Experiment 7:

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

Theory:

Multiple Inheritance using Interfaces:

In Java, while classes cannot use multiple inheritance directly, interfaces provide a way to achieve a similar effect. An interface in Java is a reference type that can declare method signatures and constants but cannot implement methods. A class can implement multiple interfaces, thereby inheriting the abstract methods from all of them and providing concrete implementations. This allows for combining functionalities from different sources and facilitates a form of multiple inheritance by enabling a class to conform to various interfaces.

Syntax:

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

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

Example:

```
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 in Java refers to the process of resolving method calls at runtime based on the actual object's type, rather than the type of reference used to call the method. This is achieved through method overriding, where a subclass provides its own implementation of a method defined in its superclass. The Java Virtual Machine (JVM) uses the actual object's class to determine which method to execute, allowing for flexible and dynamic method resolution.

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:

Runtime polymorphism in Java allows a superclass reference to point to a subclass object, enabling the method that gets executed to be determined at runtime based on the actual object's type. This mechanism supports method overriding and allows a single reference type to call methods of multiple subclasses, providing the flexibility to execute the most specific method implementation available at runtime.

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

Demonstrate Dynamic Method Dispatch:

```
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
```

Demonstrate Multiple Inheritance:

```
// 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();
}

// DepositAccount 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);
    }
}

// BalanceAccount 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 successfully applied the concept of interfaces to achieve multiple inheritance and dynamic method dispatch in Java.