LockedMe.com

Virtual Key for your Repositories

Developer Details

This project was developed by Ashraf M. Husain and stored in a repository in GitHub created using the email account ashhusai@cisco.com.

GitHub Repo Link: https://github.com/AshrafMH22/LockedMe.git

Project Details

The project consists of an application that displays a welcome screen and performs the following operations:

- Retrieving the file names in an ascending order
- Business-level operations:
  - Option to add a user specified file to the application
  - Option to delete a user specified file from the application
  - Option to search a user specified file from the application
  - Navigation option to close the current execution context and return to the main context
- Option to close the application

Sprint Planning

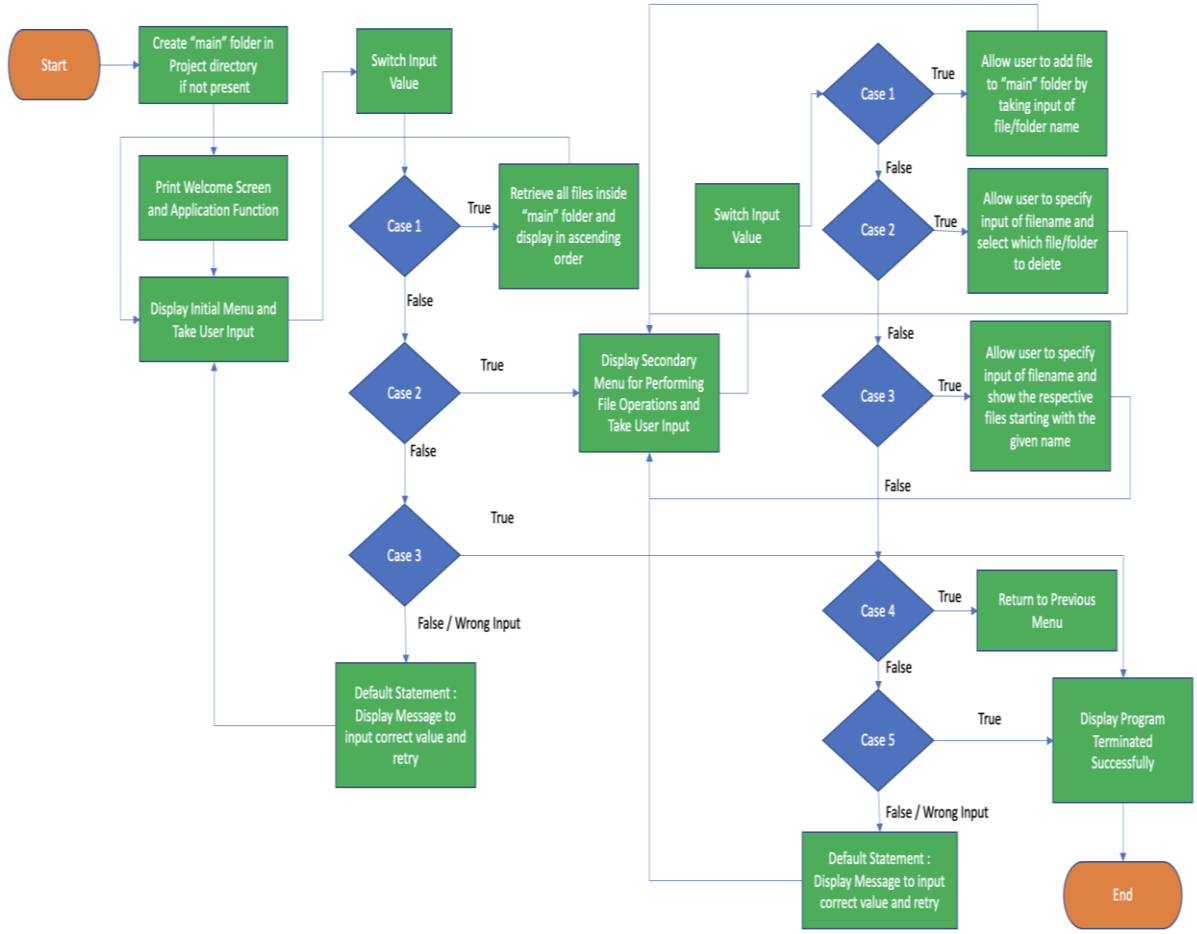
In order to complete the project, a single sprint with a time frame of 1 week was agreed to be enough. The tasks achieved during the sprint are listed as below:
• Determining the flow of the application and algorithms required for the proper functioning of the application
• Setting up a git repository to monitor changes as development moves forward.
• Writing the Java application to match the purpose of the prototype.
• Pushing the code to GitHub
• Testing the Java program with various forms of user input.
• Producing a specification document that highlights the application capabilities, appearance, and user interactions.

Core Concepts used in the Project

- Object Oriented Concepts
- Searching and Sorting Techniques
- File Handling
- Exception Handling
- Data Structures such as Lists and Collections

- Recursion

Flowchart of the Application



Demonstrating the product capabilities, appearance, and user interactions

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 VirtualKeyforRepo;

public class LockedMeMain {

	public static void main(String[] args) {

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

		WelcomeScreen.welcScreen("LockedMe", "Ashraf M. Husain");

		MethodSelector.priMenuOptions();
	}


}
```

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

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

3.1. Displaying the Welcome Screen

3.1. Displaying Primary menu

3.2. Displaying Secondary Menu for file operations available

Step 3.1: Writing method to display Welcome Screen

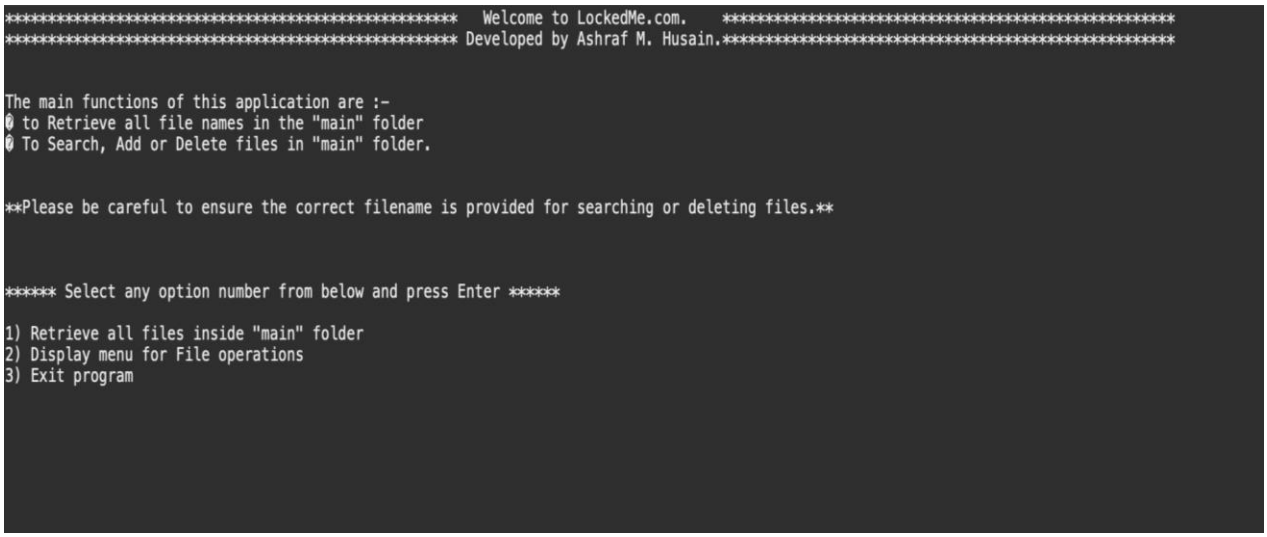
```
	public static void welcScreen(String appName, String developerName) {

	System.out.printf("**********************************************************"
				+ "   Welcome to %s.com.
**********************************************************\n"
				+
"********************************************************** Developed by %s."
				+
"**********************************************************\n", appName,
developerName);
```

```
		System.out.println("\n\nThe main functions of this application are :-\n"
					+ "? To Retrieve all file names in the \"main\" folder\n"
					+ "? To Search, Add or Delete files in \"main\" folder.\n"
					+ "\n\n**Please be careful to ensure the correct filename is
provided for searching or deleting files.**\n");

	}
```

Step 3.2: Writing method to display Primary Menu

```
public static void primaryMenu() {
		System.out.println("\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");

	}
```

Output:



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

```
public static void secondaryMenu() {
		System.out.println("\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");
	}
```

}

```
****** 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 select the appropriate method to be executed according to the menu option selected by the user (MethodSelector.java)

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

4.1. Handling input selected by user in Primary Menu

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

Step 4.1: Writing method to handle user input in Primary Menu.

```
public static void priMenuOptions() {
        boolean contInUE = true;
        Scanner sc = new Scanner(System.in);
        do {
            try {
                WelcomeScreen.primaryMenu();
                int input = sc.nextInt();

                switch (input) {
                case 1:
                    FileOps.displayAllFiles("main");
                    break;
                case 2:
```

```
                                MethodSelector.secMenuOptions();
                                break;
                        case 3:
                                System.out.println("Program exited successfully.");
                                continUE = 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());
                        priMenuOptions();
                }
        } while (continUE == true);
    }
```

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

```
    public static void secMenuOptions() {
        boolean running = true;
        Scanner sc = new Scanner(System.in);
        do {
            try {
                WelcomeScreen.secondaryMenu();
                FileOps.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();

                    FileOps.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();

                        FileOps.createMainFolderIfNotPresent("main");
                        List<String> filesToDelete =
FileOps.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) {

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

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

                        FileOps.createMainFolderIfNotPresent("main");
                        FileOps.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());
                                secMenuOptions();
                        }
                } while (running == true);
        }
```

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

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

5.1. Creating "main" folder in project if it's not already present

5.2. Displaying all files in "main" folder in ascending order and also with directory structure.

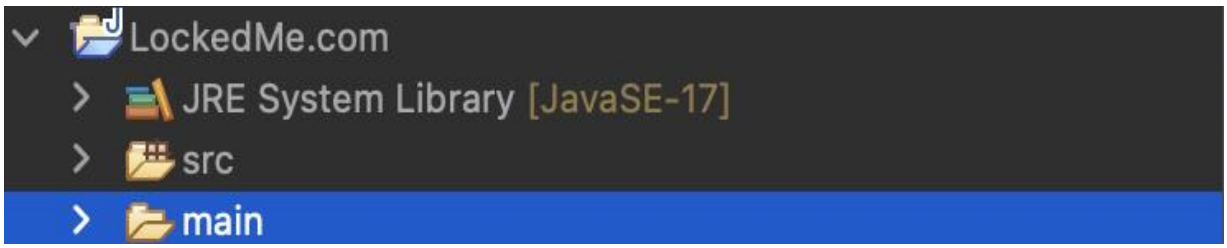
5.3. Creating a file/folder as specified by user input.

5.4. Search files as specified by user input in "main" folder and it's subfolders.

5.5. Deleting a file/folder from "main" folder

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();
                }
        }
```



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

```
public static void displayAllFiles(String path) {
		FileOps.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 = FileOps.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

|-- .DS_Store
|-- Dumbell.txt
|-- Raccoon.txt
`-- Something
  |-- Rocket.txt

`-- Test
  |-- Testing.txt


Displaying all files in ascending order

.DS_Store
Dumbell.txt
Raccoon.txt
Rocket.txt
Something
Test
Testing.txt


****** 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
```

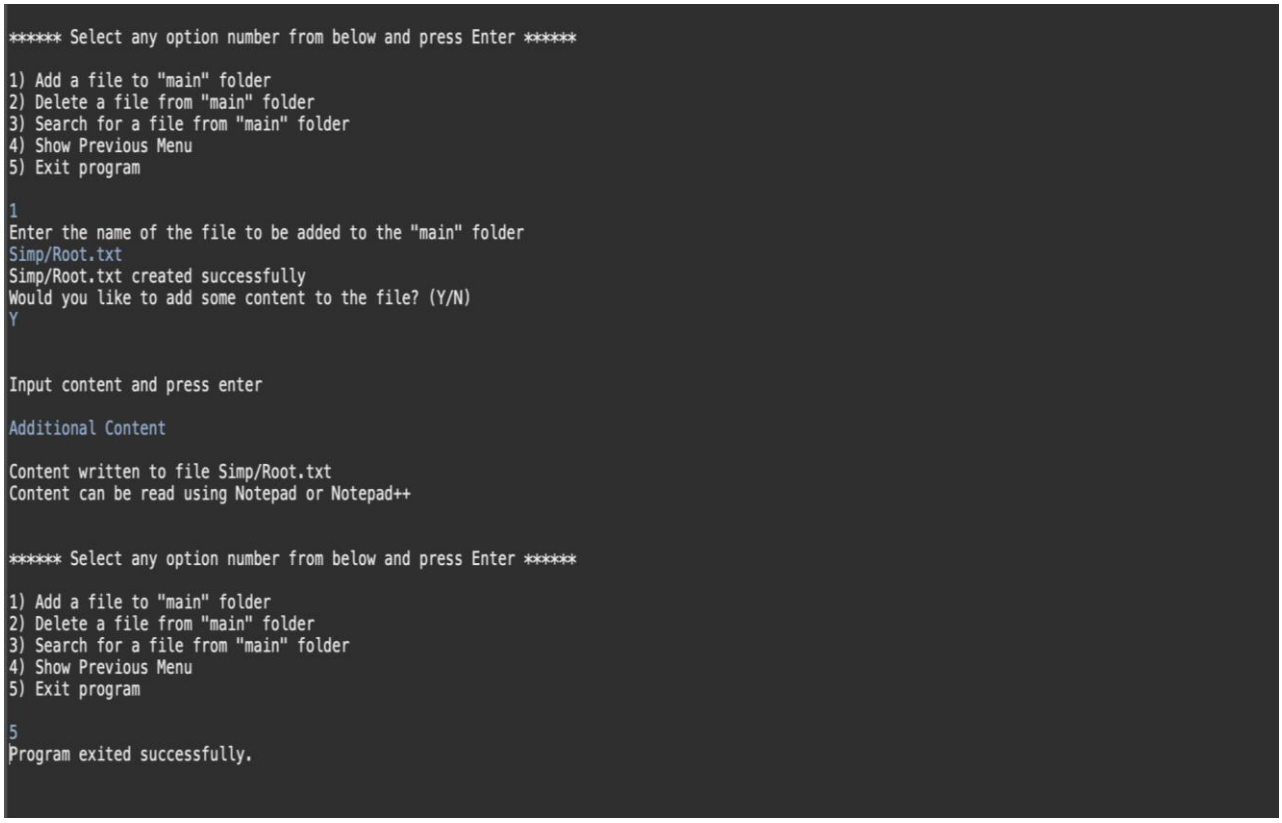
Step 5.3: Writing method to create a file/folder as specified by user input.

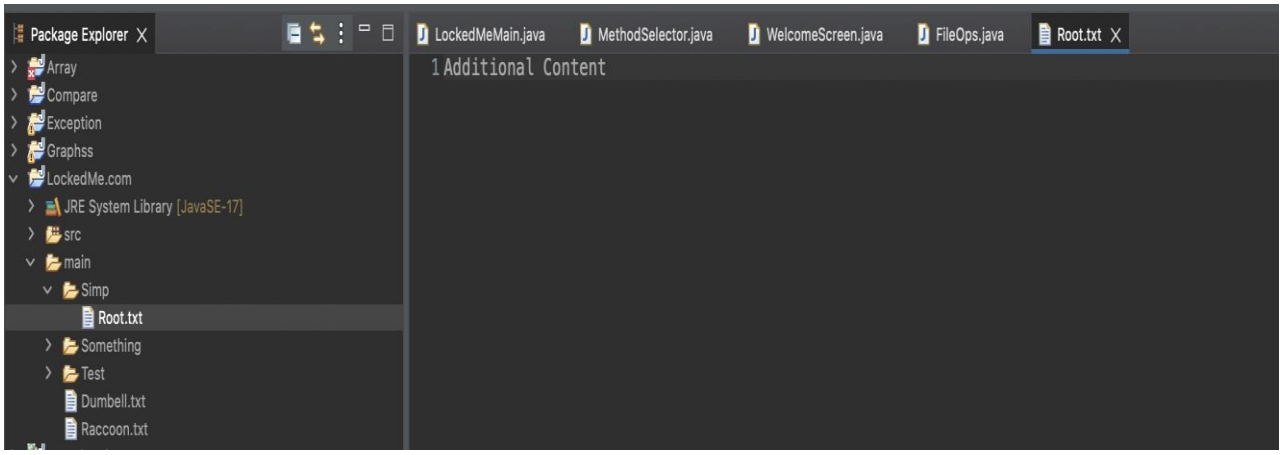
```
public static void createFile(String fileToAdd, Scanner sc) {
		FileOps.createMainFolderIfNotPresent("main");
		Path pathToFile = Paths.get("./main/" + fileToAdd);
		try {
			Files.createDirectories(pathToFile.getParent());
			Files.createFile(pathToFile);
			System.out.println(fileToAdd + " created 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



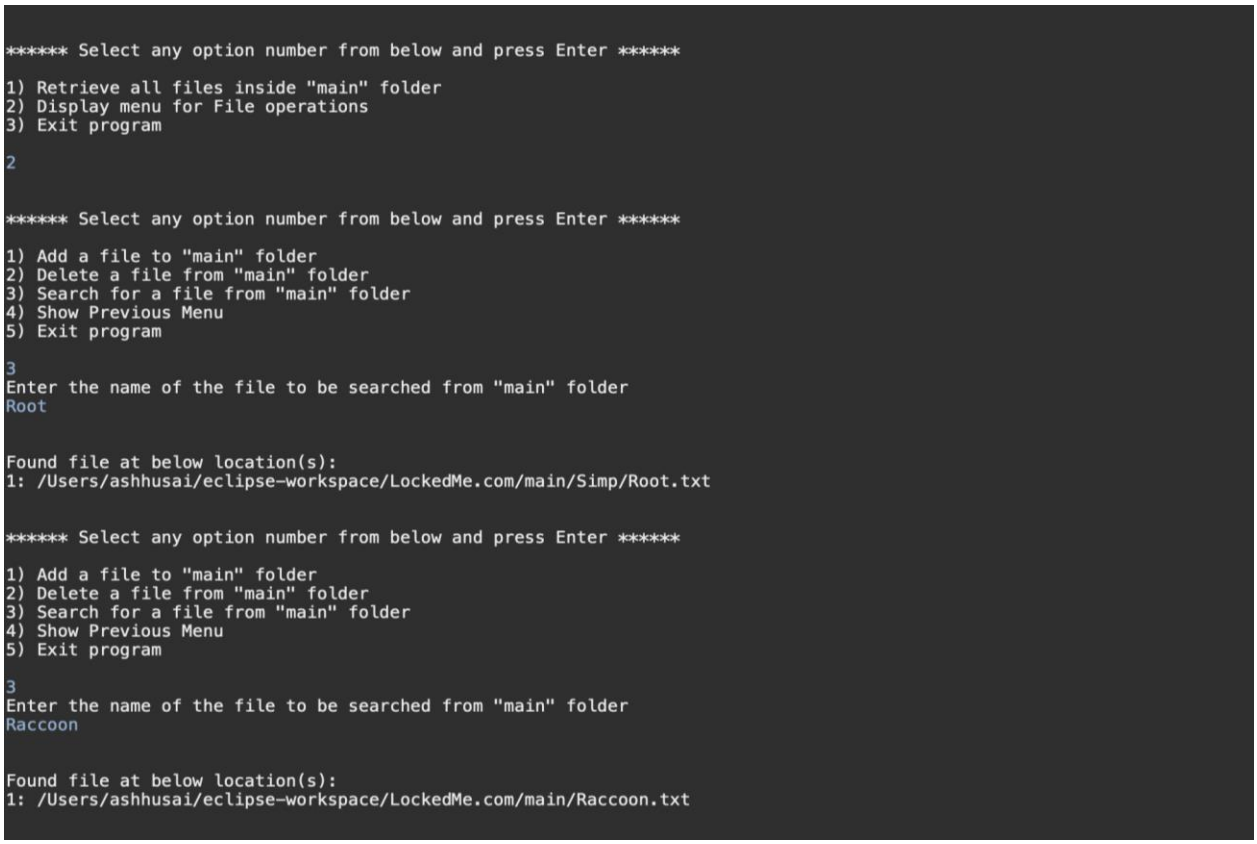


Step 5.4: Writing method to search for all files as specified by user input in “main” folder and it’s subfolders.

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



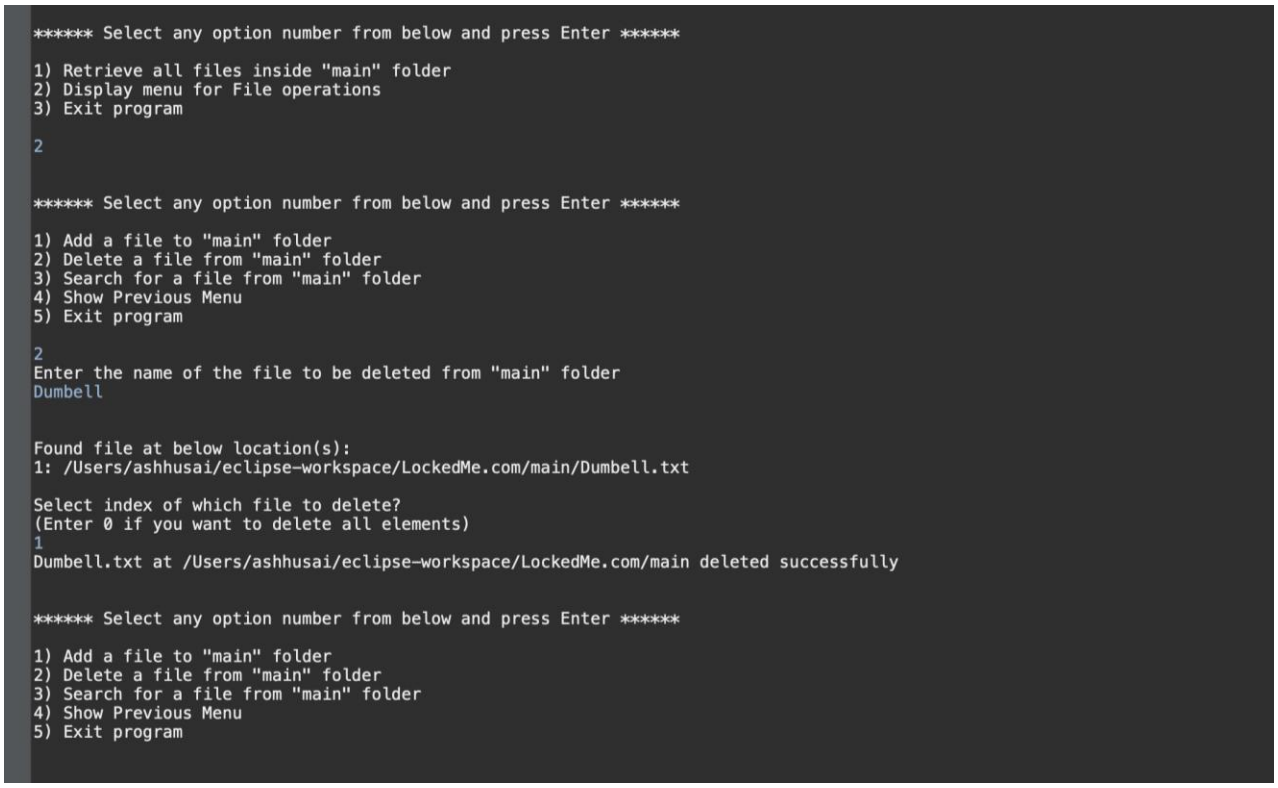
Step 5.5: Writing method to delete file/folder specified by user input in "main" folder and it's subfolders. It uses the searchFilesRecursively method and prompts user to specify which index to delete. If folder selected, all it's child files and folder will be deleted recursively. If user wants to delete 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".



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>

- Link the remote and local repositories

  git remote add origin <remote URL>

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

  git push -u origin master

Unique Selling Points

1. Any file or folder name can be entered into the program. 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.

2. If the user chooses, they have the option of adding content to the newly created file.

3. The application allows users to search for and remove files and folders without having to provide the precise filename. They can define the starting value, and the application will display the results after searching all files and folders that begin with that value. After that, the user is given the choice to remove a particular index or all files.

4. The user can fluidly navigate between options or return to the previous menu even after the requested operation, such as adding, searching, deleting, or retrieving data, has been completed.

5. The application also enables users to remove folders that aren't empty.

6. Even when an exception occurs, the application is intended to continue operating and accepting user input. The program can only be terminated when the correct option to do so is selected.

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

- Ascending order of folders first which have files sorted in them,
- Ascending order of all files and folders inside the “main” folder.

8. Modularity was considered when creating the application. Even if someone wishes to adjust the path, they can do it by editing the source code. Program was created with the idea that there should be very little "hardcoding" of data.

Scope for Improvement

The application may also receive further updates, such as:

• Criteria to determine whether a user is permitted to add or delete a file at a particular place.
• If the selected directory is not empty, the user is prompted to a confirmation dialogue box.
• Finding files and folders using different filters, such as Type, Last Modified, etc.
• Allowing user to append data to the file.