HW03 Code – Noam Adda – ID 209087634

Loan calculations:

```
/**
 * 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;

        // Iterate until you're within epsilon
        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) {
        double L = epsilon;
        double H = loan;
        double g = (L + H) / 2;
        iterationCounter = 0;

        // Iterate over the calculation using the bisection method
        while ((H - L) > epsilon) {
            if ((endBalance(loan, rate, n, g) * endBalance(loan, rate, n, L)) > 0) {
                L = g;
            } else {
                H = g;
            }
            g = (L + H) / 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 amountLeftToPay = loan;
```

```
        // Calculate the amount left to pay
        for (int i = 0; i < n; i++) {
            amountLeftToPay = (amountLeftToPay - payment) * (1 + rate / 100);
        }
        return amountLeftToPay;
    }
}
```

Lower Case:

```
/**
 * 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) {

        // Create an empty string
        String lowerCaseWord = "";

        // Iterate through all the letters in the string and switch the upper case
and lower case letters
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if ((ch >= 'A') && (ch <= 'Z')) {
                lowerCaseWord += (char) (ch + 32);
            } else {
                lowerCaseWord += ch;
            }
        }
        return lowerCaseWord;
```

```
    }
}
```

Unique Characters:

```
/**
 * 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 uniqueWord = "";

        // Iterate through all the letters in the string and add the unique ones to the empty string
        for (int i = 0; i < s.length(); i++) {
            char currentCharacter = s.charAt(i);
            if (uniqueWord.indexOf(currentCharacter) == -1 || currentCharacter == ' ') {
                uniqueWord += currentCharacter;

            }
        }
        return uniqueWord;
    }
}
```

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++) {
            int numOfDaysInMonth = nDaysInMonth(i, year);
            System.out.println("Month " + i + " has " + numOfDaysInMonth + "
days");
        }
```

```
    }


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

    // 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.
    public static int nDaysInMonth(int month, int year) {
        switch (month){
            case 1,3,5,7,8,10,12:
                return 31;
            case 2:
                if (isLeapYear(year)){
                    return 29;
                } else {
                    return 28;
                }
            case 4,6,9,11:
                return 30;
            default:
                return -1;
        }
    }
}
```

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


    /**
     * 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 numOfSundaysOnTheFirstDay = 0;
        //// Write the necessary initialization code, and replace the condition
        //// of the while loop with the necessary condition
        while (year <= 1999) {
            if (dayOfWeek == 1) {
                System.out.printf("%d/%d/%d Sunday\n", dayOfMonth, month, year);
```

```
            if (dayOfMonth == 1) {
                numOfSundaysOnTheFirstDay++;
            }
        } else {
            System.out.printf("%d/%d/%d\n", 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 (debugDaysCounter == n) {
        //              break;
        //          }
    }
    System.out.printf("During the 20th century, %d Sundays fell on the first day of the month\n", numOfSundaysOnTheFirstDay);
  }


  // 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() {
    // Advances day in the week
    if (++dayOfWeek > 7) {
        dayOfWeek = 1;
    }
    // Advances day in month, month and year
    if (++dayOfMonth > nDaysInMonth(month, year)) {
        dayOfMonth = 1;
```

```
        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);
}

// 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, 3, 5, 7, 8, 10, 12:
            return 31;
        case 2:
            if (isLeapYear(year)) {
                return 29;
            } else {
                return 28;
            }
        case 4, 6, 9, 11:
            return 30;
        default:
            return -1;
```

```
        }
    }
}
```

Calendar:

```
/**
 * Prints the calendars of all the years in the 20th century.
 */
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

    /**
     * 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;

        // Parse the command line
        int givenYear = Integer.parseInt(args[0]);

        // Make sure the given year is greater than 1900
        if (givenYear < 1900) {
            System.out.printf("%d is an invalid year\n", givenYear);
```

```
            System.exit(1);
        }
        //// Write the necessary initialization code, and replace the condition
        //// of the while loop with the necessary condition
        while (year <= givenYear) {
            if (year == givenYear) {
                if (dayOfWeek == 1) {
                    System.out.printf("%d/%d/%d Sunday\n", dayOfMonth, month,
year);
                } else {
                    System.out.printf("%d/%d/%d\n", 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 (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 advance() {
        // Advances day in the week
        if (++dayOfWeek > 7) {
```

```
        dayOfWeek = 1;
    }
    // Advances day in month, month and year
    if (++dayOfMonth > nDaysInMonth(month, year)) {
        dayOfMonth = 1;
        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);
}

// 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, 3, 5, 7, 8, 10, 12:
            return 31;
        case 2:
            if (isLeapYear(year)) {
                return 29;
            } else {
                return 28;
```

```
            }
        case 4, 6, 9, 11:
            return 30;
        default:
            return -1;
      }
   }
}
```