA PROGRAMMING

TEST – 8

SET – 2

1.Scenario:

You have an interface Shape with a method getArea().Several classes implement this interface, but there's an issue with one of the implementations.

interface Shape {

Medium

double getArea();

}

class Rectangle implements Shape {

private double width;

private double height;

public Rectangle(double width, double height) {

this.width = width;

this.height = height;

}

@Override

public double getArea() {

return width * height;

}

}

class Circle implements Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

```
}
@Override
public double getArea() {
return 3.14 * radius * radius; // Incorrect value for PI
}
}
public class Main {
public static void main(String[] args) {
Shape rectangle = new Rectangle(5, 10);
Shape circle = new Circle(7);
System.out.println(&quot;Rectangle Area: &quot; +
rectangle.getArea());
System.out.println(&quot;Circle Area: &quot; +
circle.getArea());
}
}
```

Issue:

The Circle class is using an incorrect value for PI, which affects the accuracy of the area calculation.

Question:

What is wrong with the Circle class implementation?

The issue with the Circle class is that it is using an incorrect value for the mathematical constant π (Pi). In the implementation, Pi is approximated as 3.14, which is not accurate enough for precise calculations, especially in applications requiring high precision.

How can you fix it to ensure accurate area calculation?

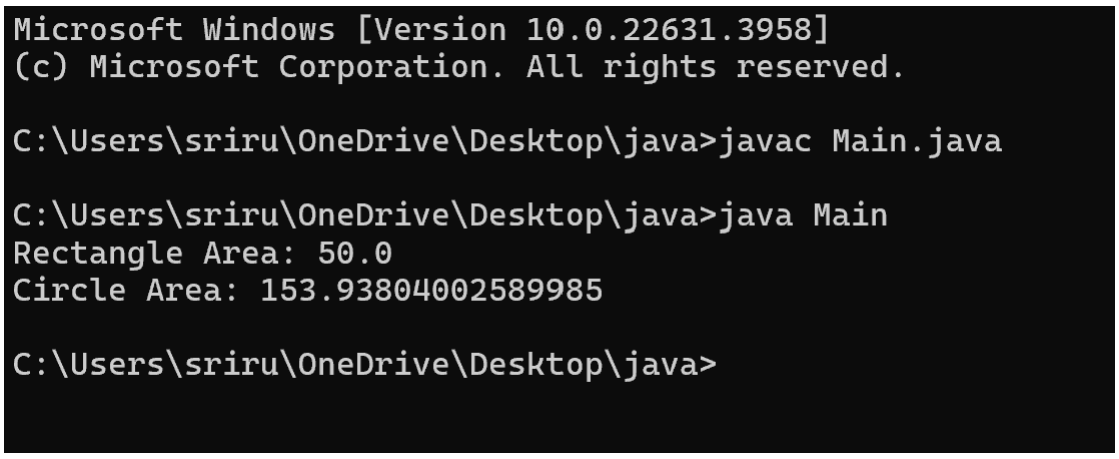
To ensure an accurate area calculation for the circle, you should use the more precise value of Pi provided by Java's Math class. The Math.PI constant in Java provides a much more accurate representation of Pi (approximately 3.141592653589793).

CORRECTED CODE:

```
interface Shape {
    double getArea();
}

class Rectangle implements Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double getArea() {
        return width * height;
    }
}

class Circle implements Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double getArea() {
```

```
        return Math.PI * radius * radius;
    }
}

public class Main {
    public static void main(String[] args) {
        Shape rectangle = new Rectangle(5, 10);
        Shape circle = new Circle(7);
        System.out.println("Rectangle Area: " + rectangle.getArea());
        System.out.println("Circle Area: " + circle.getArea());
    }
}
```

OUTPUT:



2.Scenario:

You have an abstract class Employee with a constructor that initializes the name field. Two subclasses, Manager and Developer, extend this class. There's an issue with how the Employee constructor is being called from the subclasses.

```
abstract class Employee {
protected String name;
```

```
// Constructor
public Employee(String name) {
this.name = name;
}
abstract void performDuties();
}
class Manager extends Employee {
public Manager(String name) {
super(name);
}
@Override
void performDuties() {
System.out.println(name + &amp;quot; is managing the team.&amp;quot;);
}
}
class Developer extends Employee {
public Developer(String name) {
super(name);
}
@Override
void performDuties() {

Medium

System.out.println(name + &amp;quot; is coding.&amp;quot;);
}
}
```

```
public class Main {
public static void main(String[] args) {
Employee manager = new Manager(&amp;quot;Alice&amp;quot;);
Employee developer = new Developer(&amp;quot;Bob&amp;quot;);
manager.performDuties();
developer.performDuties();
}
}
```

Issue:

The name field in the Employee class is not being printed correctly, which could be due to issues with the constructor invocation or field initialization.

Question:

What could be the issue with the Employee class or its subclasses?

The issue could be due to field shadowing if a subclass accidentally redeclares the name field, or an incorrect use of super(name) in the subclass constructors, which might prevent the proper initialization of the name field.

How can you ensure that the name field is properly initialized and used?

To ensure the name field is properly initialized and used, make sure super(name) is called as the first statement in the subclass constructors, avoid redeclaring the name field in the subclasses, and consider adding print statements in the constructor for debugging purposes to verify correct initialization.

CORRECTED CODE:

```
abstract class Employee {
    protected String name;

    // Constructor
    public Employee(String name) {
        this.name = name;
```

```
        System.out.println("Employee constructor: Name initialized to " + name);
// Debugging line
    }

    abstract void performDuties();
}

class Manager extends Employee {
    public Manager(String name) {
        super(name);
    }

    @Override
    void performDuties() {
        System.out.println(name + " is managing the team.");
    }
}

class Developer extends Employee {
    public Developer(String name) {
        super(name);
    }

    @Override
    void performDuties() {
        System.out.println(name + " is coding.");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Employee manager = new Manager("Alice");
        Employee developer = new Developer("Bob");
        manager.performDuties();
        developer.performDuties();
    }
}
```

OUTPUT:

```
Microsoft Windows [Version 10.0.22631.3958]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sriru\OneDrive\Desktop\java>javac Main.java

C:\Users\sriru\OneDrive\Desktop\java>java Main
Employee constructor: Name initialized to Alice
Employee constructor: Name initialized to Bob
Alice is managing the team.
Bob is coding.

C:\Users\sriru\OneDrive\Desktop\java>
```