# 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;    // 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) {
```

```
        double g = loan / n; // Starting point for the search

        while (endBalance(loan, rate, n, g) >= epsilon) {
            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) {
        iterationCounter = 0;
        double L = loan / n;
        double H = loan;
        double g = L + H / 2;

        while (H - L > epsilon && (Math.abs(g * g - endBalance(loan, rate, n, g))
>= epsilon)) {
            if (endBalance(loan, rate, n, g) * endBalance(loan, rate, n, L) > 0) {
                L = g;
            }
            else {
                H = g;
            }
            g = (L + H) / 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) {
```

```
        for (int i = 0; i < n; i++) {
            loan -= payment;
            loan += loan * (rate / 100);
        }

        return loan;

}
```

# Calendar

```
/**
 * Prints the calendars of all the years in the 20th century.
 */
public class Calendar {
    static int year = 1900;
    static int inputYear;
    static int dayOfMonth = 1;
    static int month = 1;
    static int dayOfWeek = 2;     // 1.1.1900 was a Monday
    static int totalDays = 0;
    static int nDaysInMonth = 31; // Number of days in January
    static int specialSunday = 0; // Number of sundays that fall on the 1st of
each month

    public static void main(String args[]) {
        inputYear = Integer.parseInt(args[0]);

        while (year < inputYear + 1) {
            if (!(year < inputYear)) {
                System.out.println(dayOfMonth + "/" + month + "/" + inputYear +
(dayOfWeek == 1 ? " Sunday" : ""));
            }
            advance();
        }
    }

    // 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 (year < inputYear) {
            totalDays++;

            if (dayOfWeek < 7) {
                dayOfWeek++;
            } else {
                dayOfWeek = 1;
            }

            if (isLeapYear(year) && totalDays == 366 || !isLeapYear(year) &&
totalDays == 365) {
                year++;
                totalDays = 0;
            }
            return;
```

```
    }

    if (dayOfWeek < 7) {
        dayOfWeek++;
    } else {
        dayOfWeek = 1;
    }

    if (dayOfMonth == nDaysInMonth(month, inputYear)) {
        dayOfMonth = 1;
        if (month == 12) {
            month = 0;
            year++;
        }
        month = month + 1;
    } else {
        dayOfMonth++;
    }
 }

// Returns true if the given year is a leap year, false otherwise.
private static boolean isLeapYear(int year) {
    // Checks if the year is divisible by 400
    boolean leapYear = ((year % 400) == 0);
    // Then checks if the year is divisible by 4 but not by 100
    leapYear = leapYear || (((year % 4) == 0) && ((year % 100) != 0));

    return leapYear;
}

// 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) {
    int days = 0;

    switch (month) {
        case 1:
            days = 31;
            break;
        case 2:
            if (isLeapYear(year)) {
                days = 29;
                break;
            }
            days = 28;
                break;
```

```
            case 3:
                days = 31;
                    break;
            case 4:
                days = 30;
                    break;
                            case 5:
                days = 31;
                    break;
            case 6:
                days = 30;
                    break;
                            case 7:
                days = 31;
                    break;
                            case 8:
                days = 31;
                    break;
            case 9:
                days = 30;
                    break;
                            case 10:
                days = 31;
                    break;
            case 11:
                days = 30;
                    break;
            case 12:
                days = 31;
                    break;
        }
        return days;
    }
}
```

# 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 changed = "";

        for (int i = 0; i < s.length(); i++) {
            if ((int) s.charAt(i) > 64 && (int) s.charAt(i) < 91) { // Checks if letter is
upper case using ASCII table numbers
                changed += (char) ((int) s.charAt(i) + 32); // difference between
upper case and lower case of same letter
            } else {
                changed += s.charAt(i); // If not an upper case letter, then continue
            }
        }
        return changed;
    }
}
```

# 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 uniqueString = "";

        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) == ' ') { // if char is space, add and go on
                uniqueString += s.charAt(i);
            }
            if (uniqueString.contains(String.valueOf(s.charAt(i)))) {
                // checks if a letter is already inside the string or not. If yes, do
nothing
            } else {
                uniqueString += s.charAt(i); // If not, meaning that it's unique - add it
            }
        }

        return uniqueString;
    }
}
```