# LoanCalc.java

```
1. /**
 2. * Computes the periodical payment necessary to re-pay a given loan.
 3. */
 4. public class LoanCalc {
 5.
 6.     static double epsilon = 0.001;  // The computation tolerance (estimation
error)
 7.     static int iterationCounter;    // Monitors the efficiency of the
calculation
 8.     static int iterationCounter1 = 0;
 9.
10.     /**
11.      * Gets the loan data and computes the periodical payment.
12.      * Expects to get three command-line arguments: sum of the loan
(double),
13.      * interest rate (double, as a percentage), and number of payments
(int).
14.      */
15.     public static void main(String[] args) {
16.             // Gets the loan data
17.             double loan = Double.parseDouble(args[0]);
18.             double rate = Double.parseDouble(args[1]);
19.             int n = Integer.parseInt(args[2]);
20.             System.out.println("Loan sum = " + loan + ", interest rate = " +
rate + "%, periods = " + n);
21.
22.             // Computes the periodical payment using brute force search
23.             System.out.print("Periodical payment, using brute force: ");
24.             System.out.printf("%.2f", bruteForceSolver(loan, rate, n,
epsilon));
25.             System.out.println();
26.             System.out.println("number of iterations: " + iterationCounter);
27.
28.
29.
30.
31.             // Computes the periodical payment using bisection search
32.             System.out.print("Periodical payment, using bi-section search:
");
33.             System.out.printf("%.2f", bisectionSolver(loan, rate, n,
epsilon));
34.             System.out.println();
35.             System.out.println("number of iterations: " + iterationCounter1);
36.     }
37.
38.     /**
39.     * Uses a sequential search method  ("brute force") to compute an
approximation
40.     * of the periodical payment that will bring the ending balance of a loan
close to 0.
```

```
41.     * Given: the sum of the loan, the periodical interest rate (as a
percentage),
42.     * the number of periods (n), and epsilon, a tolerance level.
43.     */
44.     // Side effect: modifies the class variable iterationCounter.
45.     public static double bruteForceSolver(double loan, double rate, int n,
double epsilon) {
46.             iterationCounter = 0;
47.             double payment = loan / n;
48.             double sum1 = loan;
49.
50.             while ( endBalance ( loan,  rate,  n,  payment)>= 0){
51.                     payment= payment + epsilon;
52.                     iterationCounter++;
53.             }
54.                     return payment;
55.
56.     }
57.
58.     /**
59.     * Uses bisection search to compute an approximation of the periodical
payment
60.     * that will bring the ending balance of a loan close to 0.
61.     * Given: the sum of theloan, the periodical interest rate (as a
percentage),
62.     * the number of periods (n), and epsilon, a tolerance level.
63.     */
64.     // Side effect: modifies the class variable iterationCounter.
65.     public static double bisectionSolver(double loan, double rate, int n,
double epsilon) {
66.             double L= loan/n;
67.             double H= loan;
68.             double payment= (L+H)/2;
69.
70.             while ((H-L) > epsilon) {
71.                     payment = (L + H) / 2;
72.                     if ((endBalance( loan,  rate,  n,  payment)* endBalance(
loan,  rate,  n,  L))>0) {
73.                             L= payment;
74.                     } else {
75.                             H= payment;
76.                     }
77.                     iterationCounter1++;
78.             }
79.
80.             return payment;
81.     }
82.
83.     /**
84.     * Computes the ending balance of a loan, given the sum of the loan, the
periodical
85.     * interest rate (as a percentage), the number of periods (n), and the
periodical payment.
86.     */
```

```
87.     private static double endBalance(double loan, double rate, int n, double
payment) {
88.             for (int i=0 ; i < n ; i++) {
89.                     loan = (loan - payment)*(1 + rate/100);
90.
91.             }
92.
93.             return loan;
94.     }
95. }
96.
```

# LowerCase.java

```
**/ 1. String processing exercise 1. */
 2. public class LowerCase {
 3.     public static void main(String[] args) {
 4.         String s = args[0];
 5.         System.out.println(lowerCase(s));
 6.     }
 7.
 8.    /**
 9.     * Returns a string which is identical to the original string,
10.     * except that all the upper-case letters are converted to lower-case
letters.
11.     * Non-letter characters are left as is.
12.     */
13.     public static String lowerCase(String s) {
14.         String non = "";
15.            for (int i=0 ; i < s.length () ; i++) {
16.                    if ((s.charAt(i) >= 65) && (s.charAt(i) <= 90)) {
17.                            int new1 = s.charAt(i) + 32;
18.                            non += (char) new1;
19.                    } else non += s.charAt(i);
20.            }
21.                     return non;
22.
23.                    }
24.            }
25.
```

# UniqueChars.java

```
**/ 1. String processing exercise 2. */
 2. public class UniqueChars {
 3.     public static void main(String[] args) {
 4.         String str = args[0];
 5.         System.out.println(uniqueChars(str));
 6.     }
 7.
 8.     /**
 9.      * Returns a string which is identical to the original string,
10.      * except that all the duplicate characters are removed,
11.      * unless they are space characters.
12.      */
13.     public static String uniqueChars(String s) {
14.     String non= "";
15.     for (int i = 0 ; i < s.length() ; i++) {
16.             if ((non.indexOf((s.charAt(i))) == -1) || (s.charAt(i) == 32)) {
17.                     non= non + s.charAt(i);
18.             }
19.     }
20.         return non;
21.     }
22. }
23.
```

# Calendar.java

```
 1. public class Calendar {
 2.     // Starting the calendar on 1/1/1900
 3.     static int dayOfMonth = 1;
 4.     static int month = 1;
 5.     static int year = 1900;
 6.     static int dayOfWeek = 2;      // 1.1.1900 was a Monday
 7.     static int nDaysInMonth = 31; // Number of days in January
 8.
 9.     public static void main(String args[]) {
10.            int givenyear = Integer.parseInt(args[0]);
11.         int debugDaysCounter = 0;
12.
13.             while (year < givenyear) {
14.                    advance();
15.             }
16.             int nDaysInGivenyear;
17.            if (isLeapYear(year)) {
18.                    nDaysInGivenyear= 366;
19.            } else {
20.                    nDaysInGivenyear= 365;
21.            }
22.            for (int i = 0; i < nDaysInGivenyear; i++) {
23.                    System.out.print(dayOfMonth + "/" + month + "/" + year);

24.                    if (dayOfWeek == 1) {
25.                           System.out.print (" Sunday");
26.                    }
27.                    System.out.println();
28.
29.                    advance();
30.         }
31.     }
32.
33.      private static void advance() {
34.            dayOfWeek = dayOfWeek % 7;
35.            dayOfWeek++;
36.
37.            if(dayOfMonth < nDaysInMonth) {
38.                    dayOfMonth++;
39.            } else {
40.            dayOfMonth =1;
41.            month++;
42.            nDaysInMonth= nDaysInMonth(month, year);
43.            if (dayOfWeek == 1) {
44.            }
45.            if (month == 13) {
46.                    month = 1;
47.                    year++;
48.                    nDaysInMonth = nDaysInMonth(month, year);
49.                    }
50.            }
```

```
51.      }
52.
53.     // Returns true if the given year is a leap year, false otherwise.
54.     private static boolean isLeapYear(int year) {
55.         if (((year % 100 !=0) && (year % 4 ==0)) || (year % 400 == 0)) {
56.             return true;
57.     } else {
58.             return false;
59.     }
60.     }
61.      private static int nDaysInMonth(int month,int year) {
62.              int daysInMonth=0;
63.     switch (month) {
64.             case 1:
65.             case 3 :
66.             case 5 :
67.             case 7:
68.             case 8 :
69.             case 10 :
70.             case 12 :
71.             daysInMonth = 31;
72.     break;
73.             case 4 :
74.             case 6 :
75.             case 9 :
76.             case 11 :
77.             daysInMonth = 30;
78.     break;
79.     case 2:
80.     if (isLeapYear(year) == true) {
81.             daysInMonth = 29;
82.     } else {
83.     daysInMonth= 28;
84.     }
85.     break;
86.     default :
87.     break;
88.     }
89.             return daysInMonth;
90.     }
91. }
92.
```