HW3 Code – Alon Morad

1. LoanCalc.java

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

2. 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) {
        String emptyStr = "";
        // runs on the number of chars in the string
        for (int i = 0; i < s.length(); i++) {
            // if asci code of char is between 65 to 90 it's a capital letter
            if (s.charAt(i) >= 65 & s.charAt(i) <= 90) {
                // +32 in asci flips capital letter to small letter
                emptyStr = emptyStr + (char) (s.charAt(i) + 32);
            }
            // if it's not capital letter keep the char the same as is
            else
            emptyStr = emptyStr + s.charAt(i);
        }
        return emptyStr;
    }
}
```

3. UniqueChars.java

```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) {
        String emptyStr = "";
        // varibale that keeps the char at the current i
        // starting from 0 placement in the string
        char charAti = s.charAt(0);
        // runs on the number of chars in the string
        for (int i = 0; i < s.length(); i++) {
            // sets the value of the current char
            charAti = s.charAt(i);
            // adds all space chars to the new string
            if (charAti == 32) {
                emptyStr = emptyStr + charAti;
            }
            // removes duplicates from the string unless it's space char
            else if (s.indexOf(charAti) == i) {
                emptyStr = emptyStr + charAti;
            }
        }
        return emptyStr;
    }
}
```

4. 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) {
        // runs on all months of the year and prints the relevant number of
days in the month
        // uses the nDaysInMonth on every month
        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) {
        // leap year is divisible by 400 || divisible by 4 and not by 100
        if (year % 400 == 0) {
            return true;
        } else if (((year % 4) == 0) && ((year % 100) != 0) ) {
            return true;
        }
        return false;
    }
```

```java
    // 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) {
            // feb has 29 days in leap year, 28 in common year
            case 2:
            if (isLeapYear(year)) {
                return 29;
            }
            else
            return 28;
            case 4: return 30;
            case 6: return 30;
            case 9: return 30;
            case 11: return 30;
            // all other months (default) have 31 days
            default: return 31;
        }
    }
}
```

5. Calendar1.java

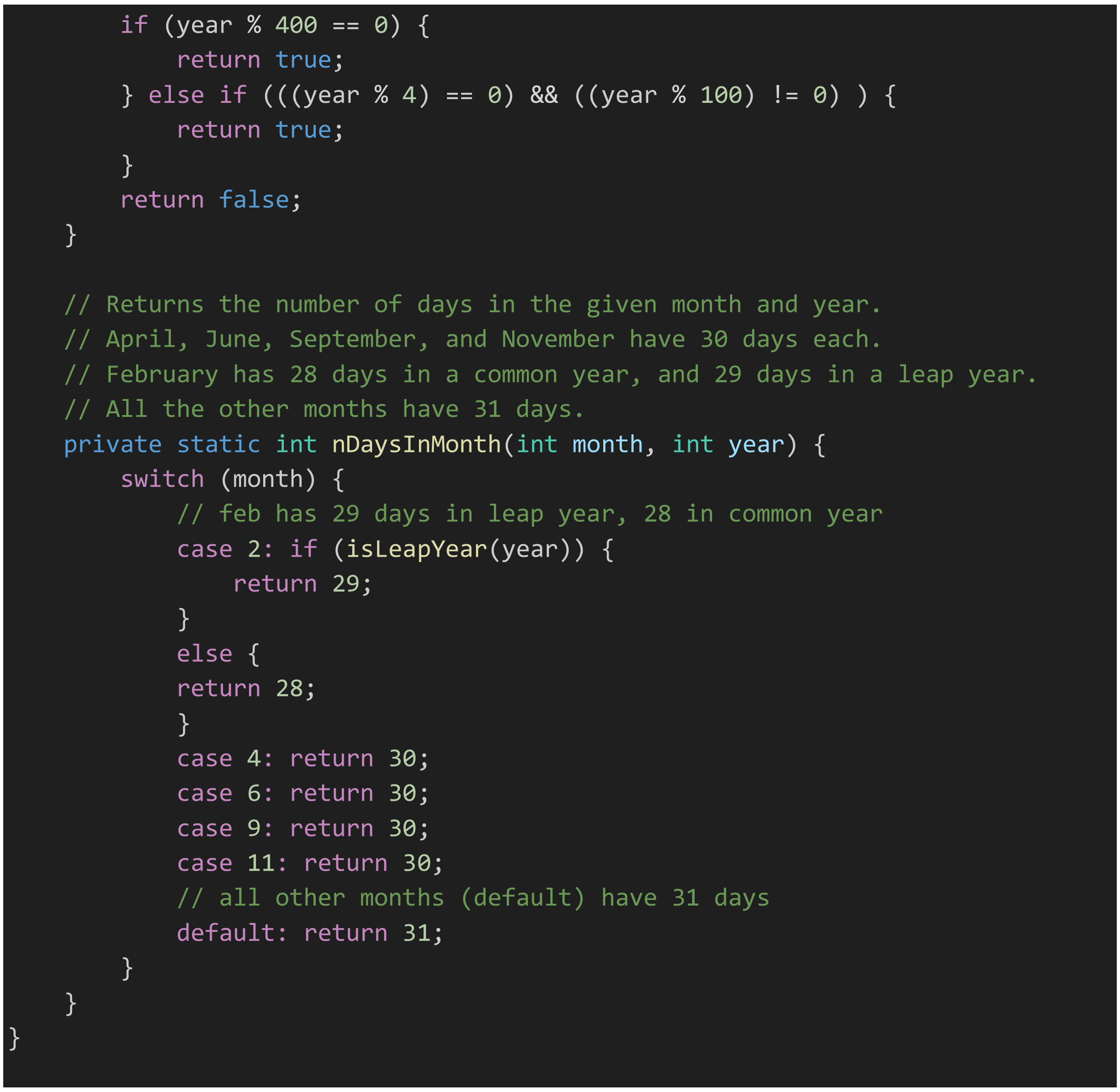
```java
/**
 * Prints the calendars of all the years in the 20th century.
 */
public class Calendar1 {
    // Starting the calendar on 1/1/1900
    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 1900;
    static int dayOfWeek = 2;     // 1.1.1900 was a Monday
    static int nDaysInMonth = 31; // Number of days in January
    static int sundaysfellCount = 0; // counter of sundays that are both the first day of the week and first day of the month
    /**
     * 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;
        // advances the date as long as the year is smaller than 2000
        while (year < 2000) {
            // if it's first day of the week print sunday after the date
            if (dayOfWeek == 1) {
                System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");
            }
            // if it's not first day of the week print just the date
            else
            System.out.println(dayOfMonth + "/" + month + "/" + year);

            // if the day is both first in the week and first in the month add 1 to counter
            if (dayOfWeek == 1 & dayOfMonth == 1) {
                sundaysfellCount++;
            }
            // runs the fucntion that advances the date
            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 (false) {
                break;
            }
        }
        System.out.println("During the 20th century, " + sundaysfellCount + "
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 it's the last day of the month : check the cases inside
        if (dayOfMonth == nDaysInMonth(month, year)) {
            // if it's the last month of the year: advance year, bring month
to january and set day of the month back to 1
            if (month == 12) {
                year++;
                month = 1;
                dayOfMonth = 1;
            }
            // if it's not the last month of the year: advance month and set
day of the month back to 1
            else {
                month++;
                dayOfMonth = 1;
            }
        }
        // if it's not last day of the month: advance day of month
        else
        dayOfMonth++;

        // if it's the last day of the week: set day of the week back to 1
        if (dayOfWeek == 7) {
            dayOfWeek = 1;
        }
        // if it's not the last day of the week: advance day of the week
        else
        dayOfWeek++;
     }

    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        // leap year is divisible by 400 || divisible by 4 and not by 100
```



```java
        if (year % 400 == 0) {
            return true;
        } else if (((year % 4) == 0) && ((year % 100) != 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) {
        switch (month) {
            // feb has 29 days in leap year, 28 in common year
            case 2: if (isLeapYear(year)) {
                return 29;
            }
            else {
            return 28;
            }
            case 4: return 30;
            case 6: return 30;
            case 9: return 30;
            case 11: return 30;
            // all other months (default) have 31 days
            default: return 31;
        }
    }
}
```

6. Calendar.java (Final)

```java
/**
 * Prints the calendar of the user given year.
 */
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 calendar of the user given year.
     */
    public static void main(String args[]) {
        // Advances the date and the day-of-the-week from 1/1/1900 till
31/12/1999, inclusive.
        // The following variable, used for debugging purposes, counts how
many days were advanced so far.
        int debugDaysCounter = 0;
        int yearUsergiven = Integer.parseInt(args[0]);
        // advances the date as long as the year is smaller than the user
given year
        // no need to print
        while (year < yearUsergiven) {
            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 (false) {
                break;
            }
        }
        // sets year as the user given year, in order of the fucntion
'advance' to work properly without changing the varibales name in it
        year = yearUsergiven;
        // advances the date until the year finishes
        while (year < yearUsergiven + 1) {
            // if it's first day of the week print sunday after the date
            if (dayOfWeek == 1) {
                System.out.println(dayOfMonth + "/" + month + "/" + year + "
Sunday");
            }
            // if it's not first day of the week print just the date
            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 (false) {
                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() {
        // if it's the last day of the month : check the cases inside
        if (dayOfMonth == nDaysInMonth(month, year)) {
            // if it's the last month of the year: advance year, bring month
to january and set day of the month back to 1
            if (month == 12) {
                year++;
                month = 1;
                dayOfMonth = 1;
            }
            // if it's not the last month of the year: advance month and set
day of the month back to 1
            else {
                month++;
                dayOfMonth = 1;
            }
        }
        // if it's not last day of the month: advance day of month
        else
        dayOfMonth++;

        // if it's the last day of the week: set day of the week back to 1
        if (dayOfWeek == 7) {
            dayOfWeek = 1;
        }
        // if it's not the last day of the week: advance day of the week
        else
        dayOfWeek++;
     }
```

```java
    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        // leap year is divisible by 400 || divisible by 4 and not by 100
        if (year % 400 == 0) {
            return true;
        } else if (((year % 4) == 0) && ((year % 100) != 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) {
        switch (month) {
            // feb has 29 days in leap year, 28 in common year
            case 2:
            if (isLeapYear(year)) {
                return 29;
            }
            else
            return 28;
            case 4: return 30;
            case 6: return 30;
            case 9: return 30;
            case 11: return 30;
            // all other months (default) have 31 days
            default: return 31;
        }
    }
}
```