

**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

SOLUTION:

import java.util.\*;

public class SpellChecker {

// Split sentence into words without using split()

public static List<String> getWords(String text) {

List<String> words = new ArrayList<>();

int start = 0;

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

char c = text.charAt(i);

if (c == ' ' || c == '.' || c == ',' || c == '!' || c == '?' || c == ';') {

if (start != i) {

words.add(text.substring(start, i));

}

start = i + 1;

}

}

if (start < text.length()) {

words.add(text.substring(start));

}

return words;

}

// Calculate string distance (Levenshtein-like simplified version)

public 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++) dp[i][0] = i;

for (int j = 0; j <= lenB; j++) dp[0][j] = j;

for (int i = 1; i <= lenA; i++) {

for (int j = 1; j <= lenB; j++) {

if (a.charAt(i - 1) == b.charAt(j - 1)) {

dp[i][j] = dp[i - 1][j - 1];

} else {

dp[i][j] = 1 + Math.min(dp[i - 1][j - 1],

Math.min(dp[i - 1][j], dp[i][j - 1]));

}

}

}

return dp[lenA][lenB];

}

// Find closest word from dictionary

public static String getClosestMatch(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 (minDistance <= 2) ? closest : "No Suggestion";

}

// Display results in tabular format

public static void displayResults(List<String> words, String[] dictionary) {

System.out.println("\n================ SPELL CHECK REPORT ================");

System.out.printf("%-15s %-20s %-10s %-12s\n", "Original", "Suggestion", "Distance", "Status");

System.out.println("-----------------------------------------------------");

for (String word : words) {

String suggestion = getClosestMatch(word, dictionary);

int distance = suggestion.equals("No Suggestion") ? -1 :

stringDistance(word.toLowerCase(), suggestion.toLowerCase());

String status = suggestion.equals(word) ? "Correct" :

suggestion.equals("No Suggestion") ? "Unknown" : "Misspelled";

System.out.printf("%-15s %-20s %-10s %-12s\n",

word, suggestion, (distance == -1 ? "--" : distance), status);

}

System.out.println("=====================================================");

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String[] dictionary = {

"java", "programming", "language", "is", "very", "powerful",

"simple", "used", "worldwide", "computer", "science", "project"

};

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

String sentence = sc.nextLine();

List<String> words = getWords(sentence);

displayResults(words, dictionary);

sc.close();

}

}

OUTPUT:

Enter a sentence: my name is xyz

================ SPELL CHECK REPORT ================

Original Suggestion Distance Status

-----------------------------------------------------

my is 2 Misspelled

name No Suggestion -- Unknown

is is 0 Correct

xyz No Suggestion -- Unknown

**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

SOLUTION:

import java.util.\*;

public class PasswordAnalyzer {

// Analyze password using ASCII values

public static int[] analyzePassword(String password) {

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

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

char c = password.charAt(i);

int ascii = (int) c;

if (ascii >= 65 && ascii <= 90) upper++; // Uppercase

else if (ascii >= 97 && ascii <= 122) lower++; // Lowercase

else if (ascii >= 48 && ascii <= 57) digits++; // Digits

else if (ascii >= 33 && ascii <= 126) special++; // Special chars

}

return new int[]{upper, lower, digits, special};

}

// Check for common weak patterns

public static boolean hasCommonPattern(String password) {

String lowerPwd = password.toLowerCase();

String[] weakPatterns = {"123", "abc", "qwerty", "password", "admin"};

for (String pattern : weakPatterns) {

if (lowerPwd.contains(pattern)) return true;

}

return false;

}

// Calculate password strength score

public static int calculateScore(String password, int upper, int lower, int digits, int special) {

int score = 0;

// Length-based points

if (password.length() > 8) score += (password.length() - 8) \* 2;

// Variety-based points

if (upper > 0) score += 10;

if (lower > 0) score += 10;

if (digits > 0) score += 10;

if (special > 0) score += 10;

// Deduct points for common patterns

if (hasCommonPattern(password)) score -= 15;

return Math.max(score, 0);

}

// Return password strength level

public static String getStrengthLevel(int score) {

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

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

else return "Strong";

}

// Generate strong random password using StringBuilder

public static String generatePassword(int length) {

if (length < 8) length = 8;

String upperChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

String lowerChars = "abcdefghijklmnopqrstuvwxyz";

String digits = "0123456789";

String specials = "!@#$%^&\*()-\_=+[]{}|;:,.<>?";

String allChars = upperChars + lowerChars + digits + specials;

Random random = new Random();

StringBuilder sb = new StringBuilder();

sb.append(upperChars.charAt(random.nextInt(upperChars.length())));

sb.append(lowerChars.charAt(random.nextInt(lowerChars.length())));

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

sb.append(specials.charAt(random.nextInt(specials.length())));

for (int i = 4; i < length; i++) {

sb.append(allChars.charAt(random.nextInt(allChars.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();

}

public static void displayResults(List<String> passwords) {

System.out.println("\n================ PASSWORD STRENGTH ANALYSIS =================");

System.out.printf("%-20s %-8s %-8s %-8s %-8s %-10s %-8s %-10s\n",

"Password", "Length", "Upper", "Lower", "Digits", "Special", "Score", "Strength");

System.out.println("--------------------------------------------------------------------------");

for (String pwd : passwords) {

int[] counts = analyzePassword(pwd);

int upper = counts[0], lower = counts[1], digits = counts[2], special = counts[3];

int score = calculateScore(pwd, upper, lower, digits, special);

String strength = getStrengthLevel(score);

System.out.printf("%-20s %-8d %-8d %-8d %-8d %-10d %-8d %-10s\n",

pwd, pwd.length(), upper, lower, digits, special, score, strength);

}

System.out.println("==========================================================================");

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of passwords to analyze: ");

int n = sc.nextInt();

sc.nextLine();

List<String> passwords = new ArrayList<>();

for (int i = 1; i <= n; i++) {

System.out.print("Enter password " + i + ": ");

passwords.add(sc.nextLine());

}

displayResults(passwords);

System.out.print("\nEnter desired length for a strong password: ");

int length = sc.nextInt();

String strongPassword = generatePassword(length);

System.out.println("\nGenerated Strong Password: " + strongPassword);

sc.close();

}

}

OUTPUT:Enter number of passwords to analyze: y\*7@P!

Exception in thread "main" java.util.InputMismatchException

at java.base/java.util.Scanner.throwFor(Scanner.java:964)

at java.base/java.util.Scanner.next(Scanner.java:1619)

at java.base/java.util.Scanner.nextInt(Scanner.java:2284)

at java.base/java.util.Scanner.nextInt(Scanner.java:2238)

at PasswordAnalyzer.main(PasswordAnalyzer.java:110)

**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

SOLUTION:

import java.util.Scanner;

public class TextCompression {

static char[] uniqueChars;

static int[] freq;

static String[] codes;

static void countCharFrequency(String text) {

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

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

int uniqueCount = 0;

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

char ch = text.charAt(i);

int index = -1;

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

if (uniqueChars[j] == ch) {

index = j;

break;

}

}

if (index == -1) {

uniqueChars[uniqueCount] = ch;

freq[uniqueCount] = 1;

uniqueCount++;

} else {

freq[index]++;

}

}

char[] tempChars = new char[uniqueCount];

int[] tempFreq = new int[uniqueCount];

System.arraycopy(uniqueChars, 0, tempChars, 0, uniqueCount);

System.arraycopy(freq, 0, tempFreq, 0, uniqueCount);

uniqueChars = tempChars;

freq = tempFreq;

}

static void generateCompressionCodes() {

codes = new String[uniqueChars.length];

int[] sortedIndices = new int[uniqueChars.length];

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

sortedIndices[i] = i;

}

for (int i = 0; i < sortedIndices.length - 1; i++) {

for (int j = i + 1; j < sortedIndices.length; j++) {

if (freq[sortedIndices[i]] < freq[sortedIndices[j]]) {

int temp = sortedIndices[i];

sortedIndices[i] = sortedIndices[j];

sortedIndices[j] = temp;

}

}

}

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

codes[sortedIndices[i]] = Integer.toBinaryString(i + 1);

}

}

static String compressText(String text) {

StringBuilder compressed = new StringBuilder();

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

char ch = text.charAt(i);

for (int j = 0; j < uniqueChars.length; j++) {

if (uniqueChars[j] == ch) {

compressed.append(codes[j]).append(" ");

break;

}

}

}

return compressed.toString().trim();

}

static String decompressText(String compressed) {

StringBuilder decompressed = new StringBuilder();

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

for (String token : tokens) {

for (int j = 0; j < codes.length; j++) {

if (codes[j].equals(token)) {

decompressed.append(uniqueChars[j]);

break;

}

}

}

return decompressed.toString();

}

static void displayCompressionAnalysis(String text, String compressed, String decompressed) {

System.out.println("\n--- CHARACTER FREQUENCY TABLE ---");

System.out.printf("%-10s%-10s%-10s\n", "Char", "Freq", "Code");

System.out.println("----------------------------------");

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

System.out.printf("%-10s%-10d%-10s\n", uniqueChars[i], freq[i], codes[i]);

}

System.out.println("\n--- COMPRESSION RESULTS ---");

System.out.println("Original Text : " + text);

System.out.println("Compressed Text : " + compressed);

System.out.println("Decompressed Text : " + decompressed);

double originalSize = text.length() \* 8;

double compressedSize = compressed.replace(" ", "").length();

double efficiency = ((originalSize - compressedSize) / originalSize) \* 100;

System.out.println("Original Size (bits): " + (int) originalSize);

System.out.println("Compressed Size(bits): " + (int) compressedSize);

System.out.printf("Compression Efficiency: %.2f%%\n", efficiency);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter text to compress: ");

String text = sc.nextLine();

countCharFrequency(text);

generateCompressionCodes();

String compressed = compressText(text);

String decompressed = decompressText(compressed);

displayCompressionAnalysis(text, compressed, decompressed);

sc.close();

}

}

OUTPUT:Enter text to compress: MyNameIsABC

--- CHARACTER FREQUENCY TABLE ---

Char Freq Code

----------------------------------

M 1 1

y 1 10

N 1 11

a 1 100

m 1 101

e 1 110

I 1 111

s 1 1000

A 1 1001

B 1 1010

C 1 1011

--- COMPRESSION RESULTS ---

Original Text : MyNameIsABC

Compressed Text : 1 10 11 100 101 110 111 1000 1001 1010 1011

Decompressed Text : MyNameIsABC

Original Size (bits): 88

Compressed Size(bits): 33

Compression Efficiency: 62.50%

**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

SOLUTION:

import java.util.\*;

public class ExpressionCalculator {

public static boolean validateExpression(String expr) {

try {

List<String> tokens = tokenize(expr);

int balance = 0;

for (int i = 0; i < tokens.size(); i++) {

String t = tokens.get(i);

if (t.equals("(")) balance++;

if (t.equals(")")) balance--;

if (balance < 0) return false;

}

if (balance != 0) return false;

if (tokens.size() == 0) return false;

String last = tokens.get(tokens.size() - 1);

if (isOperator(last) || last.equals("(")) return false;

return true;

} catch (Exception e) {

return false;

}

}

public static List<String> tokenize(String expr) {

List<String> tokens = new ArrayList<>();

int i = 0;

int n = expr.length();

while (i < n) {

char c = expr.charAt(i);

int ascii = (int) c;

if (ascii == 32) { i++; continue; }

if ((ascii >= 48 && ascii <= 57) || c == '.') {

int j = i;

while (j < n) {

char d = expr.charAt(j);

int ad = (int) d;

if ((ad >= 48 && ad <= 57) || d == '.') j++;

else break;

}

tokens.add(expr.substring(i, j));

i = j;

continue;

}

if (c == '+' || c == '\*' || c == '/' || c == '(' || c == ')') {

tokens.add(String.valueOf(c));

i++;

continue;

}

if (c == '-') {

boolean unary = tokens.isEmpty() || tokens.get(tokens.size() - 1).equals("(") || isOperator(tokens.get(tokens.size() - 1));

if (unary) {

if (i + 1 < n) {

char next = expr.charAt(i + 1);

int an = (int) next;

if ((an >= 48 && an <= 57) || next == '.') {

int j = i + 1;

while (j < n) {

char d = expr.charAt(j);

int ad = (int) d;

if ((ad >= 48 && ad <= 57) || d == '.') j++;

else break;

}

tokens.add(expr.substring(i, j));

i = j;

continue;

} else if (next == '(') {

tokens.add("0");

tokens.add("-");

i++;

continue;

} else {

tokens.add("-");

i++;

continue;

}

} else {

tokens.add("-");

i++;

continue;

}

} else {

tokens.add("-");

i++;

continue;

}

}

throw new IllegalArgumentException("Invalid character: " + c);

}

return tokens;

}

static boolean isOperator(String s) {

return s.equals("+") || s.equals("-") || s.equals("\*") || s.equals("/");

}

static String joinTokens(List<String> tokens) {

StringBuilder sb = new StringBuilder();

for (int i = 0; i < tokens.size(); i++) {

if (i > 0) sb.append(" ");

sb.append(tokens.get(i));

}

return sb.toString();

}

static String formatNumber(double v) {

double rounded = Math.rint(v);

if (Math.abs(v - rounded) < 1e-9) return String.valueOf((long) rounded);

return String.valueOf(v);

}

public static String evaluateExpression(String expr, StringBuilder steps) throws Exception {

List<String> tokens = tokenize(expr);

steps.append("Original: ").append(expr).append("\n");

while (tokens.contains("(")) {

int open = -1;

for (int i = 0; i < tokens.size(); i++) if (tokens.get(i).equals("(")) open = i;

if (open == -1) break;

int close = -1;

for (int j = open + 1; j < tokens.size(); j++) if (tokens.get(j).equals(")")) { close = j; break; }

if (close == -1) throw new IllegalArgumentException("Mismatched parentheses");

List<String> sub = new ArrayList<>();

for (int k = open + 1; k < close; k++) sub.add(tokens.get(k));

String subResult = evaluateFlat(sub, steps);

tokens.set(open, subResult);

for (int k = 0; k < close - open; k++) tokens.remove(open + 1);

steps.append("After evaluating parentheses -> ").append(joinTokens(tokens)).append("\n");

}

String finalRes = evaluateFlat(tokens, steps);

steps.append("Final Result: ").append(finalRes).append("\n");

return finalRes;

}

static String evaluateFlat(List<String> tokens, StringBuilder steps) throws Exception {

List<String> t = new ArrayList<>(tokens);

int i = 0;

while (i < t.size()) {

String op = t.get(i);

if (op.equals("\*") || op.equals("/")) {

double a = Double.parseDouble(t.get(i - 1));

double b = Double.parseDouble(t.get(i + 1));

double r;

if (op.equals("\*")) r = a \* b;

else {

if (Math.abs(b) < 1e-12) throw new ArithmeticException("Division by zero");

r = a / b;

}

String rStr = formatNumber(r);

t.set(i - 1, rStr);

t.remove(i);

t.remove(i);

steps.append("Compute ").append(a).append(" ").append(op).append(" ").append(b).append(" = ").append(rStr).append(" => ").append(joinTokens(t)).append("\n");

i = Math.max(i - 1, 0);

} else i++;

}

i = 0;

while (i < t.size()) {

String op = t.get(i);

if (op.equals("+") || op.equals("-")) {

double a = Double.parseDouble(t.get(i - 1));

double b = Double.parseDouble(t.get(i + 1));

double r = op.equals("+") ? a + b : a - b;

String rStr = formatNumber(r);

t.set(i - 1, rStr);

t.remove(i);

t.remove(i);

steps.append("Compute ").append(a).append(" ").append(op).append(" ").append(b).append(" = ").append(rStr).append(" => ").append(joinTokens(t)).append("\n");

i = Math.max(i - 1, 0);

} else i++;

}

if (t.size() == 0) throw new IllegalStateException("Empty expression");

return t.get(0);

}

public static void displayCalculation(String expr) {

StringBuilder steps = new StringBuilder();

try {

if (!validateExpression(expr)) {

System.out.println("Expression invalid: " + expr);

return;

}

String result = evaluateExpression(expr, steps);

System.out.println("\nExpression: " + expr);

System.out.println("Steps:");

System.out.println(steps.toString());

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

} catch (ArithmeticException ae) {

System.out.println("Error while evaluating '" + expr + "': " + ae.getMessage());

} catch (Exception e) {

System.out.println("Error while evaluating '" + expr + "': " + e.getMessage());

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of expressions to evaluate: ");

int n = 1;

try { n = Integer.parseInt(sc.nextLine().trim()); } catch (Exception e) { n = 1; }

for (int i = 1; i <= n; i++) {

System.out.print("Enter expression " + i + ": ");

String expr = sc.nextLine();

displayCalculation(expr);

}

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

**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

7