HW3 code

LoanCalc.java:

```
        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, ((100 + rate) / 100), 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, ((100 + rate) / 100), 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 yearlyPayment = loan / n;
                double increment = epsilon;
                double moneyLeft = endBalance(loan, rate, n, yearlyPayment) ;
                while ( moneyLeft >= epsilon ){
                        iterationCounter++;
                        yearlyPayment += increment;
                        moneyLeft = endBalance(loan, rate, n, yearlyPayment);
                }
        return yearlyPayment;
    }

    /**
        * 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 overPay = loan;
                double underPay = loan / n;
                double currentPay = (overPay + underPay) / 2;
                while (overPay - underPay > epsilon){
                        if (endBalance(loan, rate, n, currentPay) * endBalance(loan, rate, n,
underPay) > 0){
                                underPay = currentPay;
                        }else{
                                overPay = currentPay;
                        }
                        currentPay = (overPay + underPay) / 2;
                        iterationCounter++;
                }
        return currentPay;
   }

        /**
        * 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 moneyLeft = loan;
                for(int i = 0; i < n; i++){
                        moneyLeft = ((moneyLeft - payment)*rate);
                }
        return moneyLeft;
        }
}
```

LowerCase.java:

```
           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) {
        for(int i = 0; i < s.length(); i++){
            Character letter =  s.charAt(i);
            if( (((int) letter) > 64) && (((int) letter) < 91) ){
                s = s.replace(s.charAt(i), (char)(((int) letter) + 32));
            }
        }
        return s;
    }
}
```

UniqueChars.java:

```
           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) {
        int[] usedLetters = new int[1000];
        String newWord = "";
        for(int i = 0; i < s.length(); i++){
            Character letter =  s.charAt(i);
            if((usedLetters[(int) letter] != 1) || ((int) letter == 32)){
                newWord = newWord + s.charAt(i);
                usedLetters[(int) letter] = 1;
            }
        }
        return newWord;
    }
}
```

Calendar0.java:

```
    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++){
            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){
            if((year % 100 == 0) && (year % 400 != 0)){
                return false;
            }else{
                return true;
            }
        }else{
            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 == 2){
            if(isLeapYear(year)){
                return 29;
            }else{
                return 28;
            }
        }
        if((month == 4) || (month == 6) || (month == 9) || (month == 11)){
            return 30;
        }else{
            return 31;
        }
    }

}
```

Calendar1.java:

```
        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 firstSundays = 0;
           //// Write the necessary initialization code, and replace the condition
           //// of the while loop with the necessary condition
                while (true) {
                        //// Write the body of the while
                        if(dayOfWeek == 1){
                                if(dayOfMonth == 1){
                                        firstSundays += 1;
                                }
                                System.out.println(dayOfMonth + "/" + month + "/" + year + "
Sunday");
                        }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(year == 2000){
                break;
            }
   }
        System.out.println("During the 20th century, " + firstSundays + " Sundays
fell on the first day of the month");

        //// Write the necessary ending code here
    }

    // 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(month == 12){
            if(dayOfMonth == nDaysInMonth){
                month = 1;
                nDaysInMonth = nDaysInMonth(month, year);
                dayOfMonth = 1;
                year += 1;
                if(dayOfWeek == 7){
                  dayOfWeek = 1;
                }else{
                dayOfWeek += 1;
                }
            }else if(dayOfWeek == 7){
                dayOfMonth += 1;
                dayOfWeek = 1;
            }else{
              dayOfMonth += 1;
                dayOfWeek += 1;
            }


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


     }

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