# 7.INTERFACE

Experiment no. 7

Date: 21/8/24

Roll no: B014

Aim: To apply the concept of interface to achieve multiple inheritance and dynamic memory dispatch in Java.

Theory:

- Multiple inheritance using interface:
  - An interface in Java is a blueprint of a class. It has static constants and abstract methods.
  - Multiple inheritance is a feature of an object-oriented concept, where a subclass can inherit properties of more than one parent class.
  - Multiple inheritance can be achieved through the use of interfaces.

- Dynamic memory dispatch
  - Method overriding is one of the ways in which Java supports Runtime Polymorphism. Dynamic method/ memory dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. It is also known as Run-time polymorphism.

~~Runtime Polymorphism~~

- Interface
  - An interface in Java programming is defined as an abstract type used to specify the behaviour of a class.
  - A Java interface contains static constants and abstract

methods.

- The interface in Java is a mechanism to achieve abstraction.

Syntax:

```
interface <interface-name>
{
  // declare constant fields
  // declare methods (abstract)
}
```

Procedure:

1. Program 1: Dynamic method disparch.

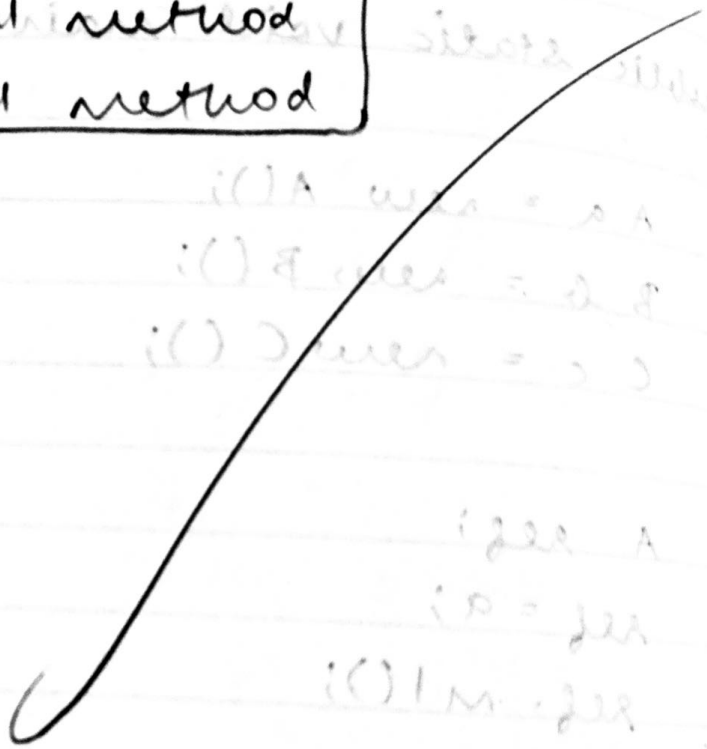
```
class A
{
  void m1()
  {
  System.out.println("Inside A's m1 method");
  }
}
class B extends A
{
  void m1()
  {
  System.out.println("Inside B's m1 method");
  }
}
class C extends A
{
  void m1()
  {
```

```
        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();
        ref = b;
        ref.M1();
        ref = c;
        ref.M1();
    }
}
```

**Output:**

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



program2:

```
import java.util.*;
interface option
{
double minbal =500;
double ROI = 9.5;
void checkbal();
void deposit();
void withdraw();
Void checkloan();
}
class Bank implements optior
 {
    double dep , with, bal, ans;
    Scanner sc = new Scanner (System.in);
    public Bank (double d, double w, double b)
      {
        this.bal = b;
        this.dep = d;
        this.with = w;
      }
    public void checkbal()
      {
      System.out.println ("Balance: " + bal);
      }
    public void deposit()
      {
      System.out.println ("enter amount: ");
       dep = sc.nextDouble();
```

```
    ans = bal + dep;
    System.out.println("balance: " + ans);
    }
    public void withdraw
    {
     System.out.println("amount to withdraw:");
    with = sc.nextDouble();
      ans = bal - with;
    if (ans < minbal)
    {
     System.out.println("Insufficient");
    }
    else
    {
    System.out.println("amount withdrawn :"
        + with + "\n Balance : " + ans);
    }
    public void checkloan()
    {
    System.out.println("ROI on loan : " + ROI);
    }
    class Main
    {
    public static void main(String[] args)
    {
      Scanner sc = new Scanner(System.in);
      int ans;
      Bank b = new Bank(0,0,6000);
    System.out.println("1.Balance " + "\n 2.Loan"
                    + "\n 3.Deposit " + "\n 4.Withdraw");
```

```
ans = sc.next
        ans = sc.nextInt();
        switch(ans)
        {
          case 1:
          b.checkbal();
          break;
          case 2:
          b.checkloan();
          break;
          case 3:
          b.deposit();
          break;
          case 4:
          b.withdraw();
          break;
          default:
          break;
        }
    }
}
```

Menu:

1. Balance
2. loan
3. deposit
4. withdraw

4.

Enter amount:
500
amount withdrawn
500.0
Balance: 5510

conclusion: Hence, we have implemented the concept of multiple inheritance using interface and dynamic method dispatch in Java.

28/8/24