Green University of Bangladesh

Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering, Semester: Spring, Year: 2024, B.Sc. in CSE (weekend)

Lab Report # 03

Course Title: Object Oriented Programming Lab

Course Code: CSE-202

Section: 223 E1

Student Details

	Name	ID
1.	Waliullah	223015026

Date : **22-03-2024**

Submission Date : **29-03-2024**

Course Teacher's Name : **Abdullah Al Farhad**

Assignment Status

Marks: …… **Signature:.....**

Comments:..... **Date:.....**

Title: Exploring Multiple Inheritance in Java

1. Introduction:

- This lab report investigates the concept of multiple inheritance in Java and explores alternative approaches for achieving similar functionality.

2. Objective:

- Understand why Java does not support traditional multiple inheritance.
- Implement Class A, Class B, and Class C, with Class C attempting to inherit from both A and B.
- Explore different combinations of access modifiers (public, private, protected) to analyze their impact on inheritance.

3. Exploring Access Modifiers:

While we cannot achieve true multiple inheritance with classes, we can analyze how access modifiers affect inheritance:

a) Public Members:

- Public members declared in A and B would be accessible in C.

b) Private Members:

- Private members in A and B would be inaccessible in C.

c) Protected Members:

- Protected members in A and B would be accessible in C and its subclasses.

Note: Since we cannot create a class inheriting from both A and B, verifying these access modifier behaviors with code is not possible.

4. Procedure:

- Understanding Multiple Inheritance Limitations
  Java enforces a single inheritance model, meaning a class can only inherit directly from one superclass. This design choice helps avoid the "diamond problem" that can arise with multiple inheritance. In the diamond problem, two superclasses have a method with the same signature, and a subclass inherits from both. This creates ambiguity about which method implementation should be called.
- Implementing Classes and Access Modifiers
  Code (compile-time error will occur):

```
class A {
    public void methodA() {
        System.out.println("Method A from Class A");
    }
}

class B {
    public void methodB() {
        System.out.println("Method B from Class B");
    }
}
```

```
// This line will cause a compile-time error
class C extends A, B {
    // ...
}
```

Explanation:

- The code attempts to create a class C that inherits from both A and B. However, Java will throw a compile-time error because multiple inheritance is not supported.

5. Implementations and Output:

- **Alternative Approaches**
  Composition (Has-A Relationship):
  A class can create an instance of another class as a member variable. This allows the class to access and use the methods of the member class.

- Code:

```
public class lab3 {
    public static void main(String[] args) {
        C c = new C();
        c.displayPublicA(); // Public method of A accessible
        // c.displayPrivateA(); // Error: displayPrivateA() has private access in A
        c.displayProtectedA(); // Protected method of A accessible
        c.displayDefaultA(); // Default method of A accessible

        c.displayPublicC(); // Public method of C accessible
        // c.displayPrivateC(); // Error: displayPrivateC() has private access in C
        c.displayProtectedC(); // Protected method of C accessible
        c.displayDefaultC(); // Default method of C accessible

        c.accessMethods(); // Accessing methods from A and B through C
    }
}




// Class A
public class A {
    public void displayPublicA() {
        System.out.println("Public method of class A");
    }

    // private void displayPrivateA() {
    //     System.out.println("Private method of class A");
    // }
```

```
    protected void displayProtectedA() {
        System.out.println("Protected method of class A");
    }

    void displayDefaultA() {
        System.out.println("Default method of class A");
    }
}
```

```
// Class B
public class B {
    public void displayPublicB() {
        System.out.println("Public method of class B");
    }

    // private void displayPrivateB() {
    //     System.out.println("Private method of class B");
    // }

    protected void displayProtectedB() {
        System.out.println("Protected method of class B");
    }

    void displayDefaultB() {
        System.out.println("Default method of class B");
    }
}
```

```
// Class C inherits from both A and B
public class C extends A{
    private B b = new B(); // Composition of B in C

    public void displayPublicC() {
        System.out.println("Public method of class C");
    }

    // private void displayPrivateC() {
    //     System.out.println("Private method of class C");
    // }

    protected void displayProtectedC() {
        System.out.println("Protected method of class C");
    }

    void displayDefaultC() {
        System.out.println("Default method of class C");
    }
```

```
    // Method to access methods from A and B
    public void accessMethods() {
        displayPublicA(); // Accessing public method of A
        // displayPrivateA(); // Error: displayPrivateA() has private access in A
        displayProtectedA(); // Accessing protected method of A
        displayDefaultA(); // Accessing default method of A

        b.displayPublicB(); // Accessing public method of B through composition
        // b.displayPrivateB(); // Error: displayPrivateB() has private access in B
        b.displayProtectedB(); // Accessing protected method of B
        b.displayDefaultB(); // Accessing default method of B
    }

}
```

- **Output:**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

Waliullah@Waliullah MINGW64 /d/OOP Lab University (main)
$ cd "/d/OOP Lab University/Lab3/" && javac lab3.java && java lab3
Public method of class A
Protected method of class A
Default method of class A
Public method of class C
Protected method of class C
Default method of class C
Public method of class A
Protected method of class A
Default method of class A
Public method of class B
Protected method of class B
Default method of class B
```

6. Limitations:

Multiple inheritance, which allows a class to inherit from more than one parent class, has been intentionally avoided in Java.

- Diamond Problem: In multiple inheritance, if a class inherits from two classes, which in turn inherit from a common base class, and both base classes have a method with the same name and signature, there arises ambiguity as to which method should be inherited.

- Complexity: Multiple inheritance can lead to complex class hierarchies, making code harder to understand, maintain, and debug.
- Java Interfaces and Composition: Java provides interfaces to achieve a form of multiple inheritance through interface implementation. This approach promotes better code organization and avoids the complexities associated with multiple inheritance.
- Fragility: Changes to one of the parent classes in a multiple inheritance hierarchy can have unforeseen effects on subclasses due to tight coupling. This can lead to fragility in the codebase.

7. Conclusion:

- While Java does not support multiple inheritance of classes, interfaces and composition offer effective alternatives that promote maintainability and avoid the "diamond problem." This lab report explored the limitations and provided alternative approaches for achieving functionality similar to multiple inheritance.