1. LoanCalc :

```
/**
* 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 = 0;    // Monitors the efficiency of the
calculation
          static int iterationCounter1 = 0;    // 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: " + iterationCounter1);
          }

          /**
          * 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 payment = loan/n ;
          while (endBalance(loan,rate,n,payment) > 0){
                              payment += epsilon ;
                              iterationCounter++ ;
                    }
          return payment;
    }

    /**
          * 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 payment = (L+H)/2 ;
                    while( ( H - L ) > epsilon ){
```

```
                                if
((endBalance(loan,rate,n,payment)*endBalance(loan,rate,n,L)) > 0 ){
                                        L = payment;}
                        else {
                                    H = payment;}
                                iterationCounter1 ++ ;
                                payment = (L+H)/2 ;
                        }
                return payment;
                }

                /**
                * 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 -= payment ;
                                loan *= ( 1 + rate/100 ) ;
                        }
                return loan;
                }
}
```

2. LowerCase :

```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) {
        int length = s.length();
                    String sLower = "";
                    for ( int i = 0 ; i < length ; i++){
                              if (s.charAt(i)>= 'A' && (s.charAt(i) <= 'Z')){
                                        sLower = sLower + ((char)(s.charAt(i)+32));
                              }
                              else {
                                        sLower = sLower + s.charAt(i);
                              }
                    }
        return sLower;

    }
}
```

3. UniqueChars

```
/** 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) {
                    int length = s.length();
                    char currentDigit ;
                    String sFinal = "" ;

        for (int i = 0 ; i < length ; i++){
                                currentDigit = s.charAt(i);
                                if (sFinal.indexOf(currentDigit) == -1 || currentDigit== '
' ){
                                        sFinal = sFinal + currentDigit ;
                                }
                                else {
                                        sFinal = sFinal ;
                                }
                    }
        return sFinal;
    }
}
```

4. Calendar :

```
/**
 * Prints the calendar of given year.
 */
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


          /**
           * caunt the calendars of all the years from 1990 till the given year
     * print it only in the given year
           */
          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;
                     int y = Integer.parseInt(args[0]); // getting year from user
                     while ( year < y ) {
                                          advance();
                                          debugDaysCounter++;
                     }

                     while ( year == y ) {
                                          System.out.print(dayOfMonth + "/" + month + "/" +
year );
                                          if (dayOfWeek == 1 ) {
                                                    System.out.print( " Sunday" );
                                                    }
                                          System.out.println();
                                          advance();
                                          debugDaysCounter++;
                     }
           }

           // 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 (dayOfMonth < nDaysInMonth(month,year)){
                               dayOfMonth++ ;
                     }
                     else {
                                dayOfMonth = 1 ;
                                if (month < 12 ) {
                                          month ++ ;
                                }
                                else {
                                          month = 1 ;
                                          year ++ ;
                                }
                     }

                     dayOfWeek++ ;
                               if ( dayOfWeek == 8 ){
                                          dayOfWeek = 1 ;
                               }
```

```
            }


    // Returns true if the given year is a leap year, false otherwise.
          private static boolean isLeapYear(int year) {
              boolean leapYear = true ;
              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.
          private static int nDaysInMonth(int month, int year) {
                    int days = 0 ;
           if ( month == 4 || month == 6 || month == 9 || month == 11 ){
                    days = 30 ;
           }
           else if (month == 2 && isLeapYear(year) == false){
                    days = 28 ;
           }
           else if (month == 2 && isLeapYear(year) == true){
                    days = 29 ;
           }
           else {
                     days = 31 ;
           }
                              return days;
          }
}
```