

�� **PRACTICE PROBLEM 1: (Any 4)**

**Understanding Classes and Objects - Real World Analogy**

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

public class Car {

// TODO: Define instance variables (attributes):

// - brand (String)

// - model (String)

// - year (int)

// - color (String)

// - isRunning (boolean)

// TODO: Create a constructor that initializes all attributes

// 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 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?

}

}



Program –

// Real-world analogy: A Car blueprint (class) and actual cars (objects)

public class Car {

// Instance variables (attributes of each car)

String brand;

String model;

int year;

String color;

boolean isRunning;

// Constructor: initializes all attributes when creating a Car object

Car(String brand, String model, int year, String color) {

this.brand = brand;

this.model = model;

this.year = year;

this.color = color;

this.isRunning = false; // by default, car is not running

}

// Method to start the engine

void startEngine() {

isRunning = true;

System.out.println(brand + " " + model + " engine started.");

}

// Method to stop the engine

void stopEngine() {

isRunning = false;

System.out.println(brand + " " + model + " engine stopped.");

}

// Method to display all car information

void displayInfo() {

System.out.println("Car Info:");

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);

System.out.println();

}

// Method to calculate the car's age

int getAge(int currentYear) {

return currentYear - year;

}

// Main method - program starts here

public static void main(String[] args) {

// Creating 3 different Car objects (like real cars)

Car car1 = new Car("Toyota", "Corolla", 2015, "Red");

Car car2 = new Car("Honda", "Civic", 2018, "Blue");

Car car3 = new Car("Tesla", "Model 3", 2022, "White");

// Demonstrating calling methods

car1.displayInfo();

car1.startEngine();

car1.displayInfo();

System.out.println("Car Age: " + car1.getAge(2025) + " years\n");

car2.displayInfo();

car2.startEngine();

car2.stopEngine();

System.out.println("Car Age: " + car2.getAge(2025) + " years\n");

car3.displayInfo();

car3.startEngine();

System.out.println("Car Age: " + car3.getAge(2025) + " years\n");

/\*

Real-world analogy explanation:

- The 'Car' class is like a blueprint for building cars.

- Each Car object (car1, car2, car3) is an actual car with its own details.

- Even though they share the same blueprint, each car has its own state

(different brand, model, year, color, and isRunning value).

- Just like in the real world, one car can be running while another is stopped.

\*/

}

}

Output –



�� **PRACTICE PROBLEM 2:**

**Class Definition and Object Creation**

**Task**: 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)

// TODO: Create a default constructor (no parameters)

// TODO: Create a parameterized constructor that accepts all attributes // TODO: Create getter and setter methods for all attributes

// TODO: Create a method calculateLetterGrade() that returns: // A (90-100), B (80-89), C (70-79), D (60-69), F (below 60)

// TODO: Create a method displayStudent() that shows all information

public static void main(String[] args) {

// TODO: Create one student using default constructor, then set values

// TODO: Create another student using parameterized constructor // TODO: Demonstrate all getter/setter methods

// TODO: Show both students' information and letter grades }

}

S

Program –

public class Student {

// Private instance variables (Encapsulation principle)

private String studentId;

private String name;

private double grade;

private String course;

// Default constructor (no parameters)

public Student() {

// values remain uninitialized until set using setters

}

// Parameterized constructor (initializes all attributes)

public Student(String studentId, String name, double grade, String course) {

this.studentId = studentId;

this.name = name;

this.grade = grade;

this.course = course;

}

// Getter and Setter methods for all attributes

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getGrade() {

return grade;

}

public void setGrade(double grade) {

this.grade = grade;

}

public String getCourse() {

return course;

}

public void setCourse(String course) {

this.course = course;

}

// Method to calculate letter grade based on numeric grade

public String calculateLetterGrade() {

if (grade >= 90) return "A";

else if (grade >= 80) return "B";

else if (grade >= 70) return "C";

else if (grade >= 60) return "D";

else return "F";

}

// Method to display student information

public void displayStudent() {

System.out.println("Student ID: " + studentId);

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

System.out.println("Course: " + course);

System.out.println("Grade: " + grade);

System.out.println("Letter Grade: " + calculateLetterGrade());

System.out.println();

}

// Main method

public static void main(String[] args) {

// Student created using default constructor + setters

Student s1 = new Student();

s1.setStudentId("S001");

s1.setName("Alice");

s1.setGrade(92.5);

s1.setCourse("Computer Science");

// Student created using parameterized constructor

Student s2 = new Student("S002", "Bob", 76.3, "Mathematics");

// Demonstrate getter usage

System.out.println("Getter Example: " + s1.getName() + " is enrolled in " + s1.getCourse());

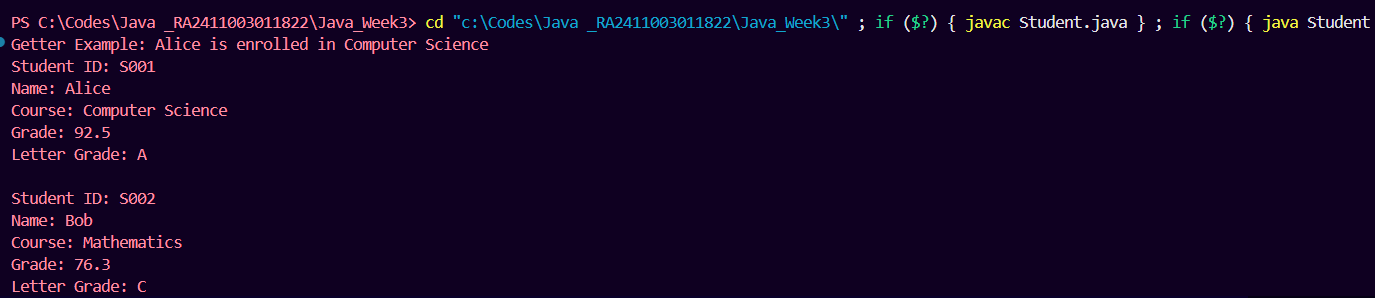
// Display both students' information

s1.displayStudent();

s2.displayStudent();

}

}

Output - 

�� **PRACTICE PROBLEM 3:**

**Instance vs Static (Class) Members**

**Task**: 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

// TODO: Create instance variables:

// - accountNumber (String) - unique for each account

// - accountHolder (String) - unique for each account

// - balance (double) - unique for each account

// TODO: Create constructor that:

// - Initializes instance variables

// - Increments totalAccounts counter

// TODO: Create static methods:

// - setBankName(String name)

// - setInterestRate(double rate)

// - getTotalAccounts() - returns count

// - displayBankInfo() - shows bank name and total accounts

// TODO: Create instance methods:

// - deposit(double amount)

// - withdraw(double amount)

// - calculateInterest() - uses static interestRate

// - displayAccountInfo()

public static void main(String[] args) {

// TODO: Set bank name and interest rate using static methods // TODO: Create multiple BankAccount objects

// TODO: Show that static members are shared across all objects // TODO: Show that instance members are unique to each object



// TODO: Demonstrate calling static methods with and without objects

}

}

Program –

public class BankAccount {

// Static variables (shared by ALL accounts, only one copy in memory)

static String bankName;

static int totalAccounts = 0;

static double interestRate;

// Instance variables (each account has its own copy)

String accountNumber;

String accountHolder;

double balance;

// Constructor: initializes instance variables and increments static counter

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

this.accountNumber = accountNumber;

this.accountHolder = accountHolder;

this.balance = balance;

totalAccounts++; // Every new account increases total count

}

// Static methods (operate on static data, not tied to any one account)

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);

System.out.println("Interest Rate: " + interestRate + "%");

System.out.println();

}

// Instance methods (operate on individual account data)

void deposit(double amount) {

balance += amount;

System.out.println(accountHolder + " deposited " + amount + ". New Balance: " + balance);

}

void withdraw(double amount) {

if (amount <= balance) {

balance -= amount;

System.out.println(accountHolder + " withdrew " + amount + ". New Balance: " + balance);

} else {

System.out.println(accountHolder + " tried to withdraw " + amount + " but has insufficient balance.");

}

}

void calculateInterest() {

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

System.out.println(accountHolder + " earned interest: " + interest);

}

void displayAccountInfo() {

System.out.println("Account Number: " + accountNumber);

System.out.println("Account Holder: " + accountHolder);

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

System.out.println();

}

// Main method - where program runs

public static void main(String[] args) {

// Set static variables (same for all accounts)

BankAccount.setBankName("OpenAI National Bank");

BankAccount.setInterestRate(5.0);

// Create multiple accounts (each with unique data)

BankAccount acc1 = new BankAccount("1001", "Alice", 5000);

BankAccount acc2 = new BankAccount("1002", "Bob", 3000);

BankAccount acc3 = new BankAccount("1003", "Charlie", 7000);

// Display bank info (static data is shared)

BankAccount.displayBankInfo();

// Show individual account info

acc1.displayAccountInfo();

acc2.displayAccountInfo();

acc3.displayAccountInfo();

// Perform some operations

acc1.deposit(1000);

acc2.withdraw(500);

acc3.calculateInterest();

// Show total accounts using static method

System.out.println("Total accounts created: " + BankAccount.getTotalAccounts());

// Demonstrate calling static method with object (not recommended, but allowed)

acc1.displayBankInfo(); // works, but better to call as BankAccount.displayBankInfo()

/\*

Real-world analogy:

- Static members = properties shared by the whole bank (bank name, interest rate, total accounts).

- Instance members = unique details of each account (account number, holder name, balance).

- Even if 1000 accounts exist, there’s only ONE copy of static variables.

- But each account has its own balance and account number.

\*/

}

}

Output-

A computer screen with blue text

AI-generated content may be incorrect.

�� **PRACTICE PROBLEM 4:**

**OOP Benefits - Reusability and Extensibility Task**: 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)

// TODO: Create constructor

// TODO: Create common methods:

// - startVehicle()

// - stopVehicle()

// - refuel(double amount)

// - displayVehicleInfo()

public static void main(String[] args) {

// TODO: Create different types of vehicles (Car, Truck, Motorcycle)

// TODO: Show how the same Vehicle class can be reused // TODO: Create an array of Vehicle objects

// TODO: Demonstrate polymorphic behavior

// TODO: In comments, explain:

// - How does this show reusability?

// - How could this be extended for new vehicle types?



// - What are the benefits over writing separate classes?

}

}

Program –

public class Vehicle {

protected String make;

protected String model;

protected int year;

protected double fuelLevel;

public Vehicle(String make, String model, int year, double fuelLevel) {

this.make = make;

this.model = model;

this.year = year;

this.fuelLevel = fuelLevel;

}

public void startVehicle() {

System.out.println(make + " " + model + " started.");

}

public void stopVehicle() {

System.out.println(make + " " + model + " stopped.");

}

public void refuel(double amount) {

fuelLevel += amount;

System.out.println(make + " " + model + " refueled. Current fuel: " + fuelLevel);

}

public void displayVehicleInfo() {

System.out.println("Vehicle Info: " + year + " " + make + " " + model + ", Fuel: " + fuelLevel);

}

// Main method is inside Vehicle now

public static void main(String[] args) {

Car car = new Car("Toyota", "Corolla", 2020, 50);

Truck truck = new Truck("Ford", "F-150", 2018, 80);

Motorcycle bike = new Motorcycle("Yamaha", "R15", 2022, 20);

car.startVehicle();

truck.refuel(20);

bike.displayVehicleInfo();

car.openTrunk();

truck.loadCargo();

bike.popWheelie();

Vehicle[] vehicles = {car, truck, bike};

System.out.println("\nPolymorphic Behavior:");

for (Vehicle v : vehicles) {

v.startVehicle();

v.displayVehicleInfo();

}

}

}

// Subclasses

class Car extends Vehicle {

public Car(String make, String model, int year, double fuelLevel) {

super(make, model, year, fuelLevel);

}

public void openTrunk() {

System.out.println(make + " " + model + " trunk opened.");

}

}

class Truck extends Vehicle {

public Truck(String make, String model, int year, double fuelLevel) {

super(make, model, year, fuelLevel);

}

public void loadCargo() {

System.out.println(make + " " + model + " cargo loaded.");

}

}

class Motorcycle extends Vehicle {

public Motorcycle(String make, String model, int year, double fuelLevel) {

super(make, model, year, fuelLevel);

}

public void popWheelie() {

System.out.println(make + " " + model + " is popping a wheelie!");

}

}

Output –

