```
/**
* Computes the periodical payment necessary to re-pay a given loan.
*/
public class LoanCalc {

	static double epsilon = 0.001;  // The computation tolerance (estimation error)
	static int iterationCounter;    // Monitors the efficiency of the calculation

   /**
    * Gets the loan data and computes the periodical payment.
    * Expects to get three command-line arguments: sum of the loan (double),
    * interest rate (double, as a percentage), and number of payments (int).
    */
	public static void main(String[] args) {
		// Gets the loan data
		double loan = Double.parseDouble(args[0]);
		double rate = Double.parseDouble(args[1]);
		int n = Integer.parseInt(args[2]);
		System.out.println("Loan sum = " + loan + ", interest rate = " + rate + "%, periods = "
+ n);

		// Computes the periodical payment using brute force search
		System.out.print("Periodical payment, using brute force: ");
		System.out.printf("%.2f", bruteForceSolver(loan, rate, n, epsilon));
		System.out.println();
		System.out.println("number of iterations: " + iterationCounter);

		// Computes the periodical payment using bisection search
		System.out.print("Periodical payment, using bi-section search: ");
		System.out.printf("%.2f", bisectionSolver(loan, rate, n, epsilon));
		System.out.println();
```

```
            System.out.println("number of iterations: " + iterationCounter);
        }

        /**
        * Uses a sequential search method  ("brute force") to compute an approximation
        * of the periodical payment that will bring the ending balance of a loan close to 0.
        * Given: the sum of the loan, the periodical interest rate (as a percentage),
        * the number of periods (n), and epsilon, a tolerance level.
        */
        // Side effect: modifies the class variable iterationCounter.
public static double bruteForceSolver(double loan, double rate, int n, double epsilon) {
            //setting 2 doubles one as balance and one as eriodicalPayment
            double periodicalPayment = loan / n;
            double balance = LoanCalc.endBalance(loan, rate, n, periodicalPayment);
            iterationCounter = 0;

            //implementing bruteforce
            while((Math.abs(balance)) >= epsilon && (balance >= 0)) {
                periodicalPayment = periodicalPayment + epsilon;
                balance = LoanCalc.endBalance(loan, rate, n, periodicalPayment);
                iterationCounter++;
            }

        return periodicalPayment;
}

/**
        * Uses bisection search to compute an approximation of the periodical payment
        * that will bring the ending balance of a loan close to 0.
        * Given: the sum of theloan, the periodical interest rate (as a percentage),
        * the number of periods (n), and epsilon, a tolerance level.
```

```
        */
        // Side effect: modifies the class variable iterationCounter.
public static double bisectionSolver(double loan, double rate, int n, double epsilon) {
                // culculate the the remain balance for this payment
   //  lsetting lower and upper payment
                double L = (loan / n), H = loan;
                double g = (H + L) / 2;
                double balance = LoanCalc.endBalance(loan, rate, n, g);
                // Reset the variable to the other search
                iterationCounter = 0;
                // implementing the bisection
                while((Math.abs(H - L)) >= epsilon) {
                        if(balance > 0) {
                                L = g;
                        } else {
                                H = g;
                        }
                        g = (L + H) / 2;
                        balance = LoanCalc.endBalance(loan, rate, n, g);
                        iterationCounter++;
                }
                return g;
}

        /**
        * Computes the ending balance of a loan, given the sum of the loan, the periodical
        * interest rate (as a percentage), the number of periods (n), and the periodical payment.
        */
        private static double endBalance(double loan, double rate, int n, double payment) {
                double balance = loan;
                for(int i = 0; i < n; i++) {
```

```
                balance = (balance - payment) * ((rate / 100) + 1);
            }
            return balance;
        }
}
```