WORD STATISTICS

Project Description:-

- This project entails the development of a user-friendly application that analyzes text files within a specified directory, providing comprehensive word statistics. The program will include a simple Graphical User Interface (GUI) to facilitate user interaction and real-time display of statistics during the processing of text files.
- Once processing is complete, the program summarizes overall statistics for the entire dataset.
- This includes the cumulative counts of words, longest and shortest words encountered.
- **Outcome:** This project aims to deliver a practical tool for users to gain insights into the textual content of files within a specified directory. The real-time updates and intuitive GUI enhance the user experience, making the analysis of word statistics more accessible and efficient.

TOOLS USED:-

- **The project is implemented using Java threads, mutex locks, GUI and Semaphore ,to coordinate the activities of Word Statistics.**

What we Have Did?

- **In this project, we have created a Java application that performs word statistics analysis on text files within a specified directory. Here's a breakdown of what the program does:**
- **Graphical User Interface (GUI):** designed a simple GUI using Java Swing components, providing an interactive interface for users to input a directory, specify whether to include subdirectories, and initiate the processing of text files.
- **Mutex Lock (“Synchronized” Keyword):** the “synchronized” keyword is used to ensure thread safety when updating shared resources, a mutex lock object is introduced to synchronize critical sections and avoid potential race conditions.
- **Thread Safety for Reading Files :** File reading operations are synchronized using the ‘ lock ‘object to ensure that file content is read safely in a multi-threaded environment.
- **Separate Thread for Processing Each File:** the ‘processFile’ method is executed using a separate thread (created by a new ‘ ExecutorService’ with a single thread) for each individual file. This enables concurrent processing of multiple files.
- **Semphore:** sed to control access to a resource with limited capacity. In this case, the semaphore is used to limit the number of concurrent threads that can execute the “Process directory” method

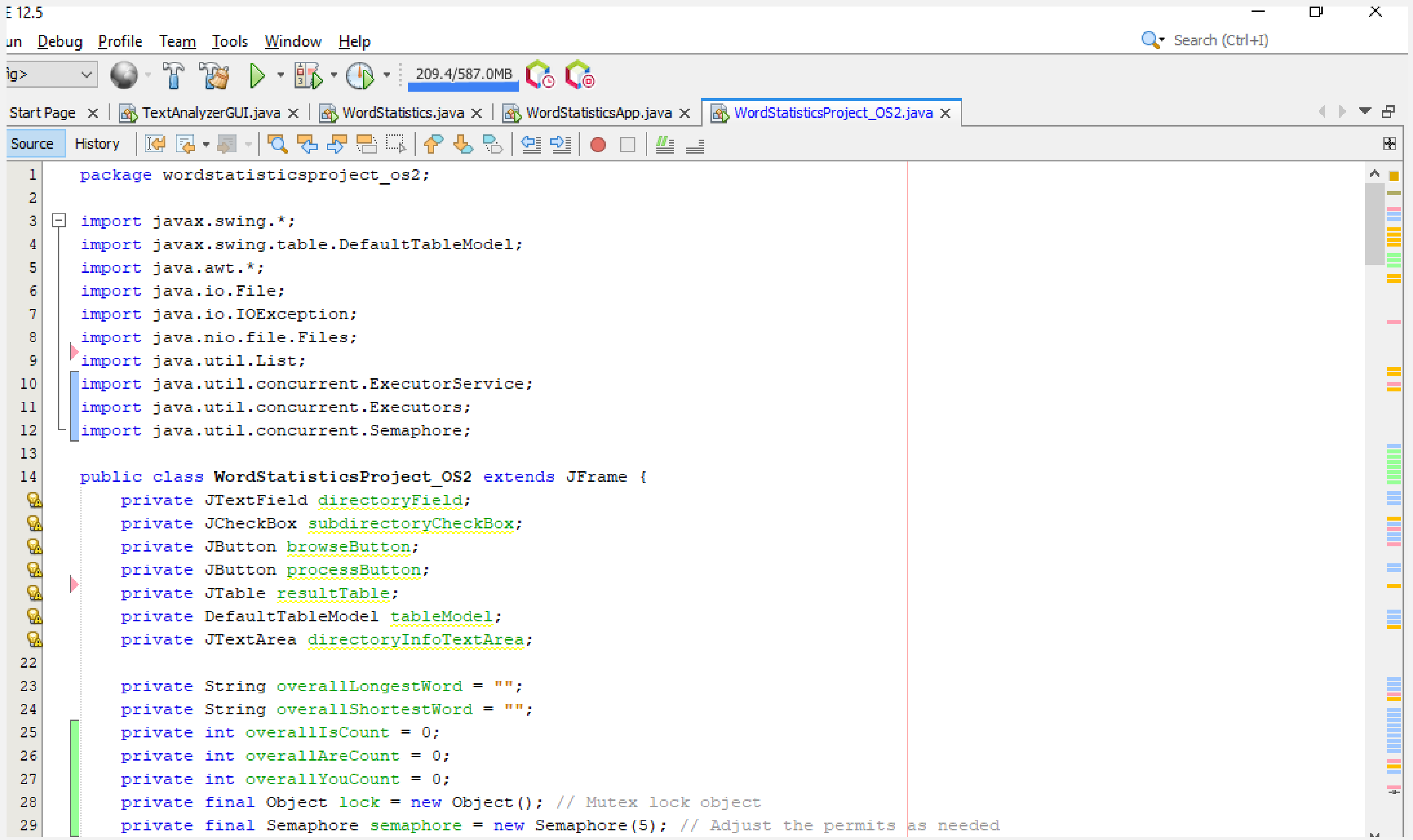
Team Members roles:-

- **Manage Classes: Omar Ashraf, Mohamed Yasser.**
- **GUI: Shehab Wael, Mahran Saber**
- **Documentation: Youssef Mohamed, Yehia Hany**

Implementation Details

This part of the code defines a class that extends “Jframe” which is a class in Java Swing used for creating GUI windows

The purpose of this code is to create a Swing-based GUI application that allows the user to select a directory, process files in the directory (and its subdirectories), and display word statistics in a table along with overall statistics in a text area. The use of the “Lock” object ensures thread safety when updating shared data structures, the semaphore is used to limit the number of concurrent threads that can executed.

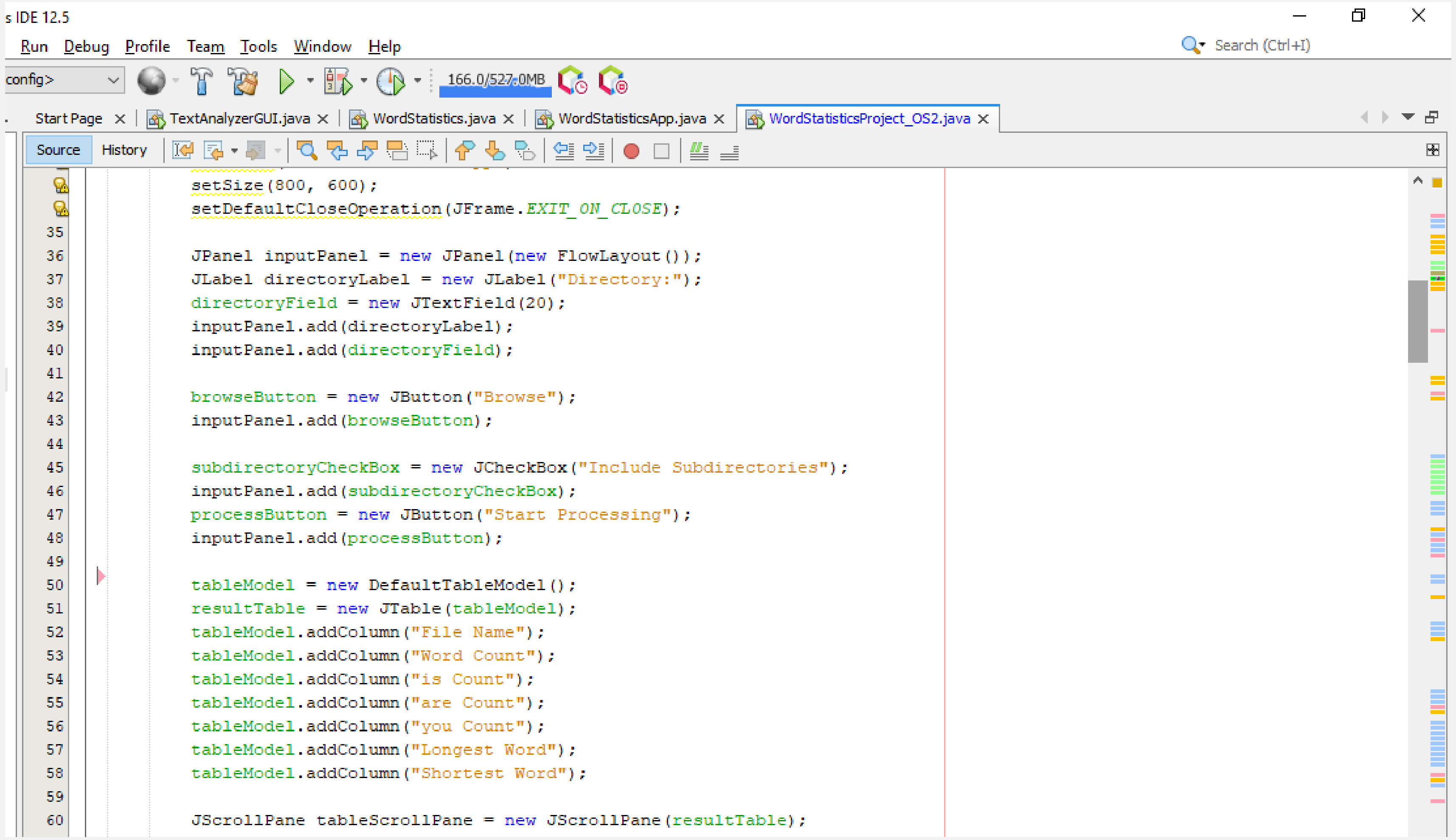


```java
package wordstatisticsproject_os2;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Semaphore;

public class WordStatisticsProject_OS2 extends JFrame {
    private JTextField directoryField;
    private JCheckBox subdirectoryCheckBox;
    private JButton browseButton;
    private JButton processButton;
    private JTable resultTable;
    private DefaultTableModel tableModel;
    private JTextArea directoryInfoTextArea;

    private String overallLongestWord = "";
    private String overallShortestWord = "";
    private int overallIsCount = 0;
    private int overallAreCount = 0;
    private int overallYouCount = 0;
    private final Object lock = new Object(); // Mutex lock object
    private final Semaphore semaphore = new Semaphore(5); // Adjust the permits as needed
```

Implementation Details

—

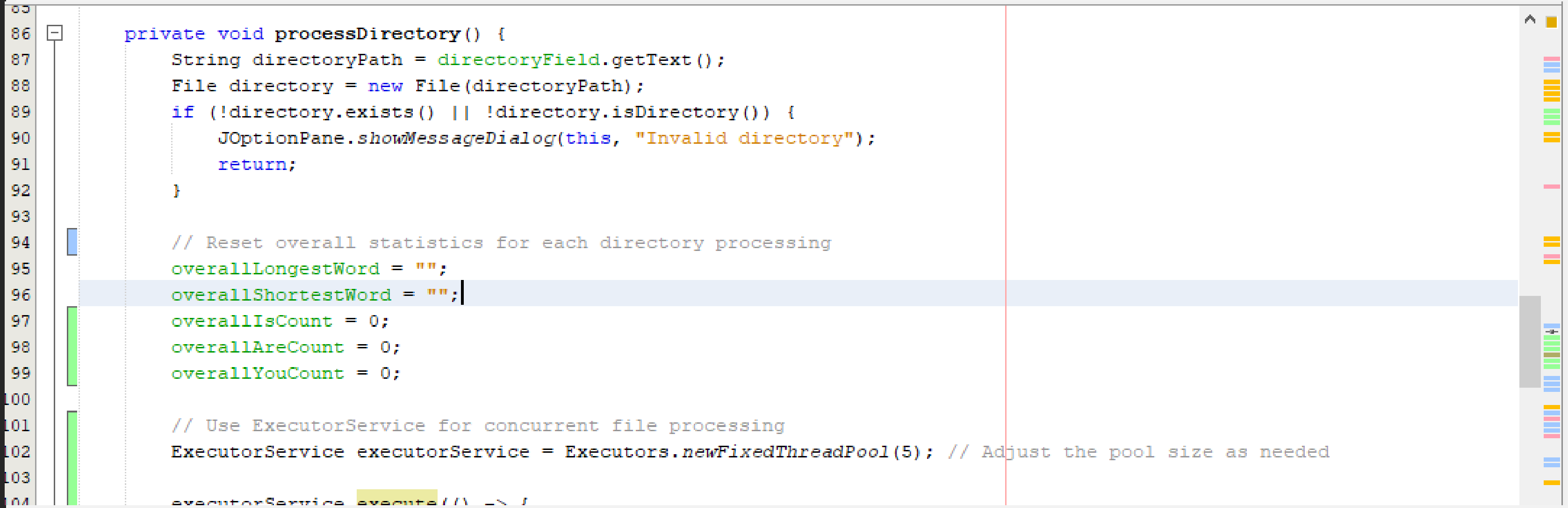
In This part we set the size of the Window of the GUI , add the buttons (Start processing,browsing,subdirectories) and the names of columns of the table



```java
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JPanel inputPanel = new JPanel(new FlowLayout());
        JLabel directoryLabel = new JLabel("Directory:");
        directoryField = new JTextField(20);
        inputPanel.add(directoryLabel);
        inputPanel.add(directoryField);

        browseButton = new JButton("Browse");
        inputPanel.add(browseButton);

        subdirectoryCheckBox = new JCheckBox("Include Subdirectories");
        inputPanel.add(subdirectoryCheckBox);
        processButton = new JButton("Start Processing");
        inputPanel.add(processButton);

        tableModel = new DefaultTableModel();
        resultTable = new JTable(tableModel);
        tableModel.addColumn("File Name");
        tableModel.addColumn("Word Count");
        tableModel.addColumn("is Count");
        tableModel.addColumn("are Count");
        tableModel.addColumn("you Count");
        tableModel.addColumn("Longest Word");
        tableModel.addColumn("Shortest Word");

        JScrollPane tableScrollPane = new JScrollPane(resultTable);
```

Implementation Details

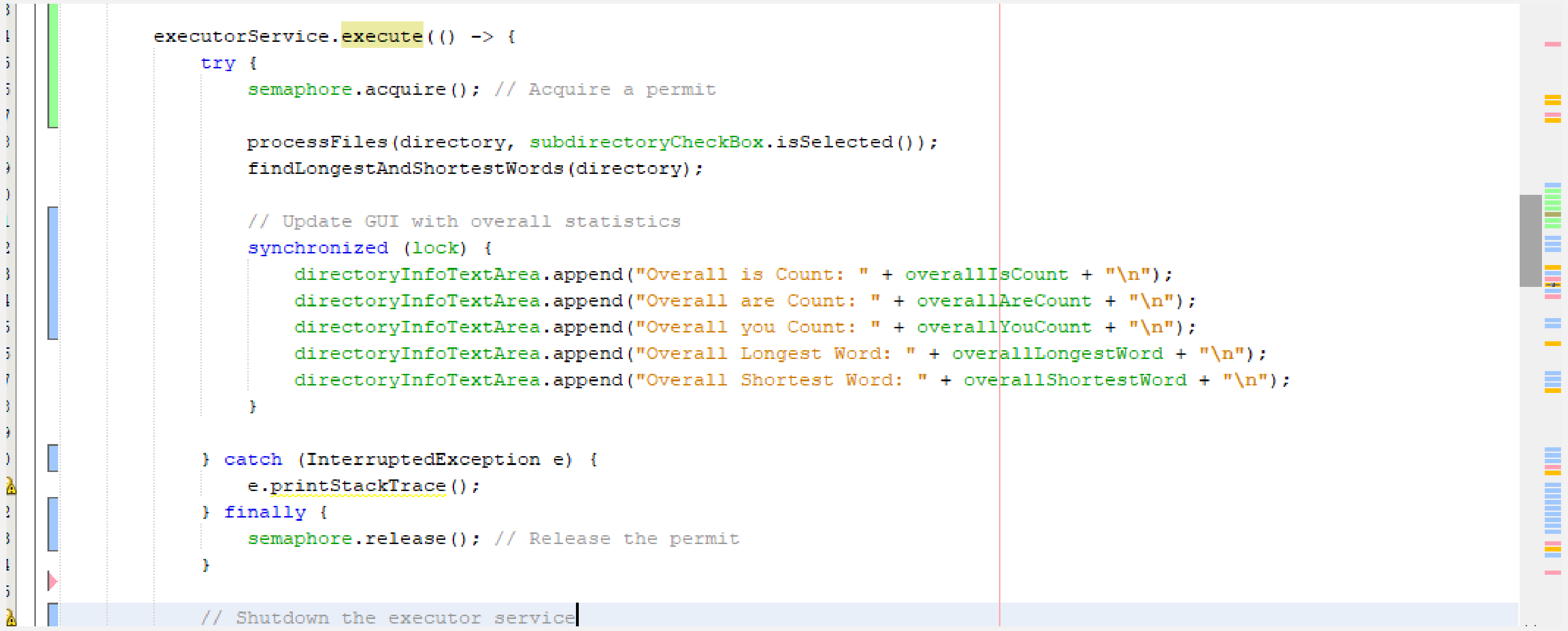
- the first part of the code ensures that the user has entered a valid directory path before proceeding with the directory processing logic. If the directory is invalid, it displays an error message.
- newFixedThreadPool(5) his means that the thread pool will have a maximum of 5 threads running concurrently. If you submit more than 5 tasks, they will be queued and executed when a thread becomes available.



```
    private void processDirectory() {
        String directoryPath = directoryField.getText();
        File directory = new File(directoryPath);
        if (!directory.exists() || !directory.isDirectory()) {
            JOptionPane.showMessageDialog(this, "Invalid directory");
            return;
        }

        // Reset overall statistics for each directory processing
        overallLongestWord = "";
        overallShortestWord = "";
        overallIsCount = 0;
        overallAreCount = 0;
        overallYouCount = 0;

        // Use ExecutorService for concurrent file processing
        ExecutorService executorService = Executors.newFixedThreadPool(5); // Adjust the pool size as needed

        executorService.execute(() -> {
```

Implementation Details

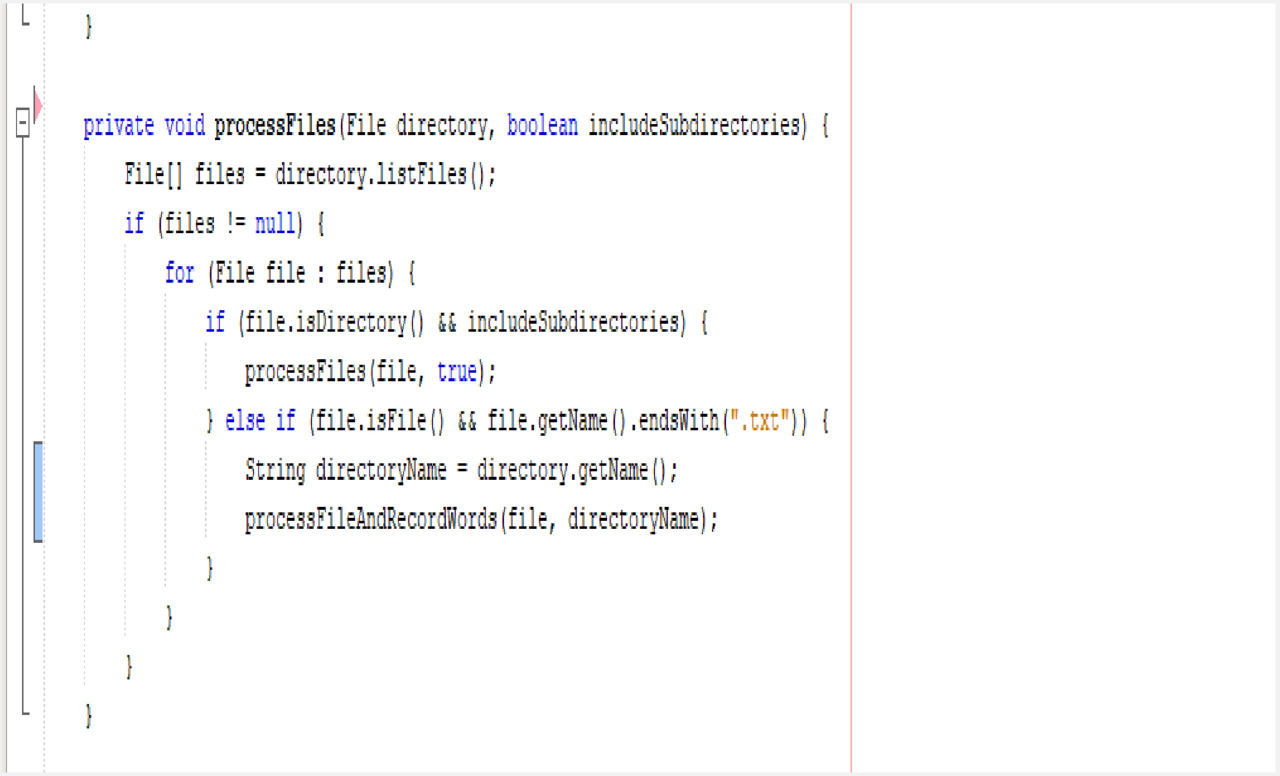
- Semaphore acuire() means:Before entering the critical section of code, a thread must acquire a permit from the semaphore. If no permits are available, the thread will block until a permit becomes available..
- Critical Section: The code inside the ‘try’ block is the critical section of code that you want to control access to.
- Semaphore release(): After the critical section is executed, the acquired permit is released using semaphore.release This allows other waiting threads to acquire a permit and enter the critical section.



```
executorService.execute(() -> {
    try {
        semaphore.acquire(); // Acquire a permit

        processFiles(directory, subdirectoryCheckBox.isSelected());
        findLongestAndShortestWords(directory);

        // Update GUI with overall statistics
        synchronized (lock) {
            directoryInfoTextArea.append("Overall is Count: " + overallIsCount + "\n");
            directoryInfoTextArea.append("Overall are Count: " + overallAreCount + "\n");
            directoryInfoTextArea.append("Overall you Count: " + overallYouCount + "\n");
            directoryInfoTextArea.append("Overall Longest Word: " + overallLongestWord + "\n");
            directoryInfoTextArea.append("Overall Shortest Word: " + overallShortestWord + "\n");
        }

    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        semaphore.release(); // Release the permit
    }

    // Shutdown the executor service
```

Implementation Details

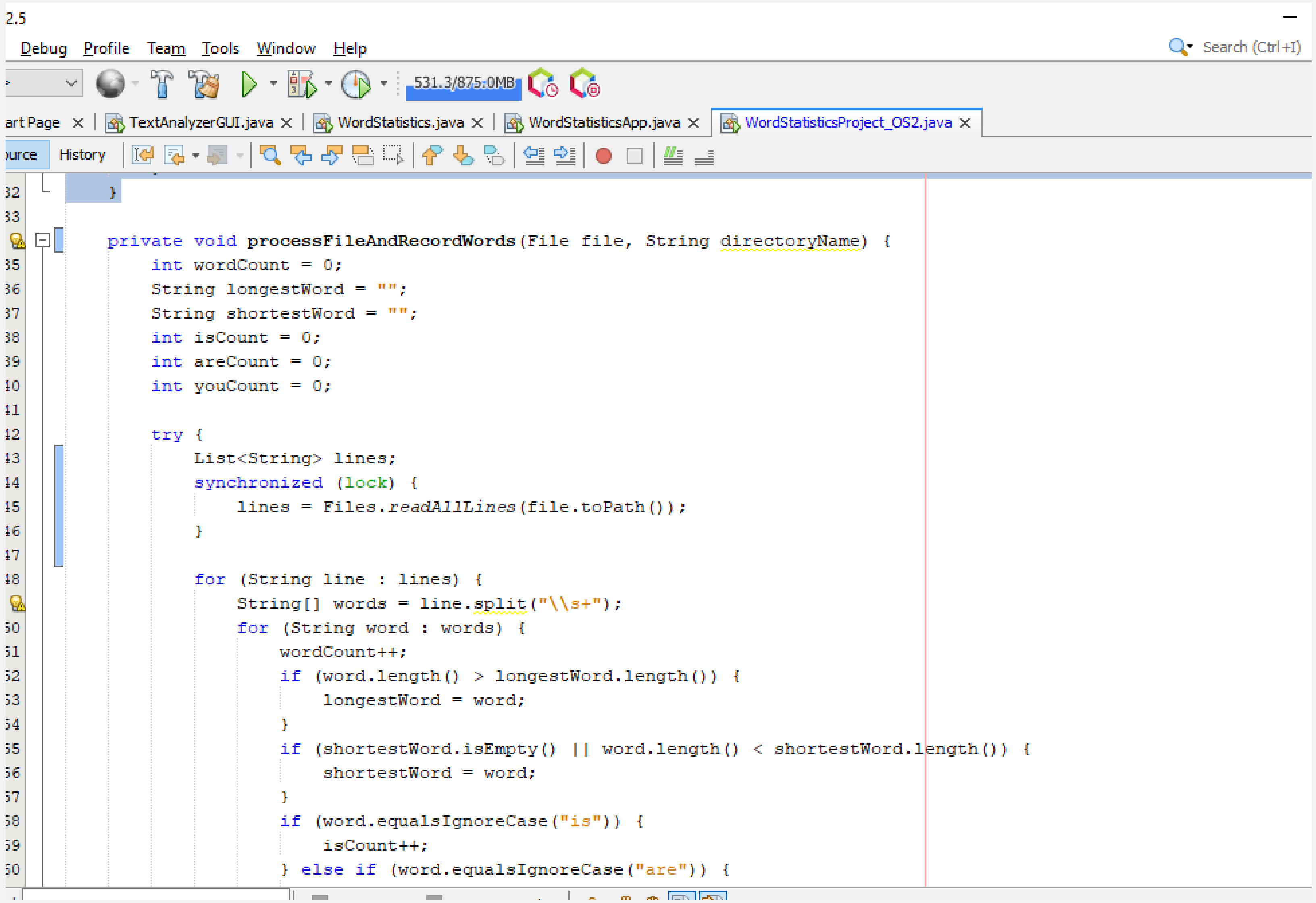
- this method recursively explores the contents of a directory, processing each text file it encounters by calling the ‘**processFileAndRecordWords**’, method. The recursion continues into subdirectories if the ‘**includeSubdirectories’** lag is set to true. This way, the method traverses the entire directory structure, processing relevant text files.



```
}

private void processFiles(File directory, boolean includeSubdirectories) {
    File[] files = directory.listFiles();
    if (files != null) {
        for (File file : files) {
            if (file.isDirectory() && includeSubdirectories) {
                processFiles(file, true);
            } else if (file.isFile() && file.getName().endsWith(".txt")) {
                String directoryName = directory.getName();
                processFileAndRecordWords(file, directoryName);
            }
        }
    }
}
```

Implementation Details

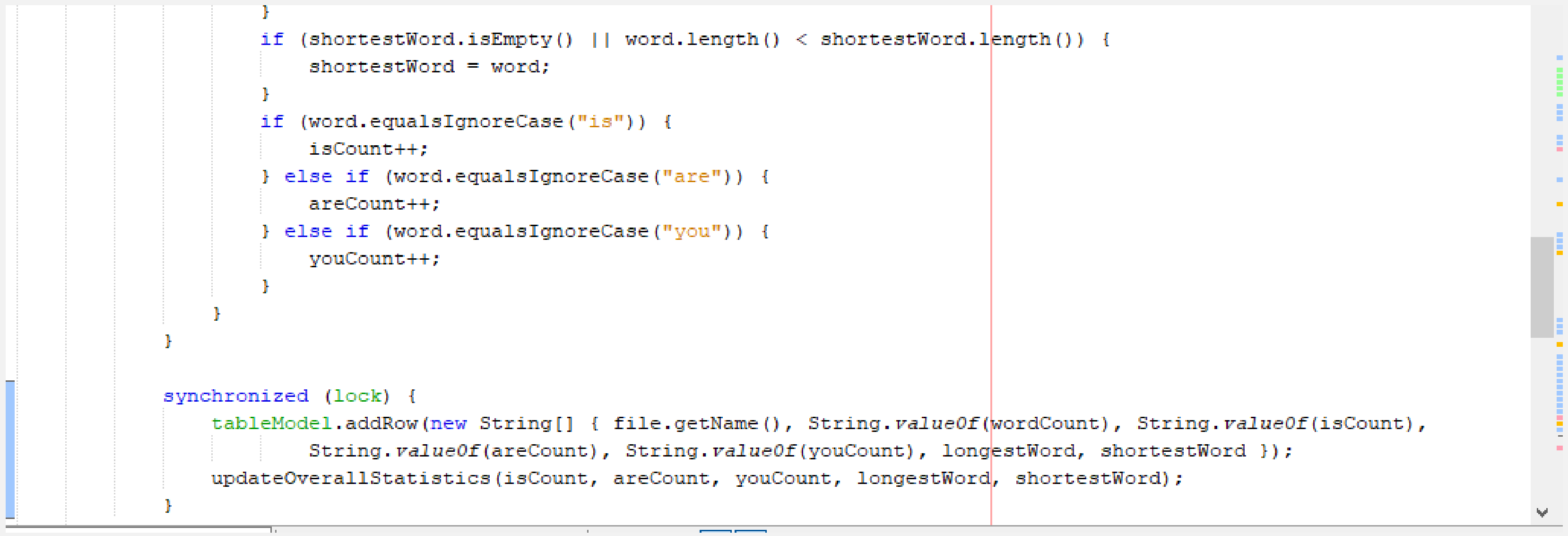
- This method is responsible for processing a given file, counting words, finding the longest and shortest words, and updating the GUI with the relevant information.
- It reads the lines when we choose the file.
- Is splits the lines into words
- The 'lock' is used to ensure that the operations inside it are executed atomically and in a thread-safe manner. In a multithreaded environment, where multiple threads can potentially access and modify shared data concurrently, using synchronization is crucial to avoid race conditions and maintain data integrity.



```
    }

    private void processFileAndRecordWords(File file, String directoryName) {
        int wordCount = 0;
        String longestWord = "";
        String shortestWord = "";
        int isCount = 0;
        int areCount = 0;
        int youCount = 0;

        try {
            List<String> lines;
            synchronized (lock) {
                lines = Files.readAllLines(file.toPath());
            }

            for (String line : lines) {
                String[] words = line.split("\\s+");
                for (String word : words) {
                    wordCount++;
                    if (word.length() > longestWord.length()) {
                        longestWord = word;
                    }
                    if (shortestWord.isEmpty() || word.length() < shortestWord.length()) {
                        shortestWord = word;
                    }
                    if (word.equalsIgnoreCase("is")) {
                        isCount++;
                    } else if (word.equalsIgnoreCase("are")) {
```

Implementation Details

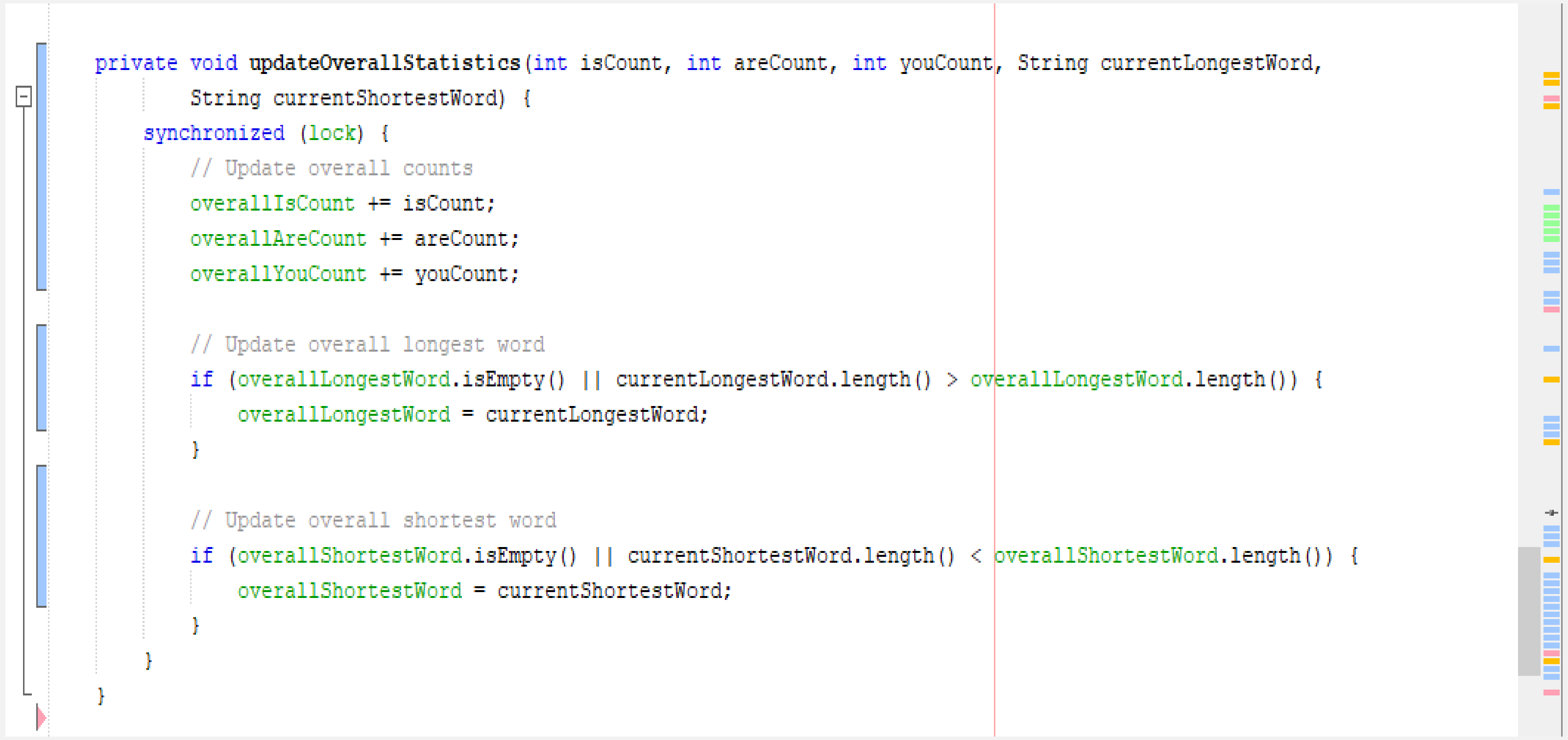
COMPLETE OF THE PREVIOUS SLIDE



```
                }
                if (shortestWord.isEmpty() || word.length() < shortestWord.length()) {
                    shortestWord = word;
                }
                if (word.equalsIgnoreCase("is")) {
                    isCount++;
                } else if (word.equalsIgnoreCase("are")) {
                    areCount++;
                } else if (word.equalsIgnoreCase("you")) {
                    youCount++;
                }
            }
        }

        synchronized (lock) {
            tableModel.addRow(new String[] { file.getName(), String.valueOf(wordCount), String.valueOf(isCount),
                    String.valueOf(areCount), String.valueOf(youCount), longestWord, shortestWord });
            updateOverallStatistics(isCount, areCount, youCount, longestWord, shortestWord);
        }
```

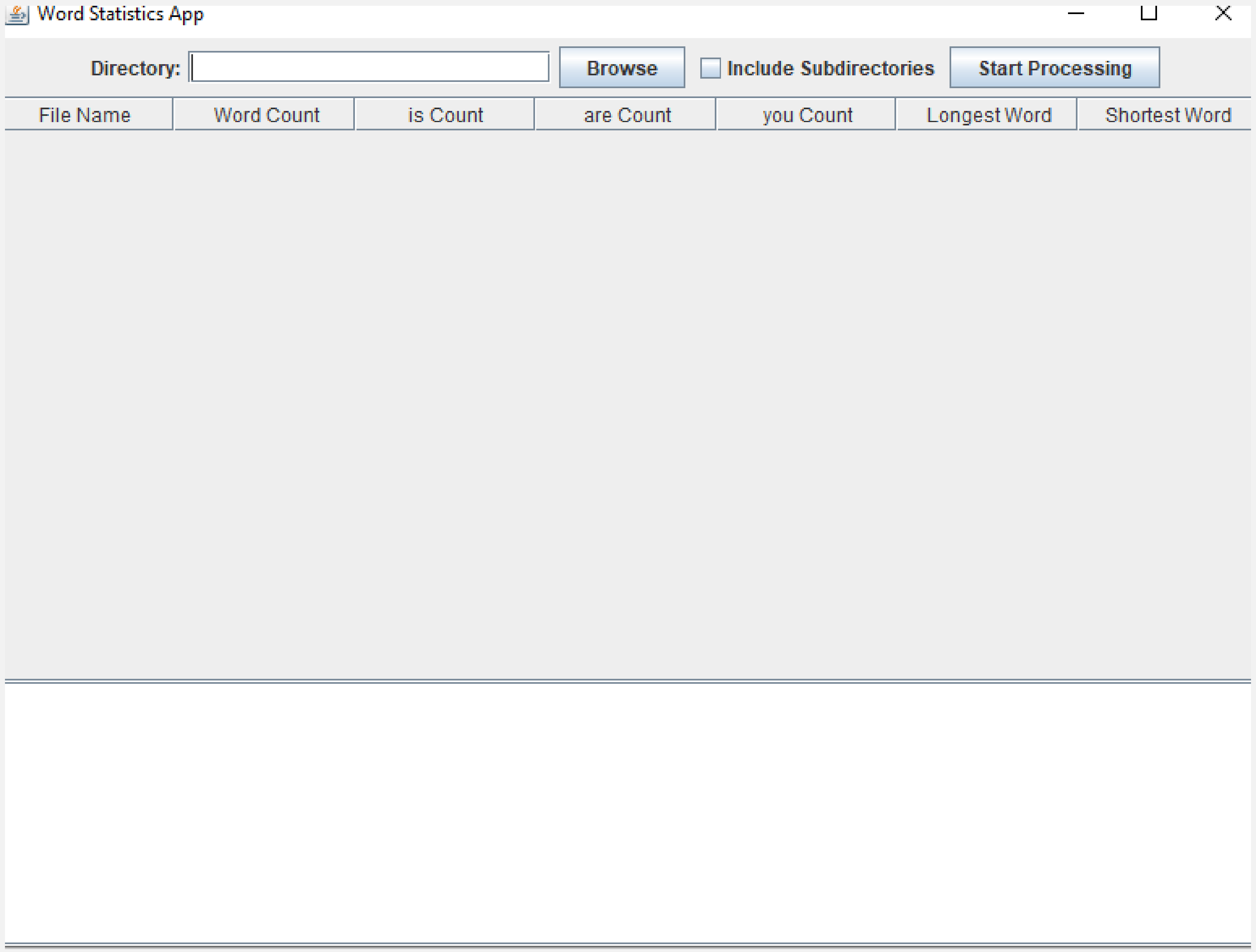
Implementation Details

- This method is responsible for updating the overall statistics (counts and longest/shortest words) in a thread-safe manner using the provided mutex lock
- It adds the (is,are ,you)
- It updates the overall longest word if the current longest word is longer than the stored overall longest word.
- It updates the overall shortest word if the current shortest word is shorter than the stored overall shortest word.



```
private void updateOverallStatistics(int isCount, int areCount, int youCount, String currentLongestWord,
        String currentShortestWord) {
    synchronized (lock) {
        // Update overall counts
        overallIsCount += isCount;
        overallAreCount += areCount;
        overallYouCount += youCount;

        // Update overall longest word
        if (overallLongestWord.isEmpty() || currentLongestWord.length() > overallLongestWord.length()) {
            overallLongestWord = currentLongestWord;
        }

        // Update overall shortest word
        if (overallShortestWord.isEmpty() || currentShortestWord.length() < overallShortestWord.length()) {
            overallShortestWord = currentShortestWord;
        }
    }
}
```

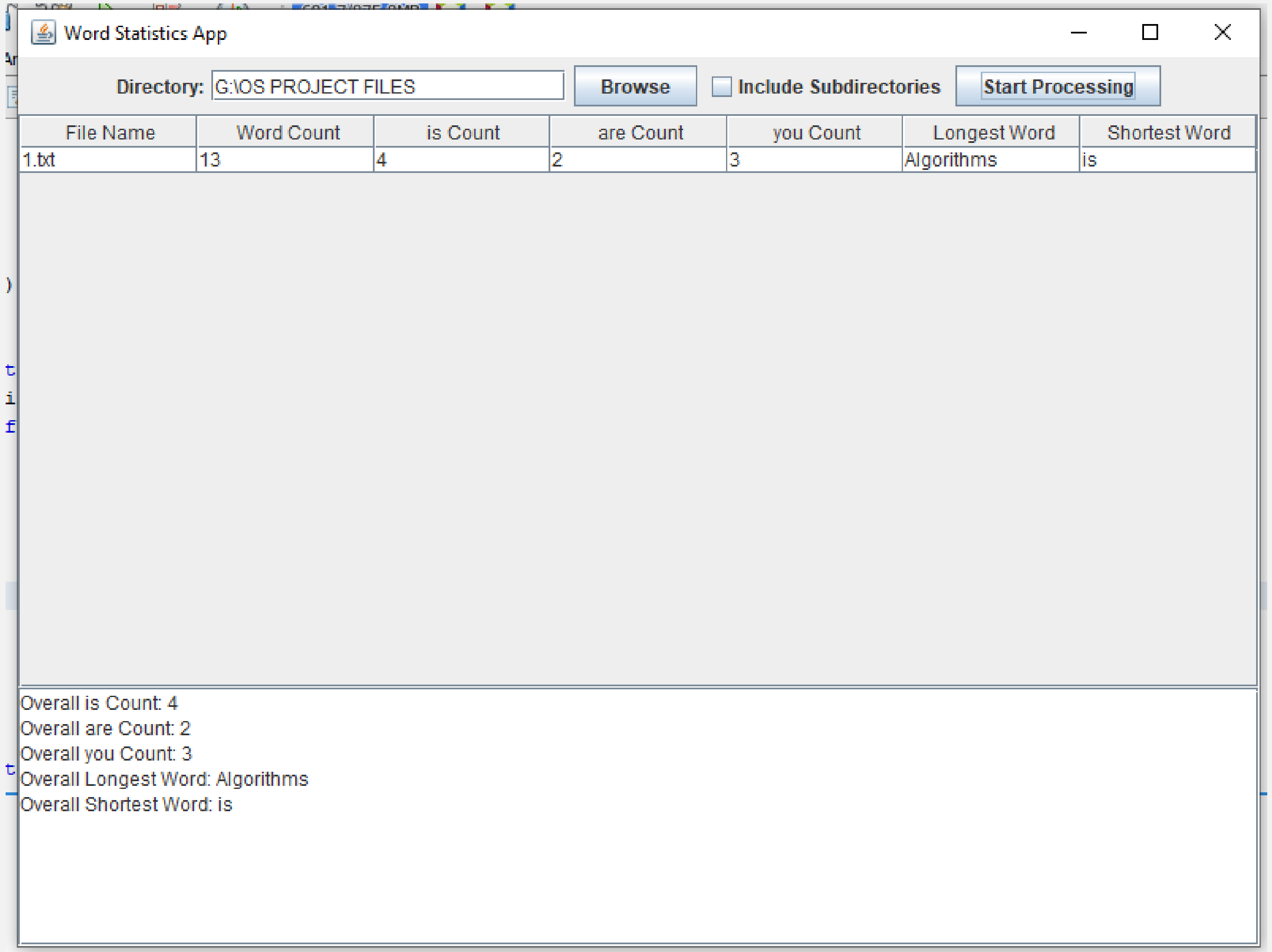
GUI



This is an example of components we have used in our project, you can Browse the folder you want and start processing and choose whether you want to show the subdirectories or not.

GUI

In this part I chooses a folder and started the processing, we se that it counts how many words, counts how many (is,are,you) and the longest word of the file and the shortest word



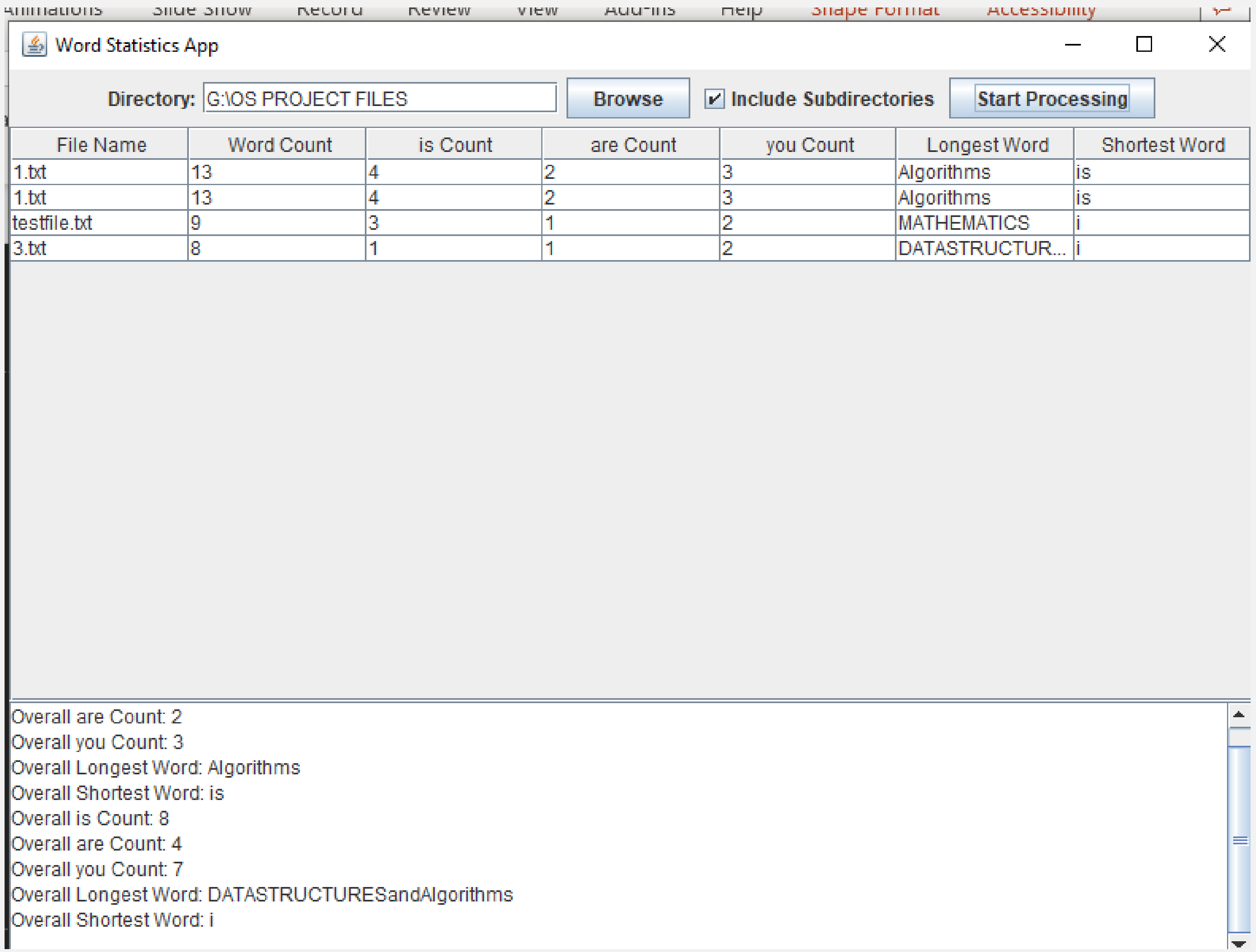
Directory: G:\OS PROJECT FILES

File Name	Word Count	is Count	are Count	you Count	Longest Word	Shortest Word
1.txt	13	4	2	3	Algorithms	is

```
Overall is Count: 4
Overall are Count: 2
Overall you Count: 3
Overall Longest Word: Algorithms
Overall Shortest Word: is
```

GUI

In this part I clicked on the subdirectories, so we see that the other files are added and at the bottom there is the overall longest world in all files and the overall shortest word in all files



Directory: G:\OS PROJECT FILES

[✓] Include Subdirectories

File Name	Word Count	is Count	are Count	you Count	Longest Word	Shortest Word
1.txt	13	4	2	3	Algorithms	is
1.txt	13	4	2	3	Algorithms	is
testfile.txt	9	3	1	2	MATHEMATICS	i
3.txt	8	1	1	2	DATASTRUCTUR...	i

```
Overall are Count: 2
Overall you Count: 3
Overall Longest Word: Algorithms
Overall Shortest Word: is
Overall is Count: 8
Overall are Count: 4
Overall you Count: 7
Overall Longest Word: DATASTRUCTURESandAlgorithms
Overall Shortest Word: i
```