HW03 – Adi Lind (322493107)

LoanCalc.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
        double g = loan/n;
        iterationCounter=0;
        double increment = epsilon;
        while(endBalance(loan, rate, n, g) >= 0)
        {
            g = g+ increment;
            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
        double high = loan;
        double low = loan/n;
        double initialGuess = (high + low) / 2;
        iterationCounter = 0;
        while( (high - low) >= epsilon)
        {
            if (endBalance(loan, rate, n, initialGuess) * endBalance(loan,
rate, n, low) > 0)
            {
                low = initialGuess;
            }
            else
            {
                high = initialGuess;
```

```
            }
            iterationCounter++;
            initialGuess = (low + high)/2;
            //System.out.println("correct intial in the loop is  " +
initialGuess);
            //System.out.println("correct endbalance is " + endBalance(loan,
rate, n, initialGuess));
        }
        return initialGuess;
    }

    /**
    * 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 = 0 ; i <= n-1 ; i++)
        {
            loan = (loan - payment) * (1 + (rate / 100));
        }
        return loan;
    }
}
```

LowerCase.java

```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
        int size = s.length();
        String newS = ""; //make the new s we get but without Upper-case.
        for(int i = 0 ; i < size; i++)
        {
            char temp = s.charAt(i);
            if (temp >= 65 && temp <= 90) // the ascii code of upper
            {
                temp = (char) (temp + 32) ;
                newS = newS + temp;
            }
            else
            {
                newS = newS + temp;
            }

        }

        return newS;
    }
}
```

UniqueChars.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
        int size = s.length();
        String finalString = "";
        for(int i = 0; i<size ; i++)
        {
            char temp = s.charAt(i);
            if ((temp == ' ') || (finalString.indexOf(String.valueOf(temp)) ==
-1))
            {
                finalString = finalString + temp;
            }

        }
        return finalString;
    }
}
```

Calendar.java-

```
public class Calendar {
    /**
 * Prints the calendars of all the years in the 20th century.
 */
    // 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

    /**
     * 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.
        int debugDaysCounter = 0;
        int SundaysCount = 0;
        int demandyear = Integer.parseInt(args[0]);
        //// Write the necessary initialization code, and replace the
condition
        //// of the while loop with the necessary condition
        while (year <= demandyear) {
            //// Write the body of the while

            if (year == demandyear)
            {
                if (dayOfWeek == 1)
            {
                System.out.println(dayOfMonth + "/" + month + "/" + year + "
Sunday");
                SundaysCount++;
            }
            else
            {
                System.out.println(dayOfMonth + "/" + month + "/" + year);
            }
            }
```

```
            advance();
            debugDaysCounter++;

            //// If you want to stop the loop after n days, replace the
condition of the
            //// if statement with the condition (debugDaysCounter == n)
            /*if ((year == 2000)) {
                break;
            } */
        }
        //// Write the necessary ending code here
        // System.out.println("During the 20th century, " + SundaysCount + "
Sundays fell on the first day of the month");
         //System.out.println("number of lines" + debugDaysCounter);
    }

    // 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() {
       // Replace this comment with your code

       if (dayOfMonth == nDaysInMonth(month, year)) {
           month++;
           dayOfMonth = 1;
           if (month > 12) {
               month = 1;
               year++;
           }
       } else {
           dayOfMonth++;
       }
       dayOfWeek = (dayOfWeek + 1) % 7; //add one for the day and save the
format of 7 days per week
    }

   // Returns true if the given year is a leap year, false otherwise.
   private static boolean isLeapYear(int year) {
       Boolean isLeapYear = false;
       isLeapYear = ((year%400) == 0);
       isLeapYear = isLeapYear || (((year %4 == 0) && (year % 100 != 0)));

       return isLeapYear;
   }

   // Returns the number of days in the given month and year.
   // April, June, September, and November have 30 days each.
```

```java
// 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) {
    // Replace the following statement with your code
    int op1 = 30;
    int op2 = 31;
    switch (month) {
        case 1:
            return op2;

        case 2:
            if(isLeapYear(year))
            {
                return 29;
            }
            else
            {
                return 28;
            }

        case 3:
            return op2;

        case 4:
            return op1;

        case 5:
            return op2;

        case 6:
            return op1;

        case 7:

            return op2;
        case 8:
            return op2;

        case 9:
            return op1;

        case 10:
            return op2;

        case 11:
            return op1;

        case 12:
```

```
                return op2;


            default:
                break;
        }
        return 0;
    }
    }
```

Bonus-

Calendar0.java

```
/*
 * 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) {
        // Replace this comment with your code
        for (int i=1 ; i<= 12; i++)
        {
            System.out.println("Month " + i + " has " + nDaysInMonth(i, year)
+ " days");
        }
    }

    // Returns true if the given year is a leap year, false otherwise.
    public static boolean isLeapYear(int year) {
        // Replace the following statement with your code
        Boolean isLeapYear = false;
        isLeapYear = ((year%400) == 0);
        isLeapYear = isLeapYear || (((year %4 == 0) && (year % 100 != 0)));

        return isLeapYear;
    }

    // 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.
```

```java
    // All the other months have 31 days.
    public static int nDaysInMonth(int month, int year) {
        // Replace the following statement with your code
        int op1 = 30;
        int op2 = 31;
        switch (month) {
            case 1:
                return op2;

            case 2:
                if(isLeapYear(year))
                {
                    return 29;
                }
                else
                {
                    return 28;
                }

            case 3:
                return op2;

            case 4:
                return op1;

            case 5:
                return op2;

            case 6:
                return op1;

            case 7:

                return op2;
            case 8:
                return op2;

            case 9:
                return op1;

            case 10:
                return op2;

            case 11:
                return op1;

            case 12:
                return op2;
```

```
            default:
                break;
        }
        return 0;
    }
}
```

Bonus 2

Calendar1.java

```
/**
 * 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

    /**
     * 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.
        int debugDaysCounter = 0;
        int SundaysCount = 0;
        //// Write the necessary initialization code, and replace the
condition
        //// of the while loop with the necessary condition
        while (year <= 1999) {
            //// Write the body of the while

            if (dayOfMonth == 1 && dayOfWeek == 1)
            {
                System.out.println(dayOfMonth + "/" + month + "/" + year + "
Sunday");
                SundaysCount++;
            }
            else
            {
                System.out.println(dayOfMonth + "/" + month + "/" + year);
            }

            advance();
```

```
            debugDaysCounter++;

            //// If you want to stop the loop after n days, replace the
condition of the
            //// if statement with the condition (debugDaysCounter == n)
            if ((year == 2000)) {
                break;
            }
        }
        //// Write the necessary ending code here
         System.out.println("During the 20th century, " + SundaysCount + "
Sundays fell on the first day of the month");
         //System.out.println("number of lines" + debugDaysCounter);
     }

     // 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() {
        // Replace this comment with your code

        if (dayOfMonth == nDaysInMonth(month, year)) {
            month++;
            dayOfMonth = 1;
            if (month > 12) {
                month = 1;
                year++;
            }
        } else {
            dayOfMonth++;
        }
        dayOfWeek = (dayOfWeek + 1) % 7; //add one for the day and save the
format of 7 days per week
     }

    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        Boolean isLeapYear = false;
        isLeapYear = ((year%400) == 0);
        isLeapYear = isLeapYear || (((year %4 == 0) && (year % 100 != 0)));

        return isLeapYear;
    }

    // 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.
```

```java
    // All the other months have 31 days.
    private static int nDaysInMonth(int month, int year) {
        // Replace the following statement with your code
        int op1 = 30;
        int op2 = 31;
        switch (month) {
            case 1:
                return op2;

            case 2:
                if(isLeapYear(year))
                {
                    return 29;
                }
                else
                {
                    return 28;
                }

            case 3:
                return op2;

            case 4:
                return op1;

            case 5:
                return op2;

            case 6:
                return op1;

            case 7:

                return op2;
            case 8:
                return op2;

            case 9:
                return op1;

            case 10:
                return op2;

            case 11:
                return op1;

            case 12:
                return op2;
```

```
            default:
                break;
        }
        return 0;
    }
    }
```

פתגם נדוש לאווירה הטובה –

"לא ניתן לשנות את כיוון הרוח אלא רק את כיוון המפרשים ⛵⛵"