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(epsilon < f){
            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 bal = loan;
        rate /= 100;
        for (int i = n; i > 0; i--) {
            bal = ((bal - payment)*(1 + rate));
        }

        return bal;
    }
}
```

LowerCase.java

```
/** String processing exercise 1. */
public class LowerCase {
    public static void main(String[] args) {
        String str = args[0];
        System.out.println(lowerCase(str));
    }
    public static String lowerCase(String s)
    {
        String newString = "";
        char c ;

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

            if (Character.isLetter(s.charAt(i))) //Is c a letter?
            {

                if ((s.charAt(i) >= 'A') && (s.charAt(i)<= 'Z')) //Is 
c upper case?
                {
                    c = Character.toLowerCase(s.charAt(i)); //change c 
to lower case
                }
                   else
                   {c = s.charAt(i);}
                }
                      else
                      {c = s.charAt(i);
             }

            newString = newString + c;
        }
        return newString;


    }
}
```

UniqueChars.java

```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 stringNew = "";
        char c;
        for (int i = 0; i < s.length(); i++)
        {
            c = s.charAt(i);
            if (c == ' ' || stringNew.indexOf(c) == -1)
            {
                stringNew = stringNew + c;
            }
        }
        return stringNew;

    }

}
```

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);
        //// 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();
            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 LeapYear;

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

        return LeapYear;
    }

    // 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 monthFeb;
        if (isLeapYear) {
            monthFeb = 29;
        }
        else {
            monthFeb = 28;
        }
        switch (month) {
            case 1:
                return 31;
            case 2:
                return monthFeb;
            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;
    }
}
```