**C-DAC Mumbai**

**OOPJ Lab**

**Assignment – 3**

**1. Bank Account Protection**

**Scenario:** You are creating a simple banking system. A user should not be able to directly change their balance.

->class BankAccountProtection {

private double balance;

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.out.println("Deposited: " + amount);

} else {

System.out.println("Deposit amount must be positive!");

}

}

public void withdraw(double amount) {

if (amount > 0) {

if (amount <= balance) {

balance -= amount;

System.out.println("Withdrawn: " + amount);

} else {

System.out.println("Insufficient Balance! Withdrawal failed.");

}

} else {

System.out.println("Withdrawal amount must be positive!");

}

}

public double getBalance() {

return balance;

}  
}

public class Main {

public static void main(String[] args) {

BankAccount account = new BankAccount();

account.deposit(5000);

account.withdraw(2000);

System.out.println("Updated Balance = " + account.getBalance());

}

}

**2. Student Marks**

**Scenario:** A teacher wants to update student marks but must ensure marks are valid.

->class StudentMarks {

private int marks;

public void setMarks(int marks) {

if (marks >= 0 && marks <= 100) {

this.marks = marks;

System.out.println("Marks updated successfully!");

} else {

System.out.println("Invalid marks! Please enter between 0 and 100.");

}

}

public int getMarks() {

return marks;

}

}

public class Main {

public static void main(String[] args) {

Student student = new Student();

student.setMarks(85);

System.out.println("Marks = " + student.getMarks());

}

}

**3. Employee Age Validation**

**Scenario:** HR wants to ensure employees entered in the system are adults.

->class EmployeeAgeVal {

private int age;

public void setAge(int age) {

if (age >= 18) {

this.age = age;

System.out.println("Age set successfully.");

} else {

System.out.println("Invalid age");

}

}

public int getAge() {

return age;

}

public static void main(String[] args) {

Employee emp = new Employee();

emp.setAge(17);

System.out.println("Current Age = " + emp.getAge());

}

}

**4. Library Book Availability**

**Scenario:** A library wants to keep track of available copies without letting external code change it directly.

->class LibBook {

private int copiesAvailable;

public void addCopies(int n) {

if (n > 0) {

copiesAvailable += n;

System.out.println(n + " copies added.");

} else {

System.out.println("Invalid number of copies!");

}

}

public void removeCopies(int n) {

if (n > 0) {

if (n <= copiesAvailable) {

copiesAvailable -= n;

System.out.println(n + " copies removed.");

} else {

System.out.println("Not enough copies available!");

}

} else {

System.out.println("Invalid number of copies!");

}

}

public int getCopiesAvailable() {

return copiesAvailable;

}

public static void main(String[] args) {

Book book = new Book();

book.addCopies(10);

book.removeCopies(3);

System.out.println("Copies Available = " + book.getCopiesAvailable());

}

}

**5. Temperature Sensor**

**Scenario:** Sensor should only accept temperatures in safe range.

->class TemperatureSensor {

private int temperature;

public void setTemperature(int t) {

if (t >= 0 && t <= 100) {

this.temperature = t;

System.out.println("Temperature set successfully.");

} else {

System.out.println("Temperature out of range");

}

}

public int getTemperature() {

return temperature;

}

public static void main(String[] args) {

TemperatureSensor sensor = new TemperatureSensor();

sensor.setTemperature(-5);

System.out.println("Current Temperature = " + sensor.getTemperature());

}

}

**6. Shape Area Calculation**

**Scenario:** You are designing a program to calculate areas of different shapes.

->class ShapeAreaCal {

public double area() {

return 0;

}

}

class Rectangle extends Shape {

private double length;

private double breadth;

public Rectangle(double length, double breadth) {

this.length = length;

this.breadth = breadth;

}

@Override

public double area() {

return length \* breadth;

}

}

class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double area() {

return Math.PI \* radius \* radius;

}

}

public class ShapeTest {

public static void main(String[] args) {

Rectangle rect = new Rectangle(5, 10); // Sample Input

Circle cir = new Circle(7); // Sample Input

System.out.println("Rectangle Area = " + rect.area());

System.out.printf("Circle Area = %.2f\n", cir.area()); // formatted to 2 decimal places

}

}

**7. Employee Hierarchy**

**Scenario:** A company has regular and contractual employees with different pay structures.

class EmployeeHierarchy {

protected String name;

protected double basicSalary;

public Employee(String name, double basicSalary) {

this.name = name;

this.basicSalary = basicSalary;

}

public double getNetSalary() {

return basicSalary;

}

}

class RegularEmployee extends Employee {

public RegularEmployee(String name, double basicSalary) {

super(name, basicSalary);

}

@Override

public double getNetSalary() {

double hra = 0.10 \* basicSalary; // 10% HRA

return basicSalary + hra;

}

}

class ContractEmployee extends Employee {

public ContractEmployee(String name, double basicSalary) {

super(name, basicSalary);

}

@Override

public double getNetSalary() {

double allowance = 0.05 \* basicSalary; // 5% Allowance

return basicSalary + allowance;

}

}

public class EmployeeTest {

public static void main(String[] args) {

RegularEmployee e1 = new RegularEmployee("Rahul", 20000);

ContractEmployee e2 = new ContractEmployee("Riya", 15000);

System.out.println(e1.name + " Net Salary = " + e1.getNetSalary());

System.out.println(e2.name + " Net Salary = " + e2.getNetSalary());

}

}

**8. Vehicle Types**

**Scenario:** You want to categorize vehicles.

->

class VehicleTypes {

protected String brand;

protected int speed;

public Vehicle(String brand, int speed) {

this.brand = brand;

this.speed = speed;

}

public void displayDetails() {

System.out.println("Brand: " + brand + ", Speed: " + speed);

}

}

class Car extends Vehicle {

private String modelType;

public Car(String brand, int speed, String modelType) {

super(brand, speed);

this.modelType = modelType;

}

@Override

public void displayDetails() {

System.out.println("Car → " + brand + " " + modelType + ", Speed = " + speed);

}

}

class Bike extends Vehicle {

private String modelType;

public Bike(String brand, int speed, String modelType) {

super(brand, speed);

this.modelType = modelType;

}

@Override

public void displayDetails() {

System.out.println("Bike → " + brand + " " + modelType + ", Speed = " + speed);

}

}

public class VehicleTest {

public static void main(String[] args) {

Car car = new Car("Honda", 180, "Civic"); // Sample Input

Bike bike = new Bike("Yamaha", 120, "R15"); // Sample Input

car.displayDetails();

bike.displayDetails();

}

}

**9. Animal Sound**

**Scenario:** You are building a zoo management system to play animal sounds.

// Superclass

class AnimalSound {

public void makeSound() {

System.out.println("Some generic animal sound");

}

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Dog → Bark");

}

}

class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("Cat → Meow");

}

}

public class AnimalTest {

public static void main(String[] args) {

Dog d = new Dog();

Cat c = new Cat();

d.makeSound();

c.makeSound();

}

}

**10. Academic Staff**

**Scenario:** University has teaching and non-teaching staff.

class AcademicStaff {

protected String name;

protected double salary;

public Staff(String name, double salary) {

this.name = name;

this.salary = salary;

}

public void displayInfo() {

System.out.println("Name: " + name + ", Salary: " + salary);

}

}

class TeachingStaff extends Staff {

private String subject;

public TeachingStaff(String name, double salary, String subject) {

super(name, salary);

this.subject = subject;

}

@Override

public void displayInfo() {

System.out.println("Teaching Staff → Name: " + name + ", Salary: " + salary + ", Subject: " + subject);

}

}

class NonTeachingStaff extends Staff {

private String department;

public NonTeachingStaff(String name, double salary, String department) {

super(name, salary);

this.department = department;

}

@Override

public void displayInfo() {

System.out.println("Non-Teaching Staff → Name: " + name + ", Salary: " + salary + ", Department: " + department);

}

}

public class StaffTest {

public static void main(String[] args) {

TeachingStaff t = new TeachingStaff("Anita", 50000, "Math");

NonTeachingStaff nt = new NonTeachingStaff("Ramesh", 40000, "Admin"

t.displayInfo();

nt.displayInfo();

}

}

**11. Bank Account Types**

**Scenario:** Bank provides different account types.

->

class AccountTypes {

protected int accountNo;

protected double balance;

public Account(int accountNo, double balance) {

this.accountNo = accountNo;

this.balance = balance;

}

public void displayDetails() {

System.out.println("Account No: " + accountNo + ", Balance: " + balance);

}

}

class SavingAccount extends Account {

private double interestRate;

public SavingAccount(int accountNo, double balance, double interestRate) {

super(accountNo, balance);

this.interestRate = interestRate;

}

@Override

public void displayDetails() {

System.out.println("Saving → " + accountNo + ", Balance = " + balance + ", Interest = " + interestRate + "%");

}

}

class CurrentAccount extends Account {

private double overdraftLimit;

public CurrentAccount(int accountNo, double balance, double overdraftLimit) {

super(accountNo, balance);

this.overdraftLimit = overdraftLimit;

}

@Override

public void displayDetails() {

System.out.println("Current → " + accountNo + ", Balance = " + balance + ", Overdraft = " + overdraftLimit);

}

}

public class AccountTest {

public static void main(String[] args) {

SavingAccount s = new SavingAccount(101, 5000, 5);

CurrentAccount c = new CurrentAccount(102, 10000, 2000);

s.displayDetails();

c.displayDetails();

}

}

**12. Payment System**

**Scenario:** A company accepts different payment modes

// Abstract superclass

abstract class PaymentSys {

public abstract void pay();

}

// Subclass for Credit Card Payment

class CreditCardPayment extends Payment {

private String cardNumber;

private double amount;

public CreditCardPayment(String cardNumber, double amount) {

this.cardNumber = cardNumber;

this.amount = amount;

}

@Override

public void pay() {

System.out.println("Payment via Credit Card " + cardNumber + " → Rs. " + amount + " Paid");

}

}

// Subclass for UPI Payment

class UPIPayment extends Payment {

private String upiId;

private double amount;

public UPIPayment(String upiId, double amount) {

this.upiId = upiId;

this.amount = amount;

}

@Override

public void pay() {

System.out.println("Payment via UPI " + upiId + " → Rs. " + amount + " Paid");

}

}

// Test class

public class PaymentTest {

public static void main(String[] args) {

CreditCardPayment cc = new CreditCardPayment("1234567890123456", 5000); // Sample Input

UPIPayment upi = new UPIPayment("rahul@upi", 2000); // Sample Input

cc.pay();

upi.pay();

}

}

**13. Shape Drawing**

**Scenario:** A graphics program needs to draw different shapes.

-> // Abstract superclass

abstract class ShapeDrawing {

public abstract void draw();

}

// Subclass for Circle

class Circle extends Shape {

private int radius;

public Circle(int radius) {

this.radius = radius;

}

@Override

public void draw() {

System.out.println("Drawing Circle of radius " + radius);

}

}

// Subclass for Rectangle

class Rectangle extends Shape {

private int length;

private int breadth;

public Rectangle(int length, int breadth) {

this.length = length;

this.breadth = breadth;

}

@Override

public void draw() {

System.out.println("Drawing Rectangle of length " + length + " and breadth " + breadth);

}

}

// Test class

public class ShapeDrawingTest {

public static void main(String[] args) {

Shape c = new Circle(7); // Sample Input

Shape r = new Rectangle(5, 10); // Sample Input

c.draw();

r.draw();

}

}

**14. Employee Bonus Calculation**

**Scenario:** A company has different types of employees with specific bonus calculation rules.

// Abstract superclass

abstract class EmployeeBonus {

protected String name;

protected double salary;

public Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

public abstract double calculateBonus();

}

// Subclass for Manager

class Manager extends Employee {

public Manager(String name, double salary) {

super(name, salary);

}

@Override

public double calculateBonus() {

return 0.20 \* salary; // 20% bonus

}

}

// Subclass for Developer

class Developer extends Employee {

public Developer(String name, double salary) {

super(name, salary);

}

@Override

public double calculateBonus() {

return 0.10 \* salary; // 10% bonus

}

}

// Test class

public class EmployeeBonusTest {

public static void main(String[] args) {

Employee m = new Manager("Anita", 50000); // Sample Input

Employee d = new Developer("Rohit", 40000); // Sample Input

System.out.println(m.name + " Bonus = " + m.calculateBonus());

System.out.println(d.name + " Bonus = " + d.calculateBonus());

}

}

**15. Shape Area Calculation**

**Scenario:** A program needs to calculate the area of different shapes using the same method name but different parameters.

->class ShapeArea {

// Overloaded method for square

public double calculateArea(int side) {

return side \* side;

}

// Overloaded method for rectangle

public double calculateArea(int length, int breadth) {

return length \* breadth;

}

// Overloaded method for circle

public double calculateArea(double radius) {

return Math.PI \* radius \* radius;

}

// Test class

public static void main(String[] args) {

ShapeArea sa = new ShapeArea();

// Sample Inputs

double squareArea = sa.calculateArea(5);

double rectangleArea = sa.calculateArea(4, 6);

double circleArea = sa.calculateArea(3.0);

// Sample Outputs

System.out.println("Square Area = " + squareArea);

System.out.println("Rectangle Area = " + rectangleArea);

System.out.printf("Circle Area = %.2f\n", circleArea); // formatted to 2 decimals

}

}

**16. Employee Salary Display**

**Scenario:** Company wants to display employee salary with different bonus calculations based on employee type.

->// Superclass

class EmployeeSalDis {

protected String name;

protected double salary;

public Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

public void displaySalary() {

System.out.println(name + " Total Salary = " + salary);

}

}

// Subclass for Manager

class Manager extends Employee {

public Manager(String name, double salary) {

super(name, salary);

}

@Override

public void displaySalary() {

double totalSalary = salary + (0.20 \* salary); // 20% bonus

System.out.println(name + " Total Salary = " + totalSalary);

}

}

// Subclass for Developer

class Developer extends Employee {

public Developer(String name, double salary) {

super(name, salary);

}

@Override

public void displaySalary() {

double totalSalary = salary + (0.10 \* salary); // 10% bonus

System.out.println(name + " Total Salary = " + totalSalary);

}

}

// Test class

public class EmployeeSalaryTest {

public static void main(String[] args) {

Employee m = new Manager("Anita", 50000); // Sample Input

Employee d = new Developer("Rohit", 40000); // Sample Input

m.displaySalary();

d.displaySalary();

**17. Vehicle Speed Display**

**Scenario:** Vehicle management system needs to display speed differently for

different vehicle types.

->// Superclass

class VehicleSpeedDis {

public void displaySpeed() {

System.out.println("Vehicle speed unknown");

}

}

// Subclass for Car

class Car extends Vehicle {

@Override

public void displaySpeed() {

System.out.println("Car speed 120 km/h");

}

}

// Subclass for Bike

class Bike extends Vehicle {

@Override

public void displaySpeed() {

System.out.println("Bike speed 80 km/h");

}

}

// Test class

public class VehicleSpeedTest {

public static void main(String[] args) {

Vehicle v1 = new Car(); // Sample Input: Car

Vehicle v2 = new Bike(); // Sample Input: Bike

v1.displaySpeed();

v2.displaySpeed();

}

}

**18. Payment Process**

**Scenario:** Company wants to process payments differently depending on mode of payment, but handle all payments through a single reference

->// Abstract superclass

abstract class Payprocess {

public abstract void pay();

}

// Subclass for Credit Card Payment

class CreditCardPayment extends Payment {

private String cardNumber;

private double amount;

public CreditCardPayment(String cardNumber, double amount) {

this.cardNumber = cardNumber;

this.amount = amount;

}

@Override

public void pay() {

System.out.println("Payment via Credit Card " + cardNumber + " → Rs. " + amount + " Paid");

}

}

// Subclass for UPI Payment

class UPIPayment extends Payment {

private String upiId;

private double amount;

public UPIPayment(String upiId, double amount) {

this.upiId = upiId;

this.amount = amount;

}

@Override

public void pay() {

System.out.println("Payment via UPI " + upiId + " → Rs. " + amount + " Paid");

}

}

// Test class

public class PaymentProcessTest {

public static void main(String[] args) {

Payment p; // Reference of abstract class

// Using runtime polymorphism

p = new CreditCardPayment("1234567890123456", 5000); // Sample Input

p.pay();

p = new UPIPayment("rahul@upi", 2000); // Sample Input

p.pay();

}

}

**19. Bank Account Types**

**Scenario:** Bank manages different types of accounts: Savings and Current. Both share basic account details, but Savings accounts have interest and Current accounts have overdraft limit.

->// Superclass

class BankAccTypes {

protected int accountNumber;

protected String accountHolder;

protected double balance;

public BankAccount(int accountNumber, String accountHolder, double balance) {

this.accountNumber = accountNumber;

this.accountHolder = accountHolder;

this.balance = balance;

}

public void displayBalance() {

System.out.println(accountHolder + " → Balance=" + balance);

}

}

// Subclass for Savings Account

class SavingsAccount extends BankAccount {

private double interestRate;

public SavingsAccount(int accountNumber, String accountHolder, double balance, double interestRate) {

super(accountNumber, accountHolder, balance);

this.interestRate = interestRate;

}

public void calculateInterest() {

double interest = balance \* (interestRate / 100);

System.out.println(accountHolder + " → Balance=" + balance + ", Interest=" + interest);

}

}

// Subclass for Current Account

class CurrentAccount extends BankAccount {

private double overdraftLimit;

public CurrentAccount(int accountNumber, String accountHolder, double balance, double overdraftLimit) {

super(accountNumber, accountHolder, balance);

this.overdraftLimit = overdraftLimit;

}

public void checkOverdraft() {

System.out.println(accountHolder + " → Balance=" + balance + ", Overdraft Limit=" + overdraftLimit);

}

}

// Test class

public class BankAccountTest {

public static void main(String[] args) {

SavingsAccount sa = new SavingsAccount(101, "Ramesh", 5000, 5);

sa.calculateInterest();

CurrentAccount ca = new CurrentAccount(102, "Anita", 2000, 1000);

ca.checkOverdraft();

}

}

**20. College Staff Hierarchy**

**Scenario:** A college has employees who can be Teaching or Non-Teaching. Teaching staff can be Professors or Lecturers.

->// Superclass

class clgstaff {

protected String name;

protected double salary;

public Employee(String name, double salary) {

this.name = name;

this.salary = salary;

}

public void displaySalary() {

System.out.println(name + " → Salary=" + salary);

}

}

// Teaching Staff

class TeachingStaff extends Employee {

protected String subject;

public TeachingStaff(String name, double salary, String subject) {

super(name, salary);

this.subject = subject;

}

@Override

public void displaySalary() {

System.out.println(name + " → Subject=" + subject + ", Salary=" + salary);

}

}

// Professor

class Professor extends TeachingStaff {

private String specialization;

public Professor(String name, double salary, String subject, String specialization) {

super(name, salary, subject);

this.specialization = specialization;

}

@Override

public void displaySalary() {

System.out.println(name + " → Subject=" + subject + ", Specialization=" + specialization + ", Salary=" + salary);

}

}

// Lecturer

class Lecturer extends TeachingStaff {

private String department;

public Lecturer(String name, double salary, String subject, String department) {

super(name, salary, subject);

this.department = department;

}

@Override

public void displaySalary() {

System.out.println(name + " → Subject=" + subject + ", Department=" + department + ", Salary=" + salary);

}

}

// Test class

public class CollegeStaffTest {

public static void main(String[] args) {

Professor prof = new Professor("Dr. Sharma", 80000, "Math", "Algebra");

prof.displaySalary();

Lecturer lec = new Lecturer("Ms. Mehta", 50000, "Physics", "Science");

lec.displaySalary();

}

}

**21. Hospital Staff**

**Scenario:** Hospital has Staff members. Both Doctors and Nurses are Staff.

->// Superclass

class HospitalStaff {

protected String name;

protected int staffId;

public Staff(String name, int staffId) {

this.name = name;

this.staffId = staffId;

}

public void displayDetails() {

System.out.println(name + " → Staff ID=" + staffId);

}

}

// Doctor Subclass

class Doctor extends Staff {

private String specialization;

public Doctor(String name, int staffId, String specialization) {

super(name, staffId);

this.specialization = specialization;

}

@Override

public void displayDetails() {

System.out.println(name + " → Staff ID=" + staffId + ", Specialization=" + specialization);

}

}

// Nurse Subclass

class Nurse extends Staff {

private String shift;

public Nurse(String name, int staffId, String shift) {

super(name, staffId);

this.shift = shift;

}

@Override

public void displayDetails() {

System.out.println(name + " → Staff ID=" + staffId + ", Shift=" + shift);

}

}

// Test class

public class HospitalTest {

public static void main(String[] args) {

Doctor d = new Doctor("Dr. Reddy", 101, "Cardiology");

Nurse n = new Nurse("Nisha", 102, "Night");

d.displayDetails();

n.displayDetails();

}

}

**22. Vehicle Types**

**Scenario:** Vehicles can be Land or Water types. Some vehicles can operate on both.

->// Interface for Land Vehicles

interface LandVehicleInterface {

void driveOnLand();

}

// Interface for Water Vehicles

interface WaterVehicle {

void driveOnWater();

}

// Amphibious Vehicle implements both interfaces

class AmphibiousVehicle implements LandVehicle, WaterVehicle {

private String name;

public AmphibiousVehicle(String name) {

this.name = name;

}

@Override

public void driveOnLand() {

System.out.println(name + " → Driving on Land");

}

@Override

public void driveOnWater() {

System.out.println(name + " → Driving on Water");

}

}

// Test class

public class VehicleTest {

public static void main(String[] args) {

AmphibiousVehicle av = new AmphibiousVehicle("HydroCar");

av.driveOnLand();

av.driveOnWater();

}

}

**23. School Members**

**Scenario:** School has members: Teachers, Students, and Staff. All share common info

->// Superclass

class Member {

protected String name;

protected int id;

public Member(String name, int id) {

this.name = name;

this.id = id;

}

public void displayInfo() {

System.out.println(name + " → ID=" + id);

}

}

// Teacher Subclass

class Teacher extends Member {

private String subject;

public Teacher(String name, int id, String subject) {

super(name, id);

this.subject = subject;

}

@Override

public void displayInfo() {

System.out.println(name + " → ID=" + id + ", Subject=" + subject);

}

}

// Student Subclass

class Student extends Member {

private int grade;

public Student(String name, int id, int grade) {

super(name, id);

this.grade = grade;

}

@Override

public void displayInfo() {

System.out.println(name + " → ID=" + id + ", Grade=" + grade);

}

}

// Staff Subclass

class Staff extends Member {

private String department;

public Staff(String name, int id, String department) {

super(name, id);

this.department = department;

}

@Override

public void displayInfo() {

System.out.println(name + " → ID=" + id + ", Department=" + department);

}

}

// Test Class

public class SchoolTest {

public static void main(String[] args) {

Teacher t = new Teacher("Mr. Kumar", 101, "English");

Student s = new Student("Riya", 201, 10);

Staff st = new Staff("Mr. Das", 301, "Maintenance");

t.displayInfo();

s.displayInfo();

st.displayInfo();

}

}

**24. Payment Gateway**

**Scenario:** An e-commerce platform supports multiple payment methods like CreditCard and PayPal. All payments must implement a pay() method.

->/ Payment Interface

interface Payment {

void pay(double amount);

}

// Credit Card Payment Class

class CreditCardPayment implements Payment {

@Override

public void pay(double amount) {

System.out.println("Processing Credit Card Payment of " + amount);

}

}

// PayPal Payment Class

class PayPalPayment implements Payment {

@Override

public void pay(double amount) {

System.out.println("Processing PayPal Payment of " + amount);

}

}

// Test Class

public class PaymentTest {

public static void main(String[] args) {

Payment credit = new CreditCardPayment();

Payment paypal = new PayPalPayment();

credit.pay(2500);

paypal.pay(1500);

}

}

**25. Media Player**

**Scenario:** A media player can play both Audio and Video files.

// Interface for Audio Player

interface AudioPlayer {

void playAudio(String song);

}

// Interface for Video Player

interface VideoPlayer {

void playVideo(String movie);

}

// MediaPlayer implements both interfaces

class MediaPlayer implements AudioPlayer, VideoPlayer {

@Override

public void playAudio(String song) {

System.out.println("Playing Audio: " + song);

}

@Override

public void playVideo(String movie) {

System.out.println("Playing Video: " + movie);

}

}

// Test Class

public class MediaPlayerTest {

public static void main(String[] args) {

MediaPlayer player = new MediaPlayer();

player.playAudio("Shape of You");

player.playVideo("Inception");

}

}

**26. Smart Devices**

**Scenario:** Smart devices can perform actions like calling, messaging, and browsing internet

-> // Callable Interface

interface Callable {

void makeCall(String number);

}

// Messaging Interface

interface Messaging {

void sendMessage(String number, String message);

}

// Internet Interface

interface Internet {

void browse(String website);

}

// SmartPhone implements all interfaces

class SmartPhone implements Callable, Messaging, Internet {

@Override

public void makeCall(String number) {

System.out.println("Calling " + number);

}

@Override

public void sendMessage(String number, String message) {

System.out.println("Sending message to " + number + ": " + message);

}

@Override

public void browse(String website) {

System.out.println("Browsing website: " + website);

}

}

// Test Class

public class SmartPhoneTest {

public static void main(String[] args) {

SmartPhone phone = new SmartPhone();

phone.makeCall("9876543210");

phone.sendMessage("9876543210", "Hello!");

phone.browse("www.google.com");

}

}

**27. Shape Area Calculator**

**Scenario:** A drawing application needs to calculate area for different shapes: Circle, Rectangle, and Square

->// Shape interface

interface Shape {

double calculateArea();

}

// Circle class

class Circle implements Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double calculateArea() {

return Math.PI \* radius \* radius;

}

}

// Rectangle class

class Rectangle implements Shape {

private double length, breadth;

public Rectangle(double length, double breadth) {

this.length = length;

this.breadth = breadth;

}

@Override

public double calculateArea() {

return length \* breadth;

}

}

// Square class

class Square implements Shape {

private double side;

public Square(double side) {

this.side = side;

}

@Override

public double calculateArea() {

return side \* side;

}

}

// Test class

public class ShapeAreaTest {

public static void main(String[] args) {

Shape circle = new Circle(5);

Shape rectangle = new Rectangle(10, 5);

Shape square = new Square(4);

System.out.printf("Circle Area = %.1f\n", circle.calculateArea());

System.out.println("Rectangle Area = " + rectangle.calculateArea());

System.out.println("Square Area = " + square.calculateArea());

}

}

**28. Online Shopping Cart System**

**Scenario:** Build a simplified shopping cart system where users can add products, calculate total cost, and apply discounts.

->import java.util.ArrayList;

import java.util.List;

// Product class (Encapsulation)

class Product {

private int productId;

private String name;

private double price;

public Product(int productId, String name, double price) {

this.productId = productId;

this.name = name;

this.price = price;

}

public String getName() { return name; }

public double getPrice() { return price; }

public void setPrice(double price) { this.price = price; }

}

// Abstract CartItem

abstract class CartItem {

public abstract double calculateTotalPrice();

}

// Interface for Discount

interface Discountable {

void applyDiscount(double percentage);

}

// Cart class

class Cart extends CartItem {

private List<Product> products;

public Cart() {

products = new ArrayList<>();

}

public void addProduct(Product p) {

products.add(p);

}

@Override

public double calculateTotalPrice() {

double total = 0;

for (Product p : products) {

total += p.getPrice();

}

return total;

}

}

// Main class

public class ShoppingCartTest {

public static void main(String[] args) {

Product p1 = new Product(1, "Laptop", 50000);

Product p2 = new Product(2, "Mouse", 500);

Product p3 = new Product(3, "Keyboard", 1200);

// Applying 10% discount to Laptop

System.out.println("Applying 10% discount to Laptop");

p1.setPrice(p1.getPrice() \* 0.9);

Cart cart = new Cart();

cart.addProduct(p1);

cart.addProduct(p2);

cart.addProduct(p3);

System.out.println("Total Cart Price = " + cart.calculateTotalPrice());

}

}

**29. Employee Management System**

**Scenario:** Manage employee details, calculate salaries, and differentiate employee types

->/ BonusEligible Interface

interface BonusEligible {

double calculateBonus();

}

// Abstract Employee Class

abstract class Employee {

protected String name;

protected int id;

public Employee(String name, int id) {

this.name = name;

this.id = id;

}

public abstract double calculateSalary();

}

// Permanent Employee

class PermanentEmployee extends Employee implements BonusEligible {

private double basicSalary;

private double hra;

public PermanentEmployee(String name, int id, double basicSalary, double hra) {

super(name, id);

this.basicSalary = basicSalary;

this.hra = hra;

}

@Override

public double calculateSalary() {

return basicSalary + hra;

}

@Override

public double calculateBonus() {

return 0.10 \* calculateSalary(); // 10% bonus

}

}

// Contract Employee

class ContractEmployee extends Employee {

private double hourlyRate;

private int hoursWorked;

public ContractEmployee(String name, int id, double hourlyRate, int hoursWorked) {

super(name, id);

this.hourlyRate = hourlyRate;

this.hoursWorked = hoursWorked;

}

@Override

public double calculateSalary() {

return hourlyRate \* hoursWorked;

}

}

// Test class

public class EmployeeManagementTest {

public static void main(String[] args) {

PermanentEmployee e1 = new PermanentEmployee("Amit", 101, 50000, 5000);

PermanentEmployee e2 = new PermanentEmployee("Sneha", 102, 60000, 6000);

ContractEmployee c1 = new ContractEmployee("Neha", 201, 300, 100);

ContractEmployee c2 = new ContractEmployee("Rohit", 202, 400, 90);

// Display salaries

System.out.println(e1.name + " Salary = " + e1.calculateSalary() + ", Bonus = " + e1.calculateBonus());

System.out.println(e2.name + " Salary = " + e2.calculateSalary() + ", Bonus = " + e2.calculateBonus());

System.out.println(c1.name + " Salary = " + c1.calculateSalary());

System.out.println(c2.name + " Salary = " + c2.calculateSalary());

}

}

**30. Library Management System**

**Scenario:** Manage books and library members with borrowing functionality.

->import java.util.ArrayList;

import java.util.List;

// Encapsulated Book Class

class Book {

private int bookId;

private String title;

private String author;

public Book(int bookId, String title, String author) {

this.bookId = bookId;

this.title = title;

this.author = author;

}

public String getTitle() { return title; }

}

// Notifyable Interface

interface Notifyable {

void sendNotification(String message);

}

// Abstract Library Member

abstract class LibraryMember {

protected int memberId;

protected String name;

protected List<Book> borrowedBooks;

public LibraryMember(int memberId, String name) {

this.memberId = memberId;

this.name = name;

borrowedBooks = new ArrayList<>();

}

public abstract void borrowBook(Book book);

}

// Student Member

class StudentMember extends LibraryMember implements Notifyable {

private static final int LIMIT = 3;

public StudentMember(int memberId, String name) {

super(memberId, name);

}

@Override

public void borrowBook(Book book) {

if (borrowedBooks.size() < LIMIT) {

borrowedBooks.add(book);

System.out.println("StudentMember " + name + " borrowed " + borrowedBooks.size() + " books");

} else {

System.out.println(name + " has reached the borrowing limit.");

}

}

@Override

public void sendNotification(String message) {

System.out.println("Notification sent to " + name + ": " + message);

}

}

// Faculty Member

class FacultyMember extends LibraryMember implements Notifyable {

private static final int LIMIT = 5;

public FacultyMember(int memberId, String name) {

super(memberId, name);

}

@Override

public void borrowBook(Book book) {

if (borrowedBooks.size() < LIMIT) {

borrowedBooks.add(book);

System.out.println("FacultyMember " + name + " borrowed " + borrowedBooks.size() + " books");

} else {

System.out.println(name + " has reached the borrowing limit.");

}

}

@Override

public void sendNotification(String message) {

System.out.println("Notification sent to " + name + ": " + message);

}

}

// Test class

public class LibraryTest {

public static void main(String[] args) {

Book b1 = new Book(1, "Math", "Author A");

Book b2 = new Book(2, "Physics", "Author B");

Book b3 = new Book(3, "Chemistry", "Author C");

Book b4 = new Book(4, "Biology", "Author D");

Book b5 = new Book(5, "History", "Author E");

StudentMember student = new StudentMember(101, "Amit");

FacultyMember faculty = new FacultyMember(201, "Prof. Singh");

// Borrow books

student.borrowBook(b1);

student.borrowBook(b2);

faculty.borrowBook(b1);

faculty.borrowBook(b2);

faculty.borrowBook(b3);

faculty.borrowBook(b4);

// Send notifications

student.sendNotification("Return books within 7 days");

faculty.sendNotification("Return books within 14 days");

}

}