

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 {
    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("Rectangle Area: " +
rectangle.getArea());
System.out.println("Circle Area: " +
circle.getArea());
}
}

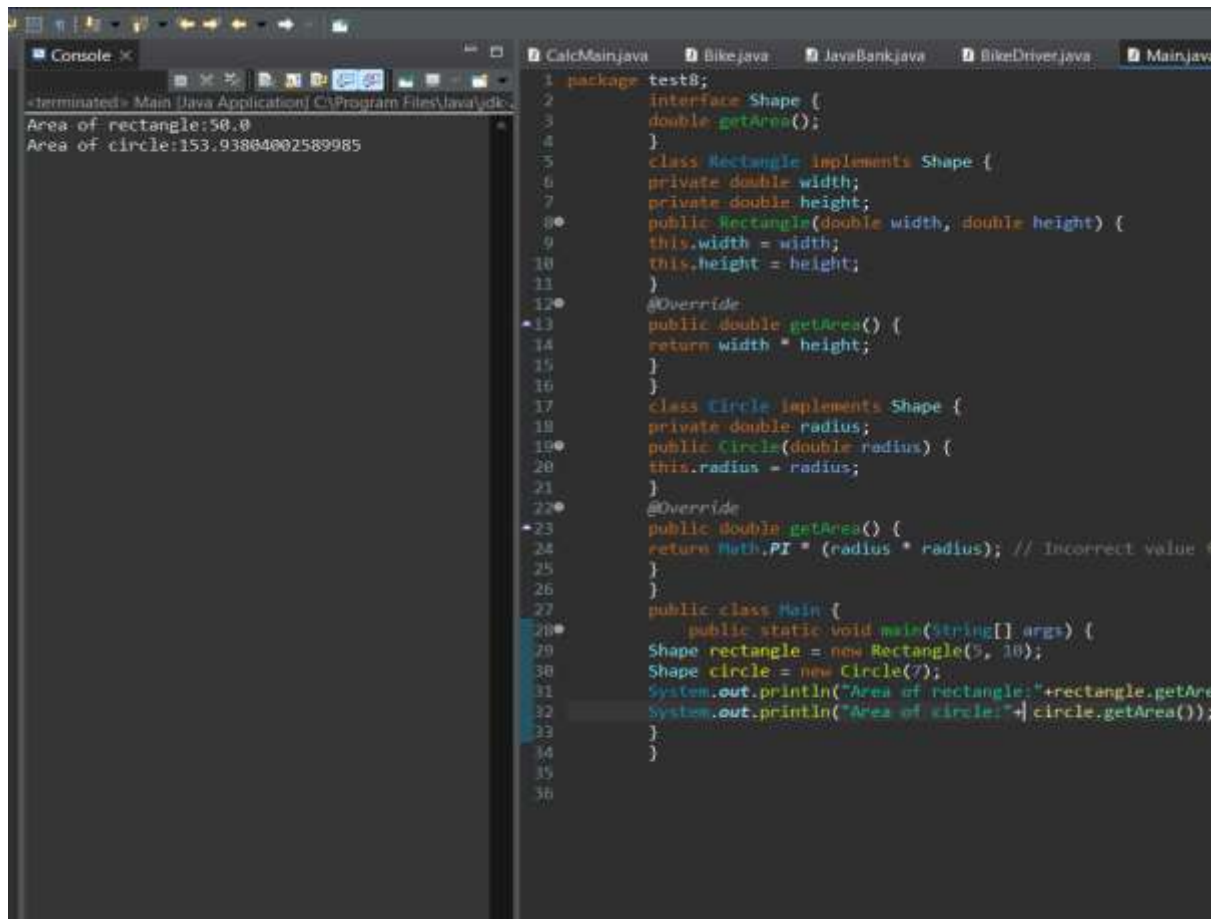
```

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?
- ☐ How can you fix it to ensure accurate area calculation?

A.



```

1 package test8;
2 interface Shape {
3     double getArea();
4 }
5 class Rectangle implements Shape {
6     private double width;
7     private double height;
8     public Rectangle(double width, double height) {
9         this.width = width;
10        this.height = height;
11    }
12    @Override
13    public double getArea() {
14        return width * height;
15    }
16 }
17 class Circle implements Shape {
18     private double radius;
19     public Circle(double radius) {
20         this.radius = radius;
21     }
22    @Override
23    public double getArea() {
24        return Math.PI * (radius * radius); // Incorrect value
25    }
26 }
27 public class Main {
28     public static void main(String[] args) {
29         Shape rectangle = new Rectangle(5, 10);
30         Shape circle = new Circle(7);
31         System.out.println("Area of rectangle:" + rectangle.getArea());
32         System.out.println("Area of circle:" + circle.getArea());
33     }
34 }
35
36

```

Console Output:

```

Area of rectangle:50.0
Area of circle:153.93804002589985

```

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 + " 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();
}
```

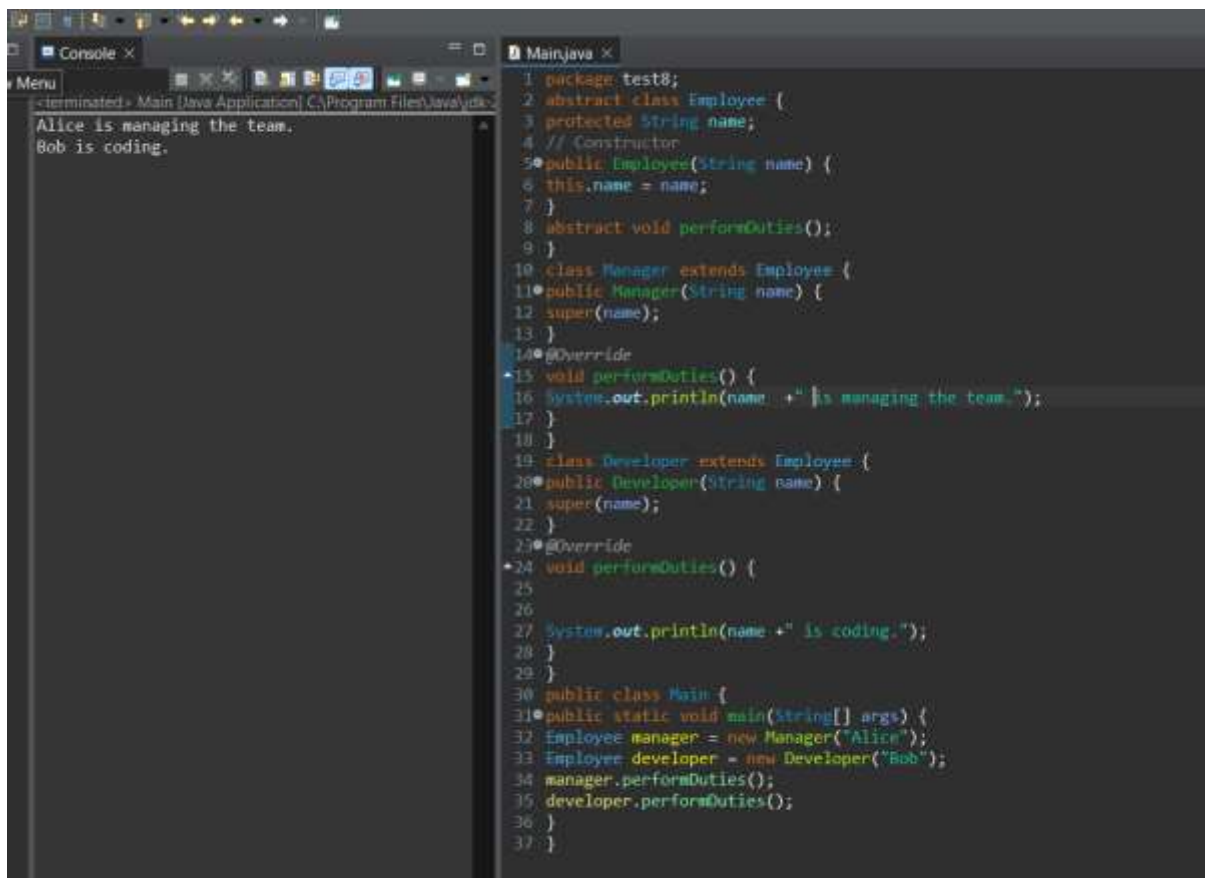
```
}  
}
```

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

A.



```
1 package test8;  
2 abstract class Employee {  
3     protected String name;  
4     // Constructor  
5     public Employee(String name) {  
6         this.name = name;  
7     }  
8     abstract void performDuties();  
9 }  
10 class Manager extends Employee {  
11     public Manager(String name) {  
12         super(name);  
13     }  
14     @Override  
15     void performDuties() {  
16         System.out.println(name + " is managing the team.");  
17     }  
18 }  
19 class Developer extends Employee {  
20     public Developer(String name) {  
21         super(name);  
22     }  
23     @Override  
24     void performDuties() {  
25  
26         System.out.println(name + " is coding.");  
27     }  
28 }  
29 }  
30 public class Main {  
31     public static void main(String[] args) {  
32         Employee manager = new Manager("Alice");  
33         Employee developer = new Developer("Bob");  
34         manager.performDuties();  
35         developer.performDuties();  
36     }  
37 }
```

Console Output:
-terminated> Main [Java Application] C:\Program Files\Java\jdk-
Alice is managing the team.
Bob is coding.