Home work 3

LoanCalc

```
/**
* 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 x = loan/n;
    double increment = 0.001;
    iterationCounter = -1;
    double value;
    do {
        value = endBalance(loan, rate, n, x);
        x = x + increment;
        iterationCounter++;
    } while ((value > epsilon) && (value > 0));

    return x;
}

/**
* 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 L = loan/n;
        iterationCounter = 0;
        double H = loan + 1;
        double g = 0;
            while ((H - L) > epsilon) {
                g = (L + H) / 2;
                if (endBalance(loan, rate, n, g) * endBalance(loan, rate, n, L) > 0){
                    L = g;
                } else {
                    H = g;
                }
                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 value = loan;
        for ( int i = 0; i < n; i++ ){
            value = (value - payment) * ((rate / 100) + 1);
        }
        return value;
    }
}
```

LowerCase

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

        }
        return ans;
    }
}
```

uniqueChars

```
/** 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 ans = "";
        for ( int i = 0; i < s.length(); i++){
            if ( s.indexOf (s.charAt(i)) == i){
                ans = ans + s.charAt(i);
            } else if ( s.charAt(i) == ' '){
                ans = ans + s.charAt(i);
            }
        }
        return ans;
    }
}
```

Calendar

```
/**
         * Prints the calendars of the chosen year. also prints sunday on the first day of every
week.
         */
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

        public static void main(String args[]) {
                int PickYear = Integer.parseInt(args[0]);
                // Advances the date and the day-of-the-week from 1/1/1900 till the chosen
year.
                //Then prints the calender of the chosen year.
           //int debugDaysCounter = 0;
                while (year < PickYear) {
                        advanceuntilPickYear();
                        year++;
                }
                if ( year == PickYear){
                        advance();
                }
                        // debugDaysCounter++;
                        //   if (debugDaysCounter == n) {
                        //         break;
                        // }
        }

         // 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 advanceuntilPickYear() {
                for ( int m = 1; m <= 12; m++){

                        dayOfMonth = nDaysInMonth(m, year);
                        dayOfWeek = (dayOfMonth + dayOfWeek - 28) % 7;

                }
        }
        private static void advance() {
                for ( int m = 1; m <= 12; m++){
                        dayOfMonth = nDaysInMonth(m, year);
                        ;
                        for (int d = 1; d <= dayOfMonth; d++ ){
                                if (dayOfWeek == 1){
                                        System.out.printf("%d/%d/%d Sunday \n", d, m,
year );
                                } else {
                                        System.out.printf("%d/%d/%d \n" , d, m, year );
                                }
                                dayOfWeek = (dayOfWeek + 1) % 7;
                        }
                }
        }


   // Returns true if the given year is a leap year, false otherwise.
        private static boolean isLeapYear(int year) {
          if ((year % 400 == 0 ) || (( year % 4 == 0 ) && ( year % 100 != 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) {
        switch (month) {
            case 1: return 31;
            case 2:
            if (isLeapYear(year) == true) {
                    return 29;
            } else {
                return 28;
            }
            case 3: return 31;
            case 4: return 30;
            case 5: return 31;
            case 6: return 30;
            case 7: return 31;
            case 8: return 31;
            case 9: return 30;
            case 10: return 31;
            case 11: return 30;
            case 12: return 31;
            default: break;
        }
        return -1;
    }
}
```