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);

        // 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;
        iterationCounter = 0;
        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 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) {
        // Replace the following statement with your code
        double L = 0;
        double H = loan;
        double guess = (L + H) / 2;
        iterationCounter = 0;
        while ((H - L) > epsilon) {
            if (endBalance(loan, rate, n, guess) * endBalance(loan, rate, n, L) >
0) {
                L = guess;
            } else {
                H = guess;
            }
            guess = (L + H) / 2;
            iterationCounter++;
        }

        return guess;
    }

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

LowerCase.java

```java
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 str2 = "";
        for (int i = 0; i < s.length(); i++){
            if ((char)(s.charAt(i)) < 91){
                if ((char)(s.charAt(i)) > 64){
                    str2 = str2 + ((char)(s.charAt(i) + 32));
                } else {
                    str2 = str2 + s.charAt(i);
                }
            } else {
                str2 = str2 + s.charAt(i);
            }
        }
        return str2;
    }
}
```

UniqueChars.java :

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

Calendar0.java

```java
/*
 * Checks if a given year is a leap year or a common year,
 * and computes the number of days in a given month and a given year.
 */
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) {
        int month = 12;
        for (int i = 1; i <= month; 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) {
        boolean LeapYear;
        if (year % 4 == 0){
            LeapYear = true;
        } else {
            LeapYear = false;
        }
        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) {
        if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 ||
month == 10 || month == 12) {
            return 31;
        } else if (month == 2) {
            if (isLeapYear(year)) {
                return 29;
            } else {
                return 28;
            }
        } else {
            return 30;
        }
    }
}
```

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

    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;
        //// 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
            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 (false) {
                //break;
            //}
        //}
        //// Write the necessary ending code here
    }

    private static void advance() {
        int yearmax = 1999;
        int monthmax = 12;
        int TotalDay = 1;
        int FirstSunday = 0;
        while (year <= yearmax){
            month = 1;
            while (month <= monthmax){
                dayOfMonth = 1;
                int dm = nDaysInMonth(month, year);
                while (dayOfMonth <= dm){
                    if (TotalDay % 7 == 0){
                        System.out.println(dayOfMonth + "/" + month + "/" + year +
" Sunday");
                        if (dayOfMonth == 1) {
                            FirstSunday++;
```

```
                        }
                    } else {
                    System.out.println(dayOfMonth + "/" + month + "/" + year);
                    }
                    dayOfMonth++;
                    TotalDay++;
                }
                month++;
            }
            year++;
        }
        System.out.println();
        System.out.println("During the 20th century, " + FirstSunday + " Sundays
fell on the first day of the month");
     }

    private static boolean isLeapYear(int year) {
        boolean LeapYear;
        if (year % 4 == 0 && year != 1900){
            LeapYear = true;
        } else {
            LeapYear = false;
        }
        return LeapYear;
    }

    private static int nDaysInMonth(int month, int year) {
        if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 ||
month == 10 || month == 12) {
            return 31;
        } else if (month == 2) {
            if (isLeapYear(year)) {
                return 29;
            } else {
                return 28;
            }
        } else {
            return 30;
        }
    }
}
```

Calendar.java

```
public class Calendar {
    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 1900;
    static int nDaysInMonth = 31;

    public static void main(String args[]) {
        int xyear = Integer.parseInt(args[0]);
        int monthmax = 12;
        int TotalDay = 1;
        while (year < xyear){
            month = 1;
            while (month <= monthmax){
                dayOfMonth = 1;
                int dm = nDaysInMonth(month, year);
                while (dayOfMonth <= dm){
                    dayOfMonth++;
                    TotalDay++;
                }
                month++;
            }
            year++;
        }
            month = 1;
            while (month <= monthmax){
                dayOfMonth = 1;
                int dm = nDaysInMonth(month, xyear);
                while (dayOfMonth <= dm){
                    if (TotalDay % 7 == 0){
                        System.out.println(dayOfMonth + "/" + month + "/" + xyear +
" Sunday");
                    } else {
                    System.out.println(dayOfMonth + "/" + month + "/" + xyear);
                    }
                    dayOfMonth++;
                    TotalDay++;
                }
                month++;
            }

     }

    private static boolean isLeapYear(int year) {
        boolean LeapYear;
        if (year % 4 == 0 && year != 1900){
            LeapYear = true;
```

```
        } else {
            LeapYear = false;
        }
        return LeapYear;
    }

    private static int nDaysInMonth(int month, int year) {
        if (month == 1 || month == 3 || month == 5 || month == 7 || month == 8 ||
month == 10 || month == 12) {
            return 31;
        } else if (month == 2) {
            if (isLeapYear(year)) {
                return 29;
            } else {
                return 28;
            }
        } else {
            return 30;
        }
    }
}
```