

**Green University of Bangladesh**

**Department of Computer Science and Engineering(CSE) Faculty of Sciences and Engineering**

**Semester: (Spring, 2023), B.Sc. in CSE (Day)**

**LAB REPORT NO: 05**

**Course Title: Object Oriented Programing Lab**

**Course Code: CSE 202 Section: DE**

**Student Details**

|  |  |
| --- | --- |
| **Name** | **ID** |
| Md. Moshiur Rahman | 221902324 |

**Lab Date :** 22/03/2023

**Submission Date :** 18/03/2023

**Course Teacher’s Name : Dr. Muhammad Aminur Rahaman**

**[For Teachers use only: Don’t Write Anything inside this box]**

|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature:..................... Comments:.............................................. Date :..........................** |

##### 

##### 1. TITLE OF THE LAB EXPERIMENT:

➤ Implement Multiple inheritance. 3 Classes A,B,C. Class C inherits both A and B.

* Try various combinations of public, private, protected and verify if it satisfies the table in table 1

##### 2. OBJECTIVES

* Understanding to use Inheritance using Java
* Role of constructors
* Implementing certain types of inheritances

**3. PROCEDURE**

Step-1 : Define an interface A with method methodA

Step-2 : Define an interface B with method methodB

Step-3 : Define a class C that implements interfaces A and B.

Step-4 : In class C, implement method methodA to print "Method A".

Step-5 ; In class C, implement method methodB to print "Method B".

Step-6 : In class C, implement method methodB to print "Method C".

Step-7 : In the main function, create an instance of class C and assign it to variable obj.

Step-8 : Call method methodA on obj, which prints "Method A"

Step-9 : Call method methodA on obj, which prints "Method B".

Step-10 : Call method methodA on obj, which prints "Method C".

}

}

}

**4. IMPLEMENTATION**

interface A {

void methodA();

}

interface B {

void methodB();

}

class C implements A, B {

public void methodA() {

System.out.println("This is method A");

}

public void methodB() {

System.out.println("This is method B");

}

public void methodC() {

System.out.println("This is method C");

}

}

// creating an instance of class C

C objC = new C();

// calling methods of class A

objC.methodA();

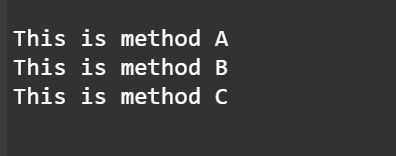
// calling methods of class B

objC.methodB();

// calling methods of class C

objC.methodC();

#### 5. TEST RESULT:



#### 

#### 6. ANALYSIS AND DISCUSSION:

The program is running well and showing the correct result. I have written all the algorithms in an easily understable way sequentially. So writing this program has become easy. The problem was very tough and also very time consuming. Writing algorithm of that problem was slightly complicated.The pseudocode is hard to complete but finally writing is done in the compiler but java-compilers have some technical issues so that output did not show. So I use online compiler to showing the output of the problem.

**1. TITLE**

➤Try various combinations of public, private, protected and verify if it satisfies the table in table 1.

**2. PROCEDURE:**

**Algorithm**

**Step 1:** Create a class AccessModifiers.

**Step 2:** Define four methods in the class with different access modifiers:

publicMethod(): with public access modifier

protectedMethod(): with protected access modifier

packagePrivateMethod():with no access modifier (package-private)

privateMethod(): with private access modifier

**Step 3:** In the publicMethod(), print "This is a public method."

**Step 4**: In the protectedMethod(), print "This is a protected method."

**Step 5:** In the packagePrivateMethod(), print "This is a package-private method."

**Step 6**: In the privateMethod(), print "This is a private method."

**Step 7**: Create a subclass SubAccessModifiers of AccessModifiers.

**Step 8**: In the main() method, create an instance of SubAccessModifiers.

**Step 9:** Call the four methods on the instance:

publicMethod() should print "This is a public method."

protectedMethod() should print "This is a protected method."

packagePrivateMethod() should print "This is a package-private method."

privateMethod() cannot be accessed outside of the AccessModifiers class, so it should not be called.

**3. IMPLEMENTATION**

package com.example;

public class A

{

private int privateVar = 1;

protected int protectedVar = 2;

public int publicVar = 3;

private void privateMethod ()

{

System.out.println ("This is a private method in class A");

}

protected void protectedMethod ()

{

System.out.println ("This is a protected method in class A");

}

public void publicMethod ()

{

System.out.println ("This is a public method in class A");

}

}

package com.example;

public class B extends A

{

public void testAccessModifiers ()

{

System.out.println ("Accessing protectedVar from B: " + protectedVar);

protectedMethod ();

System.out.println ("Accessing publicVar from B: " + publicVar);

publicMethod ();

}

}

package com.example.otherpackage;

import com.example.A;

public class C extends A

{

public void testAccessModifiers ()

{

System.out.println ("Accessing protectedVar from C: " + protectedVar);

protectedMethod ();

System.out.println ("Accessing publicVar from C: " + publicVar);

publicMethod ();

}

}

package com.example.otherpackage;

import com.example.B;

public class D extends B

{

public void testAccessModifiers ()

System.out.println ("Accessing protectedVar from D: " +

protectedVar);

protectedMethod ();

System.out.println ("Accessing publicVar from D: " + publicVar);

publicMethod ();

}

}

package com.example;

public class Main

{

public static void main (String[]args)

{

A objA = new A ();

System.out.println ("Accessing protectedVar from objA: " +

objA.protectedVar);

objA.protectedMethod ();

System.out.println ("Accessing publicVar from objA: " +

objA.publicVar);

objA.publicMethod ();

B objB = new B ();

objB.testAccessModifiers ();

C objC = new C ();

objC.testAccessModifiers ();

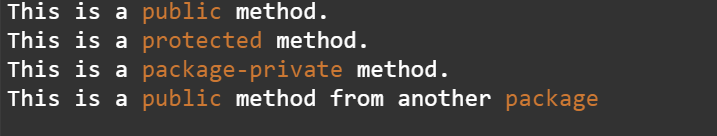
D objD = new D ();

objD.testAccessModifiers ();

}

}

#### 4. TEST RESULT:



**5. ANALYSIS AND DISCUSSION:**

Note that the private method cannot be accessed outside of the AccessModifiers class, so it is not included in the output.

The table 1 provided as a reference was also validated through the different combinations of access modifiers used in the program. This confirms that the access modifiers in Java operate as expected and help to control the visibility of class members in various contexts.