LoanCalc.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)
{
            double g = (loan/n);
            iterationCounter = 0;
            double f = endBalance(loan, rate, n, g);
            while(f > epsilon){
                if (f > 0){
                    g = g + epsilon;
                }

                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 L = loan/n;
            double H = loan;
            double g = ((L + H)/2);
            iterationCounter = 0;


            while ((H-L)> epsilon){
                double f = endBalance(loan, rate, n, g);
                double fL = endBalance(loan, rate, n, L);

                if (f * fL > 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 balance = loan;
        rate /= 100;
        for (int i = n; i > 0; i--) {
            balance = ((balance - payment)*(1 + rate));
        }

    return balance;
    }
}
```

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 str) {
        String ans = "";
        int i = 0;

        while (i < str.length()) {
            char ch = str.charAt(i);
            if (ch >= 65 && ch <= 90) {
                ans = ans + (char) (ch + 32);
            }
            else {
                ans = ans + ch;
            }
            i++;
        }
        return ans;
    }
}
```

UniqueChars.java

```
/** String processing exercise 2. */
public class UniqueChars {
    public static void main(String[] args) {
        String A = args[0];
        System.out.println(uniqueChars(A));
    }

    /**
     * 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 A) {
        String ans = "";


        for (int i = 0; i < A.length(); i++){
            if ((A.charAt(i)) != 32){
                if (ans.indexOf(A.charAt(i)) == -1) {
                    ans = ans + A.charAt(i);
                }
            }
            else{
                ans = ans + A.charAt(i);
            }
    }
    return ans;

}
}
```

Calendar.java

```
/**
 * Prints the calendars of all the years in the 20th century.
 */
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
        static int sundayC = 0;

        /**
         * 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[]) {
                String stringYearInput = args[0];
     int yearInput = Integer.parseInt(stringYearInput);
                // 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;
           //// Write the necessary initialization code, and replace the condition
           //// of the while loop with the necessary condition
                while (dayOfMonth != 31 || month != 12 || year != (yearInput)) {
                        advance();
                        debugDaysCounter++;

        if (year == yearInput) {
           if (dayOfWeek == 1) {
              sundayC++;
              System.out.println(dayOfMonth + "/" + month + "/" + year + " Sunday");
           }
           else {
           System.out.println(dayOfMonth + "/" + month + "/" + year);
        }
                        //// If you want to stop the loop after n days, replace the condition of the
                        //// if statement with the condition (debugDaysCounter == n)
              }
          }
  }

          //// 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 (dayOfWeek == 7) {
              dayOfWeek = 1;
          }
          else {
              dayOfWeek++;
          }
          if (month == 12 && dayOfMonth == 31) {
              month = 1;
              dayOfMonth = 1;
              year++;
          }
          else if (dayOfMonth == nDaysInMonth) {
              month++;
              nDaysInMonth = nDaysInMonth(month, year);
              dayOfMonth = 1;
          }
          else {
              dayOfMonth++;
          }


      }

  // Returns true if the given year is a leap year, false otherwise.
      public static boolean isLeapYear(int year) {
          boolean isLeapYear;

          isLeapYear = ((year % 400) == 0);
          isLeapYear = isLeapYear || (((year % 4) == 0) && ((year % 100) != 0));

          return isLeapYear;
      }
```

```
    // 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) {
        boolean isLeapYear = isLeapYear(year);
        int feb;
        if (isLeapYear) {
            feb = 29;
        }
        else {
            feb = 28;
        }
        switch (month) {
            case 1:
                return 31;
            case 2:
                return feb;
            case 3:
                return 31;
            case 4:
                return 30;
            case 5:
                return 31;
            case 6:
                return 30;
            case 7:
                return 31;
            case 8:
                return 31;
            case 9:
                return 30;
            case 10:
                return 31;
            case 11:
                return 30;
            case 12:
                return 31;

        }

        return 0;
    }
}
```