```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.
```

```java
    * 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) {
        iterationCounter = 0;
        double g = loan / n;
        while (endBalance(loan, rate, n, g) >= epsilon) {
            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
        iterationCounter = 0;
        double hi = loan;
        double lo = 0;
        double g = (lo + hi) / 2;
        while ((hi - lo) >= epsilon) {
            if ((endBalance(loan, rate, n, g) * endBalance(loan, rate,
n, lo)) >= 0){
                lo = g;
            } else {
                hi = g;
            }
            g = (lo + hi) / 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) {
        double balance = loan;
        for(int i = 0; i < n; i++){
            balance = (balance - payment) * (1 + (rate / 100));
        }
        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) {
        // Replace the following statement with your code
        String res = "";
        for (int i = 0; i < s.length(); i++){
            char chr = s.charAt(i);
            if (chr >= 65 && chr <= 90){
                chr = (char)(chr + 32);
                res = res + chr;
            } else {
                res = res + chr;
            }
        }
        return res;
    }
}
```

```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
        String res = "";
        for(int i = 0; i < s.length(); i++){
            char current = s.charAt(i);
            if (current == ' ' || res.indexOf(current) == -1){
                res = res + current;
            }
        }
        return res;
    }
}
```

```
public class Calendar{
    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 1900;
    static int dayOfWeek = 2;
    static int nDaysInMonth = 31;

    public static void main(String args[]) {
        int currentyear = Integer.parseInt(args[0]);
        while (year < currentyear) {
            while (month < 13) {
                nDaysInMonth = nDaysInMonth(month, year);
                for(int i = 1; i <= nDaysInMonth; i++){
                    if(dayOfWeek == 7){
                        dayOfWeek = 1;
                    }else {
                        dayOfWeek++;
                    }
                }
                month++;
            }
        year++;
        month = 1;
        }
        month = 1;
        for(int j = 1; j <= 12; j++){
            advance(dayOfMonth, month, year);
            month++;
        }
        if (month == 13){
            month = 1;
        }
        year++;

}


     private static void advance(int dayOfMonth, int month, int year) {
        for(dayOfMonth = 1; dayOfMonth <= nDaysInMonth(month, year);
dayOfMonth++){
            System.out.print(dayOfMonth + "/" + month + "/" + year);
            if(dayOfWeek == 1){
                System.out.print(" Sunday");
            }
            if(dayOfWeek == 7){
                dayOfWeek = 1;
            } else {
                dayOfWeek++;
```

```
            }

            System.out.println();
        }
    }

    public static boolean isLeapYear(int year) {
        if ((year % 400 == 0) || ((year % 100 != 0) && (year % 4 ==
0))) {
            return true;
            } else {
            return false;
            }
    }


    public 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;
        }
    }
}
```