# 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) {
        // Replace the following statement with your code
       iterationCounter = 0;
       double g = loan/n;
       while (endBalance(loan,rate,n,g) >0) {
          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) {
        // Replace the following statement with your code
       double hi = loan;
       double lo = loan/n;
       double g = (lo+hi)/2;
       iterationCounter =0;
       while ((hi-lo)>= epsilon){

        if((endBalance(loan,rate,n,g))*(endBalance(loan,rate,n,lo))>0)
        {
             lo = g;
        }
          else {
             hi = g;
          }
          g = (lo+hi)/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) {
       // Replace the following statement with your code
       double currentLoanAmount = loan;
       double interest = (rate+100)/100;
       for(int i = 0; i<n; i++) {
          currentLoanAmount = (currentLoanAmount-payment)*interest;
       }
        return currentLoanAmount;
    }
}
```

## LowerCase.java

```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) {
        // Replace the following statement with your code
        String newString = "";
        for (int i =0; i<s.length(); i++)
        {
            if (s.charAt(i) >= 'A' && s.charAt(i)<= 'Z')
            {
                newString+=(char)(s.charAt(i)+32);
            }
            else{
                newString += s.charAt(i);
            }

        }
        return newString;
    }
}
```

## UniqueChars.java

```java
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 newString = "";
        boolean isExist; // check if the character was appered id the
        string
        for (int i =0; i<s.length(); i++) {
            isExist = false;
            if(s.charAt(i) != ' ') {
                for(int j = 0; j<newString.length(); j++)
                {
                    if((s.charAt(i) == newString.charAt(j)) ) {
                        isExist = true;
                        break;
                    }
                }
            }
            if (isExist == false) {
                newString += s.charAt(i);
            }

        }
        return newString;
    }
}
```

## Calendar.java

```
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

    /**
     * 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[]) {
        // 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 givenyear = Integer.parseInt(args[0]);
        int debugDaysCounter = 0;
        //// Write the necessary initialization code, and replace the
        condition
        //// of the while loop with the necessary condition
        while (year != givenyear){
            advance();
        }
        while (year != givenyear+1) {
            //// Write the body of the while
            System.out.print(dayOfMonth + "/" + month + "/" + year);
            if(dayOfWeek == 1) {
                System.out.print(" Sunday" );
            }

            System.out.println();
            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
    }

    // 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() {
        // Replace this comment with your code
        if (dayOfMonth != nDaysInMonth) { // if the month didnt end
            dayOfMonth++;
        }
        else { // if the month end
            if(month == 12) { // if the year end
                year ++;
                month = 1;
            }
            else { //if the year didnt end
                month++;
            }
            dayOfMonth = 1;
            nDaysInMonth = nDaysInMonth(month, year); //update the
              days in the new month
        }
        dayOfWeek = (dayOfWeek%7) +1; // update the day
    }


    // Returns true if the given year is a leap year, false otherwise.
    private static boolean isLeapYear(int year) {
        // Replace the following statement with your code
        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.
```

```
    private static int nDaysInMonth(int month, int year) {
        // Replace the following statement with your code
        int daysInMonth;
        if(isLeapYear(year) && (month==2)) {
            daysInMonth = 29;
        }
        else if  ((month == 4) || (month == 6) || (month == 9) ||
        (month == 11)) {
            daysInMonth = 30;
        }
        else if (month == 2) {
            daysInMonth = 28;
        }
        else {
            daysInMonth = 31;
        }
        return daysInMonth;
    }
}
```