```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 NewPayment = loan/n;
        double increment = 0.001;
        while((endBalance(loan, rate, n, NewPayment) >=
epsilon) && (NewPayment <= loan)){
            NewPayment += increment;
            iterationCounter++;
        }
        return NewPayment;
    }

    /**
    * 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
        iterationCounter = 0;
        double H = loan + 1;
        double L = 0;
        double g = (L + H) / 2.0;
        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;
            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) {
        // Replace the following statement with your code
        double EndSum = loan;
        for (int i = 1; i <= n; i++){
            EndSum = (EndSum - payment) * (1 + rate/100);
        }
        return EndSum;
    }
}
```

```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 NewWord = "";
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == ' '){
                NewWord = NewWord + " ";
            }
            else if ((s.charAt(i) >= 65 && s.charAt(i) <=
90)){
                NewWord = NewWord + (char)(s.charAt(i) + 32);
            }
            else{
                 NewWord = NewWord + s.charAt(i);
            }
        }
        return NewWord;
    }
}
```

```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 NewWord = "";
        for (int i = 0; i < s.length(); i++) {
            boolean flag = false;
            if (s.charAt(i) == ' '){
                NewWord = NewWord + " ";
            }
            else if ((s.charAt(i) != ' ')){
                for (int j = 0; j < i; j++){
                    if (s.charAt(i) == s.charAt(j)){
                        flag = true;
                    }
                }
                if (!flag){
                   NewWord = NewWord + s.charAt(i);
                }
            }
        }
        return NewWord;
    }
}
```

```
public class Calendar {
        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

        public static void main(String args[]) {
            int GivenYear = Integer.parseInt(args[0]);
            while (year < GivenYear) {
                advance();
            }

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


         private static void advance() {

            dayOfMonth++;
            if (dayOfMonth > nDaysInMonth){
                dayOfMonth =1;
                month ++;
                 if (month > 12){
                    month = 1;
                    year ++;
                }
                nDaysInMonth = nDaysInMonth(month, year);
            }

            dayOfWeek ++;
            if (dayOfWeek==8) {
                dayOfWeek = 1;
            }
```

```
         }


        private static boolean isLeapYear(int year) {
            boolean leapyear = ((year % 400 == 0) || (year % 4
== 0 && year % 100 != 0));
            return leapyear;
        }

        private static int nDaysInMonth(int month, int year) {
            int numDays = 0;
        if (month == 1){
            numDays = 31;
        }
        if (month == 2) {
            if (isLeapYear(year)) {
                numDays = 29;
            } else {
                numDays = 28;
            }
        }
        if (month == 3) {
            numDays = 31;
        }
        if (month == 4) {
            numDays = 30;
        }
        if (month == 5) {
            numDays = 31;
        }
        if (month == 6) {
            numDays = 30;
        }
        if (month == 7){
            numDays = 31;
        }
        if (month == 8){
            numDays = 31;
        }
        if (month == 9){
            numDays = 30;
        }
        if (month == 10){
            numDays = 31;
        }
        if (month == 11){
            numDays = 30;
        }
```

```
        if (month == 12){
            numDays = 31;
        }
        return numDays;
        }

}
```