Week 2 – S2 –Assignment Homework Solution

Name: Ramesh Harisabapathi Chettiar

Roll Number: RA2411030010263

Course: Networking and Communications

Semester: 3

Date of Submission: 23/08/2025

Problem 1: Write a program to implement a simple spell checker that finds and suggests corrections for misspelled words using string distance calculation

Hint =>

a. Take user input for a sentence and a dictionary of correct words (stored in an array)

b. Create a method to split the sentence into words without using split():

- **i. Use charAt() to identify word boundaries (spaces, punctuation)**
- **ii. Extract each word using substring() method**
- **iii. Store words in an array**

c. Create a method to calculate string distance between two words:

- **i. Count character differences between words of same length**
- **ii. For different lengths, calculate insertion/deletion distance**
- **iii. Return the distance as an integer**

d. Create a method to find the closest matching word from dictionary:

- **i. Compare input word with each dictionary word**
- **ii. Find the word with minimum distance**
- **iii. Return the suggestion if distance is within acceptable range (≤ 2)**

e. Create a method to display spell check results in tabular format:

- **i. Show original word, suggested correction, distance score**
- **ii. Mark words as "Correct" or "Misspelled"**

f. The main function processes the sentence and displays comprehensive spell check report

```java
/*Problem 1: Write a program to implement a simple spell checker that finds
and suggests corrections for misspelled words using string distance
calculation
Hint =>
a. Take user input for a sentence and a dictionary of correct words (stored in an array)
b. Create a method to split the sentence into words without using split():
• i. Use charAt() to identify word boundaries (spaces, punctuation)
• ii. Extract each word using substring() method
• iii. Store words in an array
c. Create a method to calculate string distance between two words:
• i. Count character differences between words of same length
• ii. For different lengths, calculate insertion/deletion distance
• iii. Return the distance as an integer
d. Create a method to find the closest matching word from dictionary:
• i. Compare input word with each dictionary word
• ii. Find the word with minimum distance
• iii. Return the suggestion if distance is within acceptable range (≤ 2)
e. Create a method to display spell check results in tabular format:
• i. Show original word, suggested correction, distance score
• ii. Mark words as "Correct" or "Misspelled"
f. The main function processes the sentence and displays comprehensive spell check report */

import java.util.Scanner;

public class CustomSpellChecker {

    /* b. Extract words without using split() */
    public static String[] extractWords(String sentence) {
        int n = sentence.length();
        StringBuilder word = new StringBuilder();
        String[] temp = new String[n];
        int count = 0;

        for (int i = 0; i < n; i++) {
```

```
        for (int i = 0; i < n; i++) {
            if (Character.isLetter(ch)) {
                word.append(ch);
            } else {
                if (word.length() > 0) {
                    temp[count++] = word.toString();
                    word.setLength(0);
                }
            }
        }
        if (word.length() > 0) {
            temp[count++] = word.toString();
        }

        String[] words = new String[count];
        System.arraycopy(temp, 0, words, 0, count);
        return words;
    }

    /* c. String distance calculation */
    public static int stringDistance(String a, String b) {
        int lenA = a.length();
        int lenB = b.length();
        int minLen = Math.min(lenA, lenB);
        int diff = Math.abs(lenA - lenB);

        for (int i = 0; i < minLen; i++) {
            if (a.charAt(i) != b.charAt(i)) diff++;
        }
        return diff;
    }
```

```
    /* d. Find closest dictionary word */
    public static String suggestWord(String word, String[] dictionary) {
        String suggestion = word;
        int minDist = Integer.MAX_VALUE;

        for (String dictWord : dictionary) {
            int dist = stringDistance(word.toLowerCase(), dictWord.toLowerCase());
            if (dist < minDist) {
                minDist = dist;
                suggestion = dictWord;
            }
        }

        if (minDist <= 2 && !suggestion.equalsIgnoreCase(word)) {
            return suggestion;
        } else {
            return "Correct";
        }
    }

    /* e. Display spell check results */
    public static void displayResults(String[] words, String[] dictionary) {
        System.out.printf("%-15s %-20s %-10s %-15s%n", "Word", "Suggestion", "Distance", "Status");
        System.out.println("-----------------------------------------------------------------");

        for (String word : words) {
            String bestMatch = word;
            String status = "Correct";
            int minDist = Integer.MAX_VALUE;

            for (String dictWord : dictionary) {
```

```
92          for (String word : words) {
97              for (String dictWord : dictionary) {
99                  if (dist < minDist) {
101                     bestMatch = dictWord;
102                 }
103             }
104
105             String suggestion;
106             if (minDist == 0) {
107                 suggestion = "-";
108                 status = "Correct";
109             } else if (minDist <= 2) {
110                 suggestion = bestMatch;
111                 status = "Misspelled";
112             } else {
113                 suggestion = "No close match";
114                 status = "Misspelled";
115             }
116
117             System.out.printf("%-15s %-20s %-10d %-15s%n", word, suggestion, minDist, status);
118         }
119     }
120
121     /* f. Main method */
122     public static void main(String[] args) {
123         Scanner input = new Scanner(System.in);
124
125         System.out.print("Enter number of dictionary words: ");
126         int n = input.nextInt();
127         input.nextLine();
128         String[] dictionary = new String[n];
```

```
121     /* f. Main method */
122     public static void main(String[] args) {
123         Scanner input = new Scanner(System.in);
124
125         System.out.print("Enter number of dictionary words: ");
126         int n = input.nextInt();
127         input.nextLine();
128         String[] dictionary = new String[n];
129         System.out.println("Enter dictionary words:");
130         for (int i = 0; i < n; i++) {
131             dictionary[i] = input.nextLine().trim();
132         }
133
134         System.out.println("Enter a sentence to check:");
135         String sentence = input.nextLine();
136         String[] words = extractWords(sentence);
137
138         displayResults(words, dictionary);
139         input.close();
140     }
141 }
142
```

OUTPUT→

```
Enter number of dictionary words: 3
Enter dictionary words:
Ramesh
Cyber Security
Purple Teamer
Enter a sentence to check:
Hi everyone,I am Ramesh Chettiar interested in becoming a Purple Teamer
Word            Suggestion          Distance   Status
-------------------------------------------------------------
Hi              No close match      6          Misspelled
everyone        No close match      8          Misspelled
I               No close match      6          Misspelled
am              No close match      6          Misspelled
Ramesh          -                   0          Correct
Chettiar        No close match      8          Misspelled
interested      No close match      9          Misspelled
in              No close match      6          Misspelled
becoming        No close match      8          Misspelled
a               No close match      6          Misspelled
Purple          No close match      6          Misspelled
Teamer          No close match      6          Misspelled
```

Problem 2: Write a program to create a password strength analyzer and generator using ASCII values and StringBuilder

Hint =>

a. Take user input for multiple passwords to analyze

b. Create a method to analyze password strength using ASCII values:

- **i. Count uppercase letters (ASCII 65-90)**
- **ii. Count lowercase letters (ASCII 97-122)**
- **iii. Count digits (ASCII 48-57)**
- **iv. Count special characters (other printable ASCII)**
- **v. Check for common patterns and sequences**

c. Create a method to calculate password strength score:

- **i. Length points: +2 per character above 8**
- **ii. Character variety: +10 for each type present**
- **iii. Deduct points for common patterns (123, abc, qwerty)**
- **iv. Return strength level: Weak (0-20), Medium (21-50), Strong (51+)**

d. Create a method using StringBuilder to generate strong passwords:

- **i. Take desired length as parameter**
- **ii. Ensure at least one character from each category**
- **iii. Fill remaining positions with random characters**
- **iv. Shuffle the password for better randomness**

e. Create a method to display analysis results in tabular format:

i. Password, Length, Uppercase count, Lowercase count, Digits, Special chars, Score, Strength

f. The main function analyzes existing passwords and generates new strong passwords based on user requirements

```java
/*Problem 2: Write a program to calculate the strength of a password
Hint =>
a. Take user input for a password string
b. Check for different criteria:
   • i. Length of password (minimum 8 characters)
   • ii. Contains uppercase and lowercase letters
   • iii. Contains digits
   • iv. Contains special characters
c. Assign points for each criterion and calculate total score
d. Display strength level as Weak, Moderate, or Strong
*/

import java.util.Scanner;

public class CustomPasswordStrengthCalculator {

    /* b. Check for different criteria */
    public static int calculateStrength(String password) {
        int score = 0;

        // i. Length
        if (password.length() >= 8) score++;

        // ii. Uppercase + lowercase
        if (password.matches(".*[A-Z].*") && password.matches(".*[a-z].*")) score++;

        // iii. Digits
        if (password.matches(".*[0-9].*")) score++;

        // iv. Special chars
        if (password.matches(".*[^a-zA-Z0-9].*")) score++;

        return score;
    }
```

```java
    /* d. Display strength level */
    public static String strengthLevel(int score) {
        switch (score) {
            case 4: return "Strong";
            case 3: return "Moderate";
            default: return "Weak";
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter password: ");
        String pwd = sc.nextLine();

        int score = calculateStrength(pwd);
        String level = strengthLevel(score);

        System.out.println("\nPassword Strength: " + level + " (Score: " + score + "/4)");
        sc.close();
    }
}

```

OUTPUT→

```
Enter password: JaVaSTeP230825!!

Password Strength: Strong (Score: 4/4)
```

Problem 3: Write a program to implement a text-based data compression algorithm using character frequency and StringBuilder

Hint =>

a. Take user input for text to compress

b. Create a method to count character frequency without using HashMap:

- **i. Create arrays to store characters and their frequencies**
- **ii. Use charAt() to iterate through text**
- **iii. Count occurrences of each unique character**
- **iv. Return parallel arrays of characters and frequencies**

c. Create a method to create compression codes using StringBuilder:

- **i. Assign shorter codes to more frequent characters**
- **ii. Use numbers/symbols for common characters**
- **iii. Create a mapping table of original character to code**
- **iv. Return the mapping as a 2D array**

d. Create a method to compress text using the generated codes:

- **i. Replace each character with its corresponding code**
- **ii. Use StringBuilder for efficient string building**
- **iii. Calculate compression ratio (original size vs compressed size)**

e. Create a method to decompress the text:

- **i. Reverse the compression process using the mapping table**
- **ii. Validate that decompression returns original text**

f. Create a method to display compression analysis:

- **i. Show character frequency table**
- **ii. Display compression mapping**
- **iii. Show original text, compressed text, decompressed text**
- **iv. Calculate and display compression efficiency percentage**

g. The main function performs compression, decompression, and displays complete analysis

```java
/* Problem 3: Write a program to implement a text-based data compression
algorithm using character frequency and StringBuilder
Hint =>
a. Take user input for text to compress
b. Create a method to count character frequency without using HashMap:
• i. Create arrays to store characters and their frequencies
• ii. Use charAt() to iterate through text
• iii. Count occurrences of each unique character
• iv. Return parallel arrays of characters and frequencies
c. Create a method to create compression codes using StringBuilder:
• i. Assign shorter codes to more frequent characters
• ii. Use numbers/symbols for common characters
• iii. Create a mapping table of original character to code
• iv. Return the mapping as a 2D array
d. Create a method to compress text using the generated codes:
• i. Replace each character with its corresponding code
• ii. Use StringBuilder for efficient string building
• iii. Calculate compression ratio (original size vs compressed size)
e. Create a method to decompress the text:
• i. Reverse the compression process using the mapping table
• ii. Validate that decompression returns original text
f. Create a method to display compression analysis:
• i. Show character frequency table
• ii. Display compression mapping
• iii. Show original text, compressed text, decompressed text
• iv. Calculate and display compression efficiency percentage
g. The main function performs compression, decompression, and displays complete analysis*/
import java.util.Scanner;

public class CustomTextCompressionAlgorithm{
    public static Object[] countFrequencies(String text){
        char[] uniqueChars=new char[text.length()];
        int[] freq=new int[text.length()];
        int uniqueCount=0;
```

```
        for(int i=0; i<text.length(); i++){
            char c=text.charAt(i);
            boolean found=false;

            for(int j=0; j<uniqueCount; j++){
                if(uniqueChars[j]==c){
                    freq[j]++;
                    found=true;
                    break;
                }
            }
            if(!found){
                uniqueChars[uniqueCount]=c;
                freq[uniqueCount]=1;
                uniqueCount++;
            }
        }

        char[] chars=new char[uniqueCount];
        int[] frequencies=new int[uniqueCount];
        System.arraycopy(uniqueChars, 0, chars, 0, uniqueCount);
        System.arraycopy(freq, 0, frequencies, 0, uniqueCount);

        return new Object[]{chars,frequencies};
    }

    public static String[][] generateCodes(char[] chars, int[] freq){
        int n=chars.length;
        String[][] mapping=new String[n][2];
```

```
        for(int i=0; i<n; i++){
            for(int j=i+1; j<n; j++){
                if(freq[j]>freq[i]){
                    int tmpF=freq[i];  freq[i]=freq[j];  freq[j]=tmpF;
                    char tmpC=chars[i];  chars[i]=chars[j];  chars[j]=tmpC;
                }
            }
        }

        for(int i=0; i<n; i++){
            mapping[i][0]=String.valueOf(chars[i]);
            mapping[i][1]=Integer.toString(i);
        }
        return mapping;
    }

    public static String compressText(String text, String[][] mapping){
        StringBuilder compressed=new StringBuilder();

        for(int i=0; i<text.length(); i++){
            String c=String.valueOf(text.charAt(i));

            for(String[] map : mapping){
                if(map[0].equals(c)){
                    compressed.append(map[1]).append(" ");
                    break;
                }
            }
        }
        return compressed.toString().trim();
    }
```

```java
    public static String decompressText(String compressed, String[][] mapping){
        StringBuilder decompressed=new StringBuilder();
        String[] codes=compressed.split(" ");

        for(String code : codes){
            for(String[] map : mapping){
                if(map[1].equals(code)){
                    decompressed.append(map[0]);
                    break;
                }
            }
        }
        return decompressed.toString();
    }

    public static void displayFrequencies(char[] chars, int[] freq){
        System.out.println("\nCharacter Frequency Table:");
        System.out.println("Char\tFrequency");
        for(int i=0; i<chars.length; i++){
            System.out.println(chars[i]+"\t"+freq[i]);
        }
    }

    public static void displayMapping(String[][] mapping){
        System.out.println("\nCompression Mapping:");
        System.out.println("Char\tCode");
        for(String[] map : mapping){
            System.out.println(map[0]+"\t"+map[1]);
        }
    }

```

```java
    public static String decompressText(String compressed, String[][] mapping){
        StringBuilder decompressed=new StringBuilder();
        String[] codes=compressed.split(" ");

        for(String code : codes){
            for(String[] map : mapping){
                if(map[1].equals(code)){
                    decompressed.append(map[0]);
                    break;
                }
            }
        }
        return decompressed.toString();
    }

    public static void displayFrequencies(char[] chars, int[] freq){
        System.out.println("\nCharacter Frequency Table:");
        System.out.println("Char\tFrequency");
        for(int i=0; i<chars.length; i++){
            System.out.println(chars[i]+"\t"+freq[i]);
        }
    }

    public static void displayMapping(String[][] mapping){
        System.out.println("\nCompression Mapping:");
        System.out.println("Char\tCode");
        for(String[] map : mapping){
            System.out.println(map[0]+"\t"+map[1]);
        }
    }

```

```java
        System.out.print("Enter text to compress: ");
        String text=input.nextLine();

        Object[] result=countFrequencies(text);
        char[] chars=(char[])result[0];
        int[] freq=(int[])result[1];

        String[][] mapping=generateCodes(chars,freq);

        String compressed=compressText(text,mapping);

        String decompressed=decompressText(compressed,mapping);

        displayFrequencies(chars,freq);
        displayMapping(mapping);

        System.out.println("\nOriginal Text: "+text);
        System.out.println("Compressed Text: "+compressed);
        System.out.println("Decompressed Text: "+decompressed);

        int originalSize=text.length();
        int compressedSize=compressed.length();
        double efficiency=(1-(double)compressedSize/originalSize)*100;
        System.out.printf("\nCompression Ratio: %.2f%% efficiency\n", efficiency);

        if(text.equals(decompressed)){
            System.out.println("Decompression successful: Original text restored");
        }
        else{
            System.out.println("Decompression failed: Text mismatch");
        }
        input.close();
```

OUTPUT→

```
Enter text to compress: Ramesh Harisabapathi Chettiar

Character Frequency Table:
Char    Frequency
a       6
h       3
i       3
t       3
        2
r       2
s       2
e       2
R       1
m       1
b       1
p       1
H       1
C       1
```

```
Compression Mapping:
Char    Code
a       0
h       1
i       2
t       3
        4
r       5
s       6
e       7
R       8
m       9
b       10
p       11
H       12
C       13

Original Text: Ramesh Harisabapathi Chettiar
Compressed Text: 8 0 9 7 6 1 4 12 0 5 2 6 0 10 0 11 0 3 1 2 4 13 1 7 3 3 2 0 5
Decompressed Text: Ramesh Harisabapathi Chettiar
```

```
Compression Ratio: -110.34% efficiency
Decompression successful: Original text restored
```

Problem 4: Write a program to create a text-based calculator that can parse and evaluate mathematical expressions from strings

Hint =>

a. Take user input for mathematical expressions as strings (e.g., "15 + 23 * 4 - 10")

b. Create a method to validate expression format:

- **i. Check for valid characters (digits, operators, spaces, parentheses)**
- **ii. Validate operator placement and parentheses matching**
- **iii. Use ASCII values to identify different character types**
- **iv. Return boolean indicating if expression is valid**

c. Create a method to parse numbers from string:

- **i. Use charAt() to identify digit sequences**
- **ii. Extract multi-digit numbers using substring()**
- **iii. Convert string numbers to integers**
- **iv. Store numbers and operators in separate arrays**

d. Create a method to evaluate expression using order of operations:

- **i. Handle multiplication and division first**
- **ii. Then handle addition and subtraction**
- **iii. Process from left to right for same precedence**
- **iv. Use StringBuilder to show step-by-step calculation**

e. Create a method to handle parentheses:

- **i. Find innermost parentheses using indexOf() and lastIndexOf()**
- **ii. Evaluate expressions inside parentheses first**
- **iii. Replace parenthetical results in main expression**

f. Create a method to display calculation steps:

- **i. Show original expression**
- **ii. Display each step of evaluation**
- **iii. Show final result with validation**

g. The main function processes multiple expressions and shows detailed calculation process

```
/*Problem 4: Write a program to create a text-based calculator that can parse
and evaluate mathematical expressions from strings
Hint =>
a. Take user input for mathematical expressions as strings (e.g., "15 + 23 * 4 - 10")
b. Create a method to validate expression format:
• i. Check for valid characters (digits, operators, spaces, parentheses)
• ii. Validate operator placement and parentheses matching
• iii. Use ASCII values to identify different character types
• iv. Return boolean indicating if expression is valid
c. Create a method to parse numbers from string:
• i. Use charAt() to identify digit sequences
• ii. Extract multi-digit numbers using substring()
• iii. Convert string numbers to integers
• iv. Store numbers and operators in separate arrays
d. Create a method to evaluate expression using order of operations:
• i. Handle multiplication and division first
• ii. Then handle addition and subtraction
• iii. Process from left to right for same precedence
• iv. Use StringBuilder to show step-by-step calculation
e. Create a method to handle parentheses:
• i. Find innermost parentheses using indexOf() and lastIndexOf()
• ii. Evaluate expressions inside parentheses first
• iii. Replace parenthetical results in main expression
f. Create a method to display calculation steps:
• i. Show original expression
• ii. Display each step of evaluation
• iii. Show final result with validation
g. The main function processes multiple expressions and shows detailed calculation process */
import java.util.Scanner;
```

```
    public static boolean validateExpression(String expr){

        for(int i=0; i<expr.length(); i++){
            char ch=expr.charAt(i);

            if(!(ch>='0' && ch<='9') && ch!='+' && ch!='-' && ch!='*' && ch!='/' && ch!='(' && ch!=')'
                return false;
            }

            if(ch=='(')  balance++;
            if(ch==')')  balance--;

            if((ch=='+' || ch=='-' || ch=='*' || ch=='/') && (prev=='+' || prev=='-' || prev=='*' || pr
                return false;
            }

            if(balance<0)  return false;
            prev=ch;
        }
        return balance==0;
    }

```

```
31  public class CustomTextBasedMathematicalCalculator{
56      public static int evaluateExpression(String expr, StringBuilder steps){
57          while(expr.contains("(")){
59              int open=expr.lastIndexOf("(");close);
60              String inside=expr.substring(open + 1, close);
61              int val=evaluateExpression(inside,steps);
62              expr=expr.substring(0, open)+val+expr.substring(close+1);
63              steps.append("Evaluate (").append(inside).append(") → ").append(val).append("\n");
64          }
65
66          int[] numbers=new int[100];
67          char[] operators=new char[100];
68          int nIndex=0, oIndex=0;
69
70          for(int i=0; i<expr.length(); ){
71              char ch=expr.charAt(i);
72              if(ch==' '){
73                  i++;
74                  continue;
75              }
76              if(ch>='0' && ch<='9'){
77                  int j=i;
78
79                  while(j<expr.length() && expr.charAt(j)>='0' && expr.charAt(j)<='9') j++;
80                  numbers[nIndex++]=Integer.parseInt(expr.substring(i,j));
81                  i=j;
82              }
83              else{
84                  operators[oIndex++]=ch;
```

```
83              else{
84                  operators[oIndex++]=ch;
85                  i++;
86              }
87          }
88
89          for(int i=0; i<oIndex; i++){
90              if(operators[i]=='*' || operators[i]=='/'){
91                  int result=(operators[i]=='*') ? numbers[i]*numbers[i+1] : numbers[i]/numbers[i+1];
92                  steps.append(numbers[i]).append(" ").append(operators[i]).append(" ").append(numbers[i+
93
94                  numbers[i]=result;
95                  for(int j=i+1; j<nIndex-1; j++)  numbers[j]=numbers[j+1];
96                  for(int j=i; j<oIndex-1; j++)  operators[j]=operators[j+1];
97                  nIndex--;
98                  oIndex--;
99                  i--;
100             }
101         }
102
103         int result=numbers[0];
104         int k=1;
105         for(int i=0; i<oIndex; i++){
106             if(operators[i]=='+'){
107                 steps.append(result).append(" + ").append(numbers[k]).append(" = ");
108                 result+=numbers[k];
109                 steps.append(result).append("\n");
```

```
            else if(operators[i]=='-'){
                steps.append(result).append(" - ").append(numbers[k]).append(" = ");
                result-=numbers[k];
                steps.append(result).append("\n");
            }
            k++;
        }
        return result;
    }

    public static void main(String[] args){
        Scanner input=new Scanner(System.in);

        System.out.println("Text-Based Calculator");
        System.out.println("====================");
        System.out.print("Enter number of expressions: ");
        int n=input.nextInt();
        input.nextLine();

        for(int i=0; i<n; i++){
            System.out.print("\nEnter expression "+(i+1)+": ");
            String expr=input.nextLine();

            if(!validateExpression(expr)){
                System.out.println("Invalid expression format!");
```

```
            if(!validateExpression(expr)){
                System.out.println("Invalid expression format!");
                continue;
            }

            StringBuilder steps=new StringBuilder();
            steps.append("\nOriginal Expression: ").append(expr).append("\n");

            int result=evaluateExpression(expr,steps);
            steps.append("Final Result: ").append(result).append("\n");

            System.out.println("\nCalculation Steps:\n"+steps);
        }
        input.close();
    }
```

OUTPUT→

```
Text-Based Calculator
====================
Enter number of expressions: 3

Enter expression 1: 15+10

Calculation Steps:

Original Expression: 15+10
15 + 10 = 25
====================
Enter number of expressions: 3

Enter expression 1: 15+10

Calculation Steps:

Original Expression: 15+10
15 + 10 = 25

Enter expression 1: 15+10

Calculation Steps:

Original Expression: 15+10
15 + 10 = 25

Calculation Steps:

Original Expression: 15+10
15 + 10 = 25

Original Expression: 15+10
```

```
Calculation Steps:

Original Expression: 15+10
15 + 10 = 25

Original Expression: 15+10
15 + 10 = 25
Original Expression: 15+10
15 + 10 = 25
Final Result: 25
15 + 10 = 25
Final Result: 25

Final Result: 25




Enter expression 2: 20^(-0.3)
Invalid expression format!
Enter expression 2: 20^(-0.3)
Invalid expression format!
Invalid expression format!


Enter expression 3: 20^2
Invalid expression format!
```

Problem 5: Write a program to analyze and format structured data from

CSV-like text input using string manipulation methods

Hint =>

a. Take user input for CSV-like data (comma-separated values in multiple lines)

b. Create a method to parse CSV data without using split():

- **i. Use charAt() to identify commas and newlines**
- **ii. Extract each field using substring() method**
- **iii. Handle quoted fields that may contain commas**
- **iv. Store data in a 2D array structure**

c. Create a method to validate and clean data:

- **i. Remove leading/trailing spaces from each field**
- **ii. Validate numeric fields using ASCII values**
- **iii. Check for missing or invalid data**
- **iv. Apply data type conversions where needed**

d. Create a method to perform data analysis:

- **i. Calculate column statistics (min, max, average for numeric columns)**
- **ii. Count unique values in categorical columns**
- **iii. Identify data quality issues (missing, invalid entries)**

e. Create a method using StringBuilder to format output:

- **i. Create aligned tabular display with fixed column widths**
- **ii. Add borders and headers for better readability**
- **iii. Format numeric values with proper decimal places**
- **iv. Highlight data quality issues**

f. Create a method to generate data summary report:

- **i. Show total records processed**
- **ii. Display column-wise statistics**
- **iii. List data quality findings**

- iv. Calculate data completeness percentage

g. The main function processes CSV input and generates formatted output with analysis report

```
/*Problem 5: Write a program to analyze and format structured data from
CSV-like text input using string manipulation methods
Hint =>
a. Take user input for CSV-like data (comma-separated values in multiple lines)
b. Create a method to parse CSV data without using split():
• i. Use charAt() to identify commas and newlines
• ii. Extract each field using substring() method
• iii. Handle quoted fields that may contain commas
• iv. Store data in a 2D array structure
c. Create a method to validate and clean data:
• i. Remove leading/trailing spaces from each field
• ii. Validate numeric fields using ASCII values
• iii. Check for missing or invalid data
• iv. Apply data type conversions where needed
d. Create a method to perform data analysis:
• i. Calculate column statistics (min, max, average for numeric columns)
• ii. Count unique values in categorical columns
• iii. Identify data quality issues (missing, invalid entries)
e. Create a method using StringBuilder to format output:
• i. Create aligned tabular display with fixed column widths
• ii. Add borders and headers for better readability
• iii. Format numeric values with proper decimal places
• iv. Highlight data quality issues
f. Create a method to generate data summary report:
• i. Show total records processed
• ii. Display column-wise statistics
• iii. List data quality findings
• iv. Calculate data completeness percentage
g. The main function processes CSV input and generates formatted output with analysis report */
import java.util.Scanner;

public class CustomCSVFormatter{
    public static boolean validateExpression(String expr){
        int balance=0;
```

```
        for(int i=0; i<expr.length(); i++){
            char ch=expr.charAt(i);

            if(!(ch>='0' && ch<='9') && ch!='+' && ch!='-' && ch!='*' && ch!='/' && ch!='(' && ch!=')'
                return false;
            }

            if(ch=='(')  balance++;
            if(ch==')')  balance--;

            if((ch=='+' || ch=='-' || ch=='*' || ch=='/') && (prev=='+' || prev=='-' || prev=='*' || p
                return false;
            }

            if(balance<0)  return false;
            prev=ch;
        }
        return balance==0;
    }

    public static int evaluateExpression(String expr, StringBuilder steps){
        while(expr.contains("(")){
            int close=expr.indexOf(")");
            int open=expr.lastIndexOf("(",close);
            String inside=expr.substring(open + 1, close);
            int val=evaluateExpression(inside,steps);
            expr=expr.substring(0, open)+val+expr.substring(close+1);
            steps.append("Evaluate (").append(inside).append(") → ").append(val).append("\n");
        }

        int[] numbers=new int[100];
```

```
    public static int evaluateExpression(String expr, StringBuilder steps){

        int[] numbers=new int[100];
        char[] operators=new char[100];
        int nIndex=0, oIndex=0;

        for(int i=0; i<expr.length(); ){
            char ch=expr.charAt(i);
            if(ch==' '){
                i++;
                continue;
            }
            if(ch>='0' && ch<='9'){
                int j=i;

                while(j<expr.length() && expr.charAt(j)>='0' && expr.charAt(j)<='9') j++;
                numbers[nIndex++]=Integer.parseInt(expr.substring(i,j));
                i=j;
            }
            else{
                operators[oIndex++]=ch;
                i++;
            }
        }

        for(int i=0; i<oIndex; i++){
            if(operators[i]=='*' || operators[i]=='/'){
                int result=(operators[i]=='*') ? numbers[i]*numbers[i+1] : numbers[i]/numbers[i+1];
                steps.append(numbers[i]).append(" ").append(operators[i]).append(" ").append(numbers[i+

                numbers[i]=result;
                for(int j=i+1; j<nIndex-1; j++)  numbers[j]=numbers[j+1];
                for(int j=i; j<oIndex-1; j++)  operators[j]=operators[j+1];
```

```
            if(operators[i]=='*' || operators[i]=='/'){
                i--;
            }
        }

        int result=numbers[0];
        int k=1;
        for(int i=0; i<oIndex; i++){
            if(operators[i]=='+'){
                steps.append(result).append(" + ").append(numbers[k]).append(" = ");
                result+=numbers[k];
                steps.append(result).append("\n");
            }
            else if(operators[i]=='-'){
                steps.append(result).append(" - ").append(numbers[k]).append(" = ");
                result-=numbers[k];
                steps.append(result).append("\n");
            }
            k++;
        }
        return result;
    }

    public static void main(String[] args){
        Scanner input=new Scanner(System.in);

        System.out.println("Text-Based Calculator");
        System.out.println("====================");
        System.out.print("Enter number of expressions: ");
        int n=input.nextInt();
        input.nextLine();

        for(int i=0; i<n; i++){
            System.out.print("\nEnter expression "+(i+1)+": ");
            String expr=input.nextLine();

            if(!validateExpression(expr)){
                System.out.println("Invalid expression format!");
                continue;
            }

            StringBuilder steps=new StringBuilder();
            steps.append("\nOriginal Expression: ").append(expr).append("\n");

            int result=evaluateExpression(expr,steps);
            steps.append("Final Result: ").append(result).append("\n");

            System.out.println("\nCalculation Steps:\n"+steps);
        }
        input.close();
    }
}
```

OUTPUT→

```
Text-Based Calculator
=====================
Enter number of expressions: 2

Enter expression 1: 10 * 3

Calculation Steps:

Original Expression: 10 * 3
10 * 3 = 30
Final Result: 30


Enter expression 2: 10 / 3

Calculation Steps:

Original Expression: 10 / 3
10 / 3 = 3
Final Result: 3
```

Problem 6: Write a program to create a simple text-based file organizer that categorizes and renames files based on their extensions and content analysis

Hint =>

a. Take user input for multiple file names with extensions

b. Create a method to extract file components without using split():

- **i. Use lastIndexOf() to find the last dot for extension**
- **ii. Extract filename and extension using substring()**
- **iii. Validate file name format and characters**
- **iv. Store file information in structured format**

c. Create a method to categorize files by extension:

- **i. Define categories (Documents: .txt, .doc; Images: .jpg, .png; etc.)**
- **ii. Use string comparison methods to match extensions**
- **iii. Count files in each category**
- **iv. Identify unknown file types**

d. Create a method using StringBuilder to generate new file names:

- **i. Create naming convention based on category and date**
- **ii. Handle duplicate names by adding numbers**
- **iii. Ensure generated names follow proper file naming rules**
- **iv. Validate that new names don't contain invalid characters**

e. Create a method to simulate content-based analysis:

- **i. For text files, analyze content for keywords**
- **ii. Suggest subcategories based on content (Resume, Report, Code, etc.)**
- **iii. Calculate file priority based on name patterns and content**
- **iv. Use ASCII values to validate content characters**

f. Create a method to display file organization report:

- **i. Show original filename, category, new suggested name**

- ii. Display category-wise file counts in tabular format
- iii. List files that need attention (invalid names, unknown types)
- iv. Show organization statistics and recommendations

g. Create a method to generate batch rename commands:

- i. Create command strings for renaming operations
- ii. Show before/after comparison
- iii. Calculate storage organization improvement

h. The main function processes file list and generates comprehensive organization plan with statistics

```java
import java.util.*;

public class CustomTextBasedFileOrganizer {

    // ==============================
    // Category mappings for extensions
    // ==============================
    private static final Map<String, String> categories = new HashMap<>();
    static {
        categories.put("txt", "Document");
        categories.put("doc", "Document");
        categories.put("jpg", "Image");
        categories.put("png", "Image");
        categories.put("pdf", "PDF");
        categories.put("java", "Code");
        categories.put("c", "Code");
    }

    // ------------------------------------------------
    // (b) Validate file name format (only letters, numbers,
    // underscores, hyphens and must contain an extension)
    // ------------------------------------------------
    public static boolean validateFileName(String file) {
        return file.matches("^[a-zA-Z0-9._-]+\\.[a-zA-Z0-9]+$");
    }

    // ------------------------------------------------
    // (b) Extract components using lastIndexOf() and substring()
    // ------------------------------------------------
    public static String[] extractComponents(String file) {
        int dotIndex = file.lastIndexOf('.');
        String name = file.substring(0, dotIndex);
        String ext = file.substring(dotIndex + 1).toLowerCase();
        return new String[]{name, ext};
    }

    // ------------------------------------------------
    // (c) Categorize files by extension
    // ------------------------------------------------
    public static String categorize(String ext) {
        return categories.getOrDefault(ext, "Unknown");
    }

    // ------------------------------------------------
    // (e) Simulate content-based analysis for .txt files
    // ------------------------------------------------
    public static String analyzeContent(String file) {
        if (!file.endsWith(".txt")) return "N/A";

        // Fake keyword-based detection (demo only)
        if (file.toLowerCase().contains("resume")) return "Resume";
        if (file.toLowerCase().contains("report")) return "Report";
        if (file.toLowerCase().contains("code")) return "Code";
```

```
        return "GeneralText";
    }

    // ------------------------------------------------
    // (d) Generate new filename using category + date + index
    // ------------------------------------------------
    public static String generateNewName(String category, String ext, int count) {
        StringBuilder sb = new StringBuilder();
        sb.append(category).append("_2025_").append(count).append(".").append(ext);
        return sb.toString();
    }

    // ------------------------------------------------
    // (f) Display file organization report
    // ------------------------------------------------
    public static void displayReport(List<String[]> files, Map<String, Integer> counts) {
        System.out.println("\n=== File Organization Report ===");
        System.out.printf("%-20s %-12s %-20s %-15s\n", "Original Name", "Category", "New Name", "Subcat
        System.out.println("---------------------------------------------------------------------------");

        for (String[] file : files) {
            System.out.printf("%-20s %-12s %-20s %-15s\n", file[0], file[1], file[2], file[3]);
        }

```


```
        return "GeneralText";
    }

    // ------------------------------------------------
    // (d) Generate new filename using category + date + index
    // ------------------------------------------------
    public static String generateNewName(String category, String ext, int count) {
        StringBuilder sb = new StringBuilder();
        sb.append(category).append("_2025_").append(count).append(".").append(ext);
        return sb.toString();
    }

    // ------------------------------------------------
    // (f) Display file organization report
    // ------------------------------------------------
    public static void displayReport(List<String[]> files, Map<String, Integer> counts) {
        System.out.println("\n=== File Organization Report ===");
        System.out.printf("%-20s %-12s %-20s %-15s\n", "Original Name", "Category", "New Name", "Subcat
        System.out.println("---------------------------------------------------------------------------");

        for (String[] file : files) {
            System.out.printf("%-20s %-12s %-20s %-15s\n", file[0], file[1], file[2], file[3]);
        }

```

```
        System.out.println("\n--- Category Counts ---");
        for (Map.Entry<String, Integer> entry : counts.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }

    // ------------------------------------------------
    // (g) Generate batch rename commands
    // ------------------------------------------------
    public static void batchRename(List<String[]> files) {
        System.out.println("\n--- Batch Rename Commands ---");
        for (String[] file : files) {
            System.out.println("rename " + file[0] + " → " + file[2]);
        }
    }

    // ------------------------------------------------
    // (h) Main function to process file list
    // ------------------------------------------------
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String[]> reportData = new ArrayList<>();
        Map<String, Integer> counts = new HashMap<>();

        System.out.print("Enter number of files: ");
        int n = sc.nextInt();
```

```
            String ext = parts[1];

            // c(i) Categorize
            String category = categorize(ext);
            counts.put(category, counts.getOrDefault(category, 0) + 1);

            // d(i) Generate new filename
            String newName = generateNewName(category, ext, counts.get(category));

            // e(i) Simulate content analysis
            String subCategory = analyzeContent(file);

            // Store report row
            reportData.add(new String[]{name, category, newName, subCategory});
        }

        // f(i) Display report
        displayReport(reportData, counts);

        // g(i) Show batch rename simulation
        batchRename(reportData);

        sc.close();
    }
}
```

OUTPUT→

```
Enter number of files: 2
Enter file name 1: text1.txt
 }
Enter number of files: 2
Enter file name 1: text1.txt
Enter file name 2: text2.txt
Enter number of files: 2
Enter file name 1: text1.txt
Enter file name 2: text2.txt

Enter file name 1: text1.txt
Enter file name 2: text2.txt

=== File Organization Report ===
Enter file name 2: text2.txt

=== File Organization Report ===

=== File Organization Report ===
Original Name        Category     New Name             Subcategory
=== File Organization Report ===
Original Name        Category     New Name             Subcategory
------------------------------------------------------------------------
Original Name        Category     New Name             Subcategory
------------------------------------------------------------------------
------------------------------------------------------------------------
text1.txt            Document     Document_2025_1.txt  GeneralText
text1.txt            Document     Document_2025_1.txt  GeneralText
text2.txt            Document     Document_2025_2.txt  GeneralText
text2.txt            Document     Document_2025_2.txt  GeneralText
```

```
--- Category Counts ---
Document: 2


--- Batch Rename Commands ---
--- Batch Rename Commands ---
rename text1.txt ? Document_2025_1.txt
rename text2.txt ? Document_2025_2.txt
```