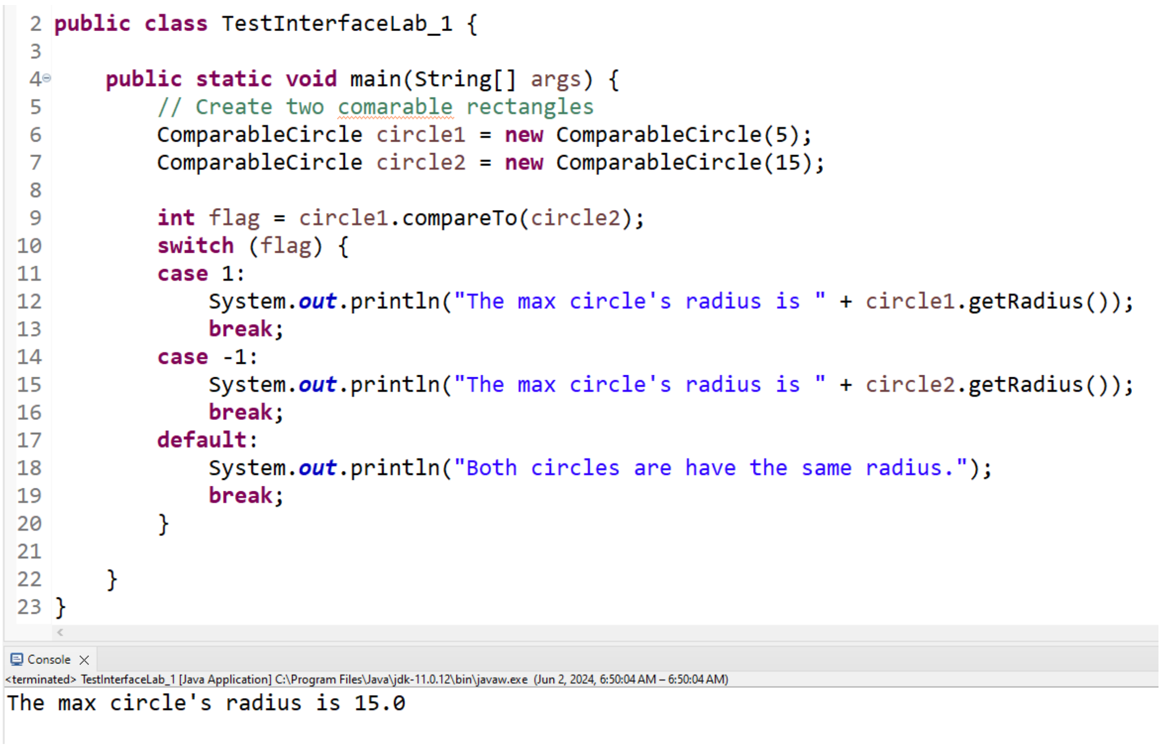
Lab: Interface

Objectives: To practice interface concept

1. (The **ComparableCircle** class) Define a class named **ComparableCircle** that extends **Circle** and implements **Comparable**. Draw the UML diagram and implement the **compareTo** method to compare the circles on the basis of radius. Write a test class to find the larger of two instances of **ComparableCircle** objects. (Given the Circle class)



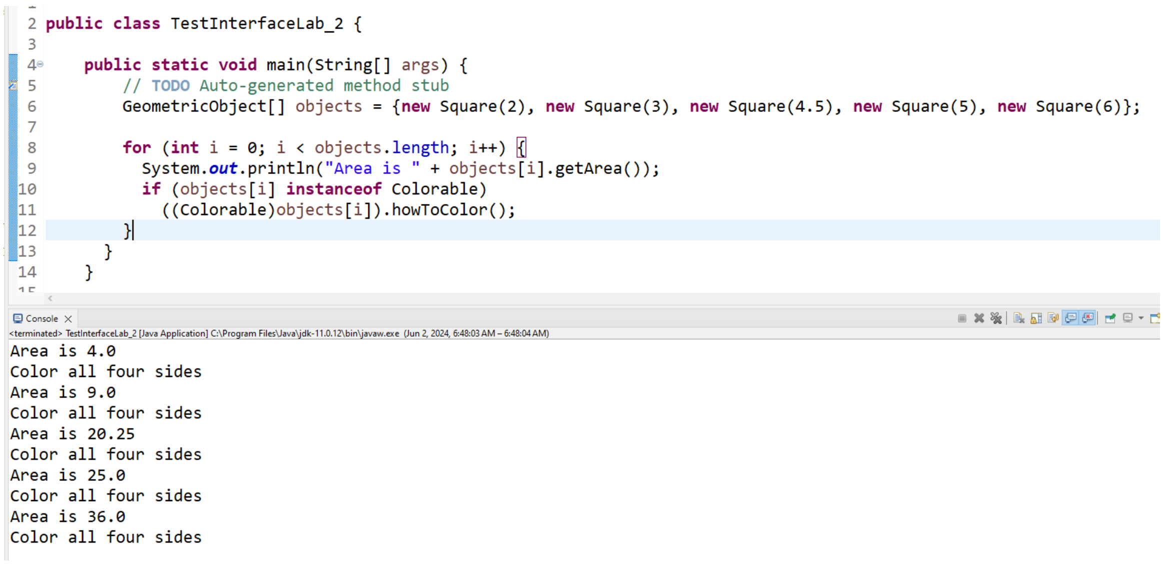
```java
public class TestInterfaceLab_1 {

    public static void main(String[] args) {
        // Create two comarable rectangles
        ComparableCircle circle1 = new ComparableCircle(5);
        ComparableCircle circle2 = new ComparableCircle(15);

        int flag = circle1.compareTo(circle2);
        switch (flag) {
        case 1:
            System.out.println("The max circle's radius is " + circle1.getRadius());
            break;
        case -1:
            System.out.println("The max circle's radius is " + circle2.getRadius());
            break;
        default:
            System.out.println("Both circles are have the same radius.");
            break;
        }

    }
}
```

Console

```
<terminated> TestInterfaceLab_1 [Java Application] C:\Program Files\Java\jdk-11.0.12\bin\javaw.exe (Jun 2, 2024, 6:50:04 AM – 6:50:04 AM)
The max circle's radius is 15.0
```

2. (The **Colorable** interface) Design an interface named **Colorable** with a void method named **howToColor**(). Every class of a colorable object must implement the **Colorable** interface. Design a class named **Square** that extends **GeometricObject** and implements **Colorable**. Implement **howToColor** to display the message "Color all four sides." The **Square** class has a private double data field named side with its getter and setter methods. It has a no-arg constructor to create a **Square** with side 0, and another constructor that creates a Square with the specified side.

   Draw a UML diagram that involves **Colorable**, **Square**, and **GeometricObject**. Write a test program that creates an array of five **GeometricObjects**. For each object in the array, display its area and invoke its **howToColor** method if it is colorable.



```java
public class TestInterfaceLab_2 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        GeometricObject[] objects = {new Square(2), new Square(3), new Square(4.5), new Square(5), new Square(6)};

        for (int i = 0; i < objects.length; i++) {
          System.out.println("Area is " + objects[i].getArea());
          if (objects[i] instanceof Colorable)
            ((Colorable)objects[i]).howToColor();
        }
      }
    }
```

Console

<terminated> TestInterfaceLab_2 [Java Application] C:\Program Files\Java\jdk-11.0.12\bin\javaw.exe (Jun 2, 2024, 6:48:03 AM – 6:48:04 AM)

```
Area is 4.0
Color all four sides
Area is 9.0
Color all four sides
Area is 20.25
Color all four sides
Area is 25.0
Color all four sides
Area is 36.0
Color all four sides
```

3. **Key-Value Storage Comparison:**

In Java, dictionaries and maps are used to store collections of key-value pairs. While dictionaries were used historically, maps offer a more modern and flexible approach.

[1] What are the result of each snippet code?

Snippet 1 (Dictionary Class):

```java
// (Assuming Hashtable is used internally)
import java.util.Dictionary;
import java.util.Hashtable;

public class StudentRecordsDictionary {

  public static void main(String[] args) {
    Dictionary<Integer, String> studentRecords = new Hashtable<>();
    studentRecords.put(123, "Alice");
    String name = studentRecords.get(123);
    System.out.println("Student with ID 123: " + name);
  }
}
```

Result: Student with ID 123: Alice

Snippet 2 (Map Interface):

```java
import java.util.HashMap;
import java.util.Map;

public class StudentRecordsMap {

  public static void main(String[] args) {
    Map<Integer, String> studentRecords = new HashMap<>();
    studentRecords.put(123, "Alice");
    if (studentRecords.containsKey(789)) {
      System.out.println("Student with ID 789 exists");
    } else {
      System.out.println("Student with ID 789 does not exist");
    }
  }
}
```

Result: Student with ID 789 does not exist

[2] Based on your analysis, list three key differences between using the Dictionary class and the Map interface for storing student records.

1. The dictionary is older than a map and limited functionally
2. The dictionary is an abstract class and must use a subclass (hashmap) to create an object; therefore, it isn't instantiated. So the map is more short code.
3. Map interface can be accessing keys and values separately, be checking key

[3] Which approach (Dictionary class or Map interface) do you think is more modern and flexible? Why?

I think Map interface because it can acess key and values separately and don't use Hashtable subclass to create object.