```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);
    }

    /**
     * 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),
```

```java
     * 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 g = loan / n; // Initial guess of payment
        iterationCounter = 0;

        while (endBalance(loan, rate, n, g) > 0) {
            g += epsilon;
            iterationCounter++;
        }

        return g;
    }

    /**
     * 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) {
        double L = loan / n;
        double H = loan;
        double g = (L + H) / 2.0;
        iterationCounter = 0;

        while (H - L > epsilon) {
            if (endBalance(loan, rate, n, g) * endBalance(loan, rate, n, L) > 0) {
                L = g;
            } else {
                H = g;
            }

            g = (L + H) / 2.0;
            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 needToPay = loan;
        for (int i = 0; i < n; i++) {
            needToPay = (needToPay - payment) * (1 + rate / 100);
        }

        return needToPay;
    }
}
```

```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 lowerCaseString = "";

        for (int i = 0; i < s.length(); i++) {
            char character = s.charAt(i);
            if ((character >= 'A' && character <= 'Z')) {
                lowerCaseString += (char) (character + 32);
            } else {
                lowerCaseString += character;
            }
        }

        return lowerCaseString;
    }
}
```

```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 uniqueCharsString = "";

        for (int i = 0; i < s.length(); i++) {
            char character = s.charAt(i);
            if (uniqueCharsString.indexOf(character) == -1 || character == ' ') {
                uniqueCharsString += character;
            }
        }

        return uniqueCharsString;
    }
}
```

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

    /**
     * Prints the calendar of a specific year
     */
    public static void main(String args[]) {
        int year = Integer.parseInt(args[0]);

        // Advances the date and the day-of-the-week from 1/1/1900 till 31/12/year, inclusive.
        while (currentYear < year) {
            advance();
        }

        // Prints each date dd/mm/yyyy in a separate line. If the day is a Sunday, prints "Sunday".
        while (currentYear == year) {
            String dateMessage = dayOfMonth + "/" + month + "/" + currentYear;

            if (dayOfWeek == 1) {
                System.out.println(dateMessage + " Sunday");
            } else {
                System.out.println(dateMessage);
            }

            advance();
        }
    }
```

```java
// 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 = 1;
    } else {
        dayOfWeek++;
    }

    if (dayOfMonth == nDaysInMonth) {
        dayOfMonth = 1;

        if (month == 12) {
            month = 1;
            currentYear++;
        } else {
            month++;
        }

        nDaysInMonth = nDaysInMonth(month, currentYear);
    } else {
        dayOfMonth++;
    }
}

// Returns true if the given year is a leap year, false otherwise.
private static boolean isLeapYear(int year) {
    return (year % 400 == 0) || ((year % 4 == 0) && (year % 100 != 0));
}
```

```
    // 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) {
        switch (month) {
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
            case 2:
                if (isLeapYear(year)) {
                    return 29;
                } else {
                    return 28;
                }
            default:
                return 31;
        }
    }
}
```