HW3 Code – Tomer Shulner

1. 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);

        iterationCounter = 0;

        // 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.
    */
```

```java
    // Side effect: modifies the class variable iterationCounter.
    public static double bruteForceSolver(double loan, double rate, int n, double epsilon)
{
        double g = loan / n;
        double increment = 0.001;
        double f = endBalance(loan, rate, n, g);
        while (f >= epsilon && f >= 0) {
            g += increment;
            iterationCounter++;
            f = endBalance(loan, rate, n, g);
        }
        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 H = loan;
        double L = loan / n;
        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) {
        for (int i = 0; i < n; i++) {
            loan = (loan - payment) * (1 + 0.01 * rate);
        }
        return loan;
    }
}
```

2. LowerCase

```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 lower_s = "";
        for(int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (c >= 65 && c <= 90) {
                c = (char)(c + 32);
            }
            lower_s += c;
        }
        // Replace the following statement with your code
        return lower_s;
    }
}
```

3. 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) {
        String unique = "";
        for (int i = 0; i < s.length(); i++) {
            char current = s.charAt(i);
            if ((unique.indexOf(current) == -1) || current == 32) {
                unique += current;
            }
        }
        return unique;
    }
}
```

4. Calendar

```
/**
 * Prints the calendar of a specific 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

    public static void main(String args[]) {
        int given_year = Integer.parseInt(args[0]);
        String current_date = dayOfMonth + "/" + month + "/" + year;

        while (year != given_year + 1) {
            if (year == given_year) {
                System.out.print("\n" + current_date);
                if (dayOfWeek == 1) {
                    System.out.print(" Sunday");
            }
            }
            advance();
            current_date = dayOfMonth + "/" + month + "/" + year;
        }
     }

     // 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(month, year)) {
            dayOfMonth = 1;
            month ++;
        }
        else {
            dayOfMonth++;
        }
        if (dayOfWeek == 7) {
            dayOfWeek = 1;
        }
        else {
            dayOfWeek++;
        }

        if (month == 13) {
            month = 1;
```

```
            year++;
        }
     }

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

    // 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 4:
                return 30;
            case 6:
                return 30;
            case 9:
                return 30;
            case 11:
                return 30;
            case 2:
                if (isLeapYear(year)) {
                    return 29;
                }
                else {
                    return 28;
                }
            default:
                return 31;
        }
    }
}
```