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, 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) >= epsilon){
                        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 = 0 ;
                double H = loan ;
                double g = (L + H)/2 ;
                while ((H-L) >= epsilon){
                        if ((endBalance(loan,rate,n,g))*(endBalance(loan,rate,n,H))  < 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 = 0 ;
                double currentBalance = loan ;

                for(int i =1 ; i <= n ; i++){

                        balance = (currentBalance - payment)*(1+(rate/100)) ;
                        currentBalance = balance ;
                }

        return balance ;
        }
}
```

Calendar.java

```
public class Calendar {


    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 1900;
    static int dayOfWeek = 2;
    static int nDaysInMonth = 31;


    public static void main(String args[]) {

    int IsYear = Integer.parseInt(args[0]);

       String s = "Sunday";


           year = IsYear;


           while (month <= 12 && year==IsYear){



                 if ( dayOfWeek==7){
                     System.out.println(dayOfMonth+"/"+month+"/"+year + " " + s);
                } else {
                     System.out.println(dayOfMonth+"/"+month+"/"+year);
      }
       advance();
           }
     }






      // 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() {

           dayOfMonth++;
           dayOfWeek++;
           if (dayOfMonth>nDaysInMonth(month, year)){
                month++;
```

```
            dayOfMonth = 1;
        }
        if (month > 12) {
            year++;
            month = 1;
        }
        if (dayOfWeek>7){
            dayOfWeek= 1;
        }

        }


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

        if (year%4==0){
            return true;
        } else {

            return false;
        }


    }

    // Returns the number of days in the given month and year.
    public static int nDaysInMonth(int month, int IsYear) {
        if (isLeapYear(year) && month ==2){

    return 29;

        } if (!isLeapYear(year) && month == 2){

            return 28;

        } if (month == 1 || month==3|| month==5 || month == 7|| month == 8||
month==10||month==12){

            return 31;

        } else {

            return 30;
        }

    }
}
```

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 newString= "";

       for (int i = 0; i<s.length(); i++){
        char ch = s.charAt(i);
         if (newString.indexOf(ch) == -1){
          newString+=ch;
           }  else if(ch == ' ') {
            newString +=ch;
           }
       }

        return newString;
    }
}
```

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) {
       String newString = "";
       for (int i = 0; i< s.length(); i++){
        char c ='a';
        int charIndex = s.charAt(i);
        if (64<charIndex&&charIndex<91)
        c = (char)(charIndex + 32);
        else
        c = (char)(charIndex);
        newString+=c;

       }

        return newString;
    }
}
```