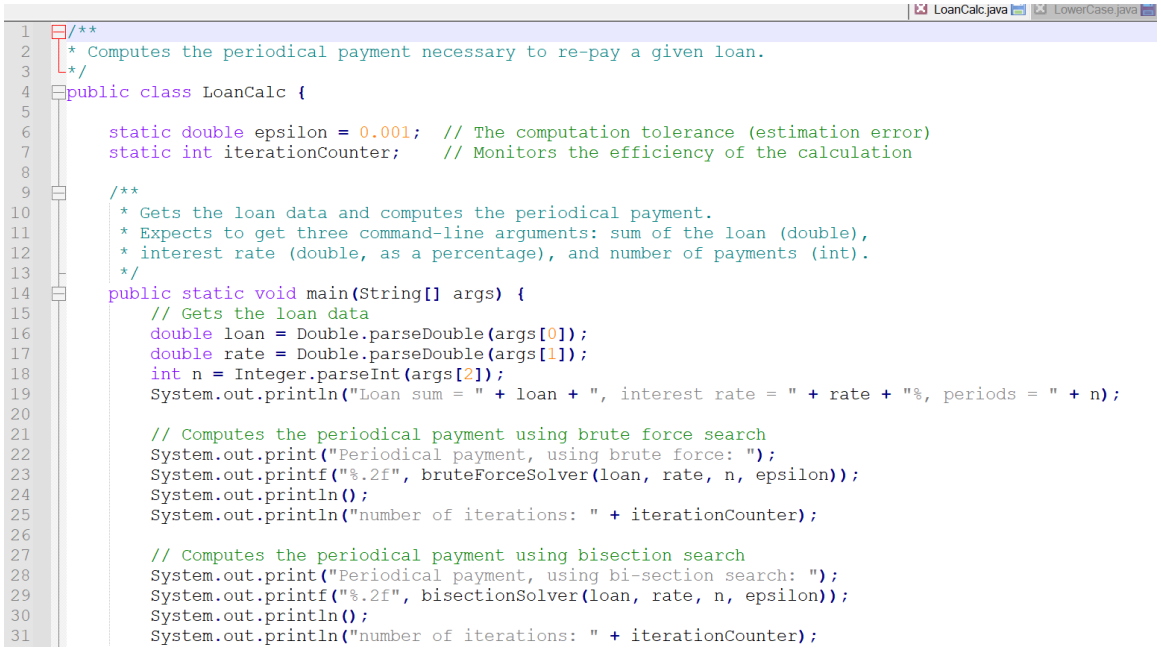
UniqueChars

```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 s1="";
        int length=s.length();

        for(int i=0; i<length; i++)
        {
            if((int)s.charAt(i)==32)
                s1=s1+" ";
            if(s1.indexOf(s.charAt(i))<0)
                s1=s1+s.charAt(i);

        }

        return s1;
    }
}

```

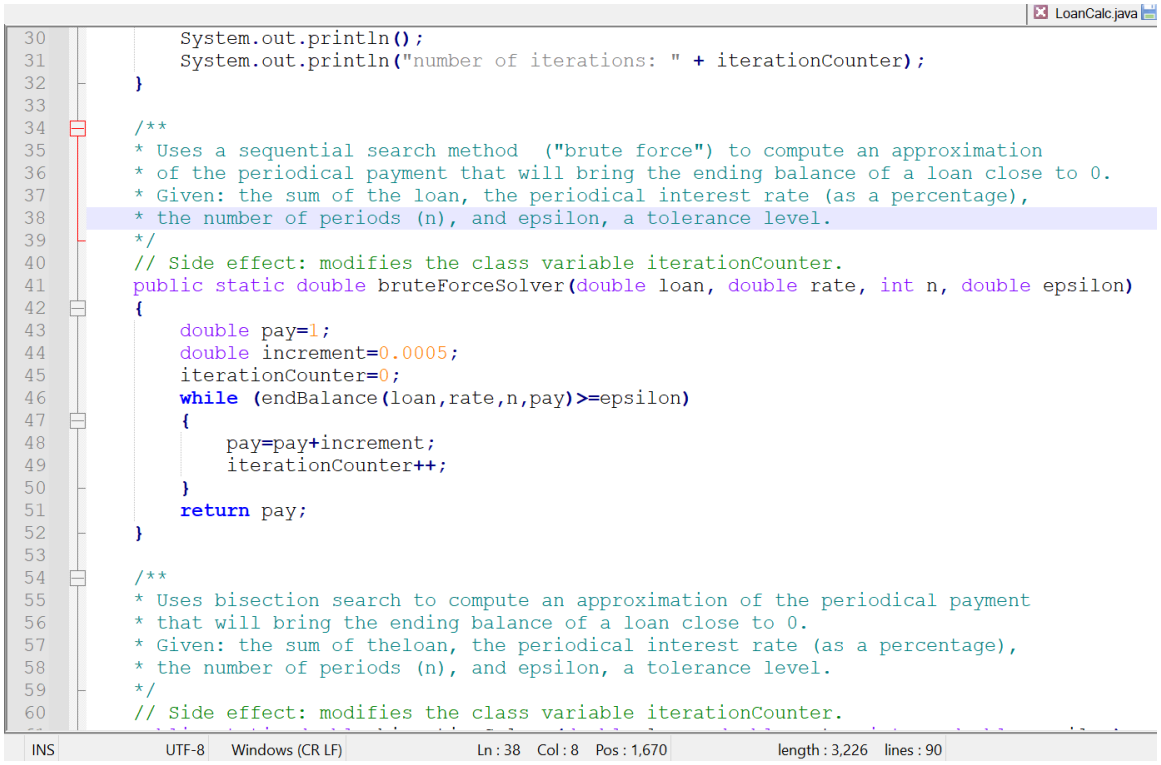
LowerCase

```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)
    {
       String s1="";
       int lenght=s.length();

       for(int i=0; i<lenght; i++)
       {
            int asciivalue=(int)(s.charAt(i));
            if(asciivalue>=65 && asciivalue<=90)
                s1=s1+(char)(asciivalue+32);
            else
                s1=s1+s.charAt(i);

        }

        return s1;

    }
}
```

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);
```



```java
        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 pay=1;
        double increment=0.0005;
        iterationCounter=0;
        while (endBalance(loan,rate,n,pay)>=epsilon)
        {
            pay=pay+increment;
            iterationCounter++;
        }
        return pay;
    }

    /**
    * 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.
```

LoanCalc.java

```
58      * the number of periods (n), and epsilon, a tolerance level.
59      */
60      // Side effect: modifies the class variable iterationCounter.
61      public static double bisectionSolver(double loan, double rate, int n, double epsilon)
62      {
63          iterationCounter=0;
64          double L=loan/n, H=loan;
65          double pay=(L+H)/2.0;
66
67          while (Math.abs(H-L)>epsilon)
68          {
69              if(endBalance(loan,rate,n,pay)<=0)
70                  H=pay;
71              else
72                  L=pay;
73              pay=(L+H)/2.0;
74              iterationCounter++;
75          }
76          return pay;
77      }
78
79      /**
80      * Computes the ending balance of a loan, given the sum of the loan, the periodical
81      * interest rate (as a percentage), the number of periods (n), and the periodical payment.
82      */
83      private static double endBalance(double loan, double rate, int n, double payment)
84      {
85          double remain=loan;
86          for (int i=1; i<=n; i++)
87              remain =(remain-payment)*(1+(rate/100));
88          return remain;
```

LoanCalc.java

```
61      public static double bisectionSolver(double loan, double rate, int n, double epsilon)
62      {
63          iterationCounter=0;
64          double L=loan/n, H=loan;
65          double pay=(L+H)/2.0;
66
67          while (Math.abs(H-L)>epsilon)
68          {
69              if(endBalance(loan,rate,n,pay)<=0)
70                  H=pay;
71              else
72                  L=pay;
73              pay=(L+H)/2.0;
74              iterationCounter++;
75          }
76          return pay;
77      }
78
79      /**
80      * Computes the ending balance of a loan, given the sum of the loan, the periodical
81      * interest rate (as a percentage), the number of periods (n), and the periodical payment.
82      */
83      private static double endBalance(double loan, double rate, int n, double payment)
84      {
85          double remain=loan;
86          for (int i=1; i<=n; i++)
87              remain =(remain-payment)*(1+(rate/100));
88          return remain;
89      }
90  }
```

Calender0

```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)
    {
        for (int i=1; i<=12; 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)
```

INS UTF-8 Windows (CR LF) Ln : 1 Col : 1 Pos : 1 length : 1,589 lines : 61

Calenda

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

    // 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==4||month==6||month==9||month==11)
            return 30;
        else
        {
            if(month==2)
            {
                if(isLeapYear(year)==true)
                    return 29;
                else
                    return 28;
            }
            else
                return 31;
        }
    }
}
```

Calendar1

```
/**
 * Prints the calendars of all the years in the 20th century.
 */
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


    /**
     * 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 debugDaysCounter = 0;
        //// Write the necessary initialization code, and replace the condition
        //// of the while loop with the necessary condition
        while (year<2000)
        {
            if(dayOfWeek==1)
                System.out.println(dayOfMonth+ "/" + month + "/" + year + " Sunday");
            else
                System.out.println(dayOfMonth+ "/" + month + "/" + year);
            if(dayOfWeek==1&&dayOfMonth==1)
                System.out.println(dayOfMonth+ "/" + month + "/" + year + " Sunday");
            else
                System.out.println(dayOfMonth+ "/" + month + "/" + year);
            if(dayOfWeek==1&&dayOfMonth==1)
                debugDaysCounter++;
            else
                debugDaysCounter=debugDaysCounter+0;
            advance();
        }
        System.out.println("During the 20th century, " + debugDaysCounter + " Sundays fell on the first day of the month");
    }

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

```
53                  year++;
54                  month=1;
55              }
56              else
57                  month++;
58              dayOfMonth=1;
59          }
60          else
61              dayOfMonth++;
62
63       }
64
65
66      // Returns true if the given year is a leap year, false otherwise.
67      private static boolean isLeapYear(int year)
68      {
69          if((year%4==0&&year%100!=0)||year%400==0)
70              return true;
71          return false;
72      }
73
74      // Returns the number of days in the given month and year.
75      // April, June, September, and November have 30 days each.
76      // February has 28 days in a common year, and 29 days in a leap year.
77      // All the other months have 31 days.
78      private static int nDaysInMonth(int month, int year)
79      {
80          if(month==4||month==6||month==9||month==11)
81              return 30;
82          else
83          {
```

```
66      // Returns true if the given year is a leap year, false otherwise.
67      private static boolean isLeapYear(int year)
68      {
69          if((year%4==0&&year%100!=0)||year%400==0)
70              return true;
71          return false;
72      }
73
74      // Returns the number of days in the given month and year.
75      // April, June, September, and November have 30 days each.
76      // February has 28 days in a common year, and 29 days in a leap year.
77      // All the other months have 31 days.
78      private static int nDaysInMonth(int month, int year)
79      {
80          if(month==4||month==6||month==9||month==11)
81              return 30;
82          else
83          {
84              if(month==2)
85              {
86                  if(isLeapYear(year)==true)
87                      return 29;
88                  else
89                      return 28;
90              }
91              else
92                  return 31;
93          }
94      }
95  }
96
```