HW03 Code – CS

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);

        // Resets the iteration counter
        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.
     */
    // Side effect: modifies the class variable iterationCounter.
    public static double bruteForceSolver(double loan, double rate,
int n, double epsilon) {
        double guess = loan / n; // Initial guess

        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 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 bisectionSolver(double loan, double rate,
int n, double epsilon) {
        double low = 0;
        double high = loan;

        while (high - low > epsilon) {
            double guess = (low + high) / 2;
            if (endBalance(loan, rate, n, guess) > 0) {
                low = guess;
            } else {
                high = guess;
            }
            iterationCounter++;
        }

        return (low + high) / 2;
    }

    /**
     * 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 balance = loan;

        for (int i = 0; i < n; i++) {
            balance = (balance - payment) * (1 + rate / 100);
        }

        return balance;
    }
}
```

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 str) {
         String result = "";

        for (int i = 0; i < str.length(); i++) {
            char currentChar = str.charAt(i);

            // Check if the character is an uppercase letter
            if (currentChar >= 'A' && currentChar <= 'Z') {
                // Convert to lowercase by adding the ASCII
difference
                result += (char) (currentChar + ('a' - 'A'));
            } else {
                // If not an uppercase letter, leave the character
unchanged
                result += currentChar;
            }
        }

        return result;
    }
}
```

UniqueChars.java

```
public class uniquechars {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(uniqueChars(str));
    }

    public static String uniqueChars(String s) {
        String result = "";

        for (int i = 0; i < s.length(); i++) {
            char currentChar = s.charAt(i);

            // Check if the character is not a space or if it has not
been added to the result yet
            if (currentChar != ' ' && result.indexOf(currentChar) ==
-1) {
                result += currentChar;
            } else if (currentChar == ' ') {
                // Space characters are always added to the result
                result += currentChar;
            }
        }

        return result;
    }
}
```

Calendar.java

```
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 occurred on the first day of the month
during this
     * period.
     */
    public static void main(String args[]) {
        int debugDaysCounter = 0;
        int sundayscount = 0;
        int inputYear = Integer.parseInt(args[0]); // Input year from
command line

        while (year <= inputYear) {
            if (year == inputYear) {
                if (dayOfWeek == 1) {
                    System.out.println(dayOfMonth + "/" + month + "/"
+ year + " Sunday");
                    if (dayOfMonth == 1) {
                        sundayscount++;
                    }
                } else {
                    System.out.println(dayOfMonth + "/" + month + "/"
+ year);
                }
            }
            advance();
            debugDaysCounter++;

            if (debugDaysCounter == 0) {
                break;
            }
        }
        System.out.println("During the 20th century, " + sundayscount
+ " 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 (dayOfWeek < 7) {
            dayOfWeek++;
        } else if (dayOfWeek == 7) {
            dayOfWeek = 1;
        }
        if (dayOfMonth < nDaysInMonth) {
            dayOfMonth++;
        } else if (dayOfMonth == nDaysInMonth) {
            dayOfMonth = 1;
```

```java
            if (month < 12) {
                month++;
                nDaysInMonth = nDaysInMonth(month, year);
            } else if (month == 12) {
                month = 1;
                nDaysInMonth = nDaysInMonth(month, year);
                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 4:
            case 6:
            case 9:
            case 11:
                return 30;
            case 2:
                return (isLeapYear(year)) ? 29 : 28;
            default:
                return 31;
        }
    }
}
```