AI Assisted Coding

# ASSIGNMENT 2.3

Name: R.NIHARSHITH

HT No: 2303A52044

Batch: 31

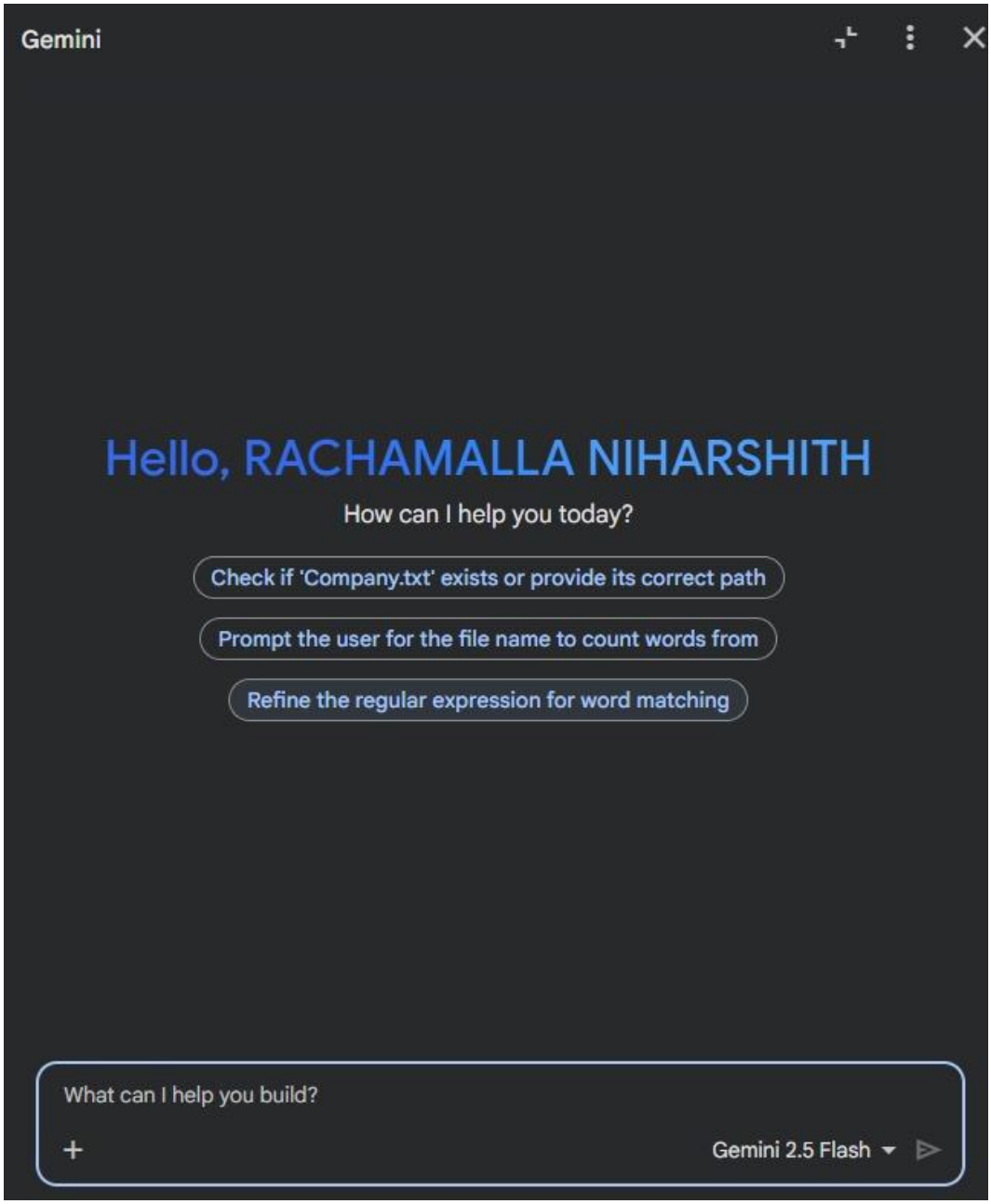
## Question:

Task 1:

Word Frequency from Text File

Scenario: You are analyzing log files for keyword frequency.

Task: Use Gemini to generate java code that reads a text file and counts word frequency, then explains the code


Gemini
Hello, RACHAMALLA NIHARSHITH
How can I help you today?
Check if 'Company.txt' exists or provide its correct path
Prompt the user for the file name to count words from
Refine the regular expression for word matching
What can I help you build?
Gemini 2.5 Flash

Prompt :

Task 1:

Word Frequency from Text File

Scenario: You are analyzing log files for keyword frequency.

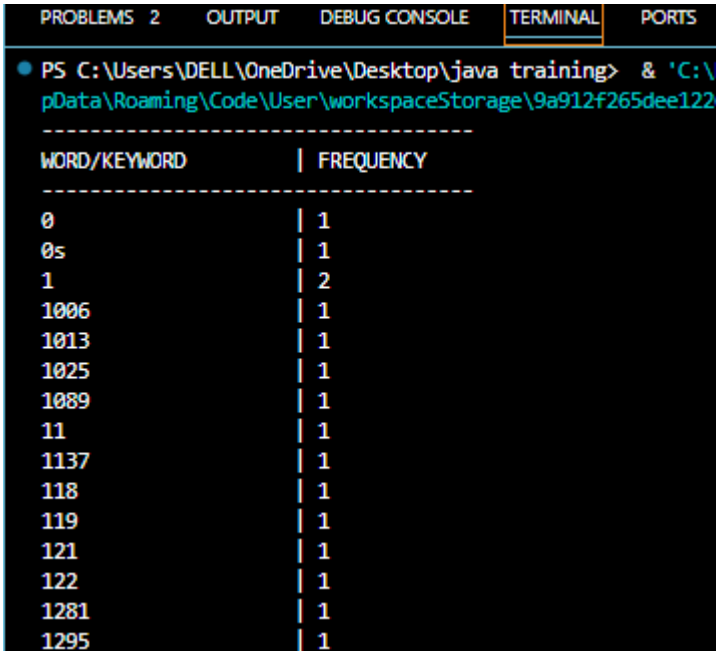
Task: Use Gemini to generate java code that reads a text file and counts word frequency,

Code

WordFrequencyCounter.java > WordFrequencyCounter

```java
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.Map;
import java.util.TreeMap;
import java.util.function.Function;
import java.util.stream.Collectors;
Windsurf: Refactor | Explain
public class WordFrequencyCounter {
    Run | Debug | Windsurf: Refactor | Explain | Generate Javadoc |
    public static void main(String[] args) {
        // Your specific Windows file path
        String filePath = "C:\\Users\\DELL\\OneDrive\\Desktop\\java training\\SRU JAVA B1.txt";

        try {
            // 1. Read all lines from the file
            // 2. Convert to lowercase
            // 3. Split into words using regex (\\W+ matches any non-word character)
            // 4. Count the occurrences of each word
            Map<String, Long> wordCounts = Files.lines(Paths.get(filePath), StandardCharsets.UTF_8)
                .flatMap(line -> Arrays.stream(line.toLowerCase().split(regex: "\\W+")))
                .filter(word -> !word.isEmpty())
                .collect(Collectors.groupingBy(
                    Function.identity(),
                    TreeMap::new, // This sorts the words alphabetically
                    Collectors.counting()
                ));

            // Display the results
            System.out.println(x: "----------------------------------");
            System.out.printf(format: "%-20s | %s%n", ...args: "WORD/KEYWORD", "FREQUENCY");
            System.out.println(x: "----------------------------------");

            if (wordCounts.isEmpty()) {
                System.out.println(x: "No words found in the file.");
            } else {
                wordCounts.forEach((word, count) ->
                    System.out.printf(format: "%-20s | %d%n", word, count)
                );
            }
            System.out.println(x: "----------------------------------");

        } catch (IOException e) {
            System.err.println(x: "Error: Could not find or read the file.");
            System.err.println("Make sure the path is correct: " + filePath);
            e.printStackTrace();
        }
    }
}
```

Output




Explanation :

First, it opens your text file and reads it line by line. Then, it breaks every sentence into individual words while ignoring things like commas or periods. It turns everything into **lowercase** so that "Apple" and "apple" aren't counted as two different things. Each word is then tossed into a "counting bucket" (a **Map**) where it keeps track of how many times that word has appeared. Finally, it just prints out that bucket as an organized list for you to see.

# Question:

Task 2:

File Operations Using Cursor AI

Scenario: You are automating basic file operations.

Task: Use Cursor AI to generate a program that:

Creates a text file

Writes sample text

Reads and displays the content

Prompt:

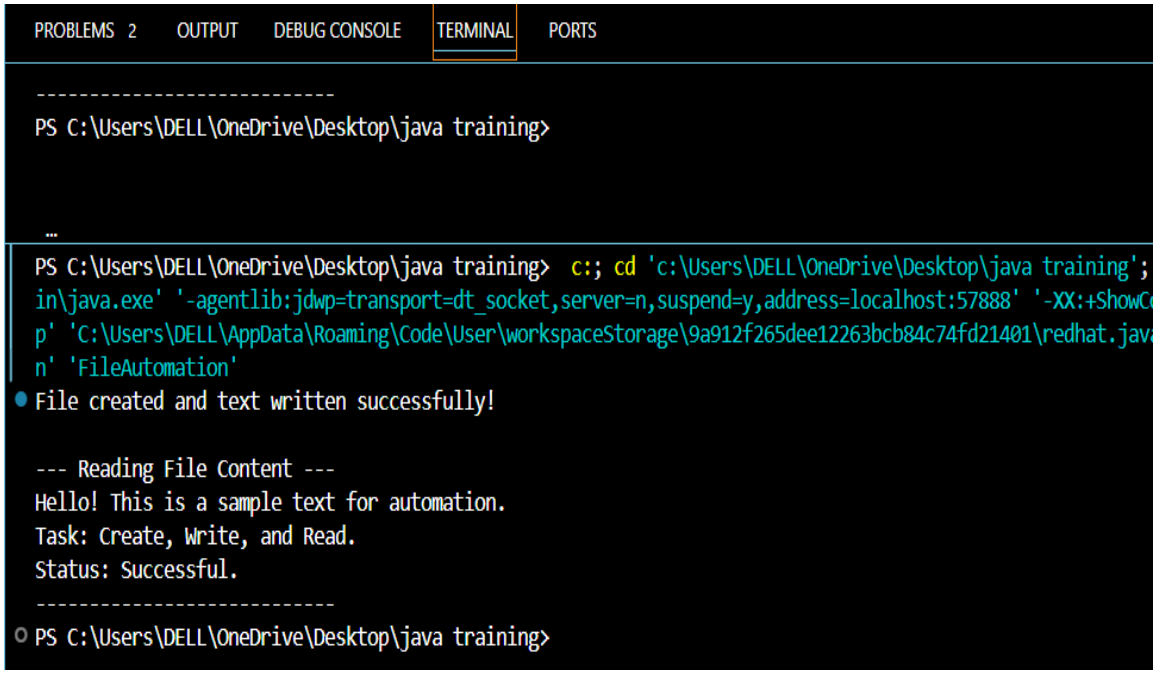
Assuming You are automating basic file operations . Your task is to
Creates a text file , Writes sample text
Reads and displays the content and print the output

Code :

FileAutomation.java > ...

```java
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.io.IOException;
import java.util.List;

Windsurf: Refactor | Explain
public class FileAutomation {
    Run | Debug | Windsurf: Refactor | Explain | Generate Javadoc |
    public static void main(String[] args) {
        // 1. Define the file path and name
        Path filePath = Paths.get(first: "automation_test.txt");
        String contentToWrite = "Hello! This is a sample text for automation.\n" +
                                "Task: Create, Write, and Read.\n" +
                                "Status: Successful.";

        try {
            // 2. CREATE AND WRITE: This one command creates the file or overwrites it
            Files.write(filePath, contentToWrite.getBytes());
            System.out.println(x: "File created and text written successfully!\n");

            // 3. READ AND DISPLAY: Read all lines and print them
            System.out.println(x: "--- Reading File Content ---");
            List<String> lines = Files.readAllLines(filePath);

            for (String line : lines) {
                System.out.println(line);
            }
            System.out.println(x: "----------------------------");

        } catch (IOException e) {
            System.err.println("An error occurred: " + e.getMessage());
        }
    }
}
```

Output:




Explanation :

 **Setting the Path:** First, we tell Java the "address" of the file we want to work with (the filename).

 **Writing the File:** We use a "write" command that automatically **creates** the file and pours our text inside.

 **Opening the File:** Next, we tell Java to go back, open that same file, and grab all the lines of text.

 **The Memory List:** Java saves those lines into a temporary list so the computer can remember them.

 **Printing:** Finally, we loop through that list and **print** each line to the screen so we can see the result.

# Question:

Task 3: CSV Data Analysis

Scenario: You are processing structured data from a CSV file.

Task: Use Gemini in Colab to read a CSV file and calculate mean, min, and max.

Prompt:

Generate java code to read a CSV file and calculate the mean, minimum, and maximum values of a numeric column.

Prompt:

Generate java
code to read a CSV file and calculate the mean, minimum, and
maximum values of
a numeric column.

Code :

```java
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.DoubleSummaryStatistics;
import java.util.List;

Windsurf: Refactor | Explain
public class CSVAutoTask {
    Run | Debug | Windsurf: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        Path filePath = Paths.get(first: "generated_scores.csv");
        // 1. CREATE: Generate the dataset inside a CSV file
        String csvData = "id,name,math_score\n" +
                         "1,Paul,73\n" +
                         "2,Danielle,90\n" +
                         "3,Tina,81\n" +
                         "4,Tara,71\n" +
                         "5,Anthony,84";
        try {
            // Write the data to a new file
            Files.write(filePath, csvData.getBytes());
            System.out.println(x: "Step 1: File 'generated_scores.csv' created successfully.");

            // 2. READ & CALCULATE: Process the math_score (Column Index 2)
            int columnId = 2;
            DoubleSummaryStatistics stats = Files.lines(filePath)
                .skip(n: 1) // Skip headers
                .map(line -> line.split(regex: ","))
                .filter(parts -> parts.length > columnId)
                .mapToDouble(parts -> Double.parseDouble(parts[columnId].trim()))
                .summaryStatistics();
            // 3. DISPLAY RESULTS
            System.out.println(x: "\nStep 2: Analysis Results from File:");
            System.out.println(x: "-----------------------------------");
            System.out.printf(format: "Average Math Score: %.2f%n", stats.getAverage());
            System.out.printf(format: "Highest Math Score: %.2f%n", stats.getMax());
            System.out.printf(format: "Lowest Math Score:  %.2f%n", stats.getMin());
            System.out.printf(format: "Total Students:     %d%n", stats.getCount());
            System.out.println(x: "-----------------------------------");
        } catch (IOException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }
}
```

Output:

```
PS C:\Users\DELL\OneDrive\Desktop\java training>  c:; cd 'c:\Users\DELL\OneDrive\Desktop\java training'
ansport=dt_socket,server=n,suspend=y,address=localhost:50399' '-XX:+ShowCodeDetailsInExceptionMessages'
a912f265dee12263bcb84c74fd21401\redhat.java\jdt_ws\java training_dab59d09\bin' 'CSVAutoTask'
Step 2: Analysis Results from File:
----------------------------------
Average Math Score: 79.80
Highest Math Score: 90.00
Lowest Math Score:  71.00
Total Students:     5
----------------------------------
```

Explanation:

 **Creating the File:** We use Files.write to create a real file called generated_scores.csv and put our student data inside it.

 **Opening to Read:** After creating it, we use Files.lines to go back and open that same file so we can look at the numbers.

 **The Header Skip:** We skip the first line (id,name,math_score) because you can't do math on words.

 **Picking the Column:** We tell Java to look at **Index 2** (the 3rd column), which is where our scores are stored.

 **Instant Math:** The summaryStatistics tool scans all the numbers we just pulled out and finds the average, highest, and lowest scores for us.

# Question:

Task 4: Sorting Lists Manual vs Built-in

Scenario: You are reviewing algorithm choices for efficiency.

Task: Use Gemini to generate:

Bubble sort

Python’s built-in sort()

Compare both implementations

Prompt:

Generate Bubble sort java code

Generate Bubble sort code java using Bubble sort Built in function

Code :

1. Using nested loops Bubble sort

```java
import java.util.Arrays;

Windsurf: Refactor | Explain
public class BubbleSortDemo {
    Run | Debug | Windsurf: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        // 1. Create a sample array of numbers
        int[] numbers = {64, 34, 25, 12, 22, 11, 90};

        System.out.println("Before Sorting: " + Arrays.toString(numbers));

        // 2. The Bubble Sort Logic
        int n = numbers.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                // If the number on the left is bigger than the right, swap them
                if (numbers[j] > numbers[j + 1]) {
                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }

        // 3. Display the sorted result
        System.out.println("After Sorting:  " + Arrays.toString(numbers));
    }
}
```

2. Actually, Java does **not** have a specific "built-in function" named bubbleSort(). However, Java provides a very powerful built-in tool called Arrays.sort().

## The Java "Built-in" Way

BuiltInSort.java > ...

```java
import java.util.Arrays;

Windsurf: Refactor | Explain
public class BuiltInSort {
    Run | Debug | Windsurf: Refactor | Explain | Generate Javadoc | ×
    public static void main(String[] args) {
        // 1. Create your array
        int[] numbers = {64, 34, 25, 12, 22, 11, 90};

        System.out.println("Before: " + Arrays.toString(numbers));

        // 2. Use the Java Built-in sorting function
        // (This uses a high-performance algorithm behind the scenes)
        Arrays.sort(numbers);

        // 3. Show the result
        System.out.println("After:  " + Arrays.toString(numbers));
    }
}
```

Output:

```
PS C:\Users\DELL\OneDrive\Desktop\java training>  & 'C:\Program Fi
=n,suspend=y,address=localhost:50312' '-XX:+ShowCodeDetailsInExcep
orage\9a912f265dee12263bcb84c74fd21401\redhat.java\jdt_ws\java tra
Before Sorting: [64, 34, 25, 12, 22, 11, 90]
After Sorting:  [11, 12, 22, 25, 34, 64, 90]
```

```
in\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,a
p' 'C:\Users\DELL\AppData\Roaming\Code\User\workspaceStorage\9a912f26
n' 'BuiltInSort'
Before: [640, 354, 255, 112, 202, 111, 90]
After:  [90, 111, 112, 202, 255, 354, 640]
```

## Explanation :

Bubble sort is a simple sorting algorithm that repeatedly compares and swaps adjacent elements. It is easy to understand but inefficient for large data sets.

Java sbuilt-in sort function is shorter, optimized and much faster. The built-in method should be preferred in real-world applications