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(&quot;Rectangle Area: &quot; +
rectangle.getArea());
System.out.println(&quot;Circle Area: &quot; +
circle.getArea());
}
}
```

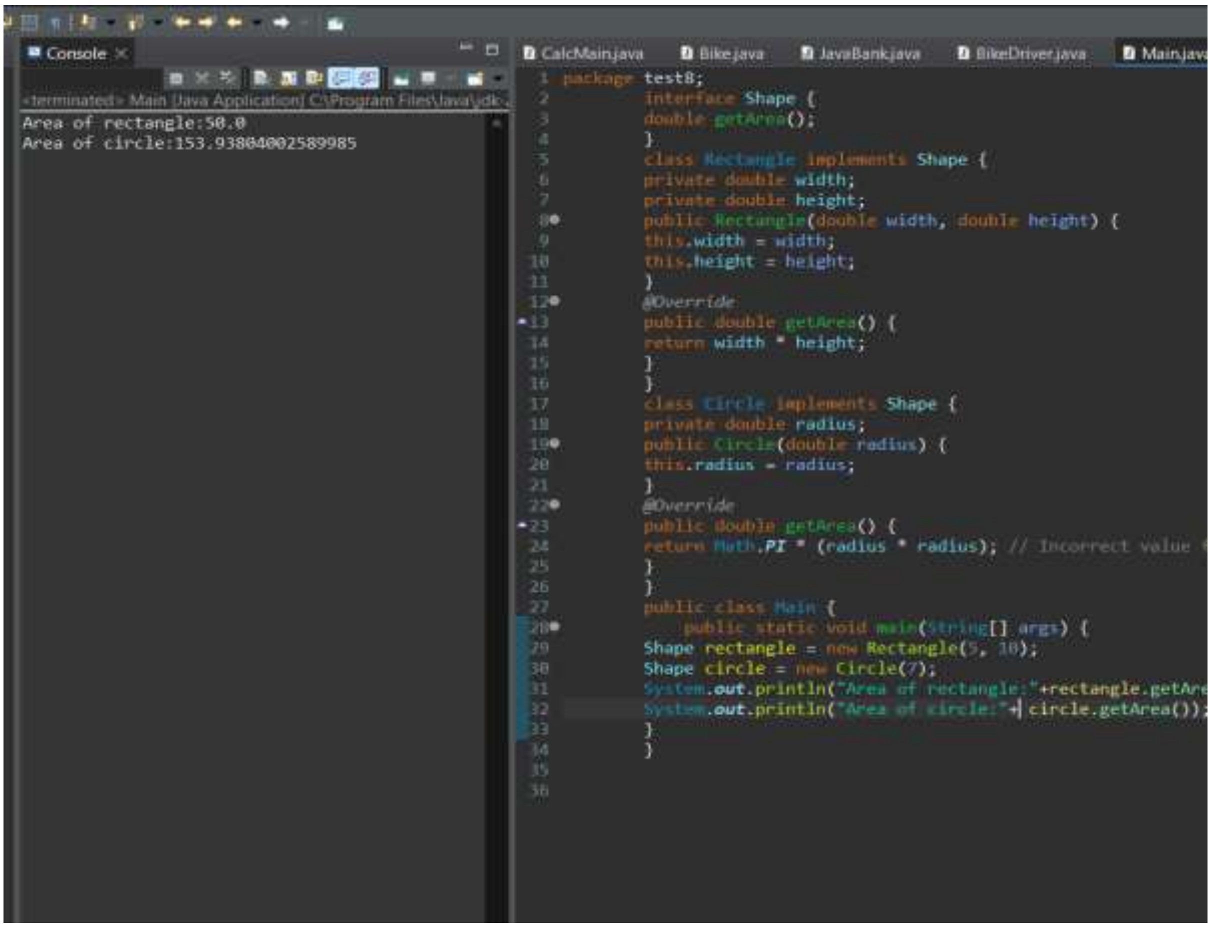
Issue: The Circle class is using an incorrect value for PI, which affects the accuracy of the area calculation.

Question:

What is wrong with the Circle classimplementation?

How can you fix it to ensure accurate area calculation?

A.



```
<terminated> Main [Java Application] C:\Program Files\Java\jdk
Area of rectangle:50.0
Area of circle:153.93804002589985
```

```
package test8;
        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 Math.PI * (radius * radius); // Incorrect value
        }
        }
        public class Main {
            public static void main(String[] args) {
        Shape rectangle = new Rectangle(5, 10);
        Shape circle = new Circle(7);
        System.out.println("Area of rectangle:"+rectangle.getAre
        System.out.println("Area of circle:"+ circle.getArea());
        }
        }
```

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 thesubclasses .

```
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 + &quot; is managing the team.&quot;);
}
}
class Developer extends Employee {
public Developer(String name) {
super(name);
}
@Override
void performDuties() {
System.out.println(name + &quot; is coding.&quot;);
}
}
public class Main {
public static void main(String[] args) {
Employee manager = new Manager(&quot;Alice&quot;);
Employee developer = new Developer(&quot;Bob&quot;);
manager.performDuties();
developer.performDuties();
```

}

}

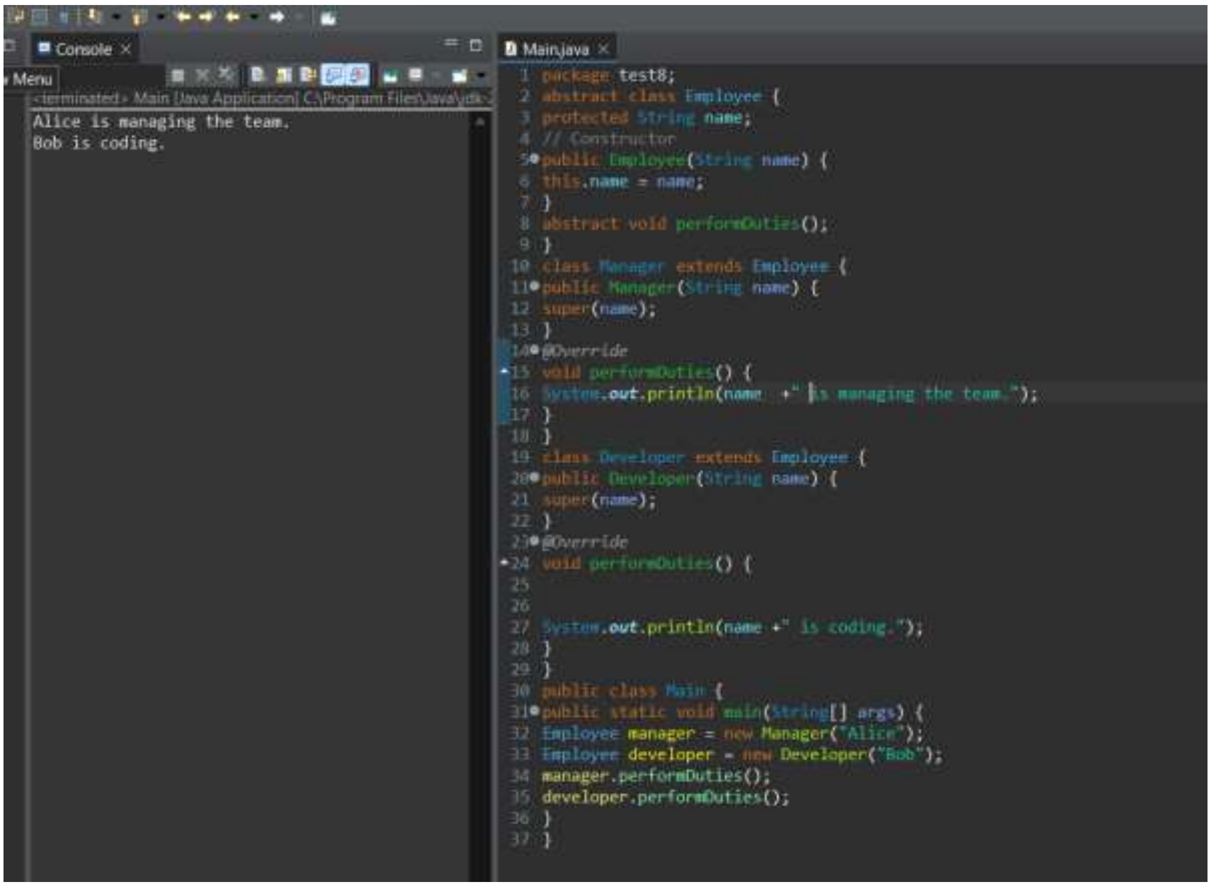
Issue: The name field in the Employee class is notbeing printed correctly, which could be due to issues withthe constructor invocation or field initialization.

Question:

** What could be the issue with the Employeeclass or its subclasses?**

** How can you ensure that the name field isproperly initialized and used?**

A.


Console
Menu
<terminated> Main [Java Application] C:\Program Files\Java\jdk
Main.java

```
Alice is managing the team.
Bob is coding.
```

```java
package test8;
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();
}
}
```