ASSIGNMENT NO.4

Note:

- The assignment is designed to practice constructor, getter/setter and toString method.
- Create a separate project for each question and create separate file for each class.
- Try to test the functionality by using menu-driven program.

1. Loan Amortization Calculator

Implement a system to calculate and display the monthly payments for a mortgage loan. The system should:

1. Accept the principal amount (loan amount), annual interest rate, and loan term (in years) from the user.
2. Calculate the monthly payment using the standard mortgage formula:
    - **Monthly Payment Calculation:**
        - `monthlyPayment = principal * (monthlyInterestRate * (1 + monthlyInterestRate)^(numberOfMonths)) / ((1 + monthlyInterestRate)^(numberOfMonths) - 1)`
        - Where `monthlyInterestRate = annualInterestRate / 12 / 100` and `numberOfMonths = loanTerm * 12`
        - Note: Here ^ means power and to find it you can use Math.pow( ) method
3. Display the monthly payment and the total amount paid over the life of the loan, in Indian Rupees (₹).

```
4. import java.util.Scanner;
5. 
6. public class MortgageCalculator {
7.     public static void main(String[] args) {
8.         Scanner scanner = new Scanner(System.in);
9. 
10.        System.out.print("Enter the principal amount (₹): ");
11.        double principal = scanner.nextDouble();
12. 
13.        System.out.print("Enter the annual interest rate (%): ");
14.        double annualInterestRate = scanner.nextDouble();
15. 
16.        System.out.print("Enter the loan term (years): ");
17.        int loanTermYears = scanner.nextInt();
18. 
19.        double monthlyInterestRate = annualInterestRate / 12 / 100;
20.        int numberOfMonths = loanTermYears * 12;
21. 
22.        double monthlyPayment = principal * (monthlyInterestRate *
   Math.pow(1 + monthlyInterestRate, numberOfMonths))
23.                                / (Math.pow(1 + monthlyInterestRate,
   numberOfMonths) - 1);
24. 
25.        double totalAmountPaid = monthlyPayment * numberOfMonths;
26. 
```

```
27.         System.out.printf("Monthly Payment: ₹%.2f\n", monthlyPayment);
28.         System.out.printf("Total Amount Paid over the life of the loan:
    ₹%.2f\n", totalAmountPaid);
29.
30.         scanner.close();
31.     }
32.}
33.
```

Define the class `LoanAmortizationCalculator` with fields, an appropriate constructor, getter and setter methods, a `toString` method and business logic methods. Define the class `LoanAmortizationCalculatorUtil` with methods `acceptRecord`, `printRecord`, and `menuList`. Define the class `Program` with a `main` method and test the functionality of the utility class.

LoanAmortizationCalculator Class

- Fields:

  - `principal`: Initial loan amount.

  - `annualInterestRate`: Annual interest rate.

  - `loanTermYears`: Loan term in years.

- Constructor: Initializes fields with provided values.

- Getters and Setters: Methods to access and modify fields.

- toString Method: Returns a string representation of loan details.

- Business Logic Methods:

  - `calculateMonthlyPayment()`: Computes the monthly payment based on the principal, interest rate, and loan term.

  - `calculateTotalPayment()`: Computes the total amount paid over the life of the loan.

LoanAmortizationCalculatorUtil Class

- Methods:

  - `acceptRecord()`: Prompts user to input loan details and returns a `LoanAmortizationCalculator` object.

- `printRecord(LoanAmortizationCalculator loan)`: Displays loan details and calculated payments.

- `menuList()`: Displays menu options for the user.

Program Class

- Main Method:

- Provides a user interface to interact with the `LoanAmortizationCalculatorUtil` class.

- Allows the user to enter loan details, view results, and navigate the menu.

2. Compound Interest Calculator for Investment

Develop a system to compute the future value of an investment with compound interest. The system should:

1. Accept the initial investment amount, annual interest rate, number of times the interest is compounded per year, and investment duration (in years) from the user.
2. Calculate the future value of the investment using the formula:
    - **Future Value Calculation:**
        - `futureValue = principal * (1 + annualInterestRate / numberOfCompounds)^(numberOfCompounds * years)`
    - **Total Interest Earned:** `totalInterest = futureValue - principal`
3. Display the future value and the total interest earned, in Indian Rupees (₹).

```
4. import java.util.Scanner;
5. 
6. public class CompoundInterestCalculator {
7.     public static void main(String[] args) {
8.         Scanner scanner = new Scanner(System.in);
9. 
10.        System.out.print("Enter the initial investment amount (₹): ");
11.        double principal = scanner.nextDouble();
12. 
13.        System.out.print("Enter the annual interest rate (%): ");
14.        double annualInterestRate = scanner.nextDouble();
15. 
16.        System.out.print("Enter the number of times the interest is
   compounded per year: ");
17.        int numberOfCompounds = scanner.nextInt();
18. 
19.        System.out.print("Enter the investment duration (years): ");
20.        int years = scanner.nextInt();
```

```
21.
22.         double futureValue = principal * Math.pow((1 +
    annualInterestRate / 100 / numberOfCompounds), (numberOfCompounds *
    years));
23.         double totalInterest = futureValue - principal;
24.
25.         System.out.printf("Future Value of Investment: ₹%.2f\n",
    futureValue);
26.         System.out.printf("Total Interest Earned: ₹%.2f\n",
    totalInterest);
27.
28.         scanner.close();
29.     }
30.}
31.
```

Define the class `CompoundInterestCalculator` with fields, an appropriate constructor, getter and setter methods, a `toString` method and business logic methods. Define the class `CompoundInterestCalculatorUtil` with methods `acceptRecord`, `printRecord`, and `menuList`. Define the class `Program` with a `main` method to test the functionality of the utility class.

CompoundInterestCalculator Class

- Fields:
  - `principal`: Initial investment amount.
  - `annualInterestRate`: Annual interest rate (in percentage).
  - `numberOfCompounds`: Number of times interest is compounded per year.
  - `years`: Duration of the investment in years.

- Constructor:
  - Initializes all fields with provided values.

- Getters and Setters:
  - Methods to access and modify each field.

- toString Method:
  - Provides a string representation of investment details.

- Business Logic Methods:
  - `calculateFutureValue()`: Computes the future value of the investment using compound interest formula.
  - `calculateTotalInterest()`: Calculates total interest earned by subtracting principal from future value.

CompoundInterestCalculatorUtil Class

- Methods:
  - `acceptRecord()`: Prompts user to enter investment details and creates a `CompoundInterestCalculator` object.
  - `printRecord(CompoundInterestCalculator calc)`: Displays investment details, future value, and total interest.
  - `menuList()`: Displays options for user interaction.

Program Class

- Main Method:
  - Tests `CompoundInterestCalculatorUtil` functionality.
  - Allows the user to input investment details, view results, and navigate through options.

3. BMI (Body Mass Index) Tracker

Create a system to calculate and classify Body Mass Index (BMI). The system should:

1. Accept weight (in kilograms) and height (in meters) from the user.
2. Calculate the BMI using the formula:
   - **BMI Calculation:** `BMI = weight / (height * height)`
3. Classify the BMI into one of the following categories:
   - Underweight: BMI < 18.5
   - Normal weight: 18.5 ≤ BMI < 24.9
   - Overweight: 25 ≤ BMI < 29.9
   - Obese: BMI ≥ 30
4. Display the BMI value and its classification.

```
import java.util.Scanner;

public class BMICalculator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter weight (in kilograms): ");
        double weight = scanner.nextDouble();

        System.out.print("Enter height (in meters): ");
        double height = scanner.nextDouble();

        double bmi = weight / (height * height);

        String classification;

        if (bmi < 18.5) {
            classification = "Underweight";
        } else if (bmi < 24.9) {
            classification = "Normal weight";
        } else if (bmi < 29.9) {
            classification = "Overweight";
```

```
        } else {
            classification = "Obese";
        }

        System.out.printf("Your BMI: %.2f\n", bmi);
        System.out.println("Classification: " + classification);

        scanner.close();
    }
}

```

Define the class `BMITracker` with fields, an appropriate constructor, getter and setter methods, a `toString` method, and business logic methods. Define the class `BMITrackerUtil` with methods `acceptRecord`, `printRecord`, and `menuList`. Define the class `Program` with a `main` method to test the functionality of the utility class.

BMITracker Class

- Fields:
  - `weight`: Weight of the individual (in kilograms).
  - `height`: Height of the individual (in meters).

- Constructor:
  - Initializes `weight` and `height` with provided values.

- Getters and Setters:
  - Methods to get and set the values of `weight` and `height`.

- toString Method:
  - Returns a string representation of the BMI details including weight and height.

- Business Logic Methods:
  - `calculateBMI()`: Computes the BMI using the formula: \(\text{BMI} = \frac{\text{weight}}{\text{height}^2} \).
  - `getBMICategory()`: Returns the BMI category based on the calculated BMI:
    - Underweight: BMI < 18.5
    - Normal weight: 18.5 ≤ BMI < 24.9
    - Overweight: 25 ≤ BMI < 29.9
    - Obese: BMI ≥ 30

BMITrackerUtil Class

- Methods:

- `acceptRecord()`: Prompts user to enter weight and height, then creates a `BMITracker` object.
- `printRecord(BMITracker tracker)`: Displays the BMI value and its classification based on the `BMITracker` object.
- `menuList()`: Displays a menu with options for the user to interact with the program.

Program Class

- Main Method:
  - Tests the functionality of `BMITrackerUtil`.
  - Provides a user interface to enter weight and height, view BMI results, and navigate through the menu options.

4. Discount Calculation for Retail Sales

Design a system to calculate the final price of an item after applying a discount. The system should:

1. Accept the original price of an item and the discount percentage from the user.
2. Calculate the discount amount and the final price using the following formulas:
   - **Discount Amount Calculation:** `discountAmount = originalPrice * (discountRate / 100)`
   - **Final Price Calculation:** `finalPrice = originalPrice - discountAmount`
3. Display the discount amount and the final price of the item, in Indian Rupees (₹).

```
4. import java.util.Scanner;
5.
6. public class DiscountCalculator {
7.     public static void main(String[] args) {
8.         Scanner scanner = new Scanner(System.in);
9.
10.        System.out.print("Enter the original price of the item (₹): ");
11.        double originalPrice = scanner.nextDouble();
12.
13.        System.out.print("Enter the discount percentage: ");
14.        double discountPercentage = scanner.nextDouble();
15.
16.        double discountAmount = originalPrice * (discountPercentage /
   100);
17.        double finalPrice = originalPrice - discountAmount;
18.
19.        System.out.printf("Discount Amount: ₹%.2f\n", discountAmount);
20.        System.out.printf("Final Price: ₹%.2f\n", finalPrice);
21.
22.        scanner.close();
23.    }
24.}
25.
```

Define the class `DiscountCalculator` with fields, an appropriate constructor, getter and setter methods, a `toString` method, and business logic methods. Define the class `DiscountCalculatorUtil` with methods `acceptRecord`, `printRecord`, and `menuList`. Define the class `Program` with a `main` method to test the functionality of the utility class.

DiscountCalculator Class

- Fields:
  - `originalPrice`: Original price of the item (in Indian Rupees).
  - `discountPercentage`: Discount percentage to be applied.

- Constructor:
  - Initializes `originalPrice` and `discountPercentage` with provided values.

- Getters and Setters:
  - Methods to get and set `originalPrice` and `discountPercentage`.

- toString Method:
  - Returns a string representation of the item with its original price and discount percentage.

- Business Logic Methods:
  - `calculateDiscountAmount()`: Computes the discount amount using the formula: $\text{discountAmount} = \text{originalPrice} \times \frac{\text{discountPercentage}}{100}$.
  - `calculateFinalPrice()`: Computes the final price after applying the discount using: $\text{finalPrice} = \text{originalPrice} - \text{discountAmount}$.

DiscountCalculatorUtil Class

- Methods:
  - `acceptRecord()`: Prompts the user to enter the original price and discount percentage, then creates and returns a `DiscountCalculator` object.
  - `printRecord(DiscountCalculator discountCalc)`: Displays the discount amount and final price based on the `DiscountCalculator` object.
  - `menuList()`: Displays menu options for the user to interact with the system.

Program Class

- Main Method:
  - Tests the functionality of `DiscountCalculatorUtil`.
  - Provides an interface for the user to input data, view results, and navigate through the menu options.

5. Toll Booth Revenue Management

Develop a system to simulate a toll booth for collecting revenue. The system should:

ASSIGNMENT NO.4

1. Allow the user to set toll rates for different vehicle types: Car, Truck, and Motorcycle.
2. Accept the number of vehicles of each type passing through the toll booth.
3. Calculate the total revenue based on the toll rates and number of vehicles.
4. Display the total number of vehicles and the total revenue collected, in Indian Rupees (₹).

- **Toll Rate Examples:**
  - Car: ₹50.00
  - Truck: ₹100.00
  - Motorcycle: ₹30.00

```java
import java.util.Scanner;

public class TollBoothSimulator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Define default toll rates
        double carRate = 50.00;
        double truckRate = 100.00;
        double motorcycleRate = 30.00;

        // Allow user to set toll rates
        System.out.println("Set toll rates for different vehicle types:");
        System.out.print("Enter the toll rate for Car (₹): ");
        carRate = scanner.nextDouble();

        System.out.print("Enter the toll rate for Truck (₹): ");
        truckRate = scanner.nextDouble();

        System.out.print("Enter the toll rate for Motorcycle (₹): ");
        motorcycleRate = scanner.nextDouble();

        // Accept number of vehicles
        System.out.print("Enter the number of Cars passing through: ");
        int numberOfCars = scanner.nextInt();

        System.out.print("Enter the number of Trucks passing through: ");
        int numberOfTrucks = scanner.nextInt();

        System.out.print("Enter the number of Motorcycles passing through: ");
        int numberOfMotorcycles = scanner.nextInt();

        // Calculate total revenue
        double totalRevenue = (numberOfCars * carRate) + (numberOfTrucks * truckRate) + (numberOfMotorcycles * motorcycleRate);
```

```
        // Calculate total number of vehicles
        int totalVehicles = numberOfCars + numberOfTrucks +
numberOfMotorcycles;

        // Display results
        System.out.printf("Total Number of Vehicles: %d\n",
totalVehicles);
        System.out.printf("Total Revenue Collected: ₹%.2f\n",
totalRevenue);

        scanner.close();
    }
}
```

Define the class `TollBoothRevenueManager` with fields, an appropriate constructor, getter and setter methods, a `toString` method, and business logic methods. Define the class `TollBoothRevenueManagerUtil` with methods `acceptRecord`, `printRecord`, and `menuList`. Define the class `Program` with a `main` method to test the functionality of the utility class.

1. Class: TollBoothRevenueManager
This class will manage the toll booth's core functionality, including storing toll rates, tracking the number of vehicles, and calculating the total revenue.
Fields:

- double carRate: The toll rate for cars.
- double truckRate: The toll rate for trucks.
- double motorcycleRate: The toll rate for motorcycles.
- int carCount: The number of cars that have passed through the toll booth.
- int truckCount: The number of trucks that have passed through the toll booth.
- int motorcycleCount: The number of motorcycles that have passed through the toll booth.

Constructor:

- Initializes the toll rates for cars, trucks, and motorcycles, and sets the initial vehicle counts to zero.

Getter and Setter Methods:

- getCarRate(), setCarRate(double carRate)
- getTruckRate(), setTruckRate(double truckRate)
- getMotorcycleRate(), setMotorcycleRate(double motorcycleRate)

- getCarCount(), setCarCount(int carCount)
- getTruckCount(), setTruckCount(int truckCount)
- getMotorcycleCount(), setMotorcycleCount(int motorcycleCount)

Business Logic Methods:

- void addVehicle(String vehicleType, int count): Adds a specified number of vehicles of a given type to the total count.
- double calculateTotalRevenue(): Calculates the total revenue based on the toll rates and the number of vehicles.
- int calculateTotalVehicles(): Calculates the total number of vehicles that have passed through the toll booth.

toString Method:

- Provides a string representation of the toll booth, including the toll rates for different vehicle types.

2. Class: TollBoothRevenueManagerUtil

This utility class will provide methods for user interaction, such as accepting input and displaying results.

Methods:

- void acceptRecord(): Accepts user input for setting toll rates and the number of vehicles for each type.
- void printRecord(): Displays the total number of vehicles and total revenue collected.
- void menuList(): Provides a menu for user interaction, listing available options.

3. Class: Program

This class contains the main method and is responsible for driving the application using the utility class methods.

Methods:

- public static void main(String[] args): The main method to run the application, which initializes the utility class and provides a loop to interact with the user through a menu.

Notes:

1. **TollBoothRevenueManager Class Design**:
    - This class focuses on core business logic for managing toll rates and calculating revenue.
    - It encapsulates data (fields for rates and counts) and provides methods to manipulate and retrieve this data.
2. **TollBoothRevenueManagerUtil Class Design**:
    - This class acts as a utility/helper for TollBoothRevenueManager.
    - It separates the user interface (input/output) from the business logic, adhering to the Single Responsibility Principle.
    - Methods like acceptRecord handle user inputs, while printRecord is responsible for output.
3. **Program Class Design**:
    - This is the entry point for the program.
    - It contains the main loop, which continuously interacts with the user until they choose to exit.