C-DAC Mumbai
OOPJ Lab Assignment

Answers

1. Bank Account Protection

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

Problem Statement:

Create a class BankAccount with a private variable balance. Provide deposit() and withdraw() methods to change balance. Provide a getter to view balance. Validate that withdrawal cannot exceed balance.

Class/Fields:

- private double balance

Methods:

- deposit(double amount) → add to balance
- withdraw(double amount) → subtract from balance if sufficient
- getBalance() → returns current balance

Sample Input:

Deposit = 5000
Withdraw = 2000

Sample Output:

Updated Balance = 3000

```java
class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        if (initialBalance >= 0) {
            this.balance = initialBalance;
        } else {
            this.balance = 0;
            System.out.println("Initial balance cannot be negative. Setting balance to 0.");
        }
    }

    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                System.out.println("Withdrawn: " + amount);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 BankDemo {
    public static void main(String[] args) {
        BankAccount account = new BankAccount(0);
        account.deposit(5000);
        account.withdraw(2000);
        System.out.println("Updated Balance = " + account.getBalance());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac BankDemo.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java BankDemo
Deposited: 5000.0
Withdrawn: 2000.0
Updated Balance = 3000.0
```

2. Student Marks
Scenario: A teacher wants to update student marks but must ensure marks are valid.
Problem Statement:
Create a class Student with private marks. Create getter and setter with validation (0–100).
Fields:
● private int marks
Methods:
● setMarks(int marks) → validate and set
● getMarks() → return marks
Sample Input:
marks = 85
Sample Output:
Marks = 85

```java
public class Student {
    private int marks;

    public void setMarks(int marks) {
        if (marks >= 0 && marks <= 100) {
            this.marks = marks;
        }
    }

    public int getMarks() {
        return marks;
    }

    public static void main(String[] args) {
        Student s = new Student();
        s.setMarks(85);
        System.out.println("Marks = " + s.getMarks());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac Student.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java Student
Marks = 85
```

3. Employee Age Validation
Scenario: HR wants to ensure employees entered in the system are adults.
Problem Statement:
Create a class Employee with private age. Provide getter/setter. Setter should reject age < 18.
Fields:
● private int age
Methods:
● setAge(int age) → if age >=18, set; else show error
● getAge() → returns age
Sample Input:
age = 17
Sample Output:
“Invalid age”

```java
public class Employee {
    private int age;

    public void setAge(int age) {
        if (age >= 18) {
            this.age = age;
        } else {
            System.out.println("Invalid age");
        }
    }

    public int getAge() {
        return age;
    }

    public static void main(String[] args) {
        Employee e = new Employee();
        e.setAge(22);
        System.out.println("Age = " + e.getAge());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac Employee.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java Employee
Age = 22
```

4. Library Book Availability

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

Problem Statement:

Create a class Book with private copiesAvailable. Provide getter/setter to update or read copies.

Fields:

- private int copiesAvailable

Methods:

- addCopies(int n) → add copies
- removeCopies(int n) → subtract if enough copies
- getCopiesAvailable() → return current copies

Sample Input:

add 3 copies, remove 1 copy

Sample Output:

Copies available = 2

```java
class Book {
    private int copiesAvailable;

    public void addCopies(int n) {
        copiesAvailable += n;
    }

    public void removeCopies(int n) {
        if (copiesAvailable >= n) {
            copiesAvailable -= n;
        }
    }

    public int getCopiesAvailable() {
        return copiesAvailable;
    }

    public static void main(String[] args) {
        Book b = new Book();
        b.addCopies(3);
        b.removeCopies(1);
        System.out.println("Copies available = " + b.getCopiesAvailable());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac Book.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java Book
Copies available = 2
```

5. Temperature Sensor
Scenario: Sensor should only accept temperatures in safe range.
Problem Statement:
Create a class TemperatureSensor with private temperature. Setter validates range 0–100°C.
Fields:
● private int temperature
Methods:
● setTemperature(int t) → only 0–100 valid
● getTemperature() → return temperature
Sample Input:
temperature = -5
Sample Output:
"Temperature out of range"

```java
class TemperatureSensor {
    private int temperature;

    public void setTemperature(int t) {
        if (t >= 0 && t <= 100) {
            temperature = t;
        } else {
            System.out.println("Temperature out of range");
        }
    }

    public int getTemperature() {
        return temperature;
    }

    public static void main(String[] args) {
        TemperatureSensor ts = new TemperatureSensor();
        ts.setTemperature(45);
        System.out.println("Temperature = " + ts.getTemperature());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac TemperatureSensor.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java TemperatureSensor
Temperature = 45
```

6. Shape Area Calculation
Scenario: You are designing a program to calculate areas of different shapes.
Problem Statement:
Create a superclass Shape with a method area(). Derive two subclasses Rectangle and Circle. Implement area() in each subclass.
Classes/Fields:
● Shape → area() (method stub)
● Rectangle → length, breadth
● Circle → radius
Methods:
● Rectangle.area() → length × breadth
● Circle.area() → π × radius²
Sample Input:
Rectangle → length=5, breadth=10
Circle → radius=7
Sample Output:
Rectangle Area = 50
Circle Area = 153.86

```java
abstract class Shape {
    abstract double area();
}

class Rectangle extends Shape {
    double length, breadth;

    Rectangle(double l, double b) {
        length = l;
        breadth = b;
    }

    double area() {
        return length * breadth;
    }
}

class Circle extends Shape {
    double radius;

    Circle(double r) {
        radius = r;
    }

    double area() {
        return Math.PI * radius * radius;
    }

    public static void main(String[] args) {
        Rectangle r = new Rectangle(5, 10);
        Circle c = new Circle(7);
        System.out.println("Rectangle Area = " + r.area());
        System.out.printf("Circle Area = %.2f\n", c.area());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac Circle.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java Circle
Rectangle Area = 50.0
Circle Area = 153.94
```

7. Employee Hierarchy
Scenario: A company has regular and contractual employees with different pay structures.
Problem Statement:
Create a superclass Employee with name and basicSalary. Subclass RegularEmployee adds HRA 10%, Subclass ContractEmployee adds allowance 5%. Display net salary.
Classes/Fields:
● Employee → name, basicSalary
● RegularEmployee → HRA 10%
● ContractEmployee → allowance 5%
Sample Input:
Regular → name=Rahul, basicSalary=20000
Contract → name=Riya, basicSalary=15000
Sample Output:
Rahul Net Salary = 22000
Riya Net Salary = 15750

```java
class Worker {
    String name;
    double basicSalary;

    Worker(String n, double s) {
        name = n;
        basicSalary = s;
    }

    double getNetSalary() {
        return basicSalary;
    }
}

class RegularWorker extends Worker {
    RegularWorker(String n, double s) {
        super(n, s);
    }

    double getNetSalary() {
        return basicSalary + 0.1 * basicSalary;
    }
}

class ContractWorker extends Worker {
    ContractWorker(String n, double s) {
        super(n, s);
    }

    double getNetSalary() {
        return basicSalary + 0.05 * basicSalary;
    }

    public static void main(String[] args) {
        RegularWorker r = new RegularWorker("Rahul", 20000);
        ContractWorker c = new ContractWorker("Riya", 15000);
        System.out.println(r.name + " Net Salary = " + r.getNetSalary());
        System.out.println(c.name + " Net Salary = " + c.getNetSalary());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac ContractWorker.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java ContractWorker
Rahul Net Salary = 22000.0
Riya Net Salary = 15750.0
```

8. Vehicle Types
Scenario: You want to categorize vehicles.
Problem Statement:
Create a superclass Vehicle with brand, speed. Create subclasses Car and Bike with additional modelType. Display details of each vehicle.
Classes/Fields:
● Vehicle → brand, speed
● Car → modelType
● Bike → modelType
Sample Input:
Car → brand=Honda, speed=180, modelType=Civic
Bike → brand=Yamaha, speed=120, modelType=R15
Sample Output:
Car → Honda Civic, Speed=180
Bike → Yamaha R15, Speed=120

```java
class Vehicle {
    String brand;
    int speed;

    Vehicle(String b, int s) {
        brand = b;
        speed = s;
    }
}

class Car extends Vehicle {
    String modelType;

    Car(String b, int s, String m) {
        super(b, s);
        modelType = m;
    }

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

class Bike extends Vehicle {
    String modelType;

    Bike(String b, int s, String m) {
        super(b, s);
        modelType = m;
    }

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

    public static void main(String[] args) {
        Car car = new Car("Honda", 180, "Civic");
        Bike bike = new Bike("Yamaha", 120, "R15");
        car.display();
        bike.display();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac Bike.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java Bike
Car ? Honda Civic, Speed=180
Bike ? Yamaha R15, Speed=120
```

9. Animal Sound
Scenario: You are building a zoo management system to play animal sounds.
Problem Statement:
Create a superclass Animal with method makeSound(). Subclass Dog and Cat override makeSound().
Classes/Fields:
● Animal → makeSound()
● Dog → "Bark"
● Cat → "Meow"
Sample Output:
Dog → Bark
Cat → Meow

```java
class Animal {
    void makeSound() {}
}

class Dog extends Animal {
    void makeSound() {
        System.out.println("Dog → Bark");
    }
}

class Cat extends Animal {
    void makeSound() {
        System.out.println("Cat → Meow");
    }

    public static void main(String[] args) {
        Dog d = new Dog();
        Cat c = new Cat();
        d.makeSound();
        c.makeSound();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac Cat.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java Cat
Dog ? Bark
Cat ? Meow
```

10. Academic Staff
Scenario: University has teaching and non-teaching staff.
Problem Statement:
Superclass Staff with name, salary. Subclass TeachingStaff adds subject, Subclass NonTeachingStaff adds department. Display staff info.
Classes/Fields:
- Staff → name, salary
- TeachingStaff → subject
- NonTeachingStaff → department

Sample Input:
Teaching → name=Anita, salary=50000, subject=Math
NonTeaching → name=Ramesh, salary=40000, department=Admin
Sample Output:
Anita → Math, 50000
Ramesh → Admin, 40000

```java
class Staff {
    String name;
    double salary;

    Staff(String n, double s) {
        name = n;
        salary = s;
    }
}

class TeachingStaff extends Staff {
    String subject;

    TeachingStaff(String n, double s, String sub) {
        super(n, s);
        subject = sub;
    }

    void display() {
        System.out.println(name + " ? " + subject + ", " + salary);
    }
}

class NonTeachingStaff extends Staff {
    String department;

    NonTeachingStaff(String n, double s, String dept) {
        super(n, s);
        department = dept;
    }

    void display() {
        System.out.println(name + " ? " + department + ", " + salary);
    }

    public static void main(String[] args) {
        TeachingStaff t = new TeachingStaff("Anita", 50000, "Math");
        NonTeachingStaff n = new NonTeachingStaff("Ramesh", 40000, "Admin");
        t.display();
        n.display();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac NonTeachingStaff.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java NonTeachingStaff
Anita ? Math, 50000.0
Ramesh ? Admin, 40000.0
```

11. Bank Account Types
Scenario: Bank provides different account types.
Problem Statement:
Superclass Account → accountNo, balance. Subclass SavingAccount → interestRate. Subclass CurrentAccount → overdraftLimit. Display account details.
Classes/Fields:
- Account → accountNo, balance
- SavingAccount → interestRate
- CurrentAccount → overdraftLimit

Sample Input:
Saving → accountNo=101, balance=5000, interestRate=5%
Current → accountNo=102, balance=10000, overdraftLimit=2000
Sample Output:
Saving → 101, Balance=5000, Interest=5%
Current → 102, Balance=10000, Overdraft=2000

```java
class Account {
    int accountNo;
    double balance;
    Account(int accountNo, double balance) {
        this.accountNo = accountNo;
        this.balance = balance;
    }
}
class SavingAccount extends Account {
    double interestRate;
    SavingAccount(int accountNo, double balance, double interestRate) {
        super(accountNo, balance);
        this.interestRate = interestRate;
    }
    void display() {
        System.out.println("Saving → " + accountNo + ", Balance=" + balance + ", Interest=" + interestRate + "%");
    }
}
class CurrentAccount extends Account {
    double overdraftLimit;
    CurrentAccount(int accountNo, double balance, double overdraftLimit) {
        super(accountNo, balance);
        this.overdraftLimit = overdraftLimit;
    }
    void display() {
        System.out.println("Current → " + accountNo + ", Balance=" + balance + ", Overdraft=" + overdraftLimit);
    }
}
public class BankAccountTypes {
    public static void main(String[] args) {
        SavingAccount s = new SavingAccount(101, 5000, 5);
        CurrentAccount c = new CurrentAccount(102, 10000, 2000);
        s.display();
        c.display();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac BankAccountTypes.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java BankAccountTypes
Saving ? 101, Balance=5000.0, Interest=5.0%
Current ? 102, Balance=10000.0, Overdraft=2000.0
```

12. Payment System

Scenario: A company accepts different payment modes.

Problem Statement:

Create an abstract class Payment with abstract method pay(). Create subclasses CreditCardPayment and UPIPayment that implement pay().

Classes/Fields:

- Payment → pay() (abstract)
- CreditCardPayment → cardNumber, amount
- UPIPayment → upiId, amount

Sample Input:

Credit Card → cardNumber=1234567890123456, amount=5000
UPI → upiId=rahul@upi, amount=2000

Sample Output:

Payment via Credit Card 1234567890123456 → Rs. 5000 Paid
Payment via UPI rahul@upi → Rs. 2000 Paid

```java
abstract class Payment {
    abstract void pay();
}
class CreditCardPayment extends Payment {
    String cardNumber;
    double amount;
    CreditCardPayment(String cardNumber, double amount) {
        this.cardNumber = cardNumber;
        this.amount = amount;
    }
    void pay() {
        System.out.println("Payment via Credit Card " + cardNumber + " ? Rs. " + amount + " Paid");
    }
}
class UPIPayment extends Payment {
    String upiId;
    double amount;
    UPIPayment(String upiId, double amount) {
        this.upiId = upiId;
        this.amount = amount;
    }
    void pay() {
        System.out.println("Payment via UPI " + upiId + " ? Rs. " + amount + " Paid");
    }
}
public class PaymentSystem {
    public static void main(String[] args) {
        CreditCardPayment c = new CreditCardPayment("1234567890123456", 5000);
        UPIPayment u = new UPIPayment("rahul@upi", 2000);
        c.pay();
        u.pay();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac PaymentSystem.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java PaymentSystem
Payment via Credit Card 1234567890123456 ? Rs. 5000.0 Paid
Payment via UPI rahul@upi ? Rs. 2000.0 Paid
```

13. Shape Drawing
Scenario: A graphics program needs to draw different shapes.
Problem Statement:
Create an abstract class Shape with abstract method draw(). Subclass Circle and Rectangle implement draw().
Classes/Fields:
- Shape → draw() (abstract)
- Circle → radius
- Rectangle → length, breadth

Sample Input:
Circle → radius=7
Rectangle → length=5, breadth=10
Sample Output:
Drawing Circle of radius 7
Drawing Rectangle of length 5 and breadth 10

```java
abstract class Shape {
    abstract void draw();
}
class Circle extends Shape {
    int radius;
    Circle(int radius) {
        this.radius = radius;
    }
    void draw() {
        System.out.println("Drawing Circle of radius " + radius);
    }
}
class Rectangle extends Shape {
    int length, breadth;
    Rectangle(int length, int breadth) {
        this.length = length;
        this.breadth = breadth;
    }
    void draw() {
        System.out.println("Drawing Rectangle of length " + length + " and breadth " + breadth);
    }
}
public class ShapeDrawing {
    public static void main(String[] args) {
        Circle c = new Circle(7);
        Rectangle r = new Rectangle(5, 10);
        c.draw();
        r.draw();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac ShapeDrawing.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java ShapeDrawing
Drawing Circle of radius 7
Drawing Rectangle of length 5 and breadth 10
```

14. Employee Bonus Calculation

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

Problem Statement:

Create an abstract class Employee with abstract method calculateBonus(). Subclass Manager → bonus=20% of salary, Subclass Developer → bonus=10% of salary.

Classes/Fields:

- Employee → name, salary, calculateBonus() (abstract)
- Manager → bonus=20% of salary
- Developer → bonus=10% of salary

Sample Input:

Manager → name=Anita, salary=50000
Developer → name=Rohit, salary=40000

Sample Output:

Anita Bonus = 10000
Rohit Bonus = 4000

```java
abstract class Employee {
    String name;
    double salary;
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    abstract void calculateBonus();
}
class Manager extends Employee {
    Manager(String name, double salary) {
        super(name, salary);
    }
    void calculateBonus() {
        System.out.println(name + " Bonus = " + (salary * 0.2));
    }
}
class Developer extends Employee {
    Developer(String name, double salary) {
        super(name, salary);
    }
    void calculateBonus() {
        System.out.println(name + " Bonus = " + (salary * 0.1));
    }
}
public class EmployeeBonusCalculation {
    public static void main(String[] args) {
        Manager m = new Manager("Anita", 50000);
        Developer d = new Developer("Rohit", 40000);
        m.calculateBonus();
        d.calculateBonus();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac EmployeeBonusCalculation.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java EmployeeBonusCalculation
Anita Bonus = 10000.0
Rohit Bonus = 4000.0
```

15. Shape Area Calculation

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

Problem Statement:

Create a class ShapeArea with overloaded methods calculateArea().

Methods:

- calculateArea(int side) → calculates area of square
- calculateArea(int length, int breadth) → calculates area of rectangle
- calculateArea(double radius) → calculates area of circle

Sample Input:

Square → side=5
Rectangle → length=4, breadth=6
Circle → radius=3

Sample Output:

Square Area = 25
Rectangle Area = 24
Circle Area = 28.26

```java
class ShapeArea {
    void calculateArea(int side) {
        System.out.println("Square Area = " + (side * side));
    }
    void calculateArea(int length, int breadth) {
        System.out.println("Rectangle Area = " + (length * breadth));
    }
    void calculateArea(double radius) {
        System.out.println("Circle Area = " + (3.14 * radius * radius));
    }
}
public class ShapeAreaCalculation {
    public static void main(String[] args) {
        ShapeArea s = new ShapeArea();
        s.calculateArea(5);
        s.calculateArea(4, 6);
        s.calculateArea(3.0);
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac ShapeAreaCalculation.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java ShapeAreaCalculation
Square Area = 25
Rectangle Area = 24
Circle Area = 28.259999999999998
```

16. Employee Salary Display

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

Problem Statement:

Create class Employee with method displaySalary(). Subclass Manager and Developer override displaySalary() to include bonus.

Classes/Fields:

- Employee → name, salary, displaySalary() prints salary
- Manager → overrides displaySalary() → adds 20% bonus
- Developer → overrides displaySalary() → adds 10% bonus

Sample Input:

Manager → name=Anita, salary=50000
Developer → name=Rohit, salary=40000

Sample Output:

Anita Total Salary = 60000
Rohit Total Salary = 44000

```java
class Employee {
    String name;
    double salary;
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    void displaySalary() {
        System.out.println(name + " Total Salary = " + salary);
    }
}
class Manager extends Employee {
    Manager(String name, double salary) {
        super(name, salary);
    }
    void displaySalary() {
        System.out.println(name + " Total Salary = " + (salary + salary * 0.2));
    }
}
class Developer extends Employee {
    Developer(String name, double salary) {
        super(name, salary);
    }
    void displaySalary() {
        System.out.println(name + " Total Salary = " + (salary + salary * 0.1));
    }
}
public class EmployeeSalaryDisplay {
    public static void main(String[] args) {
        Manager m = new Manager("Anita", 50000);
        Developer d = new Developer("Rohit", 40000);
        m.displaySalary();
        d.displaySalary();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac EmployeeSalaryDisplay.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java EmployeeSalaryDisplay
Anita Total Salary = 60000.0
Rohit Total Salary = 44000.0
```

17. Vehicle Speed Display

Scenario: Vehicle management system needs to display speed differently for different vehicle types.

Problem Statement:

Create class Vehicle with method displaySpeed(). Subclass Car and Bike override it.

Classes/Fields:

- Vehicle → displaySpeed() prints "Vehicle speed unknown"
- Car → overrides displaySpeed() → "Car speed 120 km/h"
- Bike → overrides displaySpeed() → "Bike speed 80 km/h"

Sample Input:

Car
Bike

Sample Output:

Car speed 120 km/h
Bike speed 80 km/h

```java
class Vehicle {
    void displaySpeed() {
        System.out.println("Vehicle speed unknown");
    }
}
class Car extends Vehicle {
    void displaySpeed() {
        System.out.println("Car speed 120 km/h");
    }
}
class Bike extends Vehicle {
    void displaySpeed() {
        System.out.println("Bike speed 80 km/h");
    }
}
public class VehicleSpeedDisplay {
    public static void main(String[] args) {
        Car c = new Car();
        Bike b = new Bike();
        c.displaySpeed();
        b.displaySpeed();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac VehicleSpeedDisplay.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java VehicleSpeedDisplay
Car speed 120 km/h
Bike speed 80 km/h
```

18. Payment Process

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

Problem Statement:

Create abstract class Payment with abstract method pay(). Subclass CreditCardPayment and UPIPayment implement pay().

Usage:

- Use Payment p reference → p = new CreditCardPayment(...) or p = new UPIPayment(...)
- Call p.pay() for runtime polymorphic behavior

Sample Input:

Credit Card → cardNumber=1234567890123456, amount=5000
UPI → upiId=rahul@upi, amount=2000

Sample Output:

Payment via Credit Card 1234567890123456 → Rs. 5000 Paid
Payment via UPI rahul@upi → Rs. 2000 Paid

```java
abstract class Payment {
    abstract void pay();
}
class CreditCardPayment extends Payment {
    String cardNumber;
    double amount;
    CreditCardPayment(String cardNumber, double amount) {
        this.cardNumber = cardNumber;
        this.amount = amount;
    }
    void pay() {
        System.out.println("Payment via Credit Card " + cardNumber + " ? Rs. " + amount + " Paid");
    }
}
class UPIPayment extends Payment {
    String upiId;
    double amount;
    UPIPayment(String upiId, double amount) {
        this.upiId = upiId;
        this.amount = amount;
    }
    void pay() {
        System.out.println("Payment via UPI " + upiId + " ? Rs. " + amount + " Paid");
    }
}
public class PaymentProcess {
    public static void main(String[] args) {
        Payment p;
        p = new CreditCardPayment("1234567890123456", 5000);
        p.pay();
        p = new UPIPayment("rahul@upi", 2000);
        p.pay();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac PaymentProcess.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java PaymentProcess
Payment via Credit Card 1234567890123456 ? Rs. 5000.0 Paid
Payment via UPI rahul@upi ? Rs. 2000.0 Paid
```

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.

Problem Statement:

Create a superclass BankAccount with:

- Fields: accountNumber, accountHolder, balance
- Method: displayBalance()

Create subclasses:

- SavingsAccount → field: interestRate, method: calculateInterest()
- CurrentAccount → field: overdraftLimit, method: checkOverdraft()

Sample Input:

SavingsAccount → accountNumber=101, accountHolder=Ramesh, balance=5000, interestRate=5%
CurrentAccount → accountNumber=102, accountHolder=Anita, balance=2000, overdraftLimit=1000

Sample Output:

Ramesh → Balance=5000, Interest=250
Anita → Balance=2000, Overdraft Limit=1000

```java
class BankAccount {
    int accountNumber;
    String accountHolder;
    double balance;
    BankAccount(int accountNumber, String accountHolder, double balance) {
        this.accountNumber = accountNumber;
        this.accountHolder = accountHolder;
        this.balance = balance;
    }
    void displayBalance() {
        System.out.println(accountHolder + " ? Balance=" + balance);
    }
}
class SavingsAccount extends BankAccount {
    double interestRate;
    SavingsAccount(int accountNumber, String accountHolder, double balance, double interestRate) {
        super(accountNumber, accountHolder, balance);
        this.interestRate = interestRate;
    }
    void calculateInterest() {
        System.out.println(accountHolder + " ? Balance=" + balance + ", Interest=" + (balance * interestRate / 100));
    }
}
class CurrentAccount extends BankAccount {
    double overdraftLimit;
    CurrentAccount(int accountNumber, String accountHolder, double balance, double overdraftLimit) {
        super(accountNumber, accountHolder, balance);
        this.overdraftLimit = overdraftLimit;
    }
    void checkOverdraft() {
        System.out.println(accountHolder + " ? Balance=" + balance + ", Overdraft Limit=" + overdraftLimit);
    }
}
public class BankAccountTypesExtended {
    public static void main(String[] args) {
        SavingsAccount s = new SavingsAccount(101, "Ramesh", 5000, 5);
        CurrentAccount c = new CurrentAccount(102, "Anita", 2000, 1000);
        s.calculateInterest();
        c.checkOverdraft();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac BankAccountTypesExtended.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java BankAccountTypesExtended
Ramesh ? Balance=5000.0, Interest=250.0
Anita ? Balance=2000.0, Overdraft Limit=1000.0
```

20. College Staff Hierarchy

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

Problem Statement:

Create classes:

- Employee → name, salary, displaySalary()
- TeachingStaff → subject, overrides displaySalary()
- Professor → specialization, overrides displaySalary()
- Lecturer → department, overrides displaySalary()

Sample Input:

Professor → name=Dr. Sharma, salary=80000, subject=Math, specialization=Algebra
Lecturer → name=Ms. Mehta, salary=50000, subject=Physics, department=Science

Sample Output:

Dr. Sharma → Subject=Math, Specialization=Algebra, Salary=80000
Ms. Mehta → Subject=Physics, Department=Science, Salary=50000

```java
class Employee {
    String name;
    double salary;
    Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }
    void displaySalary() {
        System.out.println(name + " Salary=" + salary);
    }
}
class TeachingStaff extends Employee {
    String subject;
    TeachingStaff(String name, double salary, String subject) {
        super(name, salary);
        this.subject = subject;
    }
    void displaySalary() {
        System.out.println(name + " , Subject=" + subject + ", Salary=" + salary);
    }
}
class Professor extends TeachingStaff {
    String specialization;
    Professor(String name, double salary, String subject, String specialization) {
        super(name, salary, subject);
        this.specialization = specialization;
    }
    void displaySalary() {
        System.out.println(name + " , Subject=" + subject + ", Specialization=" + specialization + ", Salary=" + salary);
    }
}
class Lecturer extends TeachingStaff {
    String department;
    Lecturer(String name, double salary, String subject, String department) {
        super(name, salary, subject);
        this.department = department;
    }
    void displaySalary() {
        System.out.println(name + " , Subject=" + subject + ", Department=" + department + ", Salary=" + salary);
    }
}
public class CollegeStaffHierarchy {
    public static void main(String[] args) {
        Professor p = new Professor("Dr. Sharma", 80000, "Math", "Algebra");
        Lecturer l = new Lecturer("Ms. Mehta", 50000, "Physics", "Science");
        p.displaySalary();
        l.displaySalary();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac CollegeStaffHierarchy.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java CollegeStaffHierarchy
Dr. Sharma , Subject=Math, Specialization=Algebra, Salary=80000.0
Ms. Mehta , Subject=Physics, Department=Science, Salary=50000.0
```

21. Hospital Staff

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

Problem Statement:

- Staff → name, staffId, displayDetails()
- Doctor → specialization, displayDetails() override
- Nurse → shift, displayDetails() override

Sample Input:

Doctor → name=Dr. Reddy, staffId=101, specialization=Cardiology
Nurse → name=Nisha, staffId=102, shift=Night

Sample Output:

Dr. Reddy → Staff ID=101, Specialization=Cardiology
Nisha → Staff ID=102, Shift=Night

```java
class Staff {
    String name;
    int staffId;
    Staff(String name, int staffId) {
        this.name = name;
        this.staffId = staffId;
    }
    void displayDetails() {
        System.out.println(name + " Staff ID=" + staffId);
    }
}
class Doctor extends Staff {
    String specialization;
    Doctor(String name, int staffId, String specialization) {
        super(name, staffId);
        this.specialization = specialization;
    }
    void displayDetails() {
        System.out.println(name + " Staff ID=" + staffId + ", Specialization=" + specialization);
    }
}
class Nurse extends Staff {
    String shift;
    Nurse(String name, int staffId, String shift) {
        super(name, staffId);
        this.shift = shift;
    }
    void displayDetails() {
        System.out.println(name + " Staff ID=" + staffId + ", Shift=" + shift);
    }
}
public class HospitalStaff {
    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();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac HospitalStaff.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java HospitalStaff
Dr. Reddy ? Staff ID=101, Specialization=Cardiology
Nisha ? Staff ID=102, Shift=Night
```

22. Vehicle Types

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

Problem Statement:

- Interface LandVehicle → method driveOnLand()
- Interface WaterVehicle → method driveOnWater()
- Class AmphibiousVehicle implements both interfaces → provides both methods

Sample Input:

AmphibiousVehicle → name=HydroCar

Sample Output:

HydroCar → Driving on Land
HydroCar → Driving on Water

```java
interface LandVehicle {
    void driveOnLand();
}
interface WaterVehicle {
    void driveOnWater();
}
class AmphibiousVehicle implements LandVehicle, WaterVehicle {
    String name;
    AmphibiousVehicle(String name) {
        this.name = name;
    }
    public void driveOnLand() {
        System.out.println(name + " ? Driving on Land");
    }
    public void driveOnWater() {
        System.out.println(name + " ? Driving on Water");
    }
}
public class VehicleTypes {
    public static void main(String[] args) {
        AmphibiousVehicle v = new AmphibiousVehicle("HydroCar");
        v.driveOnLand();
        v.driveOnWater();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac VehicleTypes.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java VehicleTypes
HydroCar ? Driving on Land
HydroCar ? Driving on Water
```

23. School Members
Scenario: School has members: Teachers, Students, and Staff. All share common info.
Problem Statement:
● Member → name, id, displayInfo()
● Teacher → subject, overrides displayInfo()
● Student → grade, overrides displayInfo()
● Staff → department, overrides displayInfo()
Sample Input:
Teacher → name=Mr. Kumar, id=101, subject=English
Student → name=Riya, id=201, grade=10
Staff → name=Mr. Das, id=301, department=Maintenance
Sample Output:
Mr. Kumar → ID=101, Subject=English
Riya → ID=201, Grade=10
Mr. Das → ID=301, Department=Maintenance

```java
class Member {
    String name;
    int id;
    Member(String name, int id) {
        this.name = name;
        this.id = id;
    }
    void displayInfo() {
        System.out.println(name + " ID=" + id);
    }
}
class Teacher extends Member {
    String subject;
    Teacher(String name, int id, String subject) {
        super(name, id);
        this.subject = subject;
    }
    void displayInfo() {
        System.out.println(name + " ID=" + id + ", Subject=" + subject);
    }
}
class Student extends Member {
    int grade;
    Student(String name, int id, int grade) {
        super(name, id);
        this.grade = grade;
    }
    void displayInfo() {
        System.out.println(name + " ID=" + id + ", Grade=" + grade);
    }
}
class Staff extends Member {
    String department;
    Staff(String name, int id, String department) {
        super(name, id);
        this.department = department;
    }
    void displayInfo() {
        System.out.println(name + " ID=" + id + ", Department=" + department);
    }
}
public class SchoolMembers {
    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();
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac SchoolMembers.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java SchoolMembers
Mr. Kumar ? ID=101, Subject=English
Riya ? ID=201, Grade=10
Mr. Das ? ID=301, Department=Maintenance
```

24. Payment Gateway
Scenario: An e-commerce platform supports multiple payment methods like CreditCard and PayPal. All payments must implement a pay() method.
Problem Statement:
● Create an interface Payment → method pay(double amount)
● Classes CreditCardPayment and PayPalPayment implement Payment → provide their own pay() implementation
● In main(), take payment amount and process payment using both methods
Sample Input:
CreditCardPayment → amount=2500
PayPalPayment → amount=1500
Sample Output:
Processing Credit Card Payment of 2500
Processing PayPal Payment of 1500

```java
interface Payment {
    void pay(double amount);
}
class CreditCardPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Processing Credit Card Payment of " + amount);
    }
}
class PayPalPayment implements Payment {
    public void pay(double amount) {
        System.out.println("Processing PayPal Payment of " + amount);
    }
}
public class PaymentGateway {
    public static void main(String[] args) {
        Payment c = new CreditCardPayment();
        Payment p = new PayPalPayment();
        c.pay(2500);
        p.pay(1500);
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac PaymentGateway.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java PaymentGateway
Processing Credit Card Payment of 2500.0
Processing PayPal Payment of 1500.0
```

25. Media Player
Scenario: A media player can play both Audio and Video files.
Problem Statement:
● Interface AudioPlayer → method playAudio(String song)
● Interface VideoPlayer → method playVideo(String movie)
● Class MediaPlayer implements both → provides implementation for both methods
Sample Input:
Audio → song="Shape of You"
Video → movie="Inception"
Sample Output:
Playing Audio: Shape of You
Playing Video: Inception

```java
interface AudioPlayer {
    void playAudio(String song);
}
interface VideoPlayer {
    void playVideo(String movie);
}
class MediaPlayer implements AudioPlayer, VideoPlayer {
    public void playAudio(String song) {
        System.out.println("Playing Audio: " + song);
    }
    public void playVideo(String movie) {
        System.out.println("Playing Video: " + movie);
    }
}
public class MediaPlayerApp {
    public static void main(String[] args) {
        MediaPlayer m = new MediaPlayer();
        m.playAudio("Shape of You");
        m.playVideo("Inception");
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac MediaPlayerApp.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java MediaPlayerApp
Playing Audio: Shape of You
Playing Video: Inception
```

26. Smart Devices
Scenario: Smart devices can perform actions like calling, messaging, and browsing internet.
Problem Statement:
● Interface Callable → method makeCall(String number)
● Interface Messaging → method sendMessage(String number, String message)
● Interface Internet → method browse(String website)
● Class SmartPhone implements all three interfaces → provide respective implementations
Sample Input:
Call → number="9876543210"
Message → number="9876543210", message="Hello!"
Browse → website="www.google.com"
Sample Output:
Calling 9876543210
Sending message to 9876543210: Hello!
Browsing website: www.google.com

```java
interface Callable {
    void makeCall(String number);
}
interface Messaging {
    void sendMessage(String number, String message);
}
interface Internet {
    void browse(String website);
}
class SmartPhone implements Callable, Messaging, Internet {
    public void makeCall(String number) {
        System.out.println("Calling " + number);
    }
    public void sendMessage(String number, String message) {
        System.out.println("Sending message to " + number + ": " + message);
    }
    public void browse(String website) {
        System.out.println("Browsing website: " + website);
    }
}
public class SmartDevices {
    public static void main(String[] args) {
        SmartPhone s = new SmartPhone();
        s.makeCall("9876543210");
        s.sendMessage("9876543210", "Hello!");
        s.browse("www.google.com");
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac SmartDevices.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java SmartDevices
Calling 9876543210
Sending message to 9876543210: Hello!
Browsing website: www.google.com
```

27. Shape Area Calculator

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

Problem Statement:

● Interface Shape → method calculateArea()
● Classes Circle, Rectangle, Square implement Shape → provide specific area calculation
● In main(), create objects of each shape, input dimensions, display calculated area

Sample Input:

Circle → radius=5
Rectangle → length=10, breadth=5
Square → side=4

Sample Output:

Circle Area = 78.5
Rectangle Area = 50
Square Area = 16

```java
interface Shape {
    double calculateArea();
}
class Circle implements Shape {
    double radius;
    Circle(double radius) {
        this.radius = radius;
    }
    public double calculateArea() {
        return 3.14 * radius * radius;
    }
}
class Rectangle implements Shape {
    double length, breadth;
    Rectangle(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
    }
    public double calculateArea() {
        return length * breadth;
    }
}
class Square implements Shape {
    double side;
    Square(double side) {
        this.side = side;
    }
    public double calculateArea() {
        return side * side;
    }
}
public class ShapeAreaCalculator {
    public static void main(String[] args) {
        Circle c = new Circle(5);
        Rectangle r = new Rectangle(10, 5);
        Square s = new Square(4);
        System.out.println("Circle Area = " + c.calculateArea());
        System.out.println("Rectangle Area = " + r.calculateArea());
        System.out.println("Square Area = " + s.calculateArea());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac ShapeAreaCalculator.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java ShapeAreaCalculator
Circle Area = 78.5
Rectangle Area = 50.0
Square Area = 16.0
```

28. Online Shopping Cart System

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

Problem Statement:

● **Class Product** → instance variables: productId, name, price (Encapsulation: use private variables with getters/setters)

● **Abstract Class CartItem** → method calculateTotalPrice() (Abstract Class: define generic behavior for cart items)

● **Class Cart extends CartItem** → store list of products, implement calculateTotalPrice()

● **Interface Discountable** → method applyDiscount(double percentage) (Interface: any item can have discounts applied)

In main(), create a cart, add 3 products, apply 10% discount to one product, display total cost

Sample Input:

Product1 → name="Laptop", price=50000

Product2 → name="Mouse", price=500

Product3 → name="Keyboard", price=1200

Sample Output:

Applying 10% discount to Laptop

Total Cart Price = 51800

```java
import java.util.*;
class Product {
    private int productId;
    private String name;
    private double price;
    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 class CartItem {
    abstract double calculateTotalPrice();
}
interface Discountable {
    void applyDiscount(double percentage);
}
class Cart extends CartItem implements Discountable {
    List<Product> products = new ArrayList<>();
    void addProduct(Product p) {
        products.add(p);
    }
    public double calculateTotalPrice() {
        double total = 0;
        for(Product p: products) total += p.getPrice();
        return total;
    }
    public void applyDiscount(double percentage) {
        if(!products.isEmpty()) {
            Product p = products.get(0);
            System.out.println("Applying " + percentage + "% discount to " + p.getName());
            p.setPrice(p.getPrice() - (p.getPrice() * percentage / 100));
        }
    }
}
public class OnlineShoppingCartSystem {
    public static void main(String[] args) {
        Cart cart = new Cart();
        Product p1 = new Product(1, "Laptop", 50000);
        Product p2 = new Product(2, "Mouse", 500);
        Product p3 = new Product(3, "Keyboard", 1200);
        cart.addProduct(p1);
        cart.addProduct(p2);
        cart.addProduct(p3);
        cart.applyDiscount(10);
        System.out.println("Total Cart Price = " + cart.calculateTotalPrice());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac OnlineShoppingCartSystem.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java OnlineShoppingCartSystem
Applying 10.0% discount to Laptop
Total Cart Price = 46700.0
```

29. Employee Management System
Scenario: Manage employee details, calculate salaries, and differentiate employee types.
Problem Statement:
● **Abstract Class Employee** → instance variables: name, id
○ Abstract method calculateSalary() → different calculation for each type
● **Class PermanentEmployee extends Employee** → include basicSalary and hra → implement calculateSalary()
● **Class ContractEmployee extends Employee** → include hourlyRate and hoursWorked → implement calculateSalary()
● **Interface BonusEligible** → method calculateBonus() → applies only to permanent employees
● In main(), create 2 permanent and 2 contract employees, display salary + bonus if eligible
Sample Input:
PermanentEmployee → name="Amit", basicSalary=50000, hra=5000
ContractEmployee → name="Neha", hourlyRate=300, hoursWorked=100
Sample Output:
Amit Salary = 55000, Bonus = 5500
Neha Salary = 30000

```java
abstract class Employee {
    String name;
    int id;
    Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }
    abstract double calculateSalary();
}
interface BonusEligible {
    double calculateBonus();
}
class PermanentEmployee extends Employee implements BonusEligible {
    double basicSalary, hra;
    PermanentEmployee(String name, int id, double basicSalary, double hra) {
        super(name, id);
        this.basicSalary = basicSalary;
        this.hra = hra;
    }
    double calculateSalary() {
        return basicSalary + hra;
    }
    public double calculateBonus() {
        return calculateSalary() * 0.1;
    }
}
class ContractEmployee extends Employee {
    double hourlyRate;
    int hoursWorked;
    ContractEmployee(String name, int id, double hourlyRate, int hoursWorked) {
        super(name, id);
        this.hourlyRate = hourlyRate;
        this.hoursWorked = hoursWorked;
    }
    double calculateSalary() {
        return hourlyRate * hoursWorked;
    }
}
public class EmployeeManagementSystem {
    public static void main(String[] args) {
        PermanentEmployee p1 = new PermanentEmployee("Amit", 1, 50000, 5000);
        ContractEmployee c1 = new ContractEmployee("Neha", 2, 300, 100);
        System.out.println(p1.name + " Salary = " + p1.calculateSalary() + ", Bonus = " + p1.calculateBonus());
        System.out.println(c1.name + " Salary = " + c1.calculateSalary());
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac EmployeeManagementSystem.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java EmployeeManagementSystem
Amit Salary = 55000.0, Bonus = 5500.0
Neha Salary = 30000.0
```

30. Library Management System
Scenario: Manage books and library members with borrowing functionality.
Problem Statement:
● **Class Book** → private variables: bookId, title, author (Encapsulation)
● **Abstract Class LibraryMember** → instance variables: memberId, name
○ Abstract method borrowBook(Book book)
● **Class StudentMember extends LibraryMember** → limit 3 books
● **Class FacultyMember extends LibraryMember** → limit 5 books
● **Interface Notifyable** → method sendNotification(String message) → notify members about overdue books
● In main(), create 1 student and 1 faculty, borrow books, send notifications
Sample Input:
Student → borrow 2 books
Faculty → borrow 4 books
Sample Output:
StudentMember Amit borrowed 2 books
FacultyMember Prof. Singh borrowed 4 books
Notification sent to Amit: Return books within 7 days
Notification sent to Prof. Singh: Return books within 14 days

```java
import java.util.*;
class Book {
    private int bookId;
    private String title;
    private String author;
    Book(int bookId, String title, String author) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
    }
    public String getTitle() { return title; }
}
abstract class LibraryMember {
    int memberId;
    String name;
    List<Book> borrowedBooks = new ArrayList<>();
    LibraryMember(int memberId, String name) {
        this.memberId = memberId;
        this.name = name;
    }
    abstract void borrowBook(Book book);
}
interface Notifyable {
    void sendNotification(String message);
}
class StudentMember extends LibraryMember implements Notifyable {
    StudentMember(int memberId, String name) {
        super(memberId, name);
    }
    void borrowBook(Book book) {
        if(borrowedBooks.size() < 3) borrowedBooks.add(book);
    }
    public void sendNotification(String message) {
        System.out.println("Notification sent to " + name + ": " + message);
    }
}
class FacultyMember extends LibraryMember implements Notifyable {
    FacultyMember(int memberId, String name) {
        super(memberId, name);
    }
    void borrowBook(Book book) {
        if(borrowedBooks.size() < 5) borrowedBooks.add(book);
    }
    public void sendNotification(String message) {
        System.out.println("Notification sent to " + name + ": " + message);
    }
}
public class LibraryManagementSystem {
    public static void main(String[] args) {
        StudentMember s = new StudentMember(1, "Amit");
        FacultyMember f = new FacultyMember(2, "Prof. Singh");
        s.borrowBook(new Book(101, "Book1", "Author1"));
        s.borrowBook(new Book(102, "Book2", "Author2"));
        f.borrowBook(new Book(103, "Book3", "Author3"));
        f.borrowBook(new Book(104, "Book4", "Author4"));
        f.borrowBook(new Book(105, "Book5", "Author5"));
        f.borrowBook(new Book(106, "Book6", "Author6"));
        System.out.println("StudentMember " + s.name + " borrowed " + s.borrowedBooks.size() + " books");
        System.out.println("FacultyMember " + f.name + " borrowed " + f.borrowedBooks.size() + " books");
        s.sendNotification("Return books within 7 days");
        f.sendNotification("Return books within 14 days");
    }
}
```

```
PS C:\Users\baenu\Test\OOPJ Assignment 7> javac LibraryManagementSystem.java
PS C:\Users\baenu\Test\OOPJ Assignment 7> java LibraryManagementSystem
StudentMember Amit borrowed 2 books
FacultyMember Prof. Singh borrowed 4 books
Notification sent to Amit: Return books within 7 days
Notification sent to Prof. Singh: Return books within 14 days
```