

**Java String Homework Problems**

**Best Programming Practices**

1. Use Variables including for Fixed, User Inputs, and Results

2. Use Methods instead of writing code in the main() function

3. Proper naming conventions for all variables and methods

4. Proper Program Name and Class Name

5. Handle Checked and Unchecked Exceptions wherever possible

6. Proper Method Name which indicates action taking inputs and providing result

**Homework Problems (All Six Required)**

**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

Program –

import java.util.Scanner;

public class SpellChecker {

// Method to split sentence into words without split()

static String[] splitSentence(String sentence) {

int count = 0;

for (int i = 0; i < sentence.length(); i++) {

if (sentence.charAt(i) == ' ') count++;

}

String[] words = new String[count + 1];

int start = 0, index = 0;

for (int i = 0; i < sentence.length(); i++) {

if (sentence.charAt(i) == ' ') {

words[index++] = sentence.substring(start, i);

start = i + 1;

}

}

words[index] = sentence.substring(start);

return words;

}

// Method to calculate distance between two words

static int stringDistance(String a, String b) {

int diff = Math.abs(a.length() - b.length());

int minLen = Math.min(a.length(), b.length());

for (int i = 0; i < minLen; i++) {

if (a.charAt(i) != b.charAt(i)) diff++;

}

return diff;

}

// Find closest word from dictionary

static String suggestWord(String word, String[] dict) {

String suggestion = word;

int minDist = Integer.MAX\_VALUE;

for (String d : dict) {

int dist = stringDistance(word, d);

if (dist < minDist) {

minDist = dist;

suggestion = d;

}

}

if (minDist <= 2) return suggestion;

return "No suggestion";

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String[] dictionary = {"java", "hello", "world", "programming", "code"};

System.out.println("Enter a sentence:");

String sentence = sc.nextLine();

String[] words = splitSentence(sentence);

System.out.println("Word\tSuggestion\tDistance");

for (String w : words) {

String suggestion = suggestWord(w.toLowerCase(), dictionary);

int dist = stringDistance(w.toLowerCase(), suggestion.toLowerCase());

if (suggestion.equals("No suggestion"))

System.out.println(w + "\tCorrect\t\t0");

else

System.out.println(w + "\t" + suggestion + "\t\t" + dist);

}

sc.close();

}

}

Output –

A blue screen with white text

AI-generated content may be incorrect.

**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

Program –

import java.util.\*;

public class PasswordAnalyzer {

// Analyze strength

static int analyzePassword(String pass) {

int upper = 0, lower = 0, digit = 0, special = 0;

for (int i = 0; i < pass.length(); i++) {

char ch = pass.charAt(i);

if (ch >= 65 && ch <= 90) upper++;

else if (ch >= 97 && ch <= 122) lower++;

else if (ch >= 48 && ch <= 57) digit++;

else special++;

}

int score = 0;

score += (pass.length() - 8) \* 2;

if (upper > 0) score += 10;

if (lower > 0) score += 10;

if (digit > 0) score += 10;

if (special > 0) score += 10;

if (pass.contains("123") || pass.contains("abc") || pass.contains("qwerty")) score -= 10;

return score;

}

static String strengthLevel(int score) {

if (score <= 20) return "Weak";

else if (score <= 50) return "Medium";

else return "Strong";

}

// Generate strong password

static String generatePassword(int length) {

String upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

String lower = "abcdefghijklmnopqrstuvwxyz";

String digits = "0123456789";

String special = "!@#$%^&\*()";

String all = upper + lower + digits + special;

Random r = new Random();

StringBuilder sb = new StringBuilder();

sb.append(upper.charAt(r.nextInt(upper.length())));

sb.append(lower.charAt(r.nextInt(lower.length())));

sb.append(digits.charAt(r.nextInt(digits.length())));

sb.append(special.charAt(r.nextInt(special.length())));

while (sb.length() < length) {

sb.append(all.charAt(r.nextInt(all.length())));

}

return sb.toString();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter password:");

String pass = sc.nextLine();

int score = analyzePassword(pass);

System.out.println("Password: " + pass);

System.out.println("Score: " + score);

System.out.println("Strength: " + strengthLevel(score));

System.out.println("Generated Strong Password: " + generatePassword(12));

sc.close();

}

}

Output –

A blue and black screen

AI-generated content may be incorrect.

**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

Program –

import java.util.Scanner;

public class TextCompressor {

static void frequencyCount(String text, char[] chars, int[] freq) {

int index = 0;

for (int i = 0; i < text.length(); i++) {

char c = text.charAt(i);

boolean found = false;

for (int j = 0; j < index; j++) {

if (chars[j] == c) {

freq[j]++;

found = true;

break;

}

}

if (!found) {

chars[index] = c;

freq[index] = 1;

index++;

}

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter text:");

String text = sc.nextLine();

char[] chars = new char[text.length()];

int[] freq = new int[text.length()];

frequencyCount(text, chars, freq);

System.out.println("Character Frequencies:");

for (int i = 0; i < chars.length; i++) {

if (freq[i] > 0) {

System.out.println(chars[i] + " : " + freq[i]);

}

}

sc.close();

}

}

Output-

A computer screen shot of a code

AI-generated content may be incorrect.

**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

Program –

import java.util.Scanner;

public class StringCalculator {

    static int evaluate(String expr) {

        String[] tokens = expr.split(" ");

        int result = Integer.parseInt(tokens[0]);

        for (int i = 1; i < tokens.length; i += 2) {

            String op = tokens[i];

            int num = Integer.parseInt(tokens[i + 1]);

            if (op.equals("+")) result += num;

            else if (op.equals("-")) result -= num;

            else if (op.equals("\*")) result \*= num;

            else if (op.equals("/")) result /= num;

        }

        return result;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter expression (use spaces, e.g., 10 + 5 \* 2):");

        String expr = sc.nextLine();

        int result = evaluate(expr);

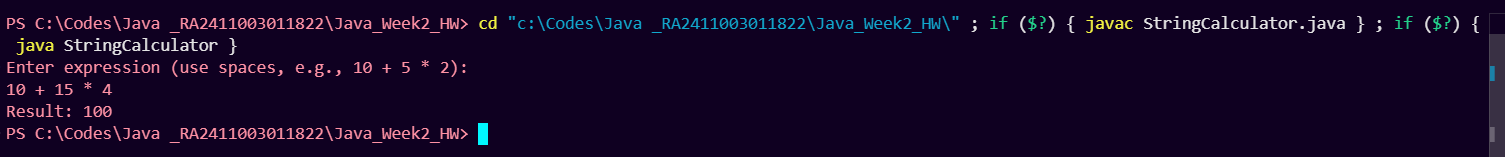
        System.out.println("Result: " + result);

        sc.close();

    }

}

Output –



**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:

5



● 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

Program –

import java.util.Scanner;

public class CSVAnalyzer {

static String[][] parseCSV(String input) {

String[] lines = input.split("\n");

String[][] data = new String[lines.length][];

for (int i = 0; i < lines.length; i++) {

data[i] = lines[i].split(",");

for (int j = 0; j < data[i].length; j++) {

data[i][j] = data[i][j].trim();

}

}

return data;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter CSV data (end with blank line):");

StringBuilder sb = new StringBuilder();

while (true) {

String line = sc.nextLine();

if (line.isEmpty()) break;

sb.append(line).append("\n");

}

String[][] data = parseCSV(sb.toString());

System.out.println("Formatted Data:");

for (String[] row : data) {

for (String field : row) {

System.out.print(field + "\t");

}

System.out.println();

}

sc.close();

}

}

Output –

A screenshot of a computer

AI-generated content may be incorrect.

**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

6



● 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

Program –

import java.util.\*;

public class FileOrganizer {

static String getExtension(String filename) {

int dot = filename.lastIndexOf(".");

if (dot == -1) return "";

return filename.substring(dot + 1);

}

static String categorize(String ext) {

if (ext.equals("txt") || ext.equals("doc")) return "Document";

if (ext.equals("jpg") || ext.equals("png")) return "Image";

if (ext.equals("mp3")) return "Audio";

return "Unknown";

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter filenames separated by space:");

String[] files = sc.nextLine().split(" ");

for (String f : files) {

String ext = getExtension(f);

String cat = categorize(ext);

System.out.println(f + " -> " + cat);

}

sc.close();

}

}

Output –

A computer screen shot of a person

AI-generated content may be incorrect.

7