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

1

- 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 SpellCheckerApp {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        try {
            String[] dictionary = {"java", "programming", "language",
"spell", "checker", "correct", "string", "distance"};
            System.out.println("Enter a sentence:");
            String inputSentence = scanner.nextLine();
            String[] words = extractWords(inputSentence);
            System.out.println("Word\tSuggestion\tDistance\tStatus");
            for (String word : words) {
                String suggestion = findClosestWord(word, dictionary);
                int distance = stringDistance(word.toLowerCase(),
```

```
suggestion.toLowerCase());
                String status = distance == 0 ? "Correct" : (distance <= 2
? "Misspelled" : "Unrecognized");
                System.out.println(word + "\t" + suggestion + "\t" +
distance + "\t" + status);
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }

    private static String[] extractWords(String sentence) {
        StringBuilder word = new StringBuilder();
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < sentence.length(); i++) {
            char c = sentence.charAt(i);
            if (Character.isLetter(c)) {
                word.append(c);
            } else {
                if (word.length() > 0) {
                    result.append(word).append(",");
                    word.setLength(0);
                }
            }
        }
        if (word.length() > 0) {
            result.append(word);
        }
        return result.toString().split(",");
    }

    private static int stringDistance(String a, String b) {
        int lenA = a.length();
        int lenB = b.length();
        int[][] dp = new int[lenA + 1][lenB + 1];
        for (int i = 0; i <= lenA; i++) {
            for (int j = 0; j <= lenB; j++) {
                if (i == 0) {
                    dp[i][j] = j;
                } else if (j == 0) {
                    dp[i][j] = i;
```

```
                } else {
                    int cost = a.charAt(i - 1) == b.charAt(j - 1) ? 0 : 1;
                    dp[i][j] = Math.min(Math.min(dp[i - 1][j] + 1, dp[i][j
- 1] + 1), dp[i - 1][j - 1] + cost);
                }
            }
        }
        return dp[lenA][lenB];
    }

    private static String findClosestWord(String word, String[] dictionary)
{
        String closest = word;
        int minDistance = Integer.MAX_VALUE;
        for (String dictWord : dictionary) {
            int distance = stringDistance(word.toLowerCase(),
dictWord.toLowerCase());
            if (distance < minDistance) {
                minDistance = distance;
                closest = dictWord;
            }
        }
        return closest;
    }
}
```

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:

2

● 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

```
import java.util.*;

public class PasswordAnalyzer {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of passwords:");
        int n = sc.nextInt(); sc.nextLine();
        String[] passwords = new String[n];
        for (int i = 0; i < n; i++) {
            System.out.println("Enter password " + (i+1) + ":");
            passwords[i] = sc.nextLine();
        }

        System.out.printf("%-15s %-7s %-10s %-10s %-10s %-10s %-7s
%-10s%n",
                "Password", "Length", "Uppercase", "Lowercase", "Digits",
"Special", "Score", "Strength");

        for (String pwd : passwords) {
            int[] counts = analyzePassword(pwd);
            int score = calculateScore(pwd, counts);
            String strength = getStrength(score);
            System.out.printf("%-15s %-7d %-10d %-10d %-10d %-10d %-7d
%-10s%n",
                    pwd, pwd.length(), counts[0], counts[1], counts[2],
counts[3], score, strength);
        }
```

```
        System.out.println("\nEnter desired length for strong password
generation:");
        int length = sc.nextInt();
        System.out.println("Generated strong password: " +
generateStrongPassword(length));
        sc.close();
    }

    public static int[] analyzePassword(String pwd) {
        int upper = 0, lower = 0, digit = 0, special = 0;
        for (int i = 0; i < pwd.length(); i++) {
            char ch = pwd.charAt(i);
            if (ch >= 65 && ch <= 90) upper++;
            else if (ch >= 97 && ch <= 122) lower++;
            else if (ch >= 48 && ch <= 57) digit++;
            else special++;
        }
        return new int[]{upper, lower, digit, special};
    }

    public static int calculateScore(String pwd, int[] counts) {
        int score = 0;
        if (pwd.length() > 8) score += (pwd.length() - 8) * 2;
        for (int c : counts) if (c > 0) score += 10;
        if (pwd.toLowerCase().contains("123") ||
pwd.toLowerCase().contains("abc") || pwd.toLowerCase().contains("qwerty"))
            score -= 10;
        return score;
    }

    public static String getStrength(int score) {
        if (score <= 20) return "Weak";
        else if (score <= 50) return "Medium";
        else return "Strong";
    }

    public static String generateStrongPassword(int length) {
        String upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        String lower = "abcdefghijklmnopqrstuvwxyz";
        String digits = "0123456789";
        String special = "!@#$%^&*()_-+=<>?";
        String all = upper + lower + digits + special;
        Random rand = new Random();
```

```
        StringBuilder sb = new StringBuilder();

        sb.append(upper.charAt(rand.nextInt(upper.length())));
        sb.append(lower.charAt(rand.nextInt(lower.length())));
        sb.append(digits.charAt(rand.nextInt(digits.length())));
        sb.append(special.charAt(rand.nextInt(special.length())));

        for (int i = 4; i < length; i++) {
            sb.append(all.charAt(rand.nextInt(all.length())));
        }

        List<Character> chars = new ArrayList<>();
        for (char c : sb.toString().toCharArray()) chars.add(c);
        Collections.shuffle(chars);
        StringBuilder shuffled = new StringBuilder();
        for (char c : chars) shuffled.append(c);
        return shuffled.toString();
    }
}
```

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

3

- 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 TextCompressor {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter text to compress:");
        String text = sc.nextLine();

        String compressed = compress(text);
        String decompressed = decompress(compressed);

        System.out.println("Original:    " + text);
        System.out.println("Compressed:  " + compressed);
        System.out.println("Decompressed:" + decompressed);
        sc.close();
    }

    public static String compress(String input) {
        if (input.isEmpty()) return "";
        StringBuilder sb = new StringBuilder();
        int count = 1;
        for (int i = 1; i < input.length(); i++) {
            if (input.charAt(i) == input.charAt(i - 1)) count++;
            else {
                sb.append(input.charAt(i - 1)).append(count);
                count = 1;
            }
        }
        sb.append(input.charAt(input.length() - 1)).append(count);
        return sb.toString();
    }
```

```
    public static String decompress(String input) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < input.length(); i++) {
            char ch = input.charAt(i);
            String num = "";
            while (i + 1 < input.length() &&
Character.isDigit(input.charAt(i + 1))) {
                num += input.charAt(++i);
            }
            int count = Integer.parseInt(num);
            for (int j = 0; j < count; j++) sb.append(ch);
        }
        return sb.toString();
    }
}
```

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:

4

- 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 class PalindromeFinder {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a paragraph:");
        String paragraph = sc.nextLine();

        String[] words = paragraph.split("[ ,.!?;:]+");
        System.out.printf("%-15s %-10s %-10s%n", "Palindrome", "Length",
"Frequency");
        System.out.println("-----------------------------------");

        for (int i = 0; i < words.length; i++) {
            String w = words[i].toLowerCase();
            if (isPalindrome(w)) {
                int freq = countFrequency(words, w);
                System.out.printf("%-15s %-10d %-10d%n", w, w.length(),
freq);
            }
        }
        sc.close();
    }

    public static boolean isPalindrome(String s) {
        if (s.length() < 2) return false;
        int i = 0, j = s.length() - 1;
        while (i < j) {
            if (s.charAt(i++) != s.charAt(j--)) return false;
        }
        return true;
    }
```

```
    public static int countFrequency(String[] arr, String word) {
        int count = 0;
        for (String w : arr) if (w.equalsIgnoreCase(word)) count++;
        return count;
    }
}
```

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.*;

public class AnagramGrouping {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of words:");
        int n = sc.nextInt(); sc.nextLine();
        String[] words = new String[n];
        for (int i = 0; i < n; i++) {
            words[i] = sc.nextLine();
        }

        Map<String, List<String>> groups = new HashMap<>();
        for (String word : words) {
            String key = sortString(word.toLowerCase());
            groups.putIfAbsent(key, new ArrayList<>());
            groups.get(key).add(word);
        }

        int groupNo = 1;
        for (List<String> group : groups.values()) {
            System.out.println("Group " + (groupNo++) + ": " + group);
        }
        sc.close();
    }

    public static String sortString(String s) {
        char[] arr = s.toCharArray();
        Arrays.sort(arr);
        return new String(arr);
    }
}
```

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

```
import java.util.*;

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

```
        System.out.println("Enter a paragraph:");
        String paragraph = sc.nextLine();

        String[] sentences = paragraph.split("[.!?]");
        String[] words = paragraph.split("[ ,.!?;:]+");

        int totalChars = paragraph.length();
        int totalWords = words.length;
        int totalSentences = sentences.length;
        String longest = getLongestWord(words);
        String shortest = getShortestWord(words);
        Map<String, Integer> freqMap = wordFrequency(words);

        System.out.println("Total Characters: " + totalChars);
        System.out.println("Total Words: " + totalWords);
        System.out.println("Total Sentences: " + totalSentences);
        System.out.println("Longest Word: " + longest);
        System.out.println("Shortest Word: " + shortest);

        System.out.println("\nWord Frequencies:");
        for (Map.Entry<String, Integer> e : freqMap.entrySet()) {
            System.out.println(e.getKey() + " : " + e.getValue());
        }

        System.out.println("\nSentence Lengths (in words):");
        for (String s : sentences) {
            String[] ws = s.trim().split("\\s+");
            if (ws.length > 0 && !ws[0].isEmpty()) {
                System.out.println("\"" + s.trim() + "\" -> " + ws.length);
            }
        }
        sc.close();
    }

    public static String getLongestWord(String[] words) {
        String longest = "";
        for (String w : words) if (w.length() > longest.length()) longest =
w;
        return longest;
    }

    public static String getShortestWord(String[] words) {
        String shortest = words[0];
```

```
        for (String w : words) if (w.length() < shortest.length()) shortest
= w;
        return shortest;
    }

    public static Map<String, Integer> wordFrequency(String[] words) {
        Map<String, Integer> map = new TreeMap<>();
        for (String w : words) {
            String lw = w.toLowerCase();
            map.put(lw, map.getOrDefault(lw, 0) + 1);
        }
        return map;
    }
}
```