Homework 3

Calendar0

```
/*
 * Checks if a given year is a leap year or a common year,
 * and computes the number of days in a given month and a
given year.
 */
public class Calendar0 {

    // Gets a year (command-line argument), and tests the
functions isLeapYear and nDaysInMonth.
    public static void main(String args[]) {
        int year = Integer.parseInt(args[0]);
        isLeapYearTest(year);
        nDaysInMonthTest(year);
    }

    // Tests the isLeapYear function.
    private static void isLeapYearTest(int year) {
        String commonOrLeap = "common";
        if (isLeapYear(year)) {
            commonOrLeap = "leap";
        }
        System.out.println(year + " is a " + commonOrLeap +
" year");
    }


    // Tests the nDaysInMonth function.
    private static void nDaysInMonthTest(int year) {
        for (int i = 1; i <= 12; i++) {
            System.out.println("Month " +i+ " has
"+nDaysInMonth(i, year)+" days");

        }
```

```
    }



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


    public static int nDaysInMonth(int month, int year) {
        switch (month) {
            case 2:
                if(isLeapYear(year))
                return 29;
                return 28;
            case 9:
              case 11:
            case 4:
              case 6:
                return 30;

            case 1:
              case 3:
            case 5:
              case 7:
              case 8:
              case 10:
              case 12:
                return 31;
        }
        return 0;
    }

}
```

Calendar1

```
/**
 * Prints the calendars of all the years in the 20th
century.
 */
public class Calendar1 {
    // 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
    static int totalSundaysOnFirst = 0;

    /**
     * Prints the calendars of all the years in the 20th
century. Also prints the
     * number of Sundays that occured on the first day of
the month during this period.
     */
    public static void main(String args[]) {
        // Advances the date and the day-of-the-week from
1/1/1900 till 31/12/1999, inclusive.
        // Prints each date dd/mm/yyyy in a separate line.
If the day is a Sunday, prints "Sunday".
        // The following variable, used for debugging
purposes, counts how many days were advanced so far.
        //// Write the necessary initialization code, and
replace the condition
        //// of the while loop with the necessary condition
        int debugDaysCounter = 0;
        while (year <= 1999) {
            advance();
            debugDaysCounter++;
            if (debugDaysCounter == 100000) {
                break;
```

```
            }
        }
        System.out.println("During the 20th century, "
+totalSundaysOnFirst+" Sundays fell on the first day of the
month");
    }


        //// 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() {
        String whereIsSunday ="Sunday";
        if (dayOfWeek == 0 || dayOfWeek == 7) {
            System.out.println(dayOfMonth +"/"+month +"/"+
year+ " "+ whereIsSunday);
            if (dayOfMonth==1){
            totalSundaysOnFirst++;
        }
        } else{
            System.out.println(dayOfMonth +"/"+month+"/"+
year);
        }


        dayOfMonth++;
        dayOfWeek = (dayOfWeek + 2) % 7;


        if (dayOfMonth > nDaysInMonth(month, year)) {
```

```
            dayOfMonth = 1;
            month++;


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

    }



    // Returns true if the given year is a leap year, false
otherwise.
    private static boolean isLeapYear(int year) {
        return (year % 4 == 0 && (year % 100 != 0 || year %
400 == 0));
    }
     public static int nDaysInMonth(int month, int year) {
        switch (month) {
            case 2:
                if(isLeapYear(year))
                return 29;
                return 28;
            case 9:
               case 11:
            case 4:
               case 6:
                return 30;

            case 1:
               case 3:
            case 5:
               case 7:
               case 8:
```

```
                case 10:
                case 12:
                  return 31;
            }
            return 0;
        }

}
```

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 keepGo = loan /n;
```

```
        while (endBalance(loan, rate, n, keepGo) >
epsilon){
            keepGo += epsilon;
            iterationCounter++;
        }
     return keepGo;
    }

    /**
     * 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 low = 0;
        double high = loan;
        iterationCounter=0;
     while (high-low>epsilon) {
            double mid = (high+low) / 2.0;
            iterationCounter++;
        if (endBalance(loan, rate, n, mid) > epsilon) {
         low = mid;
        }else{
            high = mid;
    }
}
     return (low+high)/2;
    }
```

```
    /**
     * 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;
    }
}
```

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 rasult  ="";
        for (int i = 0; i < s.length(); i++){
            if (Character.isLetter(s.charAt(i))){
                rasult +=
```

```
Character.toLowerCase(s.charAt(i));
            }
            else{
                rasult += s.charAt(i);
            }
        }
        return rasult;
    }
}
```

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