```java
/**
 * 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);
    }

    /**
     * "Brute force" computes 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) {
        double monthlyPayment = (loan / n) + epsilon;
        double balance = loan;
        double increment = 0.001;

        while (balance > epsilon) {
            balance = endBalance(loan, rate, n, monthlyPayment);
            monthlyPayment += increment;
            iterationCounter++;
        }
        return monthlyPayment;
    }

    /**
     * Bisection search computes 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 bisectionSolver(double loan, double rate, int n, double epsilon)
{
        double H = loan;
        double L = loan / n;
        double monthlyPayment = (H + L) / 2;
        iterationCounter = 0;

        while (H - L >= epsilon) {
            double balance = endBalance(loan, rate, n, monthlyPayment);
            if (balance * L > 0) {
                L = monthlyPayment;
            } else {
                H = monthlyPayment;
            }
            monthlyPayment = (L + H) / 2;
            iterationCounter++;
        }

        return monthlyPayment;
    }

    /**
     * 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 = 0;
        for (int i = 0; i < n; i++) {
            balance = (loan - payment) * (1 + (rate / 100));
            loan = balance;
        }
        return balance;
    }
}
```

```java
/** String processing exercise 1. */
public class LowerCase {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(lowerCase(str));
    }

    /**
     * Returns a string which is identical to the original string,
     * except that all the upper-case letters are converted to lower-case letters.
     * Non-letter characters are left as is.
     */
    public static String lowerCase(String s) {
        String result = "";
        for (int i = 0; i < s.length(); i++) {
            if (('A' <= s.charAt(i)) && (s.charAt(i) <= 'Z')) {
                result = result + ((char) (s.charAt(i) + 32));
            } else {
                result = result + s.charAt(i);
            }
        }
        return result;
    }
}
```

```java
/** String processing exercise 2. */
public class UniqueChars {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(uniqueChars(str));
    }

    /**
     * Returns a string which is identical to the original string,
     * except that all the duplicate characters are removed,
     * unless they are space characters.
     */
    public static String uniqueChars(String s) {
        String result = "";

        for (int i = 0; i < s.length(); i++) {
            if (result.indexOf(s.charAt(i)) == -1 || s.charAt(i) == 32) {
                result = result + s.charAt(i);
            }
        }
        return result;
    }
}
```

```java
/**
 * Prints the calendars of all the years in the 20th century.
 */
public class Calendar {
    // Starting the calendar on 1/1/1900
    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 1900;
    static int dayOfWeek = 2; // 1.1.1900 was a Monday
    static int nDaysInMonth = 31; // Number of days in January

    /**
     * Prints the calendars of all the years in the 20th century. Also prints the
     * number of Sundays that occured on the first day of the month during this
     * period.
     */
    public static void main(String args[]) {
        // Advances the date and the day-of-the-week from 1/1/1900 till 31/12/1999,
        // inclusive.
        // Prints each date dd/mm/yyyy in a separate line. If the day is a Sunday,
        // prints "Sunday".
        // The following variable, used for debugging purposes, counts how many days
        // were advanced so far.
        int givenYear = Integer.parseInt(args[0]);

        //Checks the days until the given days
        while (year < givenYear) {
            advance();
        }

        while (year < (givenYear + 1)) {
            System.out.print(dayOfMonth + "/" + month + "/" + year);
            if (dayOfWeek == 1) {
                System.out.print(" Sunday");
            }
            System.out.println();
            advance();
        }
    }

    // Advances the date (day, month, year) and the day-of-the-week.
    // If the month changes, sets the number of days in this month.
    // Side effects: changes the static variables dayOfMonth, month, year,
    // dayOfWeek, nDaysInMonth.
    private static void advance() {
        if (dayOfWeek != 7) {
            dayOfWeek++;
        } else {
            dayOfWeek = 1;
        }
        if (dayOfMonth == nDaysInMonth(month, year)) {
            if (month == 12) {
                year++;
                month = 1;
                dayOfMonth = 1;
            } else {
                month++;
                dayOfMonth = 1;
            }
        } else {
            dayOfMonth++;
        }
    }

    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        boolean isLeapYear;
        // Checks if the year is divisible by 400
        isLeapYear = ((year % 400) == 0);
        // Then checks if the year is divisible by 4 but not by 100
        isLeapYear = isLeapYear || (((year % 4) == 0) && ((year % 100) != 0));
        return isLeapYear;
    }

    // Returns the number of days in the given month and year.
    // April, June, September, and November have 30 days each.
    // February has 28 days in a common year, and 29 days in a leap year.
    // All the other months have 31 days.
    private static int nDaysInMonth(int month, int year) {
        int daysInMonth = 0;
        switch (month) {
            case 1:
                daysInMonth = 31;
                break;
            case 2:
                if (isLeapYear(year)) {
                    daysInMonth = 29;
                } else {
                    daysInMonth = 28;
                }
                break;
            case 3:
                daysInMonth = 31;
                break;
            case 4:
                daysInMonth = 30;
                break;
            case 5:
                daysInMonth = 31;
```

```
                break;
            case 6:
                daysInMonth = 30;
                break;
            case 7:
                daysInMonth = 31;
                break;
            case 8:
                daysInMonth = 31;
                break;
            case 9:
                daysInMonth = 30;
                break;
            case 10:
                daysInMonth = 31;
                break;
            case 11:
                daysInMonth = 30;
                break;
            case 12:
                daysInMonth = 31;
                break;
        }

        return daysInMonth;
    }
}
```