```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) {
        // Replace the following statement with your code
        iterationCounter = 0;
        double g = loan / n;
        // bruteForcePayment += epsilon;

        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) {
        // Replace the following statement with your code
        // Sets L and H to initial values such that f(L) > 0, f(H)
< 0,
        iterationCounter = 0;
        double L = 0;
        double H = loan;

        // Set initial guess g to the middle of L and H
        double g = (L + H) / 2;

        while ((H - L) > epsilon) {
            iterationCounter++;

            // Compute the values of the function at g, L and H
            double fG = endBalance(loan, rate, n, g);
            double fLo = endBalance(loan, rate, n, L);
            double fHi = endBalance(loan, rate, n, H);
```

```
            // Check the sign of the product of fG and fLo (or fG
and fHi)
            if (fG * fLo > 0) {
                // Solution is between g and H
                L = g;
            } else {
                // Solution is between L and g
                H = g;
            }

            // Update g for the next iteration
            g = (L + H) / 2;
        }

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

```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) {
        // Replace the following statement with your code
        String low = "";
        char currentChar;
        int charAscii = 0;
        for (int i = 0; i < s.length(); i++)
        {
            currentChar = s.charAt(i);
            charAscii = (int) currentChar;

            // if the current char ascii represents a capital
letter
            if (charAscii <= 90 && charAscii >= 65)
            {
                // the difference in the ascii table for
representing lower case letters
                // from its upper case equivelant
                charAscii += 32;
            }
            currentChar = (char) charAscii;

            low = low + currentChar;
        }
        return low;
    }
}
```

```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) {
        // Replace the following statement with your code
        char currentChar = s.charAt(0);
        String unique = "" + currentChar;

        boolean flag = true;
        for (int i = 1; i < s.length(); i++)
        {
            currentChar = s.charAt(i);

            for (int j = 0; j < unique.length(); j++)
            {
               if (unique.charAt(j) == currentChar)
               {
                   flag = false;
                   break;
               }
            }

            if (flag == true || currentChar == ' ')
            {
               unique = unique + currentChar;
            }
            flag = true;
        }
        return unique;
    }
}
```

```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 nDaysInMonth = 31; // Number of days in January

    static boolean lastDay = false;

    public static void main(String args[]) {
        int year = Integer.parseInt(args[0]);
        int dayOfWeek = whatDay(year);

        // Prints each date dd/mm/yyyy in a separate line. If the
day is a Sunday, prints "Sunday".
        int debugDaysCounter = 0;

        while (!lastDay)
        {
            if (dayOfWeek == 0)
            {
                System.out.println(dayOfMonth + "/" + month + "/"
+ year + " Sunday");
            }
            else
            {
                System.out.println(dayOfMonth + "/" + month + "/"
+ year);
            }
            dayOfWeek = (dayOfWeek + 1) % 7;
            advance(year);

            debugDaysCounter++;
        }
        //// 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(int year) {
        // Replace this comment with your code
            if (dayOfMonth == nDaysInMonth)
            {
                dayOfMonth = 1;
```

```
                if (month == 12)
                {
                    lastDay = true;
                }
                else
                {
                    month++;
                    nDaysInMonth = nDaysInMonth(month, year);
                }
            }
            else
            {
                dayOfMonth++;
            }
        }

    private static int whatDay(int year){
        int countYear = 1900;
        int weekDay = 1; // 1.1.1900 was a Monday

        for (int i = 1900; i < year; i++)
        {
            if(isLeapYear(i))
            {
                weekDay ++;
            }

            weekDay++;
            countYear ++;
        }

        return (weekDay % 7);
    }

    // Returns true if the given year is a leap year, false
otherwise.
    private static boolean isLeapYear(int year) {
        // Replace the following statement with your code
        if (year % 4 != 0)
        {
            return false;
        }

        if (year % 100 == 0 && year % 400 != 0)
        {
            return false;
        }

        return true;
    }
```

```java
    // 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) {
        // Replace the following statement with your code
        int numDays = 0;
        switch (month)
        {
            case 4:
            case 6:
            case 9:
            case 11:
                numDays = 30;
                break;

                case 2:
                    if (isLeapYear(year))
                    {
                        numDays = 29;
                    }
                    else
                    {
                        numDays = 28;
                    }
                    break;

            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numDays = 31;
                break;

        }
        return numDays;
    }
}
```