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 (₹).

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.

```
import java.util.Scanner;

public class LoanAmortizationCalculatorProgram {
   static class LoanAmortizationCalculator {
      private double principal;
      private double annualInterestRate;
      private int loanTerm;

      public LoanAmortizationCalculator(double principal, double annualInterestRate, int loanTerm) {
         this.principal = principal;
         this.annualInterestRate = annualInterestRate;
         this.loanTerm = loanTerm;
      }

      public double getPrincipal() { return principal; }
      public void setPrincipal(double principal) { this.principal = principal; }

      public double getAnnualInterestRate() { return annualInterestRate; }
      public void setAnnualInterestRate(double annualInterestRate) { this.annualInterestRate = annualInterestRate; }

      public int getLoanTerm() { return loanTerm; }
      public void setLoanTerm(int loanTerm) { this.loanTerm = loanTerm; }
```

```
    public double calculateMonthlyPayment() {
        double monthlyInterestRate = annualInterestRate / 12 / 100;
        int numberOfMonths = loanTerm * 12;
        return principal * (monthlyInterestRate * Math.pow(1 + monthlyInterestRate,
numberOfMonths)) /
                (Math.pow(1 + monthlyInterestRate, numberOfMonths) - 1);
    }

    public double calculateTotalAmountPaid() {
        return calculateMonthlyPayment() * loanTerm * 12;
    }

    @Override
    public String toString() {
        return String.format("Principal: ₹%.2f, Annual Interest Rate: %.2f%%, Loan Term:
%d years",
                principal, annualInterestRate, loanTerm);
    }
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Loan Amortization Calculator");
    System.out.print("Enter the principal amount (₹): ");
    double principal = scanner.nextDouble();
    System.out.print("Enter the annual interest rate (%): ");
    double interestRate = scanner.nextDouble();
    System.out.print("Enter the loan term (years): ");
    int term = scanner.nextInt();

    LoanAmortizationCalculator calculator = new LoanAmortizationCalculator(principal,
interestRate, term);
    System.out.println(calculator);
    System.out.printf("Monthly Payment: ₹%.2f\n", calculator.calculateMonthlyPayment());
    System.out.printf("Total Amount Paid: ₹%.2f\n",
calculator.calculateTotalAmountPaid());
  }
}
```


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 (₹).

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.

```
import java.util.Scanner;

public class CompoundInterestCalculatorProgram {
   static class CompoundInterestCalculator {
      private double principal;
      private double annualInterestRate;
      private int numberOfCompounds;
      private int years;

      public CompoundInterestCalculator(double principal, double annualInterestRate, int numberOfCompounds, int years) {
         this.principal = principal;
         this.annualInterestRate = annualInterestRate;
         this.numberOfCompounds = numberOfCompounds;
         this.years = years;
      }

      public double getPrincipal() { return principal; }
      public void setPrincipal(double principal) { this.principal = principal; }

      public double getAnnualInterestRate() { return annualInterestRate; }
      public void setAnnualInterestRate(double annualInterestRate) { this.annualInterestRate = annualInterestRate; }

      public int getNumberOfCompounds() { return numberOfCompounds; }
      public void setNumberOfCompounds(int numberOfCompounds) { this.numberOfCompounds = numberOfCompounds; }

      public int getYears() { return years; }
      public void setYears(int years) { this.years = years; }

      public double calculateFutureValue() {
         return principal * Math.pow(1 + annualInterestRate / numberOfCompounds, numberOfCompounds * years);
      }

      public double calculateTotalInterest() {
         return calculateFutureValue() - principal;
      }
```

```
    @Override
    public String toString() {
        return String.format("Principal: ₹%.2f, Annual Interest Rate: %.2f%%, Compounds per Year: %d, Duration: %d years",
                principal, annualInterestRate, numberOfCompounds, years);
    }
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Compound Interest Calculator for Investment");
    System.out.print("Enter the initial investment amount (₹): ");
    double principal = scanner.nextDouble();
    System.out.print("Enter the annual interest rate (%): ");
    double interestRate = scanner.nextDouble();
    System.out.print("Enter the number of times interest is compounded per year: ");
    int numberOfCompounds = scanner.nextInt();
    System.out.print("Enter the investment duration (years): ");
    int years = scanner.nextInt();

    CompoundInterestCalculator calculator = new CompoundInterestCalculator(principal, interestRate, numberOfCompounds, years);
    System.out.println(calculator);
    System.out.printf("Future Value: ₹%.2f\n", calculator.calculateFutureValue());
    System.out.printf("Total Interest Earned: ₹%.2f\n", calculator.calculateTotalInterest());
}
}
```

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.

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.

```
import java.util.Scanner;
```

```
public class BMITrackerProgram {
    static class BMITracker {
        private double weight;
        private double height;

        public BMITracker(double weight, double height) {
            this.weight = weight;
            this.height = height;
        }

        public double getWeight() { return weight; }
        public void setWeight(double weight) { this.weight = weight; }

        public double getHeight() { return height; }
        public void setHeight(double height) { this.height = height; }

        public double calculateBMI() {
            return weight / (height * height);
        }

        public String classifyBMI() {
            double bmi = calculateBMI();
            if (bmi < 18.5) return "Underweight";
            else if (bmi < 24.9) return "Normal weight";
            else if (bmi < 29.9) return "Overweight";
            else return "Obese";
        }

        @Override
        public String toString() {
            return String.format("Weight: %.2f kg, Height: %.2f m", weight, height);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("BMI Tracker");
        System.out.print("Enter weight (kg): ");
        double weight = scanner.nextDouble();
        System.out.print("Enter height (m): ");
        double height = scanner.nextDouble();

        BMITracker tracker = new BMITracker(weight, height);
        System.out.println(tracker);
        System.out.printf("BMI: %.2f\n", tracker.calculateBMI());
        System.out.println("Classification: " + tracker.classifyBMI());
    }
}
```

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 (₹).

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.

```
import java.util.Scanner;

public class DiscountCalculatorProgram {
   static class DiscountCalculator {
      private double originalPrice;
      private double discountRate;

      public DiscountCalculator(double originalPrice, double discountRate) {
         this.originalPrice = originalPrice;
         this.discountRate = discountRate;
      }

      public double getOriginalPrice() { return originalPrice; }
      public void setOriginalPrice(double originalPrice) { this.originalPrice = originalPrice; }

      public double getDiscountRate() { return discountRate; }
      public void setDiscountRate(double discountRate) { this.discountRate = discountRate; }

      public double calculateDiscountAmount() {
         return originalPrice * (discountRate / 100);
      }

      public double calculateFinalPrice() {
         return originalPrice - calculateDiscountAmount();
      }

      @Override
      public String toString() {
         return String.format("Original Price: ₹%.2f, Discount Rate: %.2f%%", originalPrice,
discountRate);
      }
   }

   public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
        System.out.println("Discount Calculation for Retail Sales");
        System.out.print("Enter the original price of the item (₹): ");
        double originalPrice = scanner.nextDouble();
        System.out.print("Enter the discount percentage: ");
        double discountRate = scanner.nextDouble();

        DiscountCalculator calculator = new DiscountCalculator(originalPrice, discountRate);
        System.out.println(calculator);
        System.out.printf("Discount Amount: ₹%.2f\n", calculator.calculateDiscountAmount());
        System.out.printf("Final Price: ₹%.2f\n", calculator.calculateFinalPrice());
    }
}
```

5. Toll Booth Revenue Management

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

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

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.

```
import java.util.Scanner;

public class TollBoothRevenueManagerProgram {
    static class TollBoothRevenueManager {
        private double carRate = 50.00;
        private double truckRate = 100.00;
        private double motorcycleRate = 30.00;

        private int numberOfCars;
        private int numberOfTrucks;
        private int numberOfMotorcycles;
```

```
    public TollBoothRevenueManager(int numberOfCars, int numberOfTrucks, int
numberOfMotorcycles) {
        this.numberOfCars = numberOfCars;
        this.numberOfTrucks = numberOfTrucks;
        this.numberOfMotorcycles = numberOfMotorcycles;
    }

    public int getNumberOfCars() { return numberOfCars; }
    public void setNumberOfCars(int numberOfCars) { this.numberOfCars = numberOfCars;
}

    public int getNumberOfTrucks() { return numberOfTrucks; }
    public void setNumberOfTrucks(int numberOfTrucks) { this.numberOfTrucks =
numberOfTrucks; }

    public int getNumberOfMotorcycles() { return numberOfMotorcycles; }
    public void setNumberOfMotorcycles(int numberOfMotorcycles) {
this.numberOfMotorcycles = numberOfMotorcycles; }

    public double calculateTotalRevenue() {
        return (numberOfCars * carRate) + (numberOfTrucks * truckRate) +
(numberOfMotorcycles * motorcycleRate);
    }

    @Override
    public String toString() {
        return String.format("Cars: %d, Trucks: %d, Motorcycles: %d", numberOfCars,
numberOfTrucks, numberOfMotorcycles);
    }
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Toll Booth Revenue Management");
    System.out.print("Enter the number of cars: ");
    int cars = scanner.nextInt();
    System.out.print("Enter the number of trucks: ");
    int trucks = scanner.nextInt();
    System.out.print("Enter the number of motorcycles: ");
    int motorcycles = scanner.nextInt();

    TollBoothRevenueManager manager = new TollBoothRevenueManager(cars, trucks,
motorcycles);
    System.out.println(manager);
    System.out.printf("Total Revenue: ₹%.2f\n", manager.calculateTotalRevenue());
  }
}
```