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) {
        // Replace the following statement with your code
        iterationCounter = 0;
        double guess = loan / n;
        boolean found = false;

        while (!found) {
            double end = endBalance(loan, rate, n, guess);
            if (end > 0) {
                guess += epsilon;
                iterationCounter++;
            }
            else
                found = true;
        }

        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
        // Sets L and H to initial values such that f(L) > 0, f(H) < 0,
        // implying that the function evaluates to zero somewhere between L and H.
        // So, let’s assume that L and H were set to such initial values.
        // Set g to (L + H)/2
        iterationCounter = 0;
        double low = loan / n;
        double high = loan;
        double g = (high + low) / 2;
        while ((high - low) > epsilon) {
            // Sets L and H for the next iteration
            double end = endBalance(loan, rate, n, g);
            iterationCounter++;
            if (end > 0) {
                // the solution must be between g and H
                // so set L or H accordingly
                low = g;
                g = (high + low) / 2;
            }
            else {
                // the solution must be between L and g
                // so set L or H accordingly
                // Computes the mid-value (g) for the next iteration
                high = g;
                g = (high + low) / 2;
            }
        }
        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) {
        // Replace the following statement with your code
        double subLoan = loan;
        double newRate = 1 + (rate / 100);
        for (int i = 0; i < n; i++) {
            subLoan = (subLoan - payment) * newRate;
        }
        return subLoan;
    }
}
```

LowerCase.java

```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 s) {
        // Replace the following statement with your code
        String lowerString = "";
        for(int i = 0; i < s.length(); i++) {
            if (64 < s.charAt(i) && s.charAt(i) < 91) {
                lowerString += (char)(s.charAt(i) + 32);
            }
            else {
                lowerString += s.charAt(i);
            }
        }
        return lowerString;
    }
}

```

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) {
        // Replace the following statement with your code
        String uniqueS = "";
        for (int i = 0; i < s.length(); i++) {
            char uniqueC = s.charAt(i);
            boolean found = false;
            if (uniqueC != ' ') {
                for (int j = 0; j < uniqueS.length(); j++) {
                    if (uniqueC == uniqueS.charAt(j))
                        found = true;
                }
            }
            if (!found)
                uniqueS += uniqueC;
        }
        return uniqueS;
    }
}

```

Calendar.java

```java
public class Calendar {
    static int dayOfMonth = 1;
    static int month = 1;
    static int year = 1900;
    static int dayOfWeek = 2;
    static int nDaysInMonth = 31; // Number of days in January

    /**
     * Prints the calendar of a given year.
     */
    public static void main(String args[]) {
        int newYear = Integer.parseInt(args[0]);
        while (year < newYear) {
            advance();
        }
        while (year < (newYear + 1)) {
            String date = dayOfMonth + "/" + month + "/" + year;
            if (dayOfWeek == 1) {
                date += " Sunday";
            }
            System.out.println(date);
            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() {
        if (dayOfWeek == 7)
                dayOfWeek = 1;
            else
                dayOfWeek++;
        if (dayOfMonth < nDaysInMonth) {
            dayOfMonth++;
        }
        else {
            dayOfMonth = 1;
            if (month == 12) {
                month = 1;
                year++;
            }
            else {
                month++;
                nDaysInMonth = nDaysInMonth(month, year);
            }
        }
     }

    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        boolean isLeap = ((year % 400) == 0);
        isLeap = isLeap || (((year % 4) == 0) && ((year % 100) != 0));
        return isLeap;
    }

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