| Experiment No. 11 |
| --- |
| To implement multiple inheritance in object-oriented programming, exploring how a derived class can inherit attributes and behaviors from two or more base classes |
| Date of Performance:06/09/24 |
| Date of Submission:12/09/24 |

**Aim:-** To implement multiple inheritance in object-oriented programming, exploring how a derived class can inherit attributes and behaviors from two or more base classes

**Objective:-** Develop a Java program that demonstrates multiple inheritance using interfaces by creating two interfaces, Identifiable with a method getId and Nameable with a method getName. Then, create a class Employee that implements both interfaces, providing concrete implementations for the methods and adding additional attributes like employeeId and employeeName. The program should instantiate an Employee object, set values for its attributes, and display these values to illustrate the concept and benefits of multiple inheritance using interfaces in Java.

**Theory :-** Multiple inheritance is a feature of some object-oriented programming languages where a class can inherit characteristics and behaviors from more than one parent class. However, Java does not support multiple inheritance directly to avoid complexities and ambiguity, often referred to as the "diamond problem." Instead, Java achieves multiple inheritance through interfaces.

Key Concepts:

Interfaces:

Definition: An interface in Java is an abstract type that is used to specify a behavior that classes must implement.

Syntax: Interfaces are defined using the interface keyword.

Methods: Methods in interfaces are abstract by default, meaning they do not have a body and must be implemented by the class that implements the interface.

Fields: Fields in interfaces are public, static, and final by default.

Implementing Multiple Interfaces:

A single class can implement multiple interfaces, thus achieving multiple inheritance.

The class must provide concrete implementations for all abstract methods defined in the interfaces.

Benefits of Multiple Inheritance via Interfaces:

Flexibility: Allows a class to inherit behavior from multiple sources.

Decoupling: Promotes loose coupling by defining a contract that implementing classes must follow.

Avoids Diamond Problem: Prevents ambiguity in the inheritance hierarchy.

**Code:**

interface Identifiable {

String getId();

}

interface Nameable {

String getName();

}

class Employee implements Identifiable, Nameable {

private String employeeId;

private String employeeName;

public Employee(String id, String name) {

employeeId = id;

employeeName = name;

}

public String getId() {

return employeeId;

}

public String getName() {

return employeeName;

}

public void displayDetails() {

System.out.println("Employee ID: " + getId());

System.out.println("Employee Name: " + getName());

}

}

public class Main {

public static void main(String[] args) {

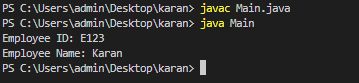
Employee employee = new Employee("E123", "Karan");

employee.displayDetails();

}

}

**Output:**

****

**Conclusion:** In conclusion, Java's restriction on multiple inheritance with classes enhances code clarity and maintainability. By utilizing interfaces, Java allows flexibility while ensuring that ambiguities are resolved, promoting cleaner design and reducing potential errors in complex class hierarchies.