**Experiment: 6**

PART A

(PART A: TO BE REFERRED BY STUDENTS)

**Aim: To study Multiple inheritance and interfaces, super and final keyword.**

**Learning Outcomes: Learner would be able to Interfaces in JAVA**

1. Understand the multiple inheritance concept.
2. Understanding the concept of Interface in JAVA.
3. Understands the concepts of super keyword.
4. Understanding the final keyword.

**Theory:**

**Multiple Inheritance**

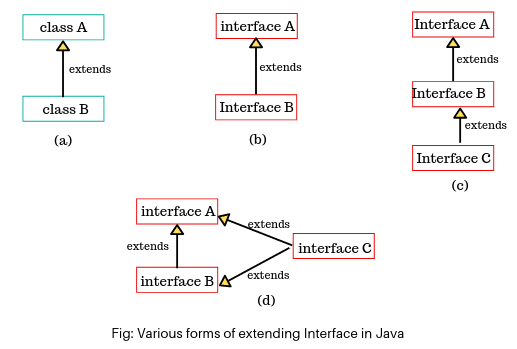
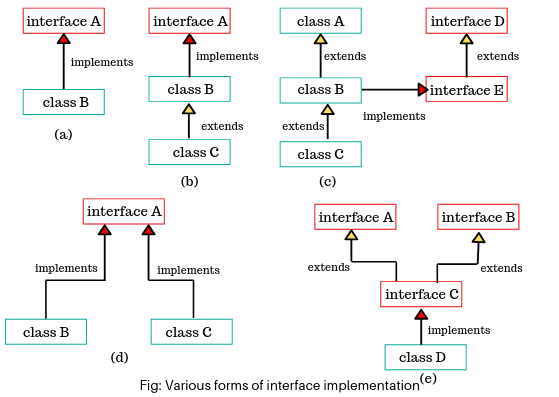
In Java, multiple inheritance refers to the ability of a class to inherit from more than one class. However, Java does not support multiple inheritance with classes directly to avoid complexity and ambiguity that might arise, such as the diamond problem (which occurs when a class inherits from two classes that have the same method).

Multiple inheritance can lead to complications if two classes provide methods with the same signature, as the subclass would not know which method to inherit. This is known as the diamond problem. Although Java does not support multiple inheritance via classes, it allows multiple inheritance through interfaces. A class can implement multiple interfaces, allowing it to inherit behavior from more than one source.

**Interface**

* An interface in Java is syntactically similar to a class but can have only abstract methods declaration and constants as members.
* Every interface in Java is abstract by default. So, it is not compulsory to write abstract keyword with an interface.
* Once an interface is defined, we can create any number of separate classes and can provide their own implementation for all the abstract methods defined by an interface.
* A class that implements an interface is called implementation class. A class can implement any number of interfaces in Java. Every implementation class can have its own implementation for abstract methods specified in the interface.

## Interface Helps Achieve Multiple Inheritance and provides the full abstraction. Various forms of interfaces

**** ****

**Syntax of Interface:**

interface InterfaceName {

// constant declarations (optional)

// method declarations (abstract by default)

// Example of abstract method

void method1();

// Example of default method (from Java 8 onwards)

default void method2() {

System.out.println("This is a default method");

}

// Example of static method (from Java 8 onwards)

static void staticMethod() {

System.out.println("This is a static method in the interface");

}

}

***super* Keyword in Java**

In Java, the super keyword is used to refer to the immediate parent class of the current object. It serves several important purposes, primarily when dealing with inheritance. It can be used to:

* Access parent class constructors.
* Access parent class methods.
* Access parent class fields.

***final* keyword in Java**

The final keyword in Java is used to define constants, prevent method overriding, and prevent inheritance. It is a very important part of the Java programming language and is often used to ensure immutability or restrict certain features in a program.

For the following Problem Statements write programs **using classes: concrete and abstract, objects, methods: abstract and non-abstract and interface.**

|  |  |
| --- | --- |
| **Program No.** | **Experiment** |
| 1. | Define an interface for shapes and implement it in classes like Circle and Rectangle.  [Hint: use the abstract class, interface, constructor] |
| 2. | WAP to create an abstract class called MobileRechargePlan with attributes like planName, planAmount, and validityDays alongside abstract methods displayPlanDetails() and calculateRechargeAmount(double talkTime). Further, create two concrete classes as prepaidRecharge and PostpaidRecharge extending the MobileRechargePlan class, provide methods for display the plan details, and calculate the recharge amount methods. Instantiate both the classes and display their respective plan details along with calculate recharge amounts.  [Hint: use the abstract class, constructor, super and this keyword] |
| 3. | WAP to create an abstract class called Payment with attributes as cardnumber,cardholder,amount and abstract methods as processPayment() and displayPaymentDetails(). Create concrete classes as creditCardPayment and DebitCardPayment acquiring the properties of the Payment class.The creditCardPayment class consist the attribute as expiryDate and implement processPayment() that displays the “credit Card Payment Processed” and displayPaymentDetails() prints details includes the cardNumber,cardholder,expiryDate and Amount. The DebitCardPayment class consist the attribute as pin and implement processPayment() that displays the “Debit Card Payment Processed” and displayPaymentDetails() prints details includes the cardNumber,cardholder,pin and Amount. Further create the objects for the both the concrete classes and invoke the processPayment() and displayPaymentDetails.  [Hint: use the abstract class, constructor and super and this keyword] |
| 4. | Demonstrate an interface is extended with another interface for following snippet code, generate the output by complete it.  // parent interface  interface Line { void drawLine(); }  // child interface Rectangle  interface Rectangle extends Line { void drawRectangle(); }  // child interface Triangle  interface Triangle extends Line { void drawTriangle(); }  // class Shape1 implements interfaces Rectangle, Triangle  class Shape1 implements Rectangle, Triangle {  void drawLine() { // display the message as” line drawn for shape1” }  void drawRectangle() { // display the message as” rectangle drawn for shape1” }  void drawTriangle() {// display the message as” triangle drawn for shape1” }  }  // class Shape2 implements interfaces Rectangle, Triangle  class Shape2 implements Rectangle, Triangle {  void drawLine() { // display the message as” line drawn for shape2” }  void drawRectangle() { // display the message as” line drawn for shape2” }  void drawTriangle() {// display the message as” triangle drawn for shape2” }  } |
| 5. | A. Created a class Animal and its subclass Dog. In the Animal class, defined a final method named display () and override this method by defining a method with the same name in the subclass Dog.  B. Similarly, create a final class as Animal and its subclass Dog. In the Animal class, defined a method named display () and override this method by defining a method with the same name in the subclass Dog.  **Implement the above A and B, Write your observations.** |
| 6 | Design a **Smart Device Control System** using **interfaces** to achieve **multiple inheritance** in Java:   1. **Interface WiFiEnabled** (provides WiFi functionality)  * connectToWiFi(String network) → Connects to a WiFi network * disconnectWiFi() → Disconnects from WiFi  1. **Interface BluetoothEnabled** (provides Bluetooth functionality)  * pairDevice(String deviceName) → Pairs with a Bluetooth device * unpairDevice() → Unpairs from the Bluetooth device  1. **Class SmartSpeaker** (implements both WiFiEnabled and BluetoothEnabled)  * Implements WiFi and Bluetooth methods * Additional method playMusic(String songName)  1. **Class SmartDeviceDemo** (Main class)  * Creates a SmartSpeaker object and performs various operations |

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the portal at the end of the practical. The filename should be **JAVA\_batch\_rollno\_experimentno Example: JAVA\_A1\_A001\_P1**

|  |  |
| --- | --- |
| **Roll No.: C126** | **Name: Rushabh Abhay Shah** |
| **Prog/Yr/Sem: BTI/4th/8th** | **Batch: 2021-2027** |
| **Date of Experiment: 22/02/2025** | **Date of Submission: 22/02/2025** |

**Program No. 1**

a. Input Statement

b. Code

public class q1{

public static void main(String[]args)

{

Circle c = new Circle(6);

c.display();

c.area();

c.perimeter();

}

}

interface Shape{

void area();

void perimeter();

}

abstract class Ashape implements Shape{

String name;

public Ashape(String name)

{

this.name = name;

}

public void display(){

System.out.println("Shape name is: "+name);

}

}

class Circle extends Ashape{

double radius;

public Circle(double radius)

{

super("Circle");

this.radius = radius;

}

@Override

public void area(){

System.out.println("Area of circle is: "+(3.14\*radius\*radius));

}

@Override

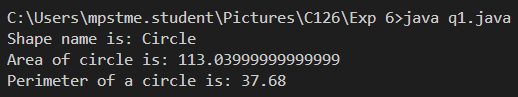
public void perimeter(){

System.out.println("Perimeter of a circle is: "+(2\*3.14\*radius));

}

}

c. Output

****

**Program No. 2**

a. Input Statement

b. Code

public class q2 {

public static void main(String[] args) {

MobileRechargePlan prepaidPlan = new PrepaidRecharge("Prepaid Plan 1", 300, 30);

MobileRechargePlan postpaidPlan = new PostpaidRecharge("Postpaid Plan 1", 500, 30);

System.out.println("Prepaid Plan Details:");

prepaidPlan.displayPlanDetails();

System.out.println("Recharge Amount for 100 minutes talk time: Rs. " + prepaidPlan.calculateRechargeAmount(100));

System.out.println("\nPostpaid Plan Details:");

postpaidPlan.displayPlanDetails();

System.out.println("Recharge Amount for 100 minutes talk time: Rs. " + postpaidPlan.calculateRechargeAmount(100));

}

}

abstract class MobileRechargePlan {

String planName;

double planAmount;

int validityDays;

public MobileRechargePlan(String planName, double planAmount, int validityDays) {

this.planName = planName;

this.planAmount = planAmount;

this.validityDays = validityDays;

}

abstract void displayPlanDetails();

abstract double calculateRechargeAmount(double talkTime);

}

class PrepaidRecharge extends MobileRechargePlan {

public PrepaidRecharge(String planName, double planAmount, int validityDays) {

super(planName, planAmount, validityDays);

}

@Override

void displayPlanDetails() {

System.out.println("Prepaid Plan: " + planName);

System.out.println("Amount: Rs. " + planAmount);

System.out.println("Validity: " + validityDays + " days");

}

@Override

double calculateRechargeAmount(double talkTime) {

return planAmount + (talkTime \* 0.5);

}

}

class PostpaidRecharge extends MobileRechargePlan {

public PostpaidRecharge(String planName, double planAmount, int validityDays) {

super(planName, planAmount, validityDays);

}

@Override

void displayPlanDetails() {

System.out.println("Postpaid Plan: " + planName);

System.out.println("Amount: Rs. " + planAmount);

System.out.println("Validity: " + validityDays + " days");

}

@Override

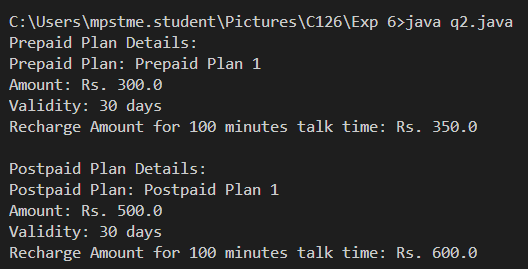
double calculateRechargeAmount(double talkTime) {

return planAmount + (talkTime \* 1.0);

}

}

c. Output



**Program No. 3**

a. Input Statement

b. Code

public class q3 {

public static void main(String[] args) {

Payment creditCardPayment = new CreditCardPayment("1234-5678-9876-5432", "John Doe", 5000, "12/25");

Payment debitCardPayment = new DebitCardPayment("4321-8765-6789-1234", "Jane Smith", 3000, "1234");

creditCardPayment.processPayment();

creditCardPayment.displayPaymentDetails();

System.out.println();

debitCardPayment.processPayment();

debitCardPayment.displayPaymentDetails();

}

}

abstract class Payment {

String cardNumber;

String cardHolder;

double amount;

public Payment(String cardNumber, String cardHolder, double amount) {

this.cardNumber = cardNumber;

this.cardHolder = cardHolder;

this.amount = amount;

}

abstract void processPayment();

abstract void displayPaymentDetails();

}

class CreditCardPayment extends Payment {

String expiryDate;

public CreditCardPayment(String cardNumber, String cardHolder, double amount, String expiryDate) {

super(cardNumber, cardHolder, amount);

this.expiryDate = expiryDate;

}

@Override

void processPayment() {

System.out.println("Credit Card Payment Processed");

}

@Override

void displayPaymentDetails() {

System.out.println("Card Number: " + cardNumber);

System.out.println("Card Holder: " + cardHolder);

System.out.println("Expiry Date: " + expiryDate);

System.out.println("Amount: Rs. " + amount);

}

}

class DebitCardPayment extends Payment {

String pin;

public DebitCardPayment(String cardNumber, String cardHolder, double amount, String pin) {

super(cardNumber, cardHolder, amount);

this.pin = pin;

}

@Override

void processPayment() {

System.out.println("Debit Card Payment Processed");

}

@Override

void displayPaymentDetails() {

System.out.println("Card Number: " + cardNumber);

System.out.println("Card Holder: " + cardHolder);

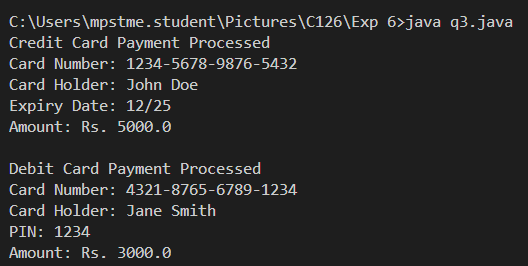
System.out.println("PIN: " + pin);

System.out.println("Amount: Rs. " + amount);

}

}

c. Output



**Program No. 4**

a. Input Statement

b. Code

public class q4 {

public static void main(String[] args) {

Shape1 shape1 = new Shape1();

Shape2 shape2 = new Shape2();

shape1.drawLine();

shape1.drawRectangle();

shape1.drawTriangle();

System.out.println();

shape2.drawLine();

shape2.drawRectangle();

shape2.drawTriangle();

}

}

interface Line {

void drawLine();

}

interface Rectangle extends Line {

void drawRectangle();

}

interface Triangle extends Line {

void drawTriangle();

}

class Shape1 implements Rectangle, Triangle {

public void drawLine() {

System.out.println("Line drawn for Shape1");

}

public void drawRectangle() {

System.out.println("Rectangle drawn for Shape1");

}

public void drawTriangle() {

System.out.println("Triangle drawn for Shape1");

}

}

class Shape2 implements Rectangle, Triangle {

public void drawLine() {

System.out.println("Line drawn for Shape2");

}

public void drawRectangle() {

System.out.println("Rectangle drawn for Shape2");

}

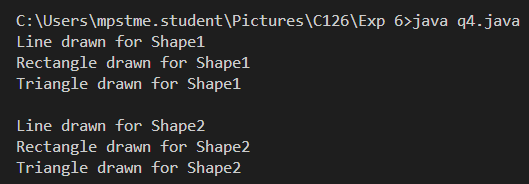
public void drawTriangle() {

System.out.println("Triangle drawn for Shape2");

}

}

c. Output



**Program No. 5a**

a. Input Statement

b. Code

public class q5a {

public static void main(String[] args) {

Dog dog = new Dog();

dog.display();

}

}

class Animal {

final void display() {

System.out.println("Display method in Animal class");

}

}

class Dog extends Animal {

void display() {

System.out.println("Display method in Dog class");

}

}

c. Output

Part A: When a method is declared as final in the parent class (Animal), it cannot be overridden by any subclass (Dog). In this case, the code in Part A would not compile because the display() method in Dog tries to override a final method from Animal, which is not allowed.

**Program No. 5b**

a. Input Statement

b. Code

final class Animal {

void display() {

System.out.println("Display method in Animal class");

}

}

class Dog extends Animal {

void display() {

System.out.println("Display method in Dog class");

}

}

public class Main {

public static void main(String[] args) {

Dog dog = new Dog();

dog.display();

}

}

c. Output

Part B: When the class Animal is declared as final, it cannot be subclassed. Therefore, in Part B, the code would not compile because the class Dog tries to extend the final class Animal, which is not allowed.

**Program No. 6**

a. Input Statement

b. Code

interface WiFiEnabled {

void connectToWiFi(String network);

void disconnectWiFi();

}

interface BluetoothEnabled {

void pairDevice(String deviceName);

void unpairDevice();

}

class SmartSpeaker implements WiFiEnabled, BluetoothEnabled {

public void connectToWiFi(String network) {

System.out.println("Connected to WiFi network: " + network);

}

public void disconnectWiFi() {

System.out.println("Disconnected from WiFi");

}

public void pairDevice(String deviceName) {

System.out.println("Paired with Bluetooth device: " + deviceName);

}

public void unpairDevice() {

System.out.println("Unpaired from Bluetooth device");

}

public void playMusic(String songName) {

System.out.println("Playing music: " + songName);

}

}

public class SmartDeviceDemo {

public static void main(String[] args) {

SmartSpeaker speaker = new SmartSpeaker();

speaker.connectToWiFi("HomeNetwork");

speaker.pairDevice("Phone");

speaker.playMusic("Song A");

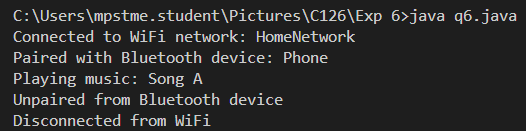
speaker.unpairDevice();

speaker.disconnectWiFi();

}

}

c. Output

****

**Conclusion (Learning Outcomes)**

Learnt about Inheritance, difference between abstract and interface.