Aishwarya TJ

RA2411033010131

1. Code

```
import java.util.*;
public class SpellChecker {
        public static String[] splitSentence(String sentence) {
      ArrayList<String> words = new ArrayList<>();
      StringBuilder word = new StringBuilder();

        for (int i = 0; i < sentence.length(); i++) {
        char ch = sentence.charAt(i);
        if (Character.isLetter(ch)) {
            word.append(ch);
        } else if (word.length() > 0) {
            words.add(word.toString());
            word.setLength(0);
        }
        }
        if (word.length() > 0) {
         words.add(word.toString());
        }
        return words.toArray(new String[0]);
```

```
    }
    public static int calculateStringDistance(String word1, String word2) {
    int len1 = word1.length();
    int len2 = word2.length();

    if (word1.equals(word2)) return 0;
    int[][] dp = new int[len1 + 1][len2 + 1];
    for (int i = 0; i <= len1; i++) dp[i][0] = i;
    for (int j = 0; j <= len2; j++) dp[0][j] = j;
    for (int i = 1; i <= len1; i++) {
    for (int j = 1; j <= len2; j++) {
        int cost = (word1.charAt(i - 1) == word2.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[len1][len2];
    }
    public static String findClosestMatch(String word, String[] dictionary) {
    String closestMatch = null;
    int minDistance = Integer.MAX_VALUE;
    for (String dictWord : dictionary) {
    int distance = calculateStringDistance(word, dictWord);
    if (distance < minDistance) {
```

```
            minDistance = distance;
            closestMatch = dictWord;
        }
        }
        return closestMatch;
        }

        public static void displaySpellCheckResults(String[] sentenceWords, String[]
dictionary) {
      System.out.printf("%-20s%-20s%-10s\n", "Original Word", "Suggested Correction",
"Distance");
      System.out.println("-------------------------------------------------------");

        for (String word : sentenceWords) {
        int distance = Integer.MAX_VALUE;
         String suggestedCorrection = "Correct";

        for (String dictWord : dictionary) {
            int currentDistance = calculateStringDistance(word, dictWord);
            if (currentDistance < distance) {
                distance = currentDistance;
                suggestedCorrection = currentDistance <= 2 ? dictWord : "Correct";
            }
        }
```

```
            System.out.printf("%-20s%-20s%-10d\n", word, suggestedCorrection, distance);
        }
        }
        public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String[] dictionary = {"apple", "banana", "orange", "grape", "melon", "pear",
"peach"};
      System.out.println("Enter a sentence for spell check:");
        String sentence = scanner.nextLine();
        String[] sentenceWords = splitSentence(sentence);

      displaySpellCheckResults(sentenceWords, dictionary);
      scanner.close();
        }
}
```

2. Code

```
import java.util.*;
import java.security.SecureRandom;
public class PasswordAnalyzerGenerator {
        public static int analyzePasswordStrength(String password) {
        int score = 0;
```

```
        int upperCaseCount = 0, lowerCaseCount = 0, digitCount = 0, specialCharCount
= 0;

        boolean hasCommonPattern = false;

        String[] commonPatterns = {"123", "abc", "qwerty", "password", "letmein"};



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

        char ch = password.charAt(i);

        if (ch >= 'A' && ch <= 'Z') upperCaseCount++;

        else if (ch >= 'a' && ch <= 'z') lowerCaseCount++;

        else if (ch >= '0' && ch <= '9') digitCount++;

        else if ((ch >= 32 && ch <= 47) || (ch >= 58 && ch <= 64) || (ch >= 91 && ch <=
96) || (ch >= 123 && ch <= 126)) specialCharCount++;

        }



        for (String pattern : commonPatterns) {

        if (password.toLowerCase().contains(pattern)) {

            hasCommonPattern = true;

            break;

        }

        }

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

        if (upperCaseCount > 0) score += 10;

        if (lowerCaseCount > 0) score += 10;

        if (digitCount > 0) score += 10;
```

```
    if (specialCharCount > 0) score += 10;
    if (hasCommonPattern) score -= 20;
    return score;
    }
    public static String passwordStrength(int score) {
    if (score >= 51) return "Strong";
    else if (score >= 21) return "Medium";
    else return "Weak";
    }
    public static String generateStrongPassword(int length) {
    if (length < 8) length = 8;
   SecureRandom random = new SecureRandom();
   StringBuilder password = new StringBuilder();

    String upperCaseChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    String lowerCaseChars = "abcdefghijklmnopqrstuvwxyz";
    String digits = "0123456789";
    String specialChars = "!@#$%^&*()-_=+[]{}|;:,.<>?/";

password.append(upperCaseChars.charAt(random.nextInt(upperCaseChars.length())));

password.append(lowerCaseChars.charAt(random.nextInt(lowerCaseChars.length())));
    password.append(digits.charAt(random.nextInt(digits.length())));
```

```
        password.append(specialChars.charAt(random.nextInt(specialChars.length())));
          String allChars = upperCaseChars + lowerCaseChars + digits + specialChars;
          for (int i = 4; i < length; i++) {
           password.append(allChars.charAt(random.nextInt(allChars.length())));
          }
        List<Character> passwordList = new ArrayList<>();
          for (int i = 0; i < password.length(); i++) {
           passwordList.add(password.charAt(i));
          }
        Collections.shuffle(passwordList);
        StringBuilder shuffledPassword = new StringBuilder();
          for (char c : passwordList) {
           shuffledPassword.append(c);
          }
          return shuffledPassword.toString();
          }
          public static void displayPasswordAnalysis(String[] passwords) {
        System.out.printf("%-20s%-10s%-15s%-15s%-10s%-15s%-10s%-10s\n",
                          "Password", "Length", "Uppercase", "Lowercase", "Digits", "Special
Chars", "Score", "Strength");
        System.out.println("--------------------------------------------------------------------------------");

          for (String password : passwords) {
          int score = analyzePasswordStrength(password);
```

```
         String strength = passwordStrength(score);

        int upperCaseCount = 0, lowerCaseCount = 0, digitCount = 0, specialCharCount
= 0;

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

            char ch = password.charAt(i);

            if (ch >= 'A' && ch <= 'Z') upperCaseCount++;

            else if (ch >= 'a' && ch <= 'z') lowerCaseCount++;

            else if (ch >= '0' && ch <= '9') digitCount++;

            else if ((ch >= 32 && ch <= 47) || (ch >= 58 && ch <= 64) || (ch >= 91 && ch
<= 96) || (ch >= 123 && ch <= 126)) specialCharCount++;

        }

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

                           password, password.length(), upperCaseCount, lowerCaseCount,
digitCount, specialCharCount, score, strength);

        }

        }

        public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

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

        int n = scanner.nextInt();

       scanner.nextLine();

        String[] passwords = new String[n];


       System.out.println("Enter the passwords:");

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

```
            passwords[i] = scanner.nextLine();
          }
        displayPasswordAnalysis(passwords);

        System.out.println("\nGenerate a strong password:");
        System.out.println("Enter desired password length:");
          int length = scanner.nextInt();

          String strongPassword = generateStrongPassword(length);
        System.out.println("Generated Strong Password: " + strongPassword);
        scanner.close();
          }
}
```

3. Code

```
import java.util.*;
public class TextCompression {
          public static char[] getUniqueChars(String text) {
        StringBuilder uniqueChars = new StringBuilder();
          for (int i = 0; i < text.length(); i++) {
          char ch = text.charAt(i);
          if (uniqueChars.indexOf(String.valueOf(ch)) == -1) {
```

```
            uniqueChars.append(ch);
        }
        }
        return uniqueChars.toString().toCharArray();
        }
        public static int[] countCharacterFrequency(String text, char[] uniqueChars) {
        int[] frequencies = new int[uniqueChars.length];
        for (int i = 0; i < text.length(); i++) {
        char ch = text.charAt(i);
        for (int j = 0; j < uniqueChars.length; j++) {
            if (ch == uniqueChars[j]) {
                frequencies[j]++;
                break;
            }
        }
        }
        return frequencies;
        }
        public static String[][] createCompressionCodes(char[] uniqueChars, int[]
frequencies) {
        int n = uniqueChars.length;
       String[][] mapping = new String[n][2];
        int[] sortedIndices = new int[n];
        for (int i = 0; i < n; i++) {
```

```
        sortedIndices[i] = i;
      }
    Arrays.sort(sortedIndices, (i1, i2) -> Integer.compare(frequencies[i2],
frequencies[i1]));

    StringBuilder code = new StringBuilder();
      for (int i = 0; i < n; i++) {
      int index = sortedIndices[i];
       code.setLength(0);
      for (int j = 0; j < i; j++) {
          code.append("0");
      }
       mapping[i][0] = String.valueOf(uniqueChars[index]);
       mapping[i][1] = code.toString();
      }
      return mapping;
      }
      public static String compressText(String text, String[][] mapping) {
    StringBuilder compressedText = new StringBuilder();
      for (int i = 0; i < text.length(); i++) {
      for (int j = 0; j < mapping.length; j++) {
          if (text.charAt(i) == mapping[j][0].charAt(0)) {
              compressedText.append(mapping[j][1]);
              break;
```

```
            }
        }
        }
        return compressedText.toString();
        }
        public static String decompressText(String compressedText, String[][] mapping) {
    StringBuilder decompressedText = new StringBuilder();
    StringBuilder code = new StringBuilder();
    Map<String, String> reverseMapping = new HashMap<>();
        for (String[] entry : mapping) {
         reverseMapping.put(entry[1], entry[0]);
        }
        for (int i = 0; i < compressedText.length(); i++) {
         code.append(compressedText.charAt(i));
        if (reverseMapping.containsKey(code.toString())) {
            decompressedText.append(reverseMapping.get(code.toString()));
            code.setLength(0);
        }
        }
        return decompressedText.toString();
        }
        public static double calculateCompressionRatio(String originalText, String
compressedText) {
        return ((double) compressedText.length() / originalText.length()) * 100;
```

```
    }

    public static void displayCompressionAnalysis(String text, String
compressedText, String decompressedText, String[][] mapping) {
        System.out.println("Character Frequency Table:");
        System.out.printf("%-10s%-10s\n", "Character", "Frequency");
        System.out.println("----------------------");
        char[] uniqueChars = getUniqueChars(text);
        int[] frequencies = countCharacterFrequency(text, uniqueChars);
        for (int i = 0; i < uniqueChars.length; i++) {
            System.out.printf("%-10c%-10d\n", uniqueChars[i], frequencies[i]);
        }
        System.out.println("\nCompression Mapping Table:");
        System.out.printf("%-10s%-10s\n", "Character", "Code");
        System.out.println("----------------------");
        for (String[] entry : mapping) {
            System.out.printf("%-10s%-10s\n", entry[0], entry[1]);
        }
        System.out.println("\nOriginal Text:");
        System.out.println(text);
        System.out.println("\nCompressed Text:");
        System.out.println(compressedText);
        System.out.println("\nDecompressed Text:");
        System.out.println(decompressedText);
```

```
        double compressionRatio = calculateCompressionRatio(text, compressedText);
        System.out.printf("\nCompression Efficiency: %.2f%%\n", compressionRatio);
    }
    public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

        System.out.println("Enter text to compress:");
    String text = scanner.nextLine();

    char[] uniqueChars = getUniqueChars(text);
    int[] frequencies = countCharacterFrequency(text, uniqueChars);
        String[][] mapping = createCompressionCodes(uniqueChars, frequencies);

    String compressedText = compressText(text, mapping);
    String decompressedText = decompressText(compressedText, mapping);

        displayCompressionAnalysis(text, compressedText, decompressedText, mapping);

        scanner.close();
    }
}
```

4. Code

```
import java.util.*;

public class TextBasedCalculator {

	public static boolean isValidExpression(String expression) {

	if (expression == null || expression.isEmpty()) return false;

	char[] chars = expression.toCharArray();



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

	char ch = chars[i];

	if (!((ch >= '0' && ch <= '9') || ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == ' ' ||
ch == '(' || ch == ')')) {

	    return false;

	}

	}

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

	char ch = chars[i];

	if (ch == '(' && i < chars.length - 1 && chars[i + 1] == ')') return false;

	if (ch == ')' && (i == 0 || chars[i - 1] == '+' || chars[i - 1] == '-' || chars[i - 1] == '*' ||
chars[i - 1] == '/')) return false;

	if (ch == '(' && (i == chars.length - 1 || chars[i + 1] == '+' || chars[i + 1] == '-' ||
chars[i + 1] == '*' || chars[i + 1] == '/')) return false;

	}

	return true;

	}

	public static List<Object> parseExpression(String expression) {

       List<Object> numbers = new ArrayList<>();
```

```
        List<Character> operators = new ArrayList<>();
          int i = 0;
          while (i < expression.length()) {
          char ch = expression.charAt(i);
          if (Character.isDigit(ch)) {
              int start = i;
              while (i < expression.length() && Character.isDigit(expression.charAt(i))) {
                  i++;
              }
              numbers.add(Integer.parseInt(expression.substring(start, i)));
              continue;
          }
          else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
              operators.add(ch);
          }
          i++;
          }
          return Arrays.asList(numbers, operators);
          }
          public static int evaluateExpression(List<Object> numbers, List<Character>
operators) {
          int result = (int) numbers.get(0);
          int numIndex = 1;
          for (int i = 0; i < operators.size(); i++) {
```

```
  char operator = operators.get(i);
  int nextNum = (int) numbers.get(numIndex);
  if (operator == '+') {
      result += nextNum;
  } else if (operator == '-') {
      result -= nextNum;
  } else if (operator == '*') {
      result *= nextNum;
  } else if (operator == '/') {
      result /= nextNum;
  }
   numIndex++;
  }
  return result;
  }
  public static String evaluateWithOrderOfOperations(String expression) {
List<Object> numbers;
List<Character> operators;

  while (expression.contains("(")) {
  int startIdx = expression.lastIndexOf('(');
  int endIdx = expression.indexOf(')', startIdx);
   String subExpression = expression.substring(startIdx + 1, endIdx);
```

```
            numbers = parseExpression(subExpression).get(0);
            operators = parseExpression(subExpression).get(1);
           int subResult = evaluateExpression(numbers, operators);
            expression = expression.substring(0, startIdx) + subResult +
expression.substring(endIdx + 1);
           }
           numbers = parseExpression(expression).get(0);
         operators = parseExpression(expression).get(1);
           int result = evaluateExpression(numbers, operators);
           return String.valueOf(result);
           }
           public static void displayCalculationSteps(String expression) {
         System.out.println("Original Expression: " + expression);
           while (expression.contains("(")) {
           int startIdx = expression.lastIndexOf('(');
           int endIdx = expression.indexOf(')', startIdx);
            String subExpression = expression.substring(startIdx + 1, endIdx);
            System.out.println("Evaluating: " + subExpression);
            String subResult = evaluateWithOrderOfOperations(subExpression);
            expression = expression.substring(0, startIdx) + subResult +
expression.substring(endIdx + 1);
            System.out.println("Step result: " + expression);
           }
```

```
        System.out.println("Final Result: " + evaluateWithOrderOfOperations(expression));
          }
          public static void main(String[] args) {
          Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a mathematical expression:");
          String expression = scanner.nextLine();
          if (!isValidExpression(expression)) {
           System.out.println("Invalid expression format.");
           return;
          }
        displayCalculationSteps(expression);
        scanner.close();
          }
}
```

5. Code

```
import java.util.*;
import java.text.*;

public class CSVAnalyzer {
          public static String[][] parseCSV(String csvData) {
        List<String[]> dataList = new ArrayList<>();
        StringBuilder field = new StringBuilder();
```

```
  boolean insideQuote = false;
List<String> row = new ArrayList<>();

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

  if (ch == '"') {
      insideQuote = !insideQuote;
  } else if (ch == ',' && !insideQuote) {
      row.add(field.toString().trim());
      field.setLength(0);
  } else if (ch == '\n' || ch == '\r') {
      if (field.length() > 0) {
          row.add(field.toString().trim());
      }
      dataList.add(row.toArray(new String[0]));
      row = new ArrayList<>();
      field.setLength(0);
  } else {
      field.append(ch);
  }
  }
  if (field.length() > 0) {
```

```
 row.add(field.toString().trim());
}
if (!row.isEmpty()) {
 dataList.add(row.toArray(new String[0]));
}
return dataList.toArray(new String[0][0]);
}
public static void cleanData(String[][] data) {
for (int i = 0; i < data.length; i++) {
for (int j = 0; j < data[i].length; j++) {
    data[i][j] = data[i][j].trim();
    if (data[i][j].matches("-?\\d+")) {
        data[i][j] = String.valueOf(Integer.parseInt(data[i][j]));
    } else if (data[i][j].matches("-?\\d*\\.\\d+")) {
        data[i][j] = String.valueOf(Double.parseDouble(data[i][j]));
    }
}
}
}

public static void performDataAnalysis(String[][] data) {
int rows = data.length;
int columns = data[0].length;
```

```
for (int j = 0; j < columns; j++) {
 Set<String> uniqueValues = new HashSet<>();
 double sum = 0;
 double min = Double.MAX_VALUE;
 double max = Double.MIN_VALUE;
int numericCount = 0;
for (int i = 1; i < rows; i++) {
    String cell = data[i][j];
    if (cell.matches("-?\\d*\\.\\d+") || cell.matches("-?\\d+")) {
        double num = Double.parseDouble(cell);
        sum += num;
        min = Math.min(min, num);
        max = Math.max(max, num);
        numericCount++;
    } else {
        uniqueValues.add(cell);
    }
}
 System.out.println("Column " + j + " Statistics:");
if (numericCount > 0) {
    double avg = sum / numericCount;
    System.out.println("Min: " + min + ", Max: " + max + ", Average: " + avg);
```

```
  }
   System.out.println("Unique Values: " + uniqueValues);
  }
  }

  public static void generateFormattedOutput(String[][] data) {
  int columnWidths[] = new int[data[0].length];

  for (int i = 0; i < data[0].length; i++) {
  for (int j = 0; j < data.length; j++) {
      columnWidths[i] = Math.max(columnWidths[i], data[j][i].length());
  }
  }
StringBuilder sb = new StringBuilder();
sb.append("| ");
  for (int i = 0; i < data[0].length; i++) {
   sb.append(String.format("%-" + columnWidths[i] + "s | ", data[0][i]));
  }
sb.append("\n");
sb.append("+");
  for (int i = 0; i < data[0].length; i++) {
   sb.append("-".repeat(columnWidths[i] + 2) + "+");
  }
```

```
sb.append("\n");
   for (int i = 1; i < data.length; i++) {
    sb.append("| ");
   for (int j = 0; j < data[i].length; j++) {
        sb.append(String.format("%-" + columnWidths[j] + "s | ", data[i][j]));
   }
    sb.append("\n");

System.out.println(sb.toString());
   }
   public static void generateSummaryReport(String[][] data) {
   int totalRecords = data.length - 1;
   int columns = data[0].length;
System.out.println("Total Records Processed: " + totalRecords);
   for (int i = 0; i < columns; i++) {
    Set<String> uniqueValues = new HashSet<>();
   int missingCount = 0;
   int invalidCount = 0;
   for (int j = 1; j < data.length; j++) {
        String cell = data[j][i];
        if (cell.isEmpty()) {
            missingCount++;
        } else if (!cell.matches("^[a-zA-Z0-9]+$")) {
```

```
            invalidCount++;
        } else {
            uniqueValues.add(cell);
        }
    }
     System.out.println("Column " + (i + 1) + " Statistics:");
     System.out.println("Unique Values: " + uniqueValues);
     System.out.println("Missing Values: " + missingCount);
     System.out.println("Invalid Entries: " + invalidCount);
    }
    int validRecords = totalRecords - missingCount - invalidCount;
    double completenessPercentage = ((double) validRecords / totalRecords) * 100;
System.out.println("Data Completeness: " + completenessPercentage + "%");
    }
    public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
System.out.println("Enter CSV-like data (Enter 'end' to stop):");
StringBuilder csvData = new StringBuilder();
    String line;
    while (!(line = scanner.nextLine()).equals("end")) {
     csvData.append(line).append("\n");
    }
String[][] data = parseCSV(csvData.toString());
```

```
        cleanData(data);
        performDataAnalysis(data);
        generateFormattedOutput(data);
        generateSummaryReport(data);
        scanner.close();
          }
}
```

6. Code

```
import java.util.*;
import java.text.*;
import java.io.*;
public class FileOrganizer {
          static class FileInfo {
          String originalName;
          String category;
          String newName;
        FileInfo(String originalName, String category, String newName) {
            this.originalName = originalName;
            this.category = category;
            this.newName = newName;
          }
          }
```

```
  public static String[] extractFileComponents(String fileName) {
  int dotIndex = fileName.lastIndexOf(".");
  if (dotIndex == -1) {
   return new String[] {fileName, ""}; // No extension
  }
  String name = fileName.substring(0, dotIndex);
  String extension = fileName.substring(dotIndex + 1);
  return new String[] {name, extension};
  }

  public static Map<String, Integer> categorizeFiles(List<String> fileNames) {
Map<String, Integer> categories = new HashMap<>();
Map<String, List<FileInfo>> categorizedFiles = new HashMap<>();
  for (String fileName : fileNames) {
   String[] components = extractFileComponents(fileName);
   String name = components[0];
   String extension = components[1].toLowerCase();
   String category = "Unknown";
  if (Arrays.asList("txt", "doc", "pdf").contains(extension)) {
      category = "Documents";
  } else if (Arrays.asList("jpg", "png", "gif").contains(extension)) {
      category = "Images";
  } else if (Arrays.asList("mp3", "wav").contains(extension)) {
```

```
            category = "Audio";
        } else if (Arrays.asList("mp4", "mkv").contains(extension)) {
            category = "Videos";
        }
        if (!categories.containsKey(category)) {
            categories.put(category, 0);
            categorizedFiles.put(category, new ArrayList<>());
        }
         categories.put(category, categories.get(category) + 1);
         categorizedFiles.get(category).add(new FileInfo(fileName, category, ""));
        }
        return categories;
        }

        public static String generateNewFileName(String originalName, String category,
int count) {
        String timestamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new
Date());
        String baseName = category + "_" + timestamp + "_" + count;
        String newName = baseName + ".txt"; // Default to .txt extension
        return newName;
        }
        public static String analyzeContent(String content) {
        if (content.contains("Resume")) {
```

```
            return "Resume";
        } else if (content.contains("Report")) {
            return "Report";
        } else if (content.contains("public class")) {
            return "Code";
        } else {
            return "General";
        }
    }
    public static void displayReport(Map<String, Integer> categories, List<FileInfo>
files) {
        System.out.println("File Organization Report:");
        System.out.println("Category-wise File Counts:");
        for (Map.Entry<String, Integer> entry : categories.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue() + " files");
        }
        System.out.println("\nDetailed File List:");
        for (FileInfo file : files) {
            System.out.println("Original Name: " + file.originalName + " | Category: " +
file.category + " | New Name: " + file.newName);
        }
    }
    public static void generateRenameCommands(List<FileInfo> files) {
        System.out.println("\nBatch Rename Commands:");
```

```
   for (FileInfo file : files) {
    System.out.println("mv " + file.originalName + " " + file.newName);
   }
   }
   public static void main(String[] args) {
   Scanner scanner = new Scanner(System.in);
List<String> fileNames = new ArrayList<>();
System.out.println("Enter file names (type 'end' to finish):");
   while (true) {
    String fileName = scanner.nextLine();
   if (fileName.equals("end")) {
       break;
   }
    fileNames.add(fileName);
   }
Map<String, Integer> categories = categorizeFiles(fileNames);
List<FileInfo> files = new ArrayList<>();
   for (String fileName : fileNames) {
    String[] components = extractFileComponents(fileName);
    String name = components[0];
    String extension = components[1].toLowerCase();
    String category = "Unknown";
   if (Arrays.asList("txt", "doc", "pdf").contains(extension)) {
```

```
            category = "Documents";
        } else if (Arrays.asList("jpg", "png", "gif").contains(extension)) {
            category = "Images";
        } else if (Arrays.asList("mp3", "wav").contains(extension)) {
            category = "Audio";
        } else if (Arrays.asList("mp4", "mkv").contains(extension)) {
            category = "Videos";
        }
        int count = categories.get(category);
         String newFileName = generateNewFileName(fileName, category, count);
         files.add(new FileInfo(fileName, category, newFileName));
        }
    displayReport(categories, files);
    generateRenameCommands(files);

    scanner.close();
    }
}
```