**Aquino, Kelvin Yap COMP1000  
10736109**

Part A – Command Line Tool

A1: Open, Read, and Display a Text File

Methodology:

1. Compile and run the command line tool with the argument “lorum.txt”.
2. Check the console for the displayed text.

Expected Result: The content of the file should be displayed in the terminal.  
Actual Result: The content of the file is displayed as expected.

A screenshot of a computer screen

Description automatically generated

A2: Specify and Perform a Search

Methodology:

1. I added the word “sample” in the lorum.txt file.
2. Compile and run the command line tool with arguments “lorum.txt sample”.
3. The tool should search for the term "sample" in the file.

Expected Result: The tool should search for "sample" and process the occurrences.  
Actual Result: The word sample has been found at line 2 as expected.

A screenshot of a computer

Description automatically generated

A3: Display Search Results

Methodology:

1. I added the words “comp1000” in the lorum.txt file.
2. Compile and run the command line tool with arguments “lorum.txt comp1000”.
3. Check the console for the average word length.

Expected Result: The results should include the line number and word number for each match.  
Actual Result: The results include line number and word number as expected.

A computer screen with white text

Description automatically generated

A4: Display Search Statistics

Methodology:

1. Compile and run the command line tool with arguments “lorum.txt comp1000”.
2. Observe if the tool calculates and displays the number of search hits as a percentage of the total number of words.

Expected Result: The number of search hits as a percentage of the total words should be displayed.  
Actual Result: The results display total words, total matches, and percentage of total number as expected.  
A screen shot of a computer

Description automatically generated

A5: Save Results

Methodology:

1. Compile and run the command line tool with arguments lorum.txt sample.
2. Check the results.csv file for an entry with the filename, search term, and frequency.

Expected Result: The results.csv file should have an entry with the filename, search term, and frequency of hits (%).  
Actual Result: Results.csv contained entries with filename, search term, and frequency of hits as expected.

A screen shot of a computer program

Description automatically generated

A6: Class Library

Methodology:

1. Check if the code is organized into classes.
2. Verify that each class is encapsulated and handles specific functionality.
3. Ensure the use of object-oriented principles like encapsulation, abstraction, inheritance, and polymorphism where applicable.

Expected Result:

1. The solution should include well-defined classes such as “TextFile”, “Search”, “Statistics”, and “Readability”.
2. Each class should encapsulate relevant data and methods.
3. The main program should create instances of these classes and use them to perform the required operations.

Actual Result: Verify the presence of these classes in the code and their proper usage.

1. TextFile Class: Handles file operations.

A screen shot of a computer code

Description automatically generated

* Encapsulation: The “lines” member variable is private, and access to it is provided through public member functions.
* Responsibility: This class is responsible for loading and providing access to the lines of text from a file.

1. Search Class: Handles search operations.

A computer code on a black background

Description automatically generated

* Encapsulation: The “searchTerm”, “isRegex”, “results”, and “matchCount” are encapsulated within the class.
* Responsibility: This class is responsible for performing the search and storing the results.

1. Statistics Class: Handles statistical calculations.

A screen shot of a computer code

Description automatically generated

* Encapsulation: The “wordCounts” member variable is private, and access to it is provided through public member functions.
* Responsibility: This class is responsible for calculating word counts, average word length, and providing the top N words.

1. Readability Class: Handles readability calculations.

A screen shot of a computer code

Description automatically generated

* Encapsulation: This class provides static methods for calculating readability scores, encapsulating the logic within the class.
* Responsibility: This class is responsible for calculating readability metrics.

Justification for Object-Oriented Techniques:

1. Encapsulation:

* Each class encapsulates data and methods relevant to its functionality, making the code modular and easy to manage.

1. Abstraction:

* By abstracting file operations, search functionality, and statistical calculations into separate classes, the code becomes more readable and maintainable.

1. Modularity:

* The modular approach allows for easy testing and debugging of individual components.

1. Reusability:

* The classes can be reused in other projects or extended with additional features without affecting other parts of the code.

Conclusion

By structuring the solution with well-defined classes and adhering to Object-Oriented principles, the code achieves modularity, encapsulation, and reusability. Each class handles a specific responsibility, making the solution easier to understand, maintain, and extend. The main function coordinates these classes to perform the overall task, demonstrating a clear separation of concerns and a robust design.

Part B – Graphical Interface

B1: Open, read, and store a text file

Methodology:

1. Run the GUI application.
2. Use the "Open" button to select and open lorum.txt.
3. Check if the file content is displayed in the GUI.

Expected Result: The contents of “lorum.txt” should be displayed in the GUI.  
Actual Result: The contents of "lorum.txt” is displayed in the GUI as expected.

A screenshot of a computer

Description automatically generated

B2: Specify a Search Term

Methodology:

1. Run the GUI application.
2. Enter a search term in the search input field.
3. Initiate the search using the "Search" button.

Expected Result: The application should search for the specified term.  
Actual Result: The application displays the specified term in the GUI as expected.

A black rectangular object with a black border

Description automatically generated

B3: Display Search Results

Methodology:

1. Run the GUI application.
2. Perform a search for the term "sample".
3. Check if the search results are displayed in the GUI.

Expected Result: The results should include the location of each match.  
Actual Result: The application displays the word “sample” and includes the location of the match as expected.

A black rectangular object with blue lines

Description automatically generated

B4: Display Search Statistics

Methodology:

1. Run the GUI application.
2. Perform a search for the term "sample".
3. Observe if the number of search hits is displayed in the GUI.

Expected Result: The number of search hits should be displayed in the GUI.  
Actual Result: The number of times the word “sample” was found is displayed in the GUI at the bottom left as expected.

A black rectangular object with blue lines

Description automatically generated

B5: Task B5: Save Results

Methodology:

1. Run the GUI application.
2. Perform a search and use the "Save Results" button.
3. Save the results to a specified location.
4. Verify the contents of the saved CSV file.

A black screen with white text

Description automatically generatedExpected Result: The results should be saved in the specified CSV file with the filename, search term, and frequency of hits (%).  
Actual Result: The results have been saved in a csv file with the filename, search term, and frequency of hits as expected.

A screenshot of a computer

Description automatically generated

GitHub Repository:

URL: <https://github.com/kjaquino1/comp1000final.git>