Experiment 7:

Aim: Apply the concept of interfaces to achieve multiple inheritance and dynamic method dispatch in Java.

Theory:

Multiple Inheritance using Interface:

In Java, interfaces enable multiple inheritance by allowing a class to implement multiple interfaces, inheriting methods from each.

Syntax:

```java
interface Interface1 {
    void method1();
}
interface Interface2 {
    void method2();
}

class ClassA implements Interface1, Interface2 {
    public void method1() { /* implementation */ }
    public void method2() { /* implementation */ }
}
```

Example:

```java
interface A {
    void methodA();
}

interface B {
    void methodB();
}

class C implements A, B {
    public void methodA() {
        System.out.println("Method A");
    }
    public void methodB() {
        System.out.println("Method B");
    }
}
```

Dynamic Method Dispatch:

Dynamic method dispatch resolves which overridden method to call at runtime, based on the object's type, enabling runtime flexibility.

Syntax:

```
class Superclass {
    void display() { /* code */ }
}

class Subclass extends Superclass {
    void display() { /* code */ }
}

Superclass obj = new Subclass();
obj.display();  // Method resolved at runtime
```

Example:

```
class A {
    void display() {
        System.out.println("A");
    }
}

class B extends A {
    void display() {
        System.out.println("B");
    }
}

A obj = new B();
obj.display();  // Output: B
```

Runtime Polymorphism:

This allows a superclass reference to dynamically call subclass methods, achieving behavior that is determined at runtime rather than compile time.

Syntax:

```
class Parent {
    void show() { /* code */ }
}

class Child extends Parent {
    void show() { /* code */ }
}

Parent obj = new Child();
obj.show();  // Calls Child's method
```

Example:

```
class Parent {
    void show() {
        System.out.println("Parent");
    }
}

class Child extends Parent {
    void show() {
        System.out.println("Child");
    }
}

Parent obj = new Child();
obj.show();  // Output: Child
```

Procedure

wap to demonstrate dyanmic method dispatch

```java
class A {
   void m1() {
      System.out.println("Inside A's m1 method");
   }
}

class B extends A {
   void m1() {
      System.out.println("Inside B's m1 method");
   }
}

class C extends A {
   void m1() {
      System.out.println("Inside C's m1 method");
   }
}

public class Dispatch {
   public static void main(String[] args) {
      A a = new A();
      B b = new B();
      C c = new C();

      A ref;

      ref = a;
      ref.m1(); // Calls A's version of m1()

      ref = b;
      ref.m1(); // Calls B's version of m1()

      ref = c;
      ref.m1(); // Calls C's version of m1()
   }
}
```

Output:

```
Inside A's m1 method
Inside B's m1 method
Inside C's m1 method
```

wap to demonstrate multiple inheritance

```java
// Interface for Bank operations
interface Bank {
    float rateOfInterest();
}

// Additional interfaces for balance and deposit operations
interface Deposit {
    void deposit(float amount);
}

interface Balance {
    float getBalance();
}

// Deposit class implementing multiple interfaces
class DepositAccount implements Bank, Balance, Deposit {
    private float balance = 0.0f;

    @Override
    public float rateOfInterest() {
        return 9.15f;
    }

    @Override
    public float getBalance() {
        return balance;
    }

    @Override
    public void deposit(float amount) {
        balance += amount;
        System.out.println("Deposited: ₹" + amount);
    }
}

// Balance class implementing multiple interfaces
```

```
class BalanceAccount implements Bank, Balance, Deposit {
    private float balance = 0.0f;

    @Override
    public float rateOfInterest() {
        return 9.7f;
    }

    @Override
    public float getBalance() {
        return balance;
    }

    @Override
    public void deposit(float amount) {
        balance += amount;
        System.out.println("Deposited: ₹" + amount);
    }
}

// Test class to demonstrate multiple inheritance
public class TestInheritance {
    public static void main(String[] args) {
        Bank bank = new DepositAccount();
        System.out.println("DepositAccount ROI: " + bank.rateOfInterest());

        // Casting Bank reference to Balance and Deposit to access additional methods
        Balance balanceable = (Balance) bank;
        Deposit depositable = (Deposit) bank;

        depositable.deposit(1000);
        System.out.println("DepositAccount Balance: ₹" + balanceable.getBalance());

        bank = new BalanceAccount();
        System.out.println("BalanceAccount ROI: " + bank.rateOfInterest());

        // Casting Bank reference to Balance and Deposit to access additional methods
        balanceable = (Balance) bank;
        depositable = (Deposit) bank;

        depositable.deposit(2000);
        System.out.println("BalanceAccount Balance: ₹" + balanceable.getBalance());
    }
}
```

output

```
Deposited: ₹1000
DepositAccount Balance: ₹1000.0
BalanceAccount ROI: 9.7
Deposited: ₹2000
BalanceAccount Balance: ₹2000.0
```

conclusion: hence we applied the concept of interfaces to achieve multiple inheritance and dynamic method dispatch in Java.

-PR7BZ
