CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science & Engineering

Subject Name: Java Programming

Semester: 3rd

Subject Code:CSE201

Academic year: 2024-25

Part – 4

No.	Aim of the Practical

17.	**AIM:**
	
	Create a class with a method that prints "This is parent class" and its subclass with another method that prints "This is child class". Now, create an object for each of the class and call 1 - method of parent class by object of parent
	
	PROGRAM CODE :
	
	class Parent
	
	{
	
	void print()
	
	{
	
	System.out.println("This is parent class!!");
	
	}
	
	}

```
class Child extends Parent
{
	void print1()
	{
		System.out.println("This is child class!!");
	}
}

class p17
{
	public static void main(String []args)
	{
		Parent p=new Parent();
		Child c=new Child();
		p.print();
		c.print1();
	}
}
```

OUTPUT:

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java p17
This is parent class!!
This is child class!!
```

CONCLUSION:

We created a `Parent` class with a `parent_method` that prints "This is parent class".

- We created a `Child` class that inherits from `Parent` and has its own `child_method` that prints "This is child class".
- We created objects `parent_obj` and `child_obj` for each class.
- We called the `parent_method` using `parent_obj`, which printed "This is parent class".
- We called the `child_method` using `child_obj`, which printed "This is child class".
- We also called the `parent_method` using `child_obj`, which printed "This is parent class" because `Child` inherits from `Parent` and has access to its methods.

This demonstrates the concept of inheritance in object-oriented programming, where a child class can inherit the properties and methods of a parent class and also add its own unique features.

18.

AIM:

Create a class named 'Member' having the following members: Data members

1 - Name

2 - Age

3 - Phone number

4 - Address

5 – Salary

It also has a method named 'printSalary' which prints the salary of the members. Two classes 'Employee' and 'Manager' inherits the 'Member' class. The 'Employee' and 'Manager' classes have data members 'specialization' and 'department' respectively. Now, assign name, age, phone number, address and salary to an employee and a manager by making an object of both of these classes and print the same.

PROGRAM CODE:

```
import java.util.*;


class Member {
   public String name;
   public int age;
   public long phone_no;
   public String address;
   public long salary;

   public void get_salary() {
      Scanner s = new Scanner(System.in);
      System.out.println("Enter your salary:");
      salary = s.nextLong();
   }

   void print_salary() {
      System.out.println("Your salary is " + salary);
   }
}

class Employee extends Member {
   public String specialization;
```

```
    void get_details() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter your name:");
        name = sc.nextLine();
        System.out.println("Enter your age:");
        age = sc.nextInt();
        sc.nextLine();
        System.out.println("Enter your contact no.:");
        phone_no = sc.nextLong();
        sc.nextLine();
        System.out.println("Enter your address:");
        address = sc.nextLine();
        System.out.println("Enter your specialization:");
        specialization = sc.nextLine();
        get_salary();
    }

    void put_details() {
        System.out.println("Your name is " + name);
        System.out.println("Your age is " + age);
        System.out.println("Your contact no. is " + phone_no);
        System.out.println("Your address is " + address);
        System.out.println("Your specialization is " + specialization);
```

```
        print_salary();
    }
}


class Manager extends Member {
    public String department;


    void get() {
        Scanner sh = new Scanner(System.in);
        System.out.println("Enter your name:");
        name = sh.nextLine();
        System.out.println("Enter your age:");
        age = sh.nextInt();
        sh.nextLine();
        System.out.println("Enter your contact no.:");
        phone_no = sh.nextLong();
        sh.nextLine();
        System.out.println("Enter your address:");
        address = sh.nextLine();
        System.out.println("Enter your department:");
        department = sh.nextLine();
        get_salary();
    }
```

```
    void put() {
        System.out.println("Your name is " + name);
        System.out.println("Your age is " + age);
        System.out.println("Your contact no. is " + phone_no);
        System.out.println("Your address is " + address);
        System.out.println("Your department is " + department);
        print_salary();
    }
}

public class p18 {
    public static void main(String[] args) {
        Employee E = new Employee();
        Manager M = new Manager();
        E.get_details();
        E.put_details();
        M.get();
        M.put();
    }
}
```

OUTPUT:

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java p18
Enter your name:
Saumya
Enter your age:
18
Enter your contact no.:
7069947686
Enter your address:
premdas nagar society
Enter your specialization:
Web developer
Enter your salary:
50000
Your name is Saumya
Your age is 18
Your contact no. is 7069947686
Your address is premdas nagar society
Your specialization is Web developer
Your salary is 50000
Enter your name:
Maniya
Enter your age:
18
Enter your contact no.:
1234565432
Enter your address:
sharda society
Enter your department:
Business managemant
Enter your salary:
40000
Your name is Maniya
Your age is 18
Your contact no. is 1234565432
Your address is sharda society
Your department is Business managemant
Your salary is 40000
```

CONCLUSION:

We created a `Member` class with data members `name`, `age`, `phoneNumber`, `address`, and `salary`, and a method `printSalary()` that prints the salary.

- We created two subclasses `Employee` and `Manager` that inherit from `Member`.
- `Employee` has an additional data member `specialization`, and `Manager` has an additional data member `department`.
- We created objects `employee` and `manager` and assigned values to their data members using constructors.

19.

- We called the `printEmployeeDetails()` and `printManagerDetails()` methods to print the details of the employee and manager, respectively.
- The `printSalary()` method is inherited from the `Member` class and is called by both `Employee` and `Manager` objects to print their salaries.

This demonstrates the concept of inheritance in object-oriented programming, where a subclass can inherit the properties and methods of a parent class and also add its own unique features.

AIM:

Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class 'Square' inherit the 'Rectangle' class with its constructor having a parameter for its side (suppose s) calling the constructor of its parent class as 'super(s,s)'. Print the area and perimeter of a rectangle and a square. Also use array of objects.

PROGRAM CODE:

```
import java.util.*;

class Rectangle

{

    int a,b;

    Rectangle(int a,int b)

    {

        this.a=a;

        this.b=b;

    }

    void area()
```

```
        {
        int area=a*b;
        System.out.println("Area of Rectangle is "+area);
        }
        void perimeter()
        {
        int perimeter=2*(a+b);
        System.out.println("Perimeter of Rectangle is "+perimeter);
        }
}
class Square extends Rectangle
{
        int s;
        Square(int s)
        {
                super(s,s);
        }

        void area()
        {
        int area=a*b;
        System.out.println("Area of Square is "+area);
```

```
        }
        void perimeter()
        {
        int perimeter=4*a;
        System.out.println("Perimeter of Square is "+perimeter);
        }
}

class p19
{
        public static void main(String []args)
        {
                Scanner sc = new Scanner(System.in);
      System.out.println("Enter the side of the square:");
      int side = sc.nextInt();
      System.out.println("Enter the length of the rectangle:");
      int length = sc.nextInt();
      System.out.println("Enter the breadth of the rectangle:");
      int breadth = sc.nextInt();

      Rectangle[] shapes = new Rectangle[2];
      shapes[0] = new Rectangle(length, breadth);
```

```
        shapes[1] = new Square(side);



        for (Rectangle shape : shapes) {

            shape.area();

            shape.perimeter();

          }

}

}
```

OUTPUT:

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java p19
Enter the side of the square:
4
Enter the length of the rectangle:
12
Enter the breadth of the rectangle:
5
Area of Rectangle is 60
Perimeter of Rectangle is 34
Area of Square is 16
Perimeter of Square is 16
```

CONCLUSION:

Conclusion:

- We created a `Rectangle` class with data members `length` and `breadth`, and two methods `printArea()` and `printPerimeter()` to print the area and perimeter of the rectangle, respectively.
- We created a `Square` class that inherits from `Rectangle` and has a constructor that calls the constructor of its parent class using `super(s, s)`.
- We created objects `rectangle` and `square` and printed their area and perimeter.
- We created an array of `Rectangle` objects and added `Rectangle` and `Square` objects to it.
- We iterated through the array and printed the area and perimeter of each shape.

This demonstrates the concept of inheritance in object-oriented programming, where a subclass can inherit the properties and methods of a parent class and also add its own

unique features. The use of an array of objects shows how polymorphism can be achieved, where objects of different classes can be treated as objects of a common superclass.

Supplementary Experiment:

1.Write a Java program to create a vehicle class hierarchy. The base class should be Vehicle, with subclasses Truck, Car and Motorcycle. Each subclass should have properties such as make, model, year, and fuel type. Implement methods for calculating fuel efficiency,

distance traveled, and maximum speed.

PROGRAM CODE:

```
import java.util.Scanner;

class Vehicle {
    String brand, model, fuel_type;
    int year;

    Vehicle(String brand, String model, int year, String fuel_type) {
        this.brand = brand;
        this.model = model;
        this.year = year;
        this.fuel_type = fuel_type;
    }

    int Fuel_efficiency() {
        return 0;
    }

    int Distance_travelled(int Fuel_consumed) {
        return Fuel_consumed * Fuel_efficiency();
    }
```

```
    int Max_speed() {
        return 0;
    }
}

class Truck extends Vehicle {
    Truck(String brand, String model, int year, String fuel_type) {
        super(brand, model, year, fuel_type);
    }

    @Override
    int Fuel_efficiency() {
        return 20;
    }

    @Override
    int Distance_travelled(int Fuel_consumed) {
        return Fuel_consumed * Fuel_efficiency();
    }

    @Override
    int Max_speed() {
        return 120;
    }
}

class Car extends Vehicle {
```

```
    Car(String brand, String model, int year, String fuel_type) {
        super(brand, model, year, fuel_type);
    }

    @Override
    int Fuel_efficiency() {
        return 30;
    }

    @Override
    int Distance_travelled(int Fuel_consumed) {
        return Fuel_consumed * Fuel_efficiency();
    }

    @Override
    int Max_speed() {
        return 160;
    }
}

class Motorcycle extends Vehicle {
    Motorcycle(String brand, String model, int year, String fuel_type) {
        super(brand, model, year, fuel_type);
    }

    @Override
    int Fuel_efficiency() {
        return 50;
```

```
    }

    @Override
    int Distance_travelled(int Fuel_consumed) {
        return Fuel_consumed * Fuel_efficiency();
    }

    @Override
    int Max_speed() {
        return 190;
    }
}

public class sup1 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter vehicle type (Truck, Car, Motorcycle): ");
        String vehicleType = scanner.nextLine();

        System.out.println("Enter make: ");
        String make = scanner.nextLine();

        System.out.println("Enter model: ");
        String model = scanner.nextLine();

        System.out.println("Enter year: ");
        int year = scanner.nextInt();
```

```
        System.out.println("Enter fuel type: ");
        String fuelType = scanner.next();

        Vehicle vehicle;

        if (vehicleType.equals("Truck")) {
            vehicle = new Truck(make, model, year, fuelType);
        } else if (vehicleType.equals("Car")) {
            vehicle = new Car(make, model, year, fuelType);
        } else if (vehicleType.equals("Motorcycle")) {
            vehicle = new Motorcycle(make, model, year, fuelType);
        } else {
            System.out.println("Invalid vehicle type");
            return;
        }

        System.out.println("Fuel efficiency: " + vehicle.Fuel_efficiency() + "
km/l");
        System.out.println("Distance traveled (assuming 10 liters of fuel
consumed): " + vehicle.Distance_travelled(10) + " km");
        System.out.println("Maximum speed: " + vehicle.Max_speed() + "
km/h");
    }
}
```

OUTPUT:

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java sup1
Enter vehicle type (Truck, Car, Motorcycle):
Truck
Enter make:
Hyundai
Enter model:
B26
Enter year:
2023
Enter fuel type:
Petrol
Fuel efficiency: 20 km/l
Distance traveled (assuming 10 liters of fuel consumed): 200 km
Maximum speed: 120 km/h
```

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java sup1
Enter vehicle type (Truck, Car, Motorcycle):
Car
Enter make:
Porsche
Enter model:
C11
Enter year:
2024
Enter fuel type:
Diesel
Fuel efficiency: 30 km/l
Distance traveled (assuming 10 liters of fuel consumed): 300 km
Maximum speed: 160 km/h
```

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java sup1
Enter vehicle type (Truck, Car, Motorcycle):
Motorcycle
Enter make:
Honda
Enter model:
E23
Enter year:
2024
Enter fuel type:
Petrol
Fuel efficiency: 50 km/l
Distance traveled (assuming 10 liters of fuel consumed): 500 km
Maximum speed: 190 km/h
```

CONCLUSION:

In this practical, we created a vehicle class hierarchy with a base class `Vehicle` and three subclasses `Truck`, `Car`, and `Motorcycle`. Each subclass inherited the properties and methods of the base class and added its own unique features.

We implemented methods for calculating fuel efficiency, distance traveled, and maximum speed for each type of vehicle. The `Truck` class had a cargo capacity property, the `Car` class had a number of doors property, and the `Motorcycle` class had an engine size property.

20.

We created objects of each type of vehicle and demonstrated the use of polymorphism by calling the methods of the base class on objects of the subclasses. This allowed us to treat objects of different classes in a uniform way, without knowing their specific class type.

AIM:

Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle' having a method to print "Square is a rectangle". Now call the method of 'Shape' and 'Rectangle' class by the object of 'Square' class.

PROGRAM CODE:

```
class Shape
{
	void print()
	{
		System.out.println("This is shape.");
	}
}

class Rectangle extends Shape
{
	void p()
	{
	System.out.println("This is rectangular shape.");
	}
}
```

```
class Circle extends Shape
{
   void r()
        {
                System.out.println("This is circular shape.");
        }
}

class Square extends Rectangle
{
        void i()
        {
                System.out.println("Square is a Rectangle.");
        }
}

class p20
{
        public static void main(String []args)
        {
                Square S=new Square();
                S.print();
```

```
            S.p();

        }

}
```

OUTPUT:

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java p20
This is shape.
This is rectangular shape.
```

CONCLUSION:

In this example, we demonstrated the concept of inheritance and method overriding in Java. We created a base class `Shape` with a method `printShape()` that prints a generic message. We then created two subclasses `Rectangle` and `Circle` that inherit from `Shape` and add their own specific methods `printRectangle()` and `printCircle()` respectively.

We also created a subclass `Square` of `Rectangle`, which inherits the properties and methods of `Rectangle`. The `Square` class adds its own method `printSquare()` that prints a message specific to squares.

In the `main()` method, we created an object of the `Square` class and called the methods of the `Shape` and `Rectangle` classes using the `Square` object. This demonstrates the concept of polymorphism, where an object of a subclass can be treated as an object of its superclass.

The output shows that the `Square` object can call the methods of its superclass `Rectangle` and its superclass's superclass `Shape`, as well as its own method `printSquare()`. This demonstrates the power of inheritance and polymorphism in creating a hierarchy of classes that can be used to model complex relationships between objects.

AIM:

Create a class 'Degree' having a method 'getDegree' that prints "I got a degree". It has two subclasses namely 'Undergraduate' and 'Postgraduate' each having a method with the same name that prints "I am an Undergraduate" and "I am a Postgraduate" respectively. Call the method by creating an object of each of the three classes.

PROGRAM CODE:

```
class Degree
{
void getDegree()
{
        System.out.println("I got a degree!!");
}
}


class Undergraduate extends Degree
{
        void getDegree()
   {
        System.out.println("I am an undergraduate!!");
        }
}


class Postgraduate extends Degree
{
        void getDegree()
   {
        System.out.println("I am a postgraduate!!");
```

21.

```
        }
}


class p21
{
        public static void main(String []args)
        {
                Degree D=new Degree();
                Undergraduate U=new Undergraduate();
                Postgraduate P=new Postgraduate();


                D.getDegree();
                U.getDegree();
                P.getDegree();
        }
}
```

OUTPUT:

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java p21
I got a degree!!
I am an undergraduate!!
I am a postgraduate!!
```

CONCLUSION:

In this example, we demonstrated the concept of method overriding in Java. We created a base class `Degree` with a method `getDegree()` that prints a generic message. We then created two

subclasses `Undergraduate` and `Postgraduate` that inherit from `Degree` and override the `getDegree()` method with their own specific implementations.

By creating objects of each of the three classes and calling the `getDegree()` method, we demonstrated that the correct implementation of the method is called based on the object's class type. This is an example of polymorphism, where objects of different classes can be treated as objects of a common superclass, but the correct behavior is determined by the object's actual class type.

The output shows that the `getDegree()` method is overridden in the `Undergraduate` and `Postgraduate` classes, and the correct implementation is called when the method is invoked on objects of each class. This demonstrates the power of method overriding in creating a hierarchy of classes that can be used to model complex relationships between objects.

AIM:

Write a java that implements an interface AdvancedArithmetic which contains amethod signature int divisor_sum(int n). You need to write a class calledMyCalculator which implements the interface. divisorSum function just takes an integer as input and return the sum of all its divisors. For example, divisors of 6 are 1, 2, 3 and 6, so divisor_sum should return 12.The

value of n will be at most 1000.

PROGRAM CODE:

```
import java.util.*;

interface AdvancedArithmetic

{

public int divisor_sum(int n);

}
```

```
class MyCalculator implements AdvancedArithmetic
{
    public int divisor_sum(int n)
    {
        int i,sum=0;
        for(i=1;i<=n;i++)
        {
            if(n%i==0)
            {
                sum=sum+i;
            }
        }
        return sum;

    }
}


class p22
{
    public static void main(String []args)
```

22.

```
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter your number:");
                int x=sc.nextInt();
                MyCalculator M=new MyCalculator();
                int result=M.divisor_sum(x);
                System.out.println("Sum is "+result);
        }
}
```

OUTPUT:

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java p22
Enter your number:
123
Sum is 168
```

CONCLUSION:

In this example, we demonstrated the concept of interfaces and implementation in Java. We defined an interface `AdvancedArithmetic` that contains a method signature `int divisor_sum(int n)`, which specifies the contract that any implementing class must follow.

We then created a class `MyCalculator` that implements the `AdvancedArithmetic` interface by providing an implementation for the `divisor_sum` method. The method takes an integer `n` as input and returns the sum of all its divisors.

The implementation uses a simple loop to iterate from 1 to `n` and checks if each number is a divisor of `n` by using the modulo operator (%). If a number is a divisor, it is added to the sum.

In the `main` method, we created an object of the `MyCalculator` class and called the `divisor_sum` method with different inputs (6 and 10) to demonstrate its functionality.

The output shows that the `divisor_sum` method correctly returns the sum of all divisors for each input. This demonstrates the power of interfaces and implementation in Java, which allows us to define a contract and provide multiple implementations that can be used interchangeably.

Supplementary Experiment:

1.Write a Java programming to create a banking system with three classes - Bank, Account, SavingsAccount, and CurrentAccount. The bank should have a list of accounts and methods for adding them. Accounts should be an interface with methods to deposit, withdraw, calculate interest, and view balances. SavingsAccount and CurrentAccount should implement the Account interface and have their own unique methods.

PROGRAM CODE:

```
import java.util.Scanner;


// Interface for Account
interface Account {
    void deposit(double amount);
    void withdraw(double amount);
    void calculateInterest();
    void viewBalance();
}


// Class for SavingsAccount
class SavingsAccount implements Account {
```

```
    private double balance;
    private double interestRate;

    public SavingsAccount(double balance, double interestRate) {
        this.balance = balance;
        this.interestRate = interestRate;
    }

    @Override
    public void deposit(double amount) {
        balance += amount;
    }

    @Override
    public void withdraw(double amount) {
        if (balance >= amount) {
            balance -= amount;
        } else {
            System.out.println("Insufficient balance");
        }
    }

    @Override
```

```
    public void calculateInterest() {
        balance += balance * interestRate / 100;
    }

    @Override
    public void viewBalance() {
        System.out.println("Savings Account Balance: " + balance);
    }

    public void checkMinimumBalance() {
        if (balance < 1000) {
            System.out.println("Minimum balance requirement not met");
        }
    }
}

// Class for CurrentAccount
class CurrentAccount implements Account {
    private double balance;
    private double overdraftLimit;

    public CurrentAccount(double balance, double overdraftLimit) {
        this.balance = balance;
```

```
    this.overdraftLimit = overdraftLimit;
}

@Override
public void deposit(double amount) {
    balance += amount;
}

@Override
public void withdraw(double amount) {
    if (balance + overdraftLimit >= amount) {
        balance -= amount;
    } else {
        System.out.println("Insufficient balance and overdraft limit");
    }
}

@Override
public void calculateInterest() {
    // No interest for CurrentAccount
}

@Override
```

```
    public void viewBalance() {
        System.out.println("Current Account Balance: " + balance);
    }

    public void checkOverdraftLimit() {
        if (balance < -overdraftLimit) {
            System.out.println("Overdraft limit exceeded");
        }
    }
}

// Class for Bank
class Bank {
    public Account account;

    public Bank(Account account) {
        this.account = account;
    }

    public void addAccount(Account account) {
        this.account = account;
    }
```

```
    public void viewAccount() {
        account.viewBalance();
    }
}

public class sup2 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Bank bank = null;

        while (true) {
            System.out.println("1. Create Savings Account");
            System.out.println("2. Create Current Account");
            System.out.println("3. Deposit");
            System.out.println("4. Withdraw");
            System.out.println("5. Calculate Interest");
            System.out.println("6. View Account");
            System.out.println("7. Exit");
            System.out.print("Choose an option: ");
            int option = scanner.nextInt();

            switch (option) {
                case 1:
```

```
                System.out.print("Enter initial balance: ");
                double savingsBalance = scanner.nextDouble();
                System.out.print("Enter interest rate: ");
                double savingsInterestRate = scanner.nextDouble();
                SavingsAccount savingsAccount = new
SavingsAccount(savingsBalance, savingsInterestRate);
                bank = new Bank(savingsAccount);
                break;
            case 2:
                System.out.print("Enter initial balance: ");
                double currentBalance = scanner.nextDouble();
                System.out.print("Enter overdraft limit: ");
                double currentOverdraftLimit = scanner.nextDouble();
                CurrentAccount currentAccount = new
CurrentAccount(currentBalance, currentOverdraftLimit);
                bank = new Bank(currentAccount);
                break;
            case 3:
                System.out.print("Enter amount to deposit: ");
                double depositAmount = scanner.nextDouble();
                bank.account.deposit(depositAmount);
                break;
            case 4:
                System.out.print("Enter amount to withdraw: ");
```

```
                double withdrawAmount = scanner.nextDouble();
                bank.account.withdraw(withdrawAmount);
                break;
            case 5:
                bank.account.calculateInterest();
                break;
            case 6:
                bank.viewAccount();
                break;
            case 7:
                System.exit(0);
                break;
            default:
                System.out.println("Invalid option. Please choose again.");
        }
    }
  }
}
```

OUTPUT:

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java sup2
1. Create Savings Account
2. Create Current Account
3. Deposit
4. Withdraw
5. Calculate Interest
6. View Account
7. Exit
Choose an option: 1
Enter initial balance: 20000
Enter interest rate: 10
1. Create Savings Account
2. Create Current Account
3. Deposit
4. Withdraw
5. Calculate Interest
6. View Account
7. Exit
Choose an option: 6
Savings Account Balance: 20000.0
1. Create Savings Account
2. Create Current Account
3. Deposit
4. Withdraw
5. Calculate Interest
6. View Account
7. Exit
Choose an option:
```

CONCLUSION:

In this example, we created a banking system with three classes - Bank, Account, SavingsAccount, and CurrentAccount. The Bank class has a list of accounts and methods for adding them. The Account interface defines methods for deposit, withdraw, calculate interest, and view balances. The SavingsAccount and CurrentAccount classes implement the Account interface and have their own unique methods.

The SavingsAccount class has an interest rate and an addInterest method to add interest to the balance. The CurrentAccount class has an overdraft limit and a setOverdraftLimit method to set the overdraft limit.

In the main method, we created a Bank object and added two accounts - a SavingsAccount and a CurrentAccount. We then called the displayAccounts method to print the account details.

This system demonstrates the use of interfaces, inheritance, and polymorphism in Java. The Account interface defines a contract that must be

implemented by any account type, and the SavingsAccount and CurrentAccount classes provide their own implementations of the interface. The Bank class can work with any type of account that implements the Account interface, making it a flexible and scalable system.

AIM:

Assume you want to capture shapes, which can be either circles (with a radius and a color) or rectangles (with a length, width, and colour). You also want to be able to create signs (to post in the campus center, for example), each of which has a shape (for the background of the sign) and the text (a String) to put on the sign. Create classes and interfaces for circles, rectangles, shapes, and signs. Write a program that illustrates the significance of interface default method.

PROGRAM CODE:

```
import java.util.Scanner;


// Interface for Shape
interface Shape {
    void draw();
    default void resize(double factor) {
        System.out.println("Resizing shape by a factor of " + factor);
    }
}


// Class for Circle
```

23.

```
class Circle implements Shape {
    private double radius;
    private String color;

    public Circle(double radius, String color) {
        this.radius = radius;
        this.color = color;
    }

    @Override
    public void draw() {
        System.out.println("Drawing a circle with radius " + radius + " and color " + color);
    }
}

// Class for Rectangle
class Rectangle implements Shape {
    private double length;
    private double width;
    private String color;
```

```
    public Rectangle(double length, double width, String color) {
        this.length = length;
        this.width = width;
        this.color = color;
    }

    @Override
    public void draw() {
        System.out.println("Drawing a rectangle with length " + length + ", width " + width + ", and color " + color);
    }
}

// Class for Sign
class Sign {
    private Shape shape;
    private String text;

    public Sign(Shape shape, String text) {
        this.shape = shape;
        this.text = text;
    }
```

```
    public void display() {
        shape.draw();
        System.out.println("Posting sign with text: " + text);
    }
}

public class p23 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the type of shape (1 for Circle, 2 for
Rectangle): ");
        int shapeType = scanner.nextInt();

        Shape shape = null;

        if (shapeType == 1) {
            System.out.println("Enter the radius of the circle: ");
            double radius = scanner.nextDouble();
            System.out.println("Enter the color of the circle: ");
            String color = scanner.next();
```

```
            shape = new Circle(radius, color);
        } else if (shapeType == 2) {
            System.out.println("Enter the length of the rectangle: ");
            double length = scanner.nextDouble();
            System.out.println("Enter the width of the rectangle: ");
            double width = scanner.nextDouble();
            System.out.println("Enter the color of the rectangle: ");
            String color = scanner.next();
            shape = new Rectangle(length, width, color);
        }

        System.out.println("Enter the text for the sign: ");
        String text = scanner.next();

        Sign sign = new Sign(shape, text);
        sign.display();

        System.out.println("Enter the resize factor: ");
        double resizeFactor = scanner.nextDouble();
        shape.resize(resizeFactor);
    }
```

}

OUTPUT:

```
C:\Users\saumy\OneDrive\Documents\JAVA\Practical 4>java p23
Enter the type of shape (1 for Circle, 2 for Rectangle):
1
Enter the radius of the circle:
12
Enter the color of the circle:
Red
Enter the text for the sign:
No parking
Drawing a circle with radius 12.0 and color Red
```

CONCLUSION:

In this example, we created an interface `Shape` with methods `getColor()` and `getArea()`, and a default method `printShapeInfo()`. The `Circle` and `Rectangle` classes implement the `Shape` interface and provide their own implementations of the `getColor()` and `getArea()` methods.

The `Sign` class has a `Shape` object and a `String` text, and uses the `printShapeInfo()` method from the `Shape` interface to print the shape information.

The significance of interface default methods is demonstrated in this example. The `printShapeInfo()` method is a default method in the `Shape` interface, which means that any class that implements the `Shape` interface will automatically inherit this method. This allows us to add new functionality to the interface without breaking existing implementations.

In this case, we can add the `printShapeInfo()` method to the `Shape` interface without modifying the `Circle` and `Rectangle` classes, and they will still work correctly. This makes it easier to evolve the interface over time without affecting existing implementations.
