Name: Gaurav Kishor Patil
Roll no:54 DIV: 2
Batch: C

Experiment No. 9
Single Inheritance
Date of Performance:29/09/23
Date of Submission: 5/10/23

Aim:- To implement the concept of single inheritance.

Objective:- To implement the concept of single inheritance

Theory:-

Single inheritance can be defined as a derived class to inherit the basic methods (data members and variables) and behaviour from a superclass. It's a basic is-a relationship concept exists here. Basically, java only uses a single inheritance as a subclass cannot extend more superclass.

Inheritance is the basic properties of object-oriented programming. Inheritance tends to make use of the properties of a class object into another object. Java uses inheritance for the purpose of code-reusability to reduce time by then enhancing reliability and to achieve run time polymorphism. As the codes are reused it makes less development cost and maintenance. Java has different types of inheritance namely single inheritance, multilevel, multiple, hybrid. In this article, we shall go through on basic understanding of single inheritance concept briefly in java with a programming example. Here we shall have a complete implementation in java.

Syntax:

The general syntax for this is given below. The inheritance concepts use the keyword 'extend' to inherit a specific class. Here you will learn how to make use of extending keyword to derive a class. An extend keyword is declared after the class name followed by another class name. Syntax is,

```
class base class
{…. methods
}
class derived class name extends base class
{
methods … along with this additional feature
}
```

Java uses a keyword 'extends' to make a new class that is derived from the existing class. The inherited class is termed as a base class or superclass, and the newly created class is called derived or subclass. The class which gives data members and methods known as the base class and the class which takes the methods is known as child class.

Code:-

```
import java.util.*;

class Airlines {
   private int ID_number;
   private String name;
   private String address;
   private String contact;
   private String email;
   public double frequentFlyerPoints;

   public Airlines(int ID_number, String name, String address, String contact, String email) {
      this.ID_number = ID_number;
      this.name = name;
      this.address = address;
      this.contact = contact;
```

```
        this.email = email;
        this.frequentFlyerPoints = 0;
    }

    public void Airlines_member() {
        System.out.println("Member Number: " + ID_number);
        System.out.println("Name: " + name);
        System.out.println("Address: " + address);
        System.out.println("Mobile Number: " + contact);
        System.out.println("Initial FrequentFlyer Points: " + frequentFlyerPoints);
    }
}

class Domestic_Flights extends Airlines {
    public Domestic_Flights(int ID_number, String name, String address, String contact, String email) {
        super(ID_number, name, address, contact, email);
    }

    public void displayAvailableFlights() {
        System.out.println("Available Domestic Flights: Flight 1, Flight 2, Flight 3, ...");
    }

    public double calculateCost(int numberOfSeats) {
        double seatCost = 1000;
        double c = seatCost / 100;
        frequentFlyerPoints = c;
        System.out.println("New Flyyer Points:" + frequentFlyerPoints);
        return seatCost * numberOfSeats;
    }
}

class International_Flights extends Airlines {
    public International_Flights(int ID_number, String name, String address, String contact, String email) {
        super(ID_number, name, address, contact, email);
    }

    public void displayAvailableFlights() {
        System.out.println("Available International Flights: Flight A, Flight B, Flight C, ...");
    }

    public double calculateCost(int numberOfSeats) {
        double seatCost = 2000;
        double c = seatCost / 100;
        frequentFlyerPoints = c;
        System.out.println("New Flyyer Points:" + frequentFlyerPoints);
        return seatCost * numberOfSeats;
    }
}

public class AirlinesInheritance {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        Domestic_Flights domesticFlight = new Domestic_Flights(101, "MR x", "vcet vasai", "9876543210",
                "xmenorigins@example.com");
        domesticFlight.Airlines_member();
        domesticFlight.displayAvailableFlights();
        int numberOfSeats = 3;
        double cost = domesticFlight.calculateCost(numberOfSeats);
        System.out.println("Cost for " + numberOfSeats + " seats: Rs. " + cost);

        International_Flights internationalFlight = new International_Flights(201, "Jada", "456 Elm St",
                "9876543211", "jada@example.com");
        internationalFlight.Airlines_member();
        internationalFlight.displayAvailableFlights();
        numberOfSeats = 2;
        cost = internationalFlight.calculateCost(numberOfSeats);
        System.out.println("Cost for " + numberOfSeats + " seats: Rs. " + cost);
        scanner.close();
    }
}
```

```
Name: MR x
Address: vcet vasai
Mobile Number: 9876543210
Initial FrequentFlyer Points: 0.0
Available Domestic Flights: Flight 1, Flight 2, Flight 3, ...
New Flyyer Points:10.0
Cost for 3 seats: Rs. 3000.0
Member Number: 201
Name: Jada
Address: 456 Elm St
Mobile Number: 9876543211
Initial FrequentFlyer Points: 0.0
Available International Flights: Flight A, Flight B, Flight C, ...
New Flyyer Points:20.0
Cost for 2 seats: Rs. 4000.0
```

Conclusion:-

Single inheritance in Java is a fundamental concept where a class can inherit properties (fields and methods) from only one superclass. This is done using the 'extends' keyword. The main advantage of single inheritance is that it provides a clear and simple hierarchical structure that is easy to understand and maintain. It also promotes reusability of code, as common properties can be defined in the superclass and inherited by subclasses. However, it's important to note that while a class can only inherit from one superclass, it can have multiple subclasses, thus forming a tree-like structure. This approach to inheritance helps to prevent ambiguity and enhances the robustness of the Java programming language.