# Day-4

# 1) package oops;

## interface Superclass1{

## void Sending();

## }

## interface Superclass2{

## void Sending();

## }

## class subclass implements Superclass1, Superclass2 {

## public void Sending() {

## System.out.println("superclass 1");

## }

## public void sended() {

## System.out.println("Superclass 2");

## }

## }

## public class Main {

## public static void main(String[] args) {

## subclass s=new subclass();

## s.sended();

## s.Sending();

## }

## }

## output

## Superclass 2

## superclass 1

#### public class Main

#### {

#### public static void main(String[] args) {

#### int a=10;

#### int[] arr = {1,2,4,3,5,12,10};

#### int max=0;

#### for(int i=0;i<arr.length;i++)

#### {

#### if(arr[i]>max)

#### {

#### max=arr[i];

#### }

#### }

#### System.out.println(max);

#### }

#### }

#### max num in an array

#### import java.util.Scanner;

#### public class Main

#### {

#### public static void main(String[] args) {

#### Scanner in = new Scanner(System.in);

#### // int a = 10;//primitive datatype

#### // Integer b = a;//

#### // System.out.println(b);

#### 

#### String str = "123@456@789@321";

#### String[] arr = str.split("@");

#### int max=0;

#### 

#### for(int i=0;i<arr.length;i++)

#### {

#### //Integer.parseInt converts string in Integer

#### if(Integer.parseInt(arr[i])>max)

#### {

#### //if the converted value is greater then max

#### //the values is stored in max

#### max=Integer.parseInt(arr[i]);

#### }

#### }

#### System.out.println(max);

#### }

#### }

#### class Main {

#### public static void main(String[] args) {

#### Account ac=new Account("Gowri",2000);

#### System.out.println("AccountHolder Name:"+ac.getAccholder());

#### ac.setAccholder("Pavithra");

#### System.out.println("update Accholder Name:"+ac.getAccholder());

#### System.out.println("Holder Balance:"+ac.getBalance());

#### ac.setBalance(3000);

#### ac.setWithdraw(1000);

#### System.out.println("current Balance:"+ac.getBalance

#### }

#### }

#### class Account{

#### private String Accholder;

#### private int Balance;

#### Account(String Accholder,int Balance){

#### this.Accholder=Accholder;

#### this.Balance=Balance;

#### }

#### public String getAccholder(){

#### return  Accholder;

#### }

#### public int getBalance(){

#### return Balance;

#### }

#### public void setBalance(int amount){

#### if(Balance > 0){

#### Balance +=amount;

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

#### System.out.println("total balance:"+Balance);

#### }

#### }

#### public void setWithdraw(int withdraw){

#### Balance -=withdraw;

#### System.out.println("Withdrwan amount:"+withdraw);

#### }

#### //to update accholder name

#### public void setAccholder(String Accholder){

#### this.Accholder=Accholder;

#### }

#### }

#### 1)Method Overloading:

#### Employee Salary Calculation:

#### Let's consider a real-world scenario where an HR system calculates employee salaries based on different types of employees:

#### 1)Full-time employees get a fixed salary.

#### 2)Part-time employees get paid based on hours worked.

#### 3)Freelancers get paid based on  hours worked and per-hour rate with bonus.

#### class Employee {

#### // Method 1: Salary for full-time employee (fixed salary)

#### void calculateSalary(double basicSalary) {

#### System.out.println("Full-time Employee Salary: ₹" + basicSalary);

#### }

#### // Method 2: Salary for part-time employee (based on hours worked)

#### void calculateSalary(int hoursWorked, double perHourRate) {

#### double salary = hoursWorked \* perHourRate;

#### System.out.println("Part-time Employee Salary: ₹" + salary);

#### }

#### // Method 3: Salary for freelancer (hours worked, per hour rate, bonus)

#### void calculateSalary(int hoursWorked, double perHourRate, double bonus) {

#### double salary = (hoursWorked \* perHourRate) + bonus;

#### System.out.println("Freelancer Salary: ₹" + salary);

#### }

#### }

#### public class SalaryExample {

#### public static void main(String[] args) {

#### Employee emp = new Employee();

#### 

#### emp.calculateSalary(50000);              // Full-time employee

#### emp.calculateSalary(20, 500);            // Part-time employee

#### emp.calculateSalary(30, 700, 5000);      // Freelancer

#### }

#### }

#### 2)Online Shopping Cart

#### Let's consider an e-commerce application where customers can add different types of items to their cart:

#### Single item with just the price.

#### Multiple items with quantity and price per unit.

#### Multiple items with discount applied.

#### class ShoppingCart {

#### // Method 1: Adding a single item with a fixed price

#### void addItem(String itemName, double price) {

#### System.out.println(itemName + " added to cart. Price: ₹" + price);

#### }

#### // Method 2: Adding multiple items with quantity

#### void addItem(String itemName, int quantity, double pricePerItem) {

#### double totalPrice = quantity \* pricePerItem;

#### System.out.println(quantity + " x " + itemName + " added to cart. Total Price: ₹" + totalPrice);

#### }

#### // Method 3: Adding multiple items with discount applied

## void addItem(String itemName, int quantity, double pricePerItem, double discount) {

## double totalPrice = (quantity \* pricePerItem) - discount;

## System.out.println(quantity + " x " + itemName + " added to cart with discount. Final Price: ₹" + totalPrice);

## }

## }

## public class ShoppingExample {

## public static void main(String[] args) {

## ShoppingCart cart = new ShoppingCart();

## 

## cart.addItem("Laptop", 55000);                  // Single item

## cart.addItem("Mouse", 2, 500);                  // Multiple items

## cart.addItem("Keyboard", 3, 1000, 500);         // Multiple items with discount

## }

## }

## Method Overridding

## class payment {

## void pay(double amount) {

## System.out.println("pay using cash:" + amount);

## }

## }

## //subclass 1: creadit card payment

## class Creditpay extends payment {

## void pay(double amount) {

## System.out.println("pay using creditcard:" + amount);

## }

## }

## class Upi extends payment {

## void pay(double amount) {

## System.out.println("pay using Upi:" + amount);

## }

## }

## public class ProgramP {

## public static void main(String[] args) {

## payment payment; // Creating a reference variable of the parent class

## payment p = new payment();

## p.pay(2000.00); // Assigning a subclass object to the parent class reference

## payment = new Creditpay();

## payment.pay(3000.00);

## payment = new Upi();

## payment.pay(40000.00);

## }

## }

## //Polymorphism allows us to perform a single action in different ways. It enhances code reusability, scalability, and maintainability by enabling the use of a common interface for multiple types.

## //There are two types of polymorphism in Java:

## //Compile-time Polymorphism (Method Overloading) – Same method name but different parameters.

## //Runtime Polymorphism (Method Overriding) – Method in the subclass overrides the method in the Superclass class.

|  |  |
| --- | --- |
|  |  |