Practical 05: Inheritance & Abstract Classes

Exercise 01:

Declare an interface called “MyFirstInterface”. Decalre integer type variable called “x”. Declare an abstract method called “display()”.

1. Try to declare the variable with/without public static final keywords. Is there any difference between these two approaches? Why?
2. Declare the abstract method with/without abstract keyword. Is there any difference between these two approaches? Why?
3. Implement this into a class called “IntefaceImplemented” . Override all the abstract methods. Try to change the value of x inside this method and print the value of x. Is it possible for you to change x? why?

```
interface MyFirstInterface {

    int x = 10; // public static final by default

    abstract void display();
}

```

1.The public static final keywords are optional when declaring a variable in an interface. If they are not specified, the variable will be implicitly declared as public static final. There is no difference between these two approaches.

2.The abstract keyword is required when declaring a method in an interface. If the abstract keyword is not specified, the compiler will generate an error.

3.The IntefaceImplemented class can override the display() method and change the value of x. However, the change will not be reflected in the interface, because the interface variable x is public static final.

```
Here is the code for the IntefaceImplemented class:
class InterfaceImplemented implements MyFirstInterface {

    @Override
    public void display() {
        x = 20;
        System.out.println(x);
    }
}
```

Output:

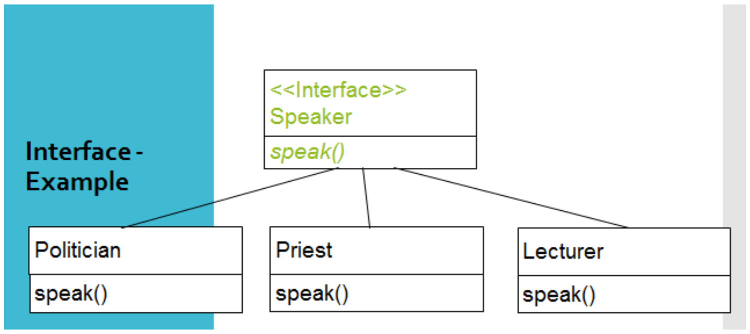
```
20
```

The value of `x` is changed to 20 in the `display()` method, but the change is not reflected in the interface. The value of `x` in the interface is still 10.

This is because the variable `x` in the interface is `public static final`, which means that it is a constant. The value of a constant cannot be changed.

<u>Exercise 02:</u>

Develop a code base for the following scenario. Recall what we have done at the lecture…



```
interface Person {

    String getName();

    int getAge();

    void sayHello();
}

class Man implements Person {

    private String name;
    private int age;

    public Man(String name, int age) {
        this.name = name;
        this.age = age;
    }
```

```java
    @Override
    public String getName() {
        return name;
    }

    @Override
    public int getAge() {
        return age;
    }

    @Override
    public void sayHello() {
        System.out.println("Hello, my name is " + name);
    }
}

class Woman implements Person {

    private String name;
    private int age;

    public Woman(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public int getAge() {
        return age;
    }

    @Override
    public void sayHello() {
        System.out.println("Hello, my name is " + name);
    }
}

public class TestPerson {
    public static void main(String[] args) {
        Person man = new Man("John Doe", 30);
        Person woman = new Woman("Jane Doe", 25);

        System.out.println("Man's name: " + man.getName());
        System.out.println("Man's age: " + man.getAge());
        man.sayHello();

        System.out.println("Woman's name: " + woman.getName());
        System.out.println("Woman's age: " + woman.getAge());
        woman.sayHello();
    }
}
```

```
/*The Output*/
Man's name: John Doe
Man's age: 30
Hello, my name is John Doe
Woman's name: Jane Doe
Woman's age: 25
Hello, my name is Jane Doe
```

Exercise 03:

Try following code. What is the outcome? Why?

Class 01:

```
final class Student {

        final int marks = 100;

        final void display();

}
```

Class 02:

```
class Undergraduate extends Student{}
```

The code will not compile because the `display()` method in the `Student` class is declared as `final`. A `final` method cannot be overridden by a subclass.

The `final` keyword in Java is used to indicate that a variable, method, or class cannot be modified. In the case of the `display()` method, the `final` keyword means that the method cannot be overridden by a subclass.

If you try to compile the code, you will get the following error message:

error: cannot override final method display()

To fix the error, you need to remove the `final` keyword from the `display()` method. Once you have done that, the code will compile and run without any problems.

```
final class Student {

    final int marks = 100;

    void display() {
        System.out.println("This is the display method");
    }
}

class Undergraduate extends Student {

    // The `display()` method is not final, so it can be overridden
    void display() {
        System.out.println("This is the Undergraduate display method");
    }
}
```

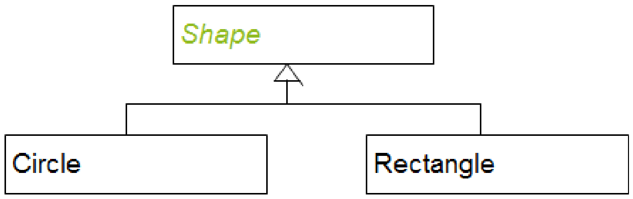
<u>Exercise 04:</u>

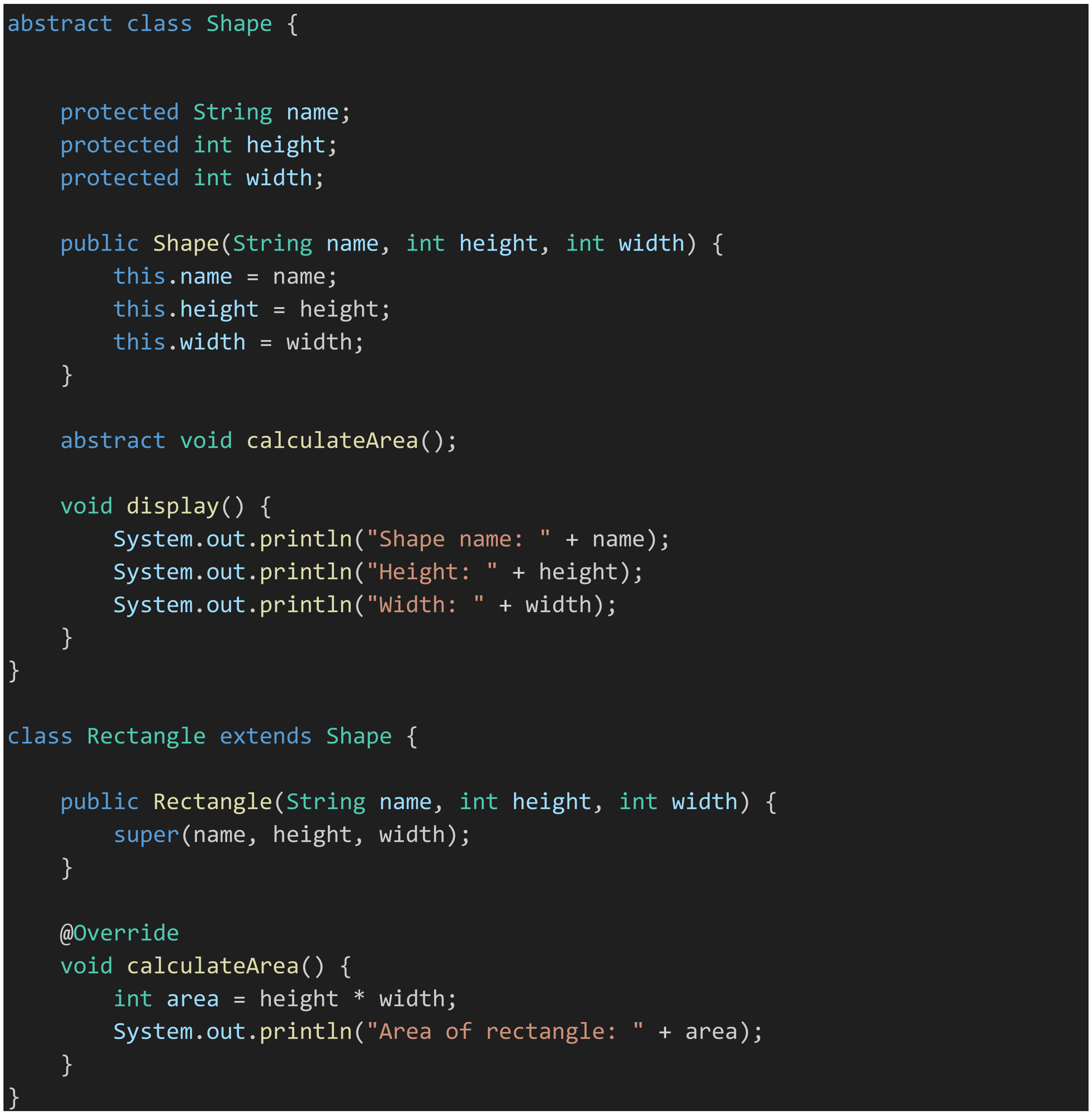
Develop a code base for the following scenario. Shape class contains an abstract method called “calculateArea” and non-abstract method called “display”. Try to pass required values at the instantiation. Recall what we have done at the lecture…

[redacted]

Abstract Class - Example

Shape is a abstract class.





```java
abstract class Shape {


    protected String name;
    protected int height;
    protected int width;

    public Shape(String name, int height, int width) {
        this.name = name;
        this.height = height;
        this.width = width;
    }

    abstract void calculateArea();

    void display() {
        System.out.println("Shape name: " + name);
        System.out.println("Height: " + height);
        System.out.println("Width: " + width);
    }
}

class Rectangle extends Shape {

    public Rectangle(String name, int height, int width) {
        super(name, height, width);
    }

    @Override
    void calculateArea() {
        int area = height * width;
        System.out.println("Area of rectangle: " + area);
    }
}
```

```java
class Circle extends Shape {

    public Circle(String name, int height, int width) {
        super(name, height, width);
    }

    @Override
    void calculateArea() {
        int area = (int) (Math.PI * Math.pow(height, 2));
        System.out.println("Area of circle: " + area);
    }
}

public class TestShape {

    public static void main(String[] args) {
        Rectangle rectangle = new Rectangle("Rectangle", 10, 20);
        rectangle.display();
        rectangle.calculateArea();

        Circle circle = new Circle("Circle", 10, 10);
        circle.display();
        circle.calculateArea();
    }
}
```

<u>Out put</u>

```
Shape name: Rectangle
Height: 10
Width: 20
Area of rectangle: 200
Shape name: Circle
Height: 10
Width: 10
Area of circle: 100
```