Week 3 – 32 – Practise Problem Solution

Name: Ramesh Harisabapathi Chettiar

Roll Number: RA2411030010263

Course: Networking and Communications

Semester: 3

Date of Submission: 29/08/2025

PRACTISE PROBLEM 1:

Create a program that demonstrates the concept of classes and objects using a real-world analogy.

```java
/*Create a program that demonstrates the concept of classes and objects using a
real-world analogy. */
import java.util.Scanner;

public class Car{
    // TODO: Define instance variables (attributes):
    // - brand (String)
    // - model (String)
    // - year (int)
    // - color (String)
    // - isRunning (boolean)
    private String brand, model, color;
    private int year;
    private boolean isRunning;

    // TODO: Create a constructor that initializes all attributes
    public Car(String brand, String model, int year, String color){
        this.brand = brand;
        this.model = model;
        this.year = year;
        this.color = color;
        this.isRunning = false; // Car is initially off
    }

    // TODO: Create instance methods:
    // - startEngine() - sets isRunning to true, prints message
    // - stopEngine() - sets isRunning to false, prints message
    // - displayInfo() - prints all car information
    // - getAge() - returns current year minus car year
    public void startEngine(){
        if(!isRunning){
            isRunning = true;
            System.out.println("The engine has started.");
        } else {
```

```java
    public void startEngine(){
        if(!isRunning){
            isRunning = true;
            System.out.println("The engine has started.");
        } else {
            System.out.println("The engine is already running.");
        }
    }

    public void stopEngine(){
        if(isRunning){
            isRunning = false;
            System.out.println("The engine has stopped.");
        } else {
            System.out.println("The engine is already off.");
        }
    }

    public void displayInfo(){
        System.out.println("Car Information:");
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
        System.out.println("Color: " + color);
        System.out.println("Is Running: " + (isRunning ? "Yes" : "No"));
    }

    public int getAge(){
        int currentYear = java.util.Calendar.getInstance().get(java.util.Calendar.YEAR);
        return currentYear - year;
    }

```

```java
    public static void main(String[] args) {
        // TODO: Create 3 different Car objects with different attributes
        // TODO: Demonstrate calling methods on each object
        // TODO: Show how each object maintains its own state
        // TODO: Explain in comments: How is this similar to real-world cars?
        Car car1 = new Car("Toyota", "Camry", 2020, "Blue");
        Car car2 = new Car("Honda", "Civic", 2018, "Red");
        Car car3 = new Car("Ford", "Mustang", 2021, "Black");

        car1.startEngine();
        car1.displayInfo();
        System.out.println("Car Age: " + car1.getAge() + " years");
        car1.stopEngine();

        car2.startEngine();
        car2.displayInfo();
        System.out.println("Car Age: " + car2.getAge() + " years");
        car2.stopEngine();

        car3.startEngine();
        car3.displayInfo();
        System.out.println("Car Age: " + car3.getAge() + " years");
        car3.stopEngine();


        /* Each Car object represents a real-world car with specific attributes (brand, model, year, color)
         and behaviors (starting/stopping the engine, displaying information).
         Just like real cars, each Car object maintains its own state (e.g., whether it's running or not)
         and can perform actions independently of other Car objects. */
    }
}
```

OUTPUT

```
The engine has started.
Car Information:
Brand: Toyota
Model: Camry
Year: 2020
Color: Blue
Is Running: Yes
Car Age: 5 years
The engine has stopped.
The engine has started.
Car Information:
Brand: Honda
Model: Civic
Year: 2018
Color: Red
Is Running: Yes
Car Age: 7 years
The engine has stopped.
The engine has started.
Car Information:
Brand: Ford
Model: Mustang
Year: 2021
Color: Black
Is Running: Yes
Car Age: 4 years
```

PRACTISE PROBLEM 2:

Create a Student class that demonstrates proper class structure and object instantiation.

```java
/*Create a Student class that demonstrates proper class structure and object instantiation. */
public class Student {
        // TODO: Define private instance variables:
        // - studentId (String)
        // - name (String)
        // - grade (double)
        // - course (String)
        private String studentId, name, course;
        private double grade;

        // TODO: Create a default constructor (no parameters)
        public Student() {
            this.studentId = "";
            this.name = "";
            this.course = "";
            this.grade = 0.0;
        }

        // TODO: Create a parameterized constructor that accepts all attributes
        public Student(String studentId, String name, double grade, String course) {
            this.studentId = studentId;
            this.name = name;
            this.grade = grade;
            this.course = course;
        }

```


```java
/*Create a Student class that demonstrates proper class structure and object instantiation. */
public class Student {
        // TODO: Define private instance variables:
        // - studentId (String)
        // - name (String)
        // - grade (double)
        // - course (String)
        private String studentId, name, course;
        private double grade;

        // TODO: Create a default constructor (no parameters)
        public Student() {
            this.studentId = "";
            this.name = "";
            this.course = "";
            this.grade = 0.0;
        }

        // TODO: Create a parameterized constructor that accepts all attributes
        public Student(String studentId, String name, double grade, String course) {
            this.studentId = studentId;
            this.name = name;
            this.grade = grade;
            this.course = course;
        }

```

```java
    }

    public void setCourse(String course) {
        this.course = course;
    }


    // TODO: Create a method calculateLetterGrade() that returns:
    // A (90-100), B (80-89), C (70-79), D (60-69), F (below 60)
    public String calculateLetterGrade() {
        if (grade >= 90 && grade <= 100) {
            return "A";
        } else if (grade >= 80 && grade < 90) {
            return "B";
        } else if (grade >= 70 && grade < 80) {
            return "C";
        } else if (grade >= 60 && grade < 70) {
            return "D";
        } else {
            return "F";
        }
    }

    // TODO: Create a method displayStudent() that shows all information
    public void displayStudent() {
        System.out.println("Student ID: " + studentId);
        System.out.println("Name: " + name);
        System.out.println("Grade: " + grade);
        System.out.println("Course: " + course);
        System.out.println("Letter Grade: " + calculateLetterGrade());
    }

    public static void main(String[] args) {
        // TODO: Create one student using default constructor, then set values
        Student student1 = new Student();
        student1.setStudentId("RA2411030010263");
        student1.setName("RAMESH");
        student1.setGrade(93.5);
        student1.setCourse("Cryptography");

        // TODO: Create another student using parameterized constructor
        Student student2 = new Student("RA2411030010264", "RAVI", 92.0, "Digital Forensics");

        // TODO: Demonstrate all getter/setter methods
        System.out.println("Student 1 ID: " + student1.getStudentId());
        System.out.println("Student 1 Name: " + student1.getName());
        System.out.println("Student 1 Grade: " + student1.getGrade());
        System.out.println("Student 1 Course: " + student1.getCourse());

        student1.setGrade(90.0);
```

```
103             student1.setGrade(90.0);
104             System.out.println("Updated Student 1 G
105
106             // TODO: Show both students' informatio
107             student1.displayStudent();
108             student2.displayStudent();
109         }
110     }
```

OUTPUT:

```
Student 1 ID: RA2411030010263
Student 1 Name: RAMESH
Student 1 Grade: 93.5
Student 1 Course: Cryptography
Updated Student 1 Grade: 90.0
Student ID: RA2411030010263
Name: RAMESH
Grade: 90.0
Course: Cryptography
Letter Grade: A
Student ID: RA2411030010264
Name: RAVI
Grade: 92.0
Course: Digital Forensics
Letter Grade: A
```

PRACTISE PROBLEM 3:

Create a program that clearly demonstrates the difference between instance and static members.

```java
/*Create a program that clearly demonstrates the difference between instance and static
members. */
public class BankAccount {
    // TODO: Create static variables:
    // - bankName (String) - same for all accounts
    // - totalAccounts (int) - count of all accounts created
    // - interestRate (double) - same rate for all accounts
    static String bankName;
    static int totalAccounts = 0;
    static double interestRate;

    // TODO: Create instance variables:
    // - accountNumber (String) - unique for each account
    // - accountHolder (String) - unique for each account
    // - balance (double) - unique for each account
    String accountNumber;
    String accountHolder;
    double balance;

    // TODO: Create constructor that:
    // - Initializes instance variables
    // - Increments totalAccounts counter
    public BankAccount(String accountNumber, String accountHolder, double initialBalance) {
        this.accountNumber = accountNumber;
        this.accountHolder = accountHolder;
        this.balance = initialBalance;
        totalAccounts++;
```

```
        totalAccounts++;
    }

    // TODO: Create static methods:
    // - setBankName(String name)
    // - setInterestRate(double rate)
    // - getTotalAccounts() - returns count
    // - displayBankInfo() - shows bank name and total accounts
    static void setBankName(String name) {
        bankName = name;
    }

    static void setInterestRate(double rate) {
        interestRate = rate;
    }

    static int getTotalAccounts() {
        return totalAccounts;
    }

    static void displayBankInfo() {
        System.out.println("Bank Name: " + bankName);
        System.out.println("Total Accounts: " + totalAccounts);
    }
```

```
    // TODO: Create instance methods:
    // - deposit(double amount)
    // - withdraw(double amount)
    // - calculateInterest() - uses static interestRate
    // - displayAccountInfo()
    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        balance -= amount;
    }

    public double calculateInterest() {
        return balance * (interestRate / 100);
    }

    public void displayAccountInfo() {
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Account Holder: " + accountHolder);
        System.out.println("Balance: " + balance);
    }
```

```java
    public static void main(String[] args) {
        // TODO: Set bank name and interest rate using static methods
        BankAccount.setBankName("My Bank");
        BankAccount.setInterestRate(5.0);

    // TODO: Create multiple BankAccount objects
    BankAccount account1 = new BankAccount("12345", "Alice", 1000.0);
    BankAccount account2 = new BankAccount("67890", "Bob", 2000.0);
    BankAccount account3 = new BankAccount("54321", "Charlie", 3000.0);

    // TODO: Show that static members are shared across all objects
    System.out.println("Bank Name: " + BankAccount.bankName);
    System.out.println("Total Accounts: " + BankAccount.totalAccounts);
    System.out.println("Interest Rate: " + BankAccount.interestRate);

    // TODO: Show that instance members are unique to each object
    System.out.println("Account 1 Holder: " + account1.accountHolder);
    System.out.println("Account 2 Holder: " + account2.accountHolder);
    System.out.println("Account 3 Holder: " + account3.accountHolder);

    // TODO: Demonstrate calling static methods with and without objects
    BankAccount.displayBankInfo();
    System.out.println("Total Accounts (via instance): " + account1.getTotalAccounts());

    }
```

OUTPUT:

```
Bank Name: My Bank
Total Accounts: 3
Interest Rate: 5.0
Account 1 Holder: Alice
Account 2 Holder: Bob
Account 3 Holder: Charlie
Bank Name: My Bank
Total Accounts: 3
Total Accounts (via instance): 3
```

PRACTISE PROBLEM 4:

Create a base class and demonstrate how OOP promotes code reusability.

```java
/*Create a base class and demonstrate how OOP promotes code reusability. */
public class Vehicle{
    // TODO: Create protected instance variables:
    // - make (String)
    // - model (String)
    // - year (int)
    // - fuelLevel (double)
    protected String make, model;
    protected int year;
    protected double fuelLevel;

    // TODO: Create constructor
    Vehicle(String make, String model, int year, double fuelLevel) {
        this.make = make;
        this.model = model;
        this.year = year;
        this.fuelLevel = fuelLevel;
    }

    // TODO: Create common methods:
    // - startVehicle()
    // - stopVehicle()
    // - refuel(double amount)
    // - displayVehicleInfo()
    public void startVehicle() {
        System.out.println("The vehicle is starting.");
    }

    public void stopVehicle() {
        System.out.println("The vehicle is stopping.");
    }

    public void refuel(double amount) {
        fuelLevel += amount;
        System.out.println("The vehicle has been refueled.");
    }

    public void displayVehicleInfo() {
        System.out.println("Vehicle Information:");
        System.out.println("Make: " + make);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
        System.out.println("Fuel Level: " + fuelLevel + " liters");
    }

```

```java
    public static void main(String[] args) {
        // TODO: Create different types of vehicles (Car, Truck,Motorcycle)
        Vehicle car = new Vehicle("Tata", "Safari", 2020, 50.0);
        Vehicle truck = new Vehicle("Mahindra", "Bolero", 2019, 80.0);
        Vehicle motorcycle = new Vehicle("Maruti Suzuki", "Wagon-R", 2021, 15.0);
        // TODO: Show how the same Vehicle class can be reused

        // TODO: Create an array of Vehicle objects
        Vehicle[] vehicles = {car, truck, motorcycle};

        // TODO: Demonstrate polymorphic behavior
        for (Vehicle vehicle : vehicles) {
            vehicle.startVehicle();
            vehicle.displayVehicleInfo();
            vehicle.stopVehicle();
        }

        // TODO: In comments, explain:
        // - How does this show reusability?
        // The Vehicle class can be reused to create different types of vehicles (car, truck, motorcycl
        // without duplicating code. Common behaviors are defined in the base class.

        // - What are the benefits over writing separate classes?
        // 1. Code Duplication: Without a base class, common code would need to be duplicated across
        //    multiple vehicle classes, making maintenance harder.
        // 2. Easier Maintenance: Changes to common behavior only need to be made in one place (the bas
        // 3. Polymorphism: The same interface can be used to interact with different vehicle types.
```

OUTPUT:

```
The vehicle is starting.
Vehicle Information:
Make: Tata
Model: Safari
Year: 2020
Fuel Level: 50.0 liters
The vehicle is stopping.
The vehicle is starting.
Vehicle Information:
Make: Mahindra
Model: Bolero
Year: 2019
Fuel Level: 80.0 liters
The vehicle is stopping.
The vehicle is starting.
Vehicle Information:
Year: 2019
Fuel Level: 80.0 liters
The vehicle is stopping.
The vehicle is starting.
Vehicle Information:
The vehicle is stopping.
The vehicle is starting.
Vehicle Information:
The vehicle is starting.
Vehicle Information:
Vehicle Information:
Make: Maruti Suzuki
Make: Maruti Suzuki
Model: Wagon-R
Year: 2021
Fuel Level: 15.0 liters
```