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 class LoanAmortizationCalculator with methods acceptRecord, calculateMonthlyPayment & printRecord and test the functionality in main method.

```
// Logic Class
class LoanAmortizationCalculator {
    double principal, annualInterestRate, loanTerm;
    double monthlyPayment;

    void acceptRecord(double principal, double annualInterestRate, double loanTerm) {
        this.principal = principal;
        this.annualInterestRate = annualInterestRate;
        this.loanTerm = loanTerm;
    }

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

    void printRecord() {
        double totalPayment = monthlyPayment * loanTerm * 12;
        System.out.println("Monthly Payment: ₹" + monthlyPayment);
        System.out.println("Total Amount Paid: ₹" + totalPayment);
    }
}

// Test Class
public class TestLoanAmortizationCalculator {
```

```
    public static void main(String[] args) {
        LoanAmortizationCalculator loanCalc = new LoanAmortizationCalculator();
        loanCalc.acceptRecord(500000, 7.5, 20); // Example values
        loanCalc.calculateMonthlyPayment();
        loanCalc.printRecord();
    }
}
```

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 class CompoundInterestCalculator with methods acceptRecord , calculateFutureValue, printRecord and test the functionality in main method.

```
// Logic Class
class CompoundInterestCalculator {
    double principal, annualInterestRate, numberOfCompounds, years;
    double futureValue;

    void acceptRecord(double principal, double annualInterestRate, double
numberOfCompounds, double years) {
        this.principal = principal;
        this.annualInterestRate = annualInterestRate;
        this.numberOfCompounds = numberOfCompounds;
        this.years = years;
    }

    void calculateFutureValue() {
        futureValue = principal * Math.pow(1 + (annualInterestRate / numberOfCompounds /
100), numberOfCompounds * years);
    }

    void printRecord() {
        double totalInterest = futureValue - principal;
        System.out.println("Future Value: ₹" + futureValue);
        System.out.println("Total Interest Earned: ₹" + totalInterest);
    }
```

```
}

// Test Class
public class TestCompoundInterestCalculator {
    public static void main(String[] args) {
        CompoundInterestCalculator interestCalc = new CompoundInterestCalculator();
        interestCalc.acceptRecord(100000, 6.5, 4, 10); // Example values
        interestCalc.calculateFutureValue();
        interestCalc.printRecord();
    }
}
```

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 class BMITracker with methods acceptRecord, calculateBMI, classifyBMI & printRecord and test the functionality in main method.

```
// Logic Class
class BMITracker {
    double weight, height;
    double bmi;

    void acceptRecord(double weight, double height) {
        this.weight = weight;
        this.height = height;
    }

    void calculateBMI() {
        bmi = weight / (height * height);
    }

    void classifyBMI() {
        System.out.println("BMI: " + bmi);
        if (bmi < 18.5) {
```

```
            System.out.println("Category: Underweight");
        } else if (bmi >= 18.5 && bmi < 24.9) {
            System.out.println("Category: Normal weight");
        } else if (bmi >= 25 && bmi < 29.9) {
            System.out.println("Category: Overweight");
        } else {
            System.out.println("Category: Obese");
        }
    }

    void printRecord() {
        calculateBMI();
        classifyBMI();
    }
}

// Test Class
public class TestBMITracker {
    public static void main(String[] args) {
        BMITracker bmiTracker = new BMITracker();
        bmiTracker.acceptRecord(70, 1.75); // Example values
        bmiTracker.printRecord();
    }
}
```

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 class DiscountCalculator with methods acceptRecord, calculateDiscount & printRecord and test the functionality in main method.

```
// Logic Class
class DiscountCalculator {
    double originalPrice, discountRate;
    double discountAmount, finalPrice;
```

```
    void acceptRecord(double originalPrice, double discountRate) {
        this.originalPrice = originalPrice;
        this.discountRate = discountRate;
    }

    void calculateDiscount() {
        discountAmount = originalPrice * (discountRate / 100);
        finalPrice = originalPrice - discountAmount;
    }

    void printRecord() {
        System.out.println("Discount Amount: ₹" + discountAmount);
        System.out.println("Final Price: ₹" + finalPrice);
    }
}

// Test Class
public class TestDiscountCalculator {
    public static void main(String[] args) {
        DiscountCalculator discountCalc = new DiscountCalculator();
        discountCalc.acceptRecord(2000, 10); // Example values
        discountCalc.calculateDiscount();
        discountCalc.printRecord();
    }
}
```

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 class TollBoothRevenueManager with methods acceptRecord, setTollRates, calculateRevenue & printRecord and test the functionality in main method.

```
// Logic Class
```

```
class TollBoothRevenueManager {
    double carRate, truckRate, motorcycleRate;
    int carCount, truckCount, motorcycleCount;
    double totalRevenue;

    void setTollRates(double carRate, double truckRate, double motorcycleRate) {
        this.carRate = carRate;
        this.truckRate = truckRate;
        this.motorcycleRate = motorcycleRate;
    }

    void acceptRecord(int carCount, int truckCount, int motorcycleCount) {
        this.carCount = carCount;
        this.truckCount = truckCount;
        this.motorcycleCount = motorcycleCount;
    }

    void calculateRevenue() {
        totalRevenue = (carRate * carCount) + (truckRate * truckCount) + (motorcycleRate *
motorcycleCount);
    }

    void printRecord() {
        int totalVehicles = carCount + truckCount + motorcycleCount;
        System.out.println("Total Vehicles: " + totalVehicles);
        System.out.println("Total Revenue: ₹" + totalRevenue);
    }
}

// Test Class
public class TestTollBoothRevenueManager {
    public static void main(String[] args) {
        TollBoothRevenueManager tollBooth = new TollBoothRevenueManager();
        tollBooth.setTollRates(50, 100, 30); // Example toll rates
        tollBooth.acceptRecord(10, 5, 7); // Example vehicle counts
        tollBooth.calculateRevenue();
        tollBooth.printRecord();
    }
}
```