Loan Calc

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

        }
        // Replace the following statement with your code
        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) {
        iterationCounter = 0;

        double L = loan/n;
        double H = loan+epsilon;
        double g = (L+H)/2;

        while (H-L > epsilon) {
            // H/=2;
            // L/=2;
            if(endBalance(loan, rate, n, g)*endBalance(loan, rate,
n, L)>0){
                L = g;
            } else{
                H = g;
            }

            g = (L+H)/2;

            iterationCounter++;
        }
        // Replace the following statement with your code
        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=1;i<=n;i++){
            loan = (loan-payment)*(1+rate/100);
        }
        // Replace the following statement with your code
        return loan;
    }
}
```

Lower Case

```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 str="";
        for(int i=0;i<s.length();i++){
            char letter = s.charAt(i);
            if(letter>='A' && letter<='Z'){
                letter+=32;
            }
            str+=letter;


        }
        // Replace the following statement with your code
        return str;
    }
}
```

Unique Character:

```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 str = "";
        // Replace the following statement with your code
        for(int i=0;i<s.length();i++){
            if(str.indexOf(s.charAt(i))==-1||s.charAt(i)==' '){
                str += s.charAt(i);
            }
        }
        return str;
    }
}
```

Calendar:

```
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;
    public static void main(String args[]) {
            int theYear = Integer.parseInt(args[0]);


        while (theYear>year) {
            advance();

        }
        while (theYear==year){

System.out.println(dayOfMonth+"/"+month+"/"+year+(dayOfWeek==1? "
Sunday":""));
            advance();
        }
     }


     private static void advance() {
            dayOfWeek++;
            dayOfWeek%=7;
        if(dayOfMonth>=nDaysInMonth(month,year)){
            dayOfMonth=1;
            month++;

            if(month>12){
                month%=12;
                year++;

            }
        } else dayOfMonth++;
     }

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


    private static int nDaysInMonth(int month, int year) {
        int daysOfMonth;
```

```
        switch (month) {

    case 2:  daysOfMonth = isLeapYear(year)? 29 : 28;
             break;

    case 4:  daysOfMonth = 30;
             break;

    case 6:  daysOfMonth = 30;
             break;

    case 9:  daysOfMonth = 30;
             break;

    case 11: daysOfMonth = 30;
             break;

    default: daysOfMonth = 31;
             break;

}
        return daysOfMonth;
    }
}
```