LockedMe – Virtual Key for Repositories

This document contains sections for:

The project is developed by Sheshank Mashetty

The code for this project is hosted at
https://github.com/sheshankmashetty7/SMProject

Sprints planning and Task completion

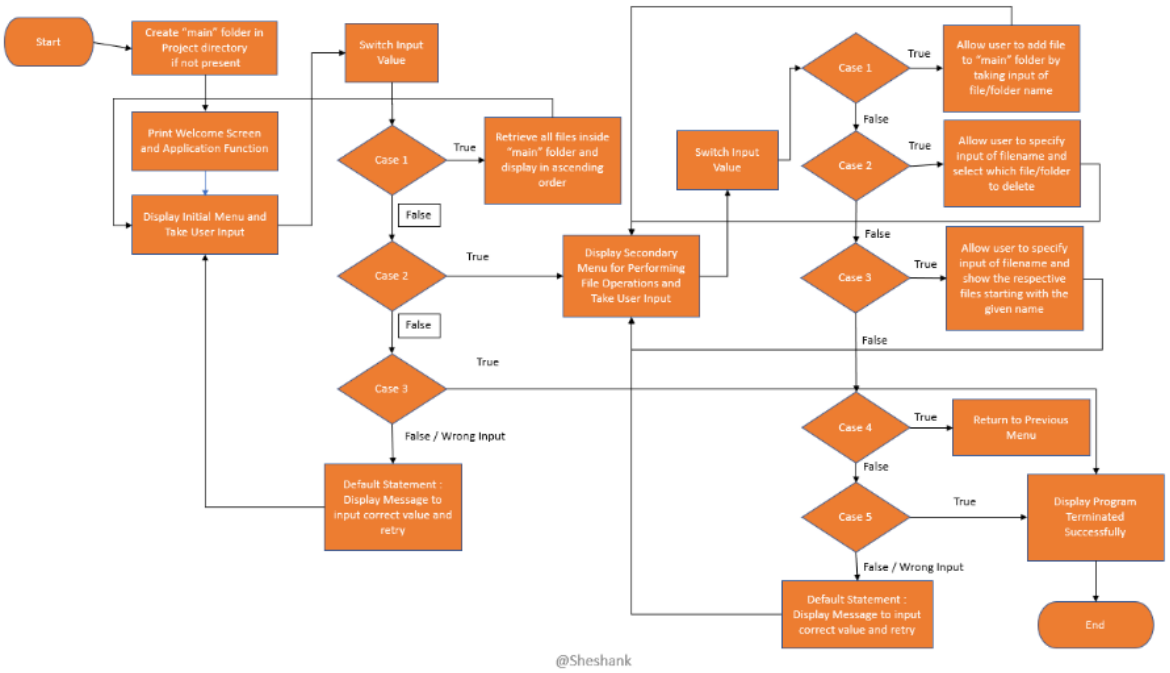
The project is planned to be completed in 1 sprint. Tasks assumed to be completed in the sprint are:

- Creating the flow of the application
- Initializing git repository to track changes as development progresses.
- Writing the Java program to fulfill the requirements of the project.
- Testing the Java program with different kinds of User input
- Pushing code to GitHub.
- Creating this specification document highlighting application capabilities, appearance, and user interactions.

Core concepts used in project

Collections framework, File Handling, Sorting, Flow Control, Recursion, Exception Handling, Streams API

Flow of the Application



Flow of Application.pptx

Demonstrating the product capabilities, appearance, and user interactions

To demonstrate the product capabilities, below are the sub-sections configured to highlight appearance and user interactions for the project:

1. Creating the project in Eclipse
2. Writing a program in Java for the entry point of the application (**LockedMeMain.java**)
3. Writing a program in Java to display Menu options available for the user (**MenuOptions.java**)

4 Writing a program in Java to handle Menu options selected by user (**HandleOptions.java**)
5 Writing a program in Java to perform the File operations as specified by user (**FileOperations.java**)
6 Pushing the code to GitHub repository

Step 1: Creating a new project in Eclipse

- Open Eclipse
- Go to File -> New -> Project -> Java Project -> Next.
- Type in any project name and click on “Finish.”
- Select your project and go to File -> New -> Class.
- Enter **LockedMeMain** in any class name, check the checkbox “public static void main(String[] args)”, and click on “Finish.”

Step 2: Writing a program in Java for the entry point of the application (**LockedMeMain.java**)

```
package sllockedme;

public class LockedMeMain {

	public static void main(String[] args) {

		// Create "main" folder if not present in current folder structure
		FileOperations.createMainFolderIfNotPresent("main");

		MenuOptions.printWelcomeScreen("LockedMe", "Sheshank");

		HandleOptions.handleWelcomeScreenInput();
	}


}
```

Step 3: Writing a program in Java to display Menu options available for the user (**MenuOptions.java**)

- Select your project and go to File -> New -> Class.
- Enter **MenuOptions** in class name and click on “Finish.”
- **MenuOptions** consists methods for -:

Step 3.1: Writing method to display Welcome Screen

```
public static void printWelcomeScreen(String appName, String developerName) {
		String companyDetails =
String.format("============================================================\n"
				+ "** Welcome to %s.com. \n" + "** This application was
developed by %s.\n"
				+
"============================================================\n", appName,
developerName);
		String appFunction = "You can use this application to :-\n"
				+ "• Access all file names in the \"main\" folder\n"
				+ "• Search, add, or delete files in \"main\" folder.\n"
				+ "\n**Please be careful to ensure the correct filename is
provided for searching or deleting files.**\n";
		System.out.println(companyDetails);

		System.out.println(appFunction);
}
```

Output:

```
|=====================================================
** Welcome to LockedMe.com.
** This application was developed by Sheshank.
=====================================================

You can use this application to :-
1. Retrieve all file names in the "main" folder
2. Search, add, or delete files in "main" folder.


====== Select any option number from below and press Enter ======

1) Retrieve all files inside "main" folder
2) Display menu for File operations
3) Exit program
```

Step 3.2: Writing method to display Initial Menu

```
public static void displayMenu() {
		String menu = "\n\n====== Select any option number from below and press Enter =======\n\n"
				+ "1) Retrieve all files inside \"main\" folder\n" + "2) Display menu for File operations\n"
				+ "3) Exit program\n";
		System.out.println(menu);

}
```

Output:

```
====== Select any option number from below and press Enter ======

1) Retrieve all files inside "main" folder
2) Display menu for File operations
3) Exit program
```

Step 3.3: Writing method to display Secondary Menu for File Operations

```
public static void displayFileMenuOptions() {
		String fileMenu = "\n\n====== Select any option number from below and press Enter ======\n\n"
				+ "1) Add a file to \"main\" folder\n" + "2) Delete a file from \"main\" folder\n"
				+ "3) Search for a file from \"main\" folder\n" + "4) Show Previous Menu\n" + "5) Exit program\n";

		System.out.println(fileMenu);
}
```

Output:

```
====== Select any option number from below and press Enter ======

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
5) Exit program
```

Step 4: Writing a program in Java to handle Menu options selected by user (**HandleOptions.java**)

- Select your project and go to File -> New -> Class.
- Enter **HandleOptions** in class name and click on “Finish.”
- **HandleOptions** consists methods for -:

4.1. Handling input selected by user in initial Menu

4.2. Handling input selected by user in secondary Menu for File Operations

Step 4.1: Writing method to handle user input in initial Menu

```
public static void handleWelcomeScreenInput() {
        boolean running = true;
        Scanner sc = new Scanner(System.in);
        do {
            try {
                MenuOptions.displayMenu();
                int input = sc.nextInt();

                switch (input) {
                case 1:
                    FileOperations.displayAllFiles("main");
                    break;
                case 2:
                    HandleOptions.handleFileMenuOptions();
                    break;
                case 3:
```

```
                                System.out.println("Program exited successfully.");
                                running = false;
                                sc.close();
                                System.exit(0);
                                break;
                        default:
                                System.out.println("Please select a valid option from
above.");
                        }
                } catch (Exception e) {
                        System.out.println(e.getClass().getName());
                        handleWelcomeScreenInput();
                }
        } while (running == true);
    }
```

Output:

```
====== Select any option number from below and press Enter ======

1) Retrieve all files inside "main" folder
2) Display menu for File operations
3) Exit program

1
Displaying all files with directory structure in ascending order

|-- s1.txt
|-- s2.txt
|-- s3.txt
|-- s4.txt
`-- sm
  |-- sm1.txt
  |-- sm2.txt
  |-- sm3.txt
  `-- ssm
    |-- sm4.txt


Displaying all files in ascending order

s1.txt
s2.txt
s3.txt
s4.txt
sm
sm1.txt
sm2.txt
sm3.txt
sm4.txt
ssm
```

Step 4.2: Writing method to handle user input in Secondary Menu for File Operations

```
public static void handleFileMenuOptions() {
		boolean running = true;
		Scanner sc = new Scanner(System.in);
		do {
			try {
				MenuOptions.displayFileMenuOptions();
				FileOperations.createMainFolderIfNotPresent("main");

				int input = sc.nextInt();
				switch (input) {
				case 1:
					// File Add
					System.out.println("Enter the name of the file to be
added to the \"main\" folder");
					String fileToAdd = sc.next();

					FileOperations.createFile(fileToAdd, sc);

					break;
				case 2:
					// File/Folder delete
					System.out.println("Enter the name of the file to be
deleted from \"main\" folder");
					String fileToDelete = sc.next();

					FileOperations.createMainFolderIfNotPresent("main");
					List<String> filesToDelete =
FileOperations.displayFileLocations(fileToDelete, "main");

					String deletionPrompt = "\nSelect index of which file to
delete?"
							+ "\n(Enter 0 if you want to delete all
elements)";
					System.out.println(deletionPrompt);

					int idx = sc.nextInt();

					if (idx != 0) {

	FileOperations.deleteFileRecursively(filesToDelete.get(idx - 1));
					} else {
```

```
                                        // If idx == 0, delete all files displayed for the
name
                                        for (String path : filesToDelete) {
                                                FileOperations.deleteFileRecursively(path);
                                        }
                                }


                                break;
                        case 3:
                                // File/Folder Search
                                System.out.println("Enter the name of the file to be
searched from \"main\" folder");
                                String fileName = sc.next();

                                FileOperations.createMainFolderIfNotPresent("main");
                                FileOperations.displayFileLocations(fileName, "main");


                                break;
                        case 4:
                                // Go to Previous menu
                                return;
                        case 5:
                                // Exit
                                System.out.println("Program exited successfully.");
                                running = false;
                                sc.close();
                                System.exit(0);
                        default:
                                System.out.println("Please select a valid option from
above.");
                        }
                } catch (Exception e) {
                        System.out.println(e.getClass().getName());
                        handleFileMenuOptions();
                }
        } while (running == true);
    }
```

Output:

```
====== Select any option number from below and press Enter ======

1) Retrieve all files inside "main" folder
2) Display menu for File operations
3) Exit program

2


====== Select any option number from below and press Enter ======

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
5) Exit program

3
Enter the name of the file to be searched from "main" folder
sm1


Found file at below location(s):
1: /Users/shashanksureshmashetty/Java/SMProject/main/sm/sm1.txt
```

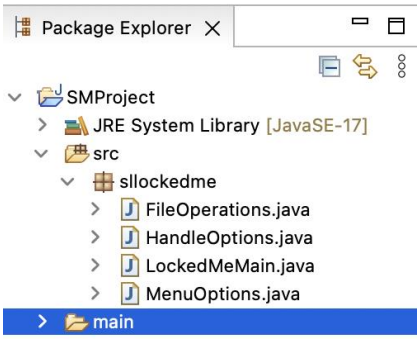
Step 5: Writing a program in Java to perform the File operations as specified by user (**FileOperations.java**)

- Select your project and go to File -> New -> Class.
- Enter **FileOperations** in class name and click on “Finish.”
- **FileOperations** consists methods for -:

Step 5.1: Writing method to create “main” folder in project if it’s not present

```
public static void createMainFolderIfNotPresent(String folderName) {
	File file = new File(folderName);

	// If file doesn't exist, create the main folder
	if (!file.exists()) {
		file.mkdirs();
	}
}
```

Output:



Step 5.2: Writing method to display all files in “main” folder in ascending order and also with directory structure. (“`--" represents a directory. “|--” represents a file.)

```
public static void displayAllFiles(String path) {
		FileOperations.createMainFolderIfNotPresent("main");
		// All required files and folders inside "main" folder relative to current
		// folder
		System.out.println("Displaying all files with directory structure in ascending
order\n");
```

```
		// listFilesInDirectory displays files along with folder structure
		List<String> filesListNames = FileOperations.listFilesInDirectory(path, 0, new
ArrayList<String>());

		System.out.println("Displaying all files in ascending order\n");
		Collections.sort(filesListNames);

		filesListNames.stream().forEach(System.out::println);
	}

	public static List<String> listFilesInDirectory(String path, int indentationCount,
List<String> fileListNames) {
		File dir = new File(path);
		File[] files = dir.listFiles();
		List<File> filesList = Arrays.asList(files);

		Collections.sort(filesList);

		if (files != null && files.length > 0) {
			for (File file : filesList) {

				System.out.print(" ".repeat(indentationCount * 2));

				if (file.isDirectory()) {
					System.out.println("`-- " + file.getName());

					// Recursively indent and display the files
					fileListNames.add(file.getName());
					listFilesInDirectory(file.getAbsolutePath(),
indentationCount + 1, fileListNames);
				} else {
					System.out.println("|-- " + file.getName());
					fileListNames.add(file.getName());
				}
			}
		} else {
			System.out.print(" ".repeat(indentationCount * 2));
			System.out.println("|-- Empty Directory");
		}
		System.out.println();
		return fileListNames;
	}
```

Output:

```
====== Select any option number from below and press Enter ======

1) Retrieve all files inside "main" folder
2) Display menu for File operations
3) Exit program

1
Displaying all files with directory structure in ascending order

|-- s1.txt
|-- s2.txt
|-- s3.txt
|-- s4.txt
`-- sm
  |-- sm1.txt
  |-- sm2.txt
  |-- sm3.txt
  `-- ssm
    |-- sm4.txt



Displaying all files in ascending order

s1.txt
s2.txt
s3.txt
s4.txt
sm
sm1.txt
sm2.txt
sm3.txt
sm4.txt
ssm
```

Step 5.3: Writing method to create a file/folder as specified by user input.

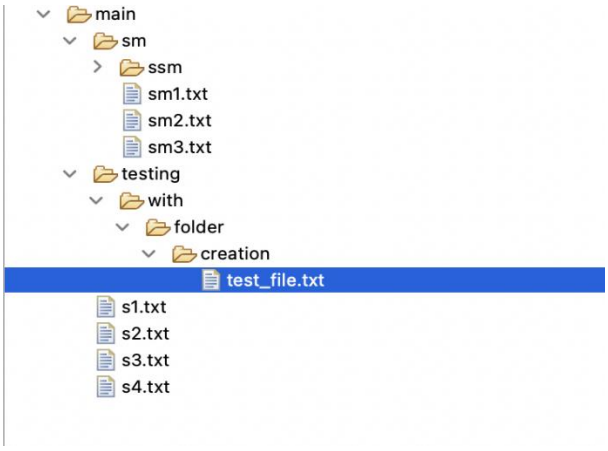
```
public static void createFile(String fileToAdd, Scanner sc) {
```

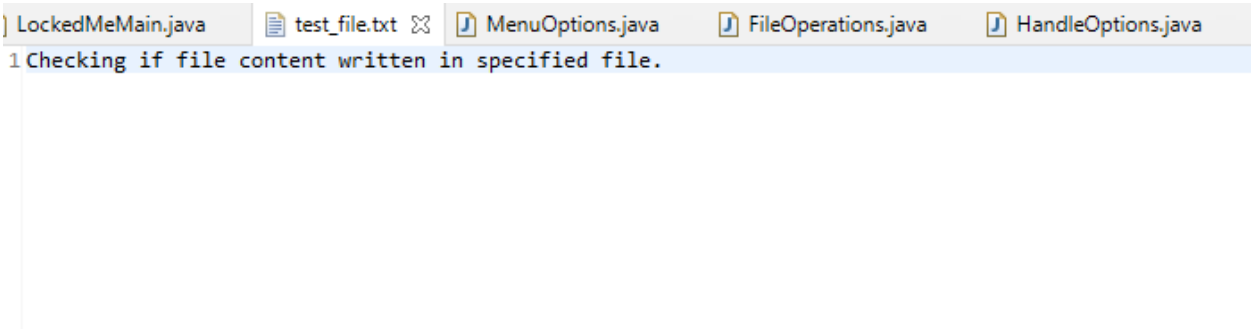
```
                FileOperations.createMainFolderIfNotPresent("main");
                Path pathToFile = Paths.get("./main/" + fileToAdd);
                try {
                        Files.createDirectories(pathToFile.getParent());
                        Files.createFile(pathToFile);
                        System.out.println(fileToAdd + " created successfully");

                        System.out.println("Would you like to add some content to the file?
(Y/N)");
                        String choice = sc.next().toLowerCase();

                        sc.nextLine();
                        if (choice.equals("y")) {
                                System.out.println("\n\nInput content and press enter\n");
                                String content = sc.nextLine();
                                Files.write(pathToFile, content.getBytes());
                                System.out.println("\nContent written to file " + fileToAdd);
                                System.out.println("Content can be read using Notepad or
Notepad++");
                        }

                } catch (IOException e) {
                        System.out.println("Failed to create file " + fileToAdd);
                        System.out.println(e.getClass().getName());
                }
        }
```

Output:

Folders are automatically created along with file

```
====== Select any option number from below and press Enter ======

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
5) Exit program

1
Enter the name of the file to be added to the "main" folder
/testing/with/folder/creation/test_file.txt
/testing/with/folder/creation/test_file.txt created successfully
Would you like to add some content to the file? (Y/N)
Y


Input content and press enter

Checking if file content is written in specified file.

Content written to file /testing/with/folder/creation/test_file.txt
Content can be read using Notepad or Notepad++
```





Step 5.4: Writing method to search for all files as specified by user input in “main” folder and it’s subfolders.

```
public static List<String> displayFileLocations(String fileName, String path) {
		List<String> fileListNames = new ArrayList<>();
		FileOperations.searchFileRecursively(path, fileName, fileListNames);

		if (fileListNames.isEmpty()) {
			System.out.println("\n\n***** Couldn't find any file with given file
name \"" + fileName + "\" *****\n\n");
		} else {
			System.out.println("\n\nFound file at below location(s):");

			List<String> files = IntStream.range(0, fileListNames.size())
					.mapToObj(index -> (index + 1) + ": " +
fileListNames.get(index)).collect(Collectors.toList());

			files.forEach(System.out::println);
		}

		return fileListNames;
	}

	public static void searchFileRecursively(String path, String fileName, List<String>
fileListNames) {
		File dir = new File(path);
		File[] files = dir.listFiles();
		List<File> filesList = Arrays.asList(files);

		if (files != null && files.length > 0) {
			for (File file : filesList) {

				if (file.getName().startsWith(fileName)) {
					fileListNames.add(file.getAbsolutePath());
				}

				// Need to search in directories separately to ensure all files of
required
				// fileName are searched
				if (file.isDirectory()) {
					searchFileRecursively(file.getAbsolutePath(), fileName,
fileListNames);
				}
			}
		}
	}
```

Output:

All files starting with the user input are displayed along with index

```
====== Select any option number from below and press Enter ======

1) Retrieve all files inside "main" folder
2) Display menu for File operations
3) Exit program

2


====== Select any option number from below and press Enter ======

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
5) Exit program

3
Enter the name of the file to be searched from "main" folder
sm1


Found file at below location(s):
1: /Users/shashanksureshmashetty/Java/SMProject/main/sm/sm1.txt
```

Step 5.5: Writing method to delete file/folder specified by user input in “main” folder and it’s subfolders. It uses the searchFilesRecursively method and prompts user to specify which index to delete. If folder selected, all it’s child files and folder will be deleted recursively. If user wants to delete all the files specified after the search, they can input value 0.

```
public static void deleteFileRecursively(String path) {

        File currFile = new File(path);
        File[] files = currFile.listFiles();
```

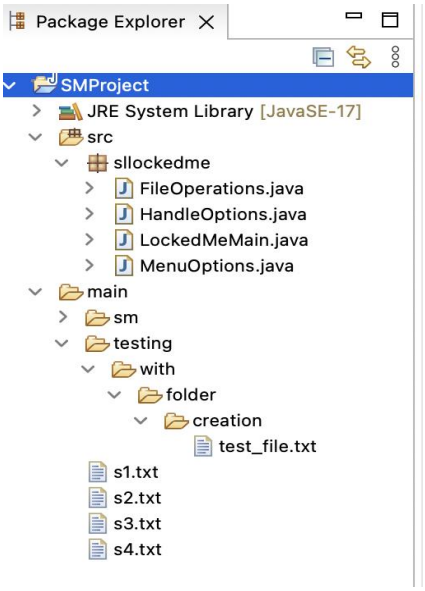
```
            if (files != null && files.length > 0) {
                for (File file : files) {

                    String fileName = file.getName() + " at " + file.getParent();
                    if (file.isDirectory()) {
                        deleteFileRecursively(file.getAbsolutePath());
                    }

                    if (file.delete()) {
                        System.out.println(fileName + " deleted successfully");
                    } else {
                        System.out.println("Failed to delete " + fileName);
                    }
                }
            }

            String currFileName = currFile.getName() + " at " + currFile.getParent();
            if (currFile.delete()) {
                System.out.println(currFileName + " deleted successfully");
            } else {
                System.out.println("Failed to delete " + currFileName);
            }
        }
```

Output:

To verify if file is deleted on Eclipse, right click on Project and click “Refresh”.

```
2


====== Select any option number from below and press Enter ======

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
5) Exit program

2
Enter the name of the file to be deleted from "main" folder
s4


Found file at below location(s):
1: /Users/shashanksureshmashetty/Java/SMProject/main/s4.txt

Select index of which file to delete?
(Enter 0 if you want to delete all elements)
0
s4.txt at /Users/shashanksureshmashetty/Java/SMProject/main deleted successfully


====== Select any option number from below and press Enter ======

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
5) Exit program
```

```
====== Select any option number from below and press Enter ======

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
5) Exit program

4


====== Select any option number from below and press Enter ======

1) Retrieve all files inside "main" folder
2) Display menu for File operations
3) Exit program

1
Displaying all files with directory structure in ascending order

|-- s1.txt
|-- s2.txt
|-- s3.txt
`-- sm
  |-- sm1.txt
  |-- sm2.txt
  |-- sm3.txt
  `-- ssm
    |-- sm4.txt


`-- testing
  `-- with
    `-- folder
      `-- creation
        |-- test_file.txt




Displaying all files in ascending order

creation
folder
s1.txt
s2.txt
s3.txt
sm
sm1.txt
sm2.txt
sm3.txt
sm4.txt
ssm
test_file.txt
testing
with
```



Step 6: Pushing the code to GitHub repository

- Open your command prompt and navigate to the folder where you have created your files.

  cd <folder path>

- Initialize repository using the following command:

  git init

- Add all the files to your git repository using the following command:

  git add .

- Commit the changes using the following command:

  git commit . -m <commit message>

- Push the files to the folder you initially created using the following command:

  git push -u origin master

Unique Selling Points of the Application

1. The application is designed to keep on running and taking user inputs even after exceptions occur. To terminate the application, appropriate option needs to be selected.

2. The application can take any file/folder name as input. Even if the user wants to create nested folder structure, user can specify the relative path, and the application takes care of creating the required folder structure.

3. User is also provided the option to write content if they want into the newly created file.

4. The application doesn't restrict user to specify the exact filename to search/delete file/folder. They can specify the starting input, and the program searches all files/folder starting with the value and displays it. The user is then provided the option to select all files or to select a specific index to delete.

5. The application also allows user to delete folders which are not empty.

6. The user is able to seamlessly switch between options or return to previous menu even after any required operation like adding, searching, deleting or retrieving of files is performed.

7. When the option to retrieve files in ascending order is selected, user is displayed with two options of viewing the files.

    7.1. Ascending order of folders first which have files sorted in them,
    7.2. Ascending order of all files and folders inside the "main" folder.

8. The application is designed with modularity in mind. Even if one wants to update the path, they can change it through the source code. Application has been developed keeping in mind that there should be very less "hardcoding" of data.

Conclusions

Further enhancements to the application can be made which may include:

- Conditions to check if user is allowed to delete the file or add the file at the specific locations.

- Asking user to verify if they really want to delete the selected directory if it’s not empty.
- Retrieving files/folders by different criteria like Last Modified, Type, etc.
- Allowing user to append data to the file.