```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),
    * 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;
        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 h = loan;
        double l = loan / n;
        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));
    }

    public static String lowerCase(String str) {
          String isSolution = " ";
               for (int i = 0; i < str.length(); i++) {
                char currentChar = str.charAt(i);
                  if (currentChar >= 'A' && currentChar <=
'Z') {
                       currentChar =  (char) (currentChar +
32);
                }

                isSolution += currentChar;

            }

            return isSolution;

        }
    }
```

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


    public static String uniqueChars(String str) {
        String isSolution = "";
          for (int i = 0; i < str.length(); i++) {
            char currentChar = str.charAt(i);

            if (isSolution.indexOf(currentChar) == -1
||(currentChar == ' ')) {
              isSolution += currentChar;



            }



       }



              return isSolution;


  }

}
```

```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
    static int sumOfSundays = 0;


    /**
     * 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[]) {
     int isTheYear = Integer.parseInt(args[0]);
        for (year = 1900; year < isTheYear; year++) {
            for (month = 1; month <= 12; month++) {
                for (dayOfMonth = 1; dayOfMonth <=
nDaysInMonth(month,year); dayOfMonth++) {
                    if(dayOfWeek == 8) {
                        dayOfWeek = 1;
                        dayOfWeek++;

                    } else {
                        dayOfWeek++;
                }

            }

        }

    }
        while (year == isTheYear) {
            for (month = 1; month <= 12; month++) {
                for (dayOfMonth = 1; dayOfMonth <=
nDaysInMonth(month,year); dayOfMonth++) {
                    if(dayOfWeek == 8) {
                    System.out.println(dayOfMonth + "/" +
month + "/" + year + " Sunday");

                    dayOfWeek = 1;
                    dayOfWeek++;
```

```
                    } else {
                    System.out.println(dayOfMonth + "/" +
month + "/" + year);
                    dayOfWeek++; }

                }

            }
                 year++;
        }

    }



        // 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 debugDaysCounter = 0;
        //// Write the necessary initialization code, and
replace the condition
        //// of the while loop with the necessary condition
        //while (true) {
            //// Write the body of the while
            //advance();
        //  debugDaysCounter++;
            //// If you want to stop the loop after n days,
replace the condition of the
            //// if statement with the condition
(debugDaysCounter == n)
        //  if (false) {
            //  break;
            //}
    //    }
        //// Write the necessary ending code here
    // }

     // 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() {
```

```
        // Replace this comment with your code
     //}

    // Returns true if the given year is a leap year, false
otherwise.
    private static boolean isLeapYear(int year) {
        if(year==1900) return false;
        if (year % 4 == 0)
        return true;
        else
        return false;
    }


    // 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) {
        if(month==4 || month==6 || month==9 || month==11)
return 30;
        if(month==1 || month==3 || month==5 || month==7 ||
month==8 || month==10 || month==12) return 31;
        if(month==2) {
            if(isLeapYear(year) == true) return 29;
            else return 28;
        }
        return 0;
    }

}
```