Yarin Raichman

HW03

Loan calculations

```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).
     */

    // main(string[]) - the entry point of a Java program.
    // args - args contains the supplied command-line
    // arguments as an array of String objects.
    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);
    }
```

```java
    /**
    * 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.

    // bruteForceSolver - search for the ideal Periodical payment using
the brute force search methood.
    public static double bruteForceSolver(double loan, double rate, int n,
double epsilon) {
        // declaring a first gueass bigger the n 0 for the implementation
of the brute force search methood.
        double guess = loan / n;
        // reseting the value of the flobal variable that counts the
iterations of each search methood to 0.
        iterationCounter = 0;
        // implementation of the brute force search methood.
        while (endBalance(loan, rate, n, guess) >= 0) {
            guess += epsilon;
            iterationCounter++;
        }
        return guess;
    }

    /**
    * 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.

    // bisectionSolver - search for the ideal Periodical payment using the
bisection search methood.
    public static double bisectionSolver(double loan, double rate, int n,
double epsilon) {
        // declaring H, L and M values for the implementation of the
bisection search methood.
        double low = 0.0;
        double high = loan;
        double middle = (high + low) / 2;
```

```
        // reseting the value of the flobal variable that counts the
iterations of each search methood to 0.
        iterationCounter = 0;
        // implementation of the bisection search methood.
        while ((high - low) > epsilon) {
            if ((endBalance(loan, rate, n, middle) * endBalance(loan,
rate, n, low)) > 0) {
                low = middle;
            }
            else{
                high = middle;
            }
            middle = (high + low) / 2;
            iterationCounter++;
        }
        return middle;
    }

    /**
    * 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.
    */

    // endBalance - calculate the n'th period's balance by loan, rate,
periods and payment
    // according to the excel's calculations.
    private static double endBalance(double loan, double rate, int n,
double payment) {
        // declaring the last balance variable starting from loan
downwards.
        double bal = loan;
        for (int i = 0; i < n; i++) {
            // calculating the next period's balance as the excel does.
            bal = (bal - payment) * (1 + rate/100);
        }
        return bal;
    }
}
```

Lower case

```java
/** String processing exercise 1. */
public class LowerCase {
    // main(string[]) - the entry point of a Java program.
    // args - args contains the supplied command-line
    // arguments as an array of String objects.
    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 loweString = "";
        for (int i = 0; i < s.length(); i++) {
            if ((s.charAt(i) >= 'A') && (s.charAt(i) <= 'Z')) {
                loweString += (char)(s.charAt(i) + 32);
            }
            else {
                loweString += s.charAt(i);
            }
        }
        return loweString;
    }
}
```

Unique characters

```java
/** String processing exercise 2. */
public class UniqueChars {
    // main(string[]) - the entry point of a Java program.
    // args - args contains the supplied command-line
    // arguments as an array of String objects.
    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.
     */

    // isIn - checks if a character(c) is in a string(s).
    private static boolean isIn(char c, String s) {
        for (int i = 0; i < s.length(); i++) {
            // check if character(c) is equal to each character in
string(s) one by one.
            if (c == s.charAt(i)) {
                return true;
            }
        }
        return false;
    }

     // uniqueChars - returns a string without any indentical characters
except for " "
     // in the order and content provided by string(s).
    public static String uniqueChars(String s) {
        // declaring the new string withe unique characters as an empty
string.
        String uniqString = "";
        for (int i = 0; i < s.length(); i++) {
            if (!(isIn(s.charAt(i), uniqString)) || (s.charAt(i) == ' '))
{
                // adding a unique character or a " " to the string withe
the unique characters.
                uniqString += s.charAt(i);
            }
        }
        return uniqString;
    }
}
```

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 {
    // main(string[]) - the entry point of a Java program.
    // args - args contains the supplied command-line
    // arguments as an array of String objects.
    // 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");
        }
    }

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

```
            return 30;
        }
        else if (month == 2) {
            if (isLeapYear(year)) {
                return 29;
            }
            else {
                return 28;
            }
        }
        else {
            return 31;
        }
    }
}
```

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.
     */

    // main(string[]) - the entry point of a Java program.
    // args - args contains the supplied command-line
    // arguments as an array of String objects.
    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 nSundaysFirstDM = 0;
        //// Write the necessary initialization code, and replace the
condition
        //// of the while loop with the necessary condition
        while (true) {
            System.out.print(dayOfMonth + "/" + month + "/" + year);
            if (dayOfWeek == 1) {
                System.out.println(" Sunday");
                if (dayOfMonth == 1){
                    nSundaysFirstDM++;
                }
            }
            else{
                System.out.println();
            }
            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 == 36524) {
                break;
            }
        }
        System.out.println("During the 20th century, " + nSundaysFirstDM +
" Sundays fell on the first day of the month");
    }

    // 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() {
        if (dayOfMonth == nDaysInMonth) {
            if (month == 12) {
                month = 1;
                year++;
            }
            else {
                month++;
            }
            nDaysInMonth = nDaysInMonth(month, year);
            dayOfMonth = 1;
        }
        else {
            dayOfMonth++;
        }

        if (dayOfWeek == 7) {
            dayOfWeek = 1;
        }
        else {
            dayOfWeek++;
        }
    }

    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        if (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)){
            return true;
        }
        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) {
        if (month == 4 || month == 6 || month == 9 || month == 11) {
            return 30;
        }
        else if (month == 2) {
            if (isLeapYear(year)) {
                return 29;
            }
            else {
                return 28;
            }
        }
        else {
            return 31;
        }
    }
}
```

Calendar

```java
/**
 * Prints the calendars of a given year.
 */
public class Calendar {
    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 0;
    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.
     */

    // main(string[]) - the entry point of a Java program.
    // args - args contains the supplied command-line
    // arguments as an array of String objects.
    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;
        year = Integer.parseInt(args[0]);
        //// Write the necessary initialization code, and replace the
condition
        //// of the while loop with the necessary condition
        while (debugDaysCounter < 365) {
            System.out.println(dayOfMonth + "/" + month + "/" + year);
            advance();
            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() {
        if (dayOfMonth == nDaysInMonth) {
            if (month == 12) {
                month = 1;
                year++;
```

```
            }
            else {
                month++;
            }
            nDaysInMonth = nDaysInMonth(month, year);
            dayOfMonth = 1;
        }
        else {
            dayOfMonth++;
        }
     }

    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        if (((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)){
            return true;
        }
        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) {
        if (month == 4 || month == 6 || month == 9 || month == 11) {
            return 30;
        }
        else if (month == 2) {
            if (isLeapYear(year)) {
                return 29;
            }
            else {
                return 28;
            }
        }
        else {
            return 31;
        }
    }
}
```