EXPERIMENT NO. 10

AIM: Edit, compile, execute and test inheritance and interfaces in Java program.

THEORY:

Java and Multiple Inheritance

Multiple Inheritance is a feature of object oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with same signature in both the super classes and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

Why Java doesn't support Multiple Inheritance?

Consider the below Java code. It shows error.

```
// First Parent class
classParent1
{
   voidfun()
   {
      System.out.println("Parent1");
   }
}

// Second Parent Class
classParent2
{
   voidfun()
   {
      System.out.println("Parent2");
   }
}

// Error : Test is inheriting from multiple
// classes
classTest extendsParent1, Parent2
{
  publicstaticvoidmain(String args[])
  {
     Test t = newTest();
     t.fun();
  }
}
```

Output :

Compiler Error

From the code, we see that, on calling the method fun() using Test object will cause complications such as whether to call Parent1's fun() or Parent2's fun() method.

1. The Diamond Problem:


```
// A Grand parent class in diamond
classGrandParent
{
   voidfun()
   {
      System.out.println("Grandparent");
   }
}

// First Parent class
classParent1 extendsGrandParent
{
   voidfun()
   {
      System.out.println("Parent1");
   }
}

// Second Parent Class
classParent2 extendsGrandParent
{
   voidfun()
   {
      System.out.println("Parent2");
   }
}


// Error : Test is inheriting from multiple
// classes
```

```
classTest extendsParent1, Parent2
{
  publicstaticvoidmain(String args[])
  {
     Test t = newTest();
     t.fun();
  }
}
```

From the code, we see that: On calling the method fun() using Test object will cause complications such as whether to call Parent1's fun() or Child's fun() method.

Therefore, in order to avoid such complications Java does not support multiple inheritance of classes.

2. Simplicity – Multiple inheritance is not supported by Java using classes , handling the complexity that causes due to multiple inheritance is very complex. It creates problem during various operations like casting, constructor chaining etc and the above all reason is that there are very few scenarios on which we actually need multiple inheritance, so better to omit it for keeping the things simple and straightforward.

Interfaces in Java

Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
- A Java library example is, Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

Syntax :

```
interface<interface_name> {


   // declare constant fields

   // declare methods that abstract

   // by default.

}
```

To declare an interface, use **interface** keyword. It is used to provide total abstraction. That means all the methods in an interface are declared with an empty body and are public and all

fields are public, static and final by default. A class that implement interface must implement all the methods declared in the interface. To implement interface use **implements** keyword.

Why do we use interface ?

- It is used to achieve total abstraction.
- Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .
- It is also used to achieve loose coupling.
- Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?
  The reason is, abstract classes may contain non-final variables, whereas variables in interface are final, public and static.

```
// A simple interface
interfacePlayer
{
   finalintid = 10;
   intmove();
}
```

To implement an interface we use keyword: implement

EXPERIMENT NO. 10

AIM: Edit, compile, execute and test inheritance and interfaces in Java program.

PROGRAM:

```
// A simple Java program to demonstrate multiple
// inheritance through default methods.
interfacePI1
{
    // default method
    defaultvoidshow()
    {
        System.out.println("Default PI1");
    }
}

interfacePI2
{
    // Default method
    defaultvoidshow()
    {
        System.out.println("Default PI2");
    }
}

// Implementation class code
classTestClass implementsPI1, PI2
{
    // Overriding default show method
    publicvoidshow()
    {
        // use super keyword to call the show
        // method of PI1 interface
        PI1.super.show();

        // use super keyword to call the show
        // method of PI2 interface
        PI2.super.show();
    }

    publicstaticvoidmain(String args[])
    {
        TestClass d = newTestClass();
        d.show();
    }
}
```

Output:

```
Default PI1
Default PI2
```

PROGRAM:

```
// A simple Java program to demonstrate how diamond
// problem is handled in case of default methods

interfaceGPI
{
    // default method
    defaultvoidshow()
    {
        System.out.println("Default GPI");
    }
}

interface PI1 extends GPI { }

interface PI2 extends GPI { }

// Implementation class code
classTestClass implementsPI1, PI2
{
    publicstaticvoidmain(String args[])
    {
        TestClass d = newTestClass();
        d.show();
    }
}
```

Output:

```
Default GPI
```

PROGRAM:

```
// Java program to demonstrate working of
// interface.
importjava.io.*;

// A simple interface
interfaceIn1
{
    // public, static and final
    final int a = 10(ROLL NUMBER);

    // public and abstract
    void display();
}

// A class that implements the interface.
class Test Class implementsIn1
{
    // Implementing the capabilities of
    // interface.
    public void display()
    {
        System.out.println("YOUR NAME");
```

```
    }

    // Driver Code
    public static void main (String[] args)
    {
        TestClass t = newTestClass();
        t.display();
        System.out.println(a);
    }
}
```

Output:

```
YOUR NAME

10
```

PROGRAM:

```
importjava.io.*;

interfaceVehicle {

    // all are the abstract methods.
    voidchangeGear(inta);
    voidspeedUp(inta);
    voidapplyBrakes(inta);
}

classBicycle implementsVehicle{

    intspeed;
    intgear;

     // to change gear
    @Override
    publicvoidchangeGear(intnewGear){

        gear = newGear;
    }

    // to increase speed
    @Override
    publicvoidspeedUp(intincrement){

        speed = speed + increment;
    }

    // to decrease speed
    @Override
    publicvoidapplyBrakes(intdecrement){

        speed = speed - decrement;
    }
```

```
    publicvoidprintStates() {
         System.out.println("speed: "+ speed
              + " gear: "+ gear);
    }
}

classBike implementsVehicle {

    intspeed;
    intgear;

    // to change gear
    @Override
    publicvoidchangeGear(intnewGear){

        gear = newGear;
    }

    // to increase speed
    @Override
    publicvoidspeedUp(intincrement){

        speed = speed + increment;
    }

    // to decrease speed
    @Override
    publicvoidapplyBrakes(intdecrement){

        speed = speed - decrement;
    }

    publicvoidprintStates() {
         System.out.println("speed: "+ speed
             + " gear: "+ gear);
    }

}
classGFG {

    publicstaticvoidmain (String[] args) {

        // creating an inatance of Bicycle
        // doing some operations
        Bicycle bicycle = newBicycle();
        bicycle.changeGear(2);
        bicycle.speedUp(3);
        bicycle.applyBrakes(1);

        System.out.println("Bicycle present state :");
        bicycle.printStates();

        // creating instance of the bike.
        Bike bike = newBike();
```

```
        bike.changeGear(1);
        bike.speedUp(4);
        bike.applyBrakes(3);

        System.out.println("Bike present state :");
        bike.printStates();
    }
}
```

Output;

```
Bicycle present state :

speed: 2 gear: 2

Bike present state :

speed: 1 gear: 1
```

CONCLUSION: Thus we have successfully studied about editing, compiling, executing and testing inheritance and interfaces in Java program.