Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester:(Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO: 05
Course Title: Object-Oriented Programming Lab
Course Code: CSE 202 **Section**: 223 D9

Lab Experiment Name: Implementation of Interface and Multiple Inheritance

Student Details

	Name	ID
1.	Promod Chandra Das	231002005

Lab Date :
Submission Date :
Course Teacher's Name : Noyan Ali

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

- **TITLE OF THE LAB REPORT EXPERIMENT:** Implementation of Interface and Multiple.

- **OBJECTIVES/AIM :**

  To gather knowledge of how the interface works.
  • Understand multiple inheritances using a class implementing multiple interfaces.
  • To solve problems using the concept of interface.

- **PROCEDURE / ANALYSIS / DESIGN :**

  - **Problem analysis:**

An interface is a reference type in Java. It is similar to class. But it is a collection of abstract methods and
static constants. Three important attributes of interfaces are :
1. Access of an interface is either public or default.
2. Variables declared inside an interface are implicitly final and static.
3. All methods and variables are implicitly public if the interface, itself, is declared as public.
Interfaces provide specifications that a class (which implements it) must follow. Interfaces are also used to
achieve multiple inheritance in Java. If a subclass is inherited from two or more classes, it's multiple inheritance.
Using the concept of interface, implement the following:
• Creating an interface Printable with an abstract method - void print();
• Creating another interface Showable with an abstract method - void show();
• Create a Drawable class which implements both Printable and Showable interfaces and implements their
methods.
• Create an object of Drawable class in the main method and call the print and show methods.

- ✓ **Keywords**

- **Interface**

Like the class keyword, the interface keyword is used to declare an interface.

```
interface Polygon{
 public static final String color = "Green";
public void getArea();
 }
```

In the example above, we have used the interface keyword to declare an interface named Polygon. It includes
a constant variable color and an abstract method getArea(). This means that any class that implements Polygon
must provide an implementation for the getArea() method.

- **implements**

Like abstract classes, we cannot create objects of interfaces. However, we can implement interfaces in other
classes. In Java, we use the implements keyword to implement interfaces. To implement an interface a class
must define each of the method declared in the interface. Each class can also add new features.

```
interface Polygon{
public static final String color = "Green";
 public void getArea();
 }

class Rectangle implements Polygon{
 public void getArea(int length, int breadth){
 System.out.println("The area of the rectangle is "+ (length *

breadth));

 }
}
```

In the above program, the Rectangle class implements Polygon interface, therefore it has the method
getArea() with implementation.

- extends

Similar to classes, interfaces can extend other interfaces. The extends keyword is used for extending interfaces.

```
1 interface Line{
2 //members of Line interface
3 }
4
5 interface Polygon extends Line{
6 //members of Polygon interface and Line interface
7 }
```

As shown in the figure given below, a class extends another class, an interface extends another interface but
a class implements an interface.

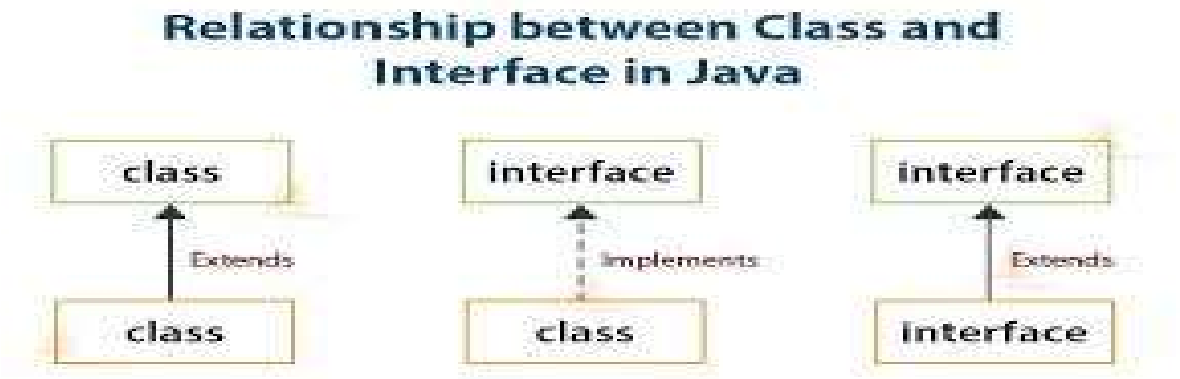

Figure 1: Relationship between inteface and class

- Multiple Inheritance

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple
inheritance.

Figure 2: Multiple inheritance using Interface

- **Implementation in Java**

```
1 interface Printable{
2 void print();

3 }
4
5 interface Showable{
6 void show();
7 }
8
9 class Drawable implements Printable,Showable{
10
11 public void print(){System.out.println("Hello");}
12 public void show(){System.out.println("Welcome");}
13 }
14
15 class Main{
16 public static void main(String args[]){
17 Drawable obj = new Drawable();
18 obj.print();
19 obj.show();
20 }
21 }
```

- **Input/Output**

The output of the program is given below.

```
Hello
Welcome
```

Lab Exercise (Submit as a report)

Create an interface isEmergency with only one method - soundSiren which takes no arguments and returns
no value.
• Write a class FireEmergency that implements the IsEmergency interface. The soundSiren method should
print "Siren Sounded".
• Write a class SmokeAlarm that does not implement any interface. The class has an empty body.
• Create an array of Object class, myArray in the main method.
• Construct 2 SmokeAlarm object and add it to the array myArray in the main method.
• Construct 2 FireEmergency object and add it to the array myArray in the main method.
• In the main method, write a for loop, to print which array elements are instances of classes that implement
the IsEmergency interface and if so, call the soundSiren method.

✓ **Answer:**

```
// Define the IsEmergency interface
interface IsEmergency {
    void soundSiren();
}

// Implement the IsEmergency interface in the FireEmergency class
class FireEmergency implements IsEmergency {
    @Override
    public void soundSiren() {
        System.out.println("Siren Sounded");
    }
}

// Define the SmokeAlarm class
class SmokeAlarm {
    // Empty body
}

public class Main {
    public static void main(String[] args) {
        // Create an array of Object class
```

```
        Object[] myArray = new Object[4];

        // Construct 2 SmokeAlarm objects and add them to myArray
        myArray[0] = new SmokeAlarm();
        myArray[1] = new SmokeAlarm();

        // Construct 2 FireEmergency objects and add them to myArray
        myArray[2] = new FireEmergency();
        myArray[3] = new FireEmergency();

        // Loop through the array
        for (Object obj : myArray) {
            // Check if the current element implements the IsEmergency interface
            if (obj instanceof IsEmergency) {
                // If it does, call the soundSiren method
                ((IsEmergency) obj).soundSiren();
            }
        }
    }
}
```

6. **ANALYSIS AND DISCUSSION**

This Java implementation demonstrates interface-based polymorphism, circumventing Java's lack of multiple class inheritance. The `IsEmergency` interface mandates a common `soundSiren()` method, facilitating diverse emergency object handling. By implementing this interface, the `FireEmergency` class achieves polymorphic behavior, enhancing code reusability and maintainability without resorting to multiple inheritance.

7. **SUMMARY:**

In this Java implementation, the interface `IsEmergency` establishes a contract for emergency-related classes, promoting code cohesion. Despite Java's absence of multiple class inheritance, the `FireEmergency` class achieves polymorphism by implementing `IsEmergency`, enabling diverse emergency object handling. This approach fosters modular design, allowing for future extension with minimal modification. By leveraging interfaces, the implementation circumvents the

limitations of single class inheritance, ensuring flexibility and scalability. Overall, the strategy underscores the power of interface-based polymorphism in Java, offering a robust solution to complex inheritance scenarios and promoting code maintainability and extensibility.