

```

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
7      static int iterationCounter;    // Monitors the efficiency of the
8
9      /**
10     * Gets the loan data and computes the periodical payment.
11     * Expects to get three command-line arguments: sum of the loan (double),
12     * interest rate (double, as a percentage), and number of payments
13     * (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
21         + "%, periods = " + n);
22
23         // Computes the periodical payment using brute force search
24         System.out.print("Periodical payment, using brute force: ");
25         System.out.printf("%.2f", bruteForceSolver(loan, rate, n, epsilon));
26         System.out.println();
27         System.out.println("number of iterations: " + iterationCounter);
28
29         // Computes the periodical payment using bisection search
30         System.out.print("Periodical payment, using bi-section search: ");
31         System.out.printf("%.2f", bisectionSolver(loan, rate, n, epsilon));
32         System.out.println();
33         System.out.println("number of iterations: " + iterationCounter);
34     }
35
36     /**
37     * Uses a sequential search method ("brute force") to compute an
38     * approximation
39     * of the periodical payment that will bring the ending balance of a loan
40     * close to 0.
41     * Given: the sum of the loan, the periodical interest rate (as a
42     * percentage),
43     * the number of periods (n), and epsilon, a tolerance level.
44     */
45     // Side effect: modifies the class variable iterationCounter.
46     public static double bruteForceSolver(double loan, double rate, int n,
47     double epsilon) {
48         iterationCounter = 0;
49         double payment = loan / n;
50         while (endBalance(loan, rate, n, payment) > epsilon) {
51             iterationCounter++;
52             payment += 0.001;
53         }
54         return payment;
55     }
56 }
57 /**

```

```

51     * Uses bisection search to compute an approximation of the periodical
52     payment
53     * that will bring the ending balance of a loan close to 0.
54     * Given: the sum of the loan, the periodical interest rate (as a
55     percentage),
56     * the number of periods (n), and epsilon, a tolerance level.
57     */
58     // Side effect: modifies the class variable iterationCounter.
59     public static double bisectionSolver(double loan, double rate, int n,
60     double epsilon) {
61         iterationCounter = 0;
62         double L = loan / n;
63         double H = loan;
64         double g = (L + H) / 2.0;
65         while ((H - L) > epsilon) {
66             iterationCounter++;
67             // Sets L and H for the next iteration
68             if ((endBalance(loan, rate, n, g) * endBalance(loan, rate, n, L))
69             > 0) {
70                 // the solution must be between g and H
71                 // so sets L accordingly
72                 L = g;
73             } else {
74                 // the solution must be between L and g
75                 // so sets H accordingly
76                 H = g;
77             }
78             // Computes the mid-value (g) for the next iteration
79             g = (L + H) / 2.0;
80         }
81         return g;
82     }
83     /**
84     * Computes the ending balance of a loan, given the sum of the loan, the
85     periodical
86     * interest rate (as a percentage), the number of periods (n), and the
87     periodical payment.
88     */
89     private static double endBalance(double loan, double rate, int n, double
90     payment) {
91         double balance = loan;
92         for (int i = 0; i < n; i++) {
93             balance = (balance - payment) * (1 + 0.01 * rate);
94         }
95         return balance;
96     }
97 }

```

```
1  /** String processing exercise 1. */
2  public class LowerCase {
3      public static void main(String[] args) {
4          String str = args[0]; // Gets the string to process
5          System.out.println(lowerCase(str)); // Prints the processed string
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
11      * letters.
12      * Non-letter characters are left as is.
13      */
14      public static String lowerCase(String s) {
15          String newString = ""; // Defines an empty string that will be
16          // gradually evolved into the answer string
17          for (int i = 0; i < s.length(); i++) {
18              char ch = s.charAt(i); // Gets the char in the i-th place at the
19              // original string
20              // Checks if the char is an upper-case letter
21              if ((ch >= 65) && (ch <= 90)) {
22                  ch += 32; // Converts an upper-case letter to a low-case
23                  // letter by changing it's ASCII value.
24              }
25              newString += ch; // Adds the processed char to the processed
26              // string
27          }
28          return newString; // Returns the processed string
29      }
30  }
```

```
1  /** String processing exercise 2. */
2  public class UniqueChars {
3      public static void main(String[] args) {
4          String str = args[0]; // Gets the string to process
5          System.out.println(uniqueChars(str)); // Prints the processed string
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 newString = ""; // Defines an empty string that will be
15         // gradually evolved into the answer string
16         for (int i = 0; i < s.length(); i++) {
17             char ch = s.charAt(i); // Gets the char in the i-th place at the
18             // original string
19             // Adds all the space characters from the original string to the
20             // processed string
21             if (ch == 32) {
22                 newString += ch; // Adds the char to the processed string
23             } else {
24                 // Removes all the duplicate characters of the original
25                 // string that are not space characters
26                 if (s.indexOf(ch) == i) {
27                     newString += ch; // Adds the char to the processed string
28                 }
29             }
30         }
31     }
32 }
```

```

1  /*
2  * Checks if a given year is a leap year or a common year,
3  * and computes the number of days in a given month and a given year.
4  */
5  public class Calendar0 {
6
7      // Gets a year (command-line argument), and tests the functions
      // isLeapYear and nDaysInMonth.
8      public static void main(String args[]) {
9          int year = Integer.parseInt(args[0]);
10         isLeapYearTest(year);
11         nDaysInMonthTest(year);
12     }
13
14     // Tests the isLeapYear function.
15     private static void isLeapYearTest(int year) {
16         String commonOrLeap = "common";
17         if (isLeapYear(year)) {
18             commonOrLeap = "leap";
19         }
20         System.out.println(year + " is a " + commonOrLeap + " year");
21     }
22
23     // Tests the nDaysInMonth function.
24     private static void nDaysInMonthTest(int year) {
25         for (int i = 1; i <= 12; i++) {
26             System.out.println("Month " + i + " has " + nDaysInMonth(i, year)
27                               + " days");
28         }
29     }
30
31     // Returns true if the given year is a leap year, false otherwise.
32     public static boolean isLeapYear(int year) {
33         if (year % 4 == 0) {
34             if (year % 100 != 0) {
35                 return true;
36             } else {
37                 if (year % 400 == 0) {
38                     return true;
39                 }
40             }
41         }
42         return false;
43     }
44
45     // Returns the number of days in the given month and year.
46     // April, June, September, and November have 30 days each.
47     // February has 28 days in a common year, and 29 days in a leap year.
48     // All the other months have 31 days.
49     public static int nDaysInMonth(int month, int year) {
50         switch (month) {
51             case 2:
52                 return (isLeapYear(year) ? 29 : 28);
53             case 4:
54                 return 30;
55             case 6:
56                 return 30;
57             case 9:

```

```
57         return 30;
58     case 11:
59         return 30;
60     default:
61         return 31;
62     }
63 }
64 }
65
66
```

```

1  /**
2   * Prints the calendars of all the years in the 20th century.
3   */
4  public class Calendar1 {
5      // Starting the calendar on 1/1/1900
6      static int dayOfMonth = 1;
7      static int month = 1;
8      static int year = 1900;
9      static int dayOfWeek = 2; // 1.1.1900 was a Monday
10     static int nDaysInMonth = 31; // Number of days in January
11
12     /**
13      * Prints the calendars of all the years in the 20th century. Also prints
14      * the
15      * number of Sundays that occurred on the first day of the month during
16      * this period.
17      */
18     public static void main(String args[]) {
19         // Advances the date and the day-of-the-week from 1/1/1900 till
20         // 31/12/1999, inclusive.
21         // Prints each date dd/mm/yyyy in a separate line. If the day is a
22         // Sunday, prints "Sunday".
23         // The following variable, used for debugging purposes, counts how
24         // many days were advanced so far.
25         int debugDaysCounter = 0;
26         int sundaysCounter = 0; // Counts the number of sundays that fell on
27         // the 1st day of month in the 20th century
28         while (year < 2000) {
29             System.out.println(dayOfMonth + "/" + month + "/" + year + ((
30                 dayOfWeek == 1) ? " Sunday" : ""));
31             if (dayOfWeek == 1 && dayOfMonth == 1) {
32                 sundaysCounter++;
33             }
34             advance();
35             debugDaysCounter++;
36             //// If you want to stop the loop after n days, replace the
37             // condition of the
38             //// if statement with the condition (debugDaysCounter == n)
39             if (debugDaysCounter == 37000) {
40                 break;
41             }
42         }
43         System.out.println("During the 20th century, " + sundaysCounter + "
44             Sundays fell on the first day of the month");
45     }
46
47     // Advances the date (day, month, year) and the day-of-the-week.
48     // If the month changes, sets the number of days in this month.
49     // Side effects: changes the static variables dayOfMonth, month, year,
50     // dayOfWeek, nDaysInMonth.
51     private static void advance() {
52         dayOfWeek = ((dayOfWeek == 7) ? 1 : dayOfWeek + 1);
53         if (dayOfMonth == nDaysInMonth(month, year)) {
54             dayOfMonth = 1;
55             if (month == 12) {
56                 year++;
57                 month = 1;
58             } else {
59

```

```

49         month++;
50     }
51     } else {
52         dayOfMonth++;
53     }
54 }
55
56 // Returns true if the given year is a leap year, false otherwise.
57 private static boolean isLeapYear(int year) {
58     if (year % 4 == 0) {
59         if (year % 100 != 0) {
60             return true;
61         } else {
62             if (year % 400 == 0) {
63                 return true;
64             }
65         }
66     }
67     return false;
68 }
69
70 // Returns the number of days in the given month and year.
71 // April, June, September, and November have 30 days each.
72 // February has 28 days in a common year, and 29 days in a leap year.
73 // All the other months have 31 days.
74 private static int nDaysInMonth(int month, int year) {
75     switch (month) {
76         case 2:
77             return (isLeapYear(year) ? 29 : 28);
78         case 4:
79             return 30;
80         case 6:
81             return 30;
82         case 9:
83             return 30;
84         case 11:
85             return 30;
86         default:
87             return 31;
88     }
89 }
90 }
91

```