Phase-1:LockedMe Virtual Key for Repository

This document contains sections for:

Core concepts used in project

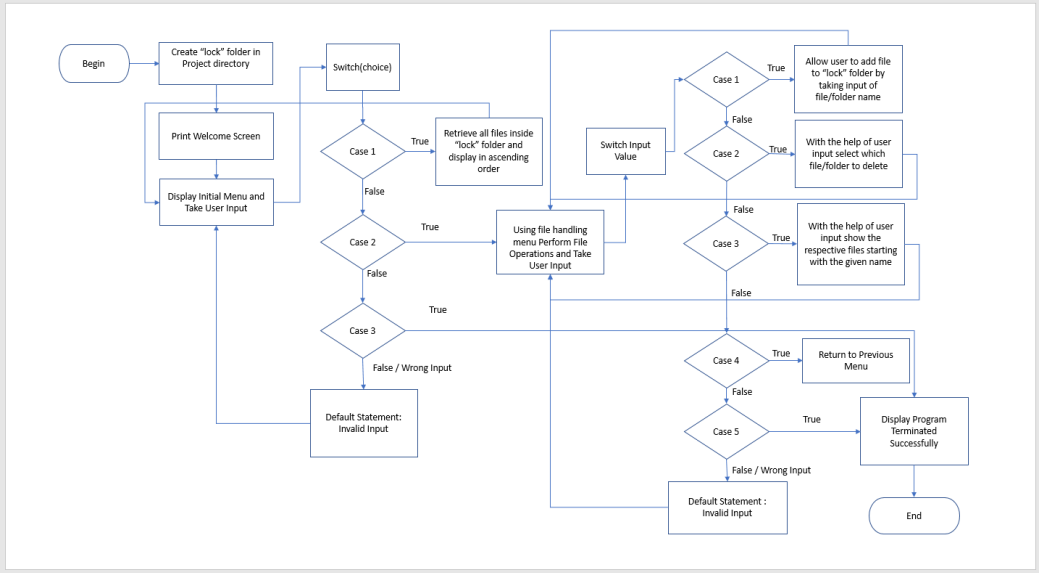
Searching, File Handling, Sorting, Flow Control, Recursion, Exception Handling, Streams API , Collections framework.

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.

Flow of the Application



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 **virtual-key-repository** 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 vkp.lockedme;

public class LockedMeMain {

	public static void main(String[] args) {

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

		MenuOptions.printWelcomeScreen("LockedMe", "Anusha Somaraddi");

		HandleOptions.handleWelcomeScreenInput();
	}


}
```

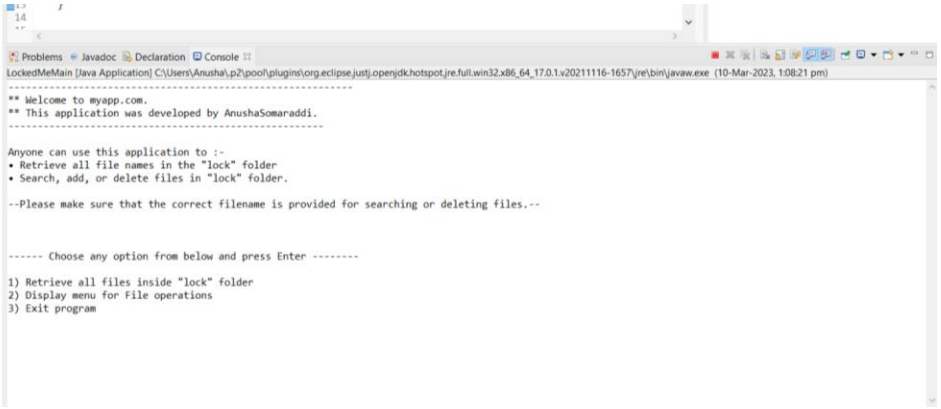
Step 3: Writing a program in Java to display Menu options available for the user (**MenuOptions.java**)

- 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 appDetails = "Anyone can use this application to :-\n"
				+ "• Retrieve all file names in the \"main\" folder\n"
				+ "• Search, add, or delete files in \"main\" folder.\n"
				+ "\n--Please make sure that the correct filename is provided for searching or deleting files.--\n";
		System.out.println(companyDetails);

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

Output:



Step 3.2: Writing method to display Initial Menu

```
public static void displayMenu() {
		String menu = "\n\n****** Select any option from below  menu 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);

}
```

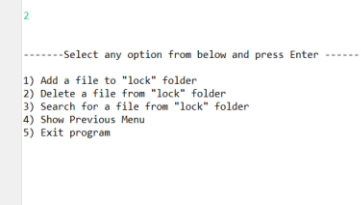
Output:

```
------ Choose any option from below and press Enter --------

1) Retrieve all files inside "lock" 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 \"lock\" folder\n" + "2) Delete a file from \"lock\" folder\n"
				+ "3) Search for a file from \"lock\" folder\n" + "4) Show Previous Menu\n" + "5) Exit program\n";

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

Output:



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

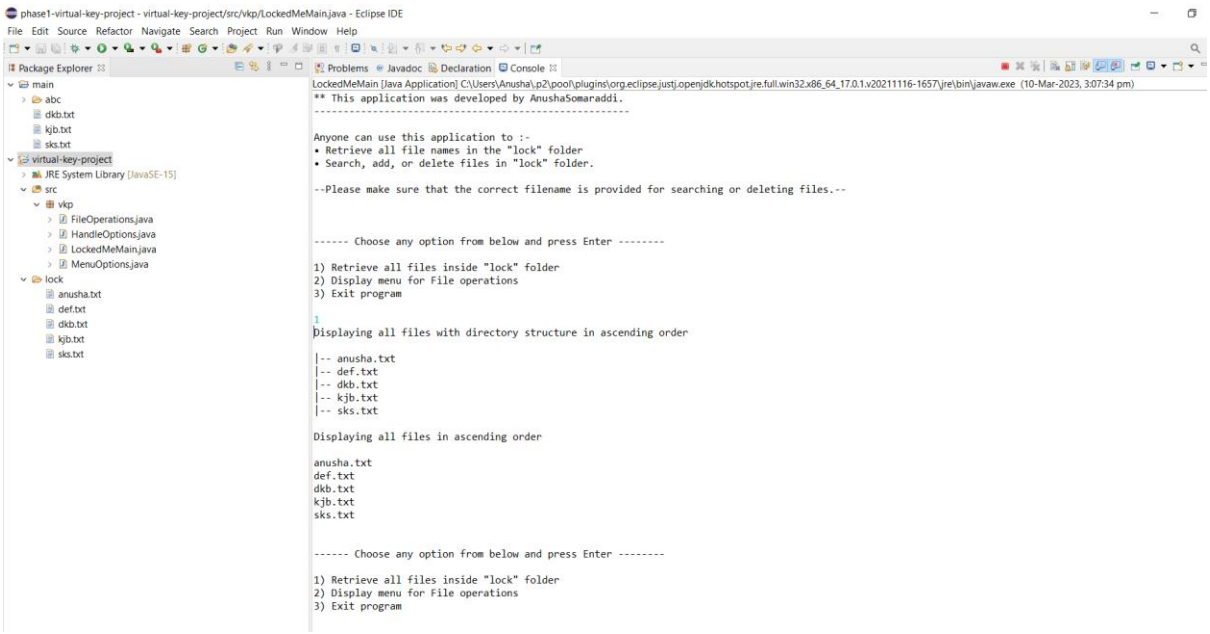
4.2. Handling input selected by user in secondary Menu for File Operations

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("lock");
					break;
				case 2:
					HandleOptions.handleFileMenuOptions();
					break;
				case 3:
					System.out.println("Program executed 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:

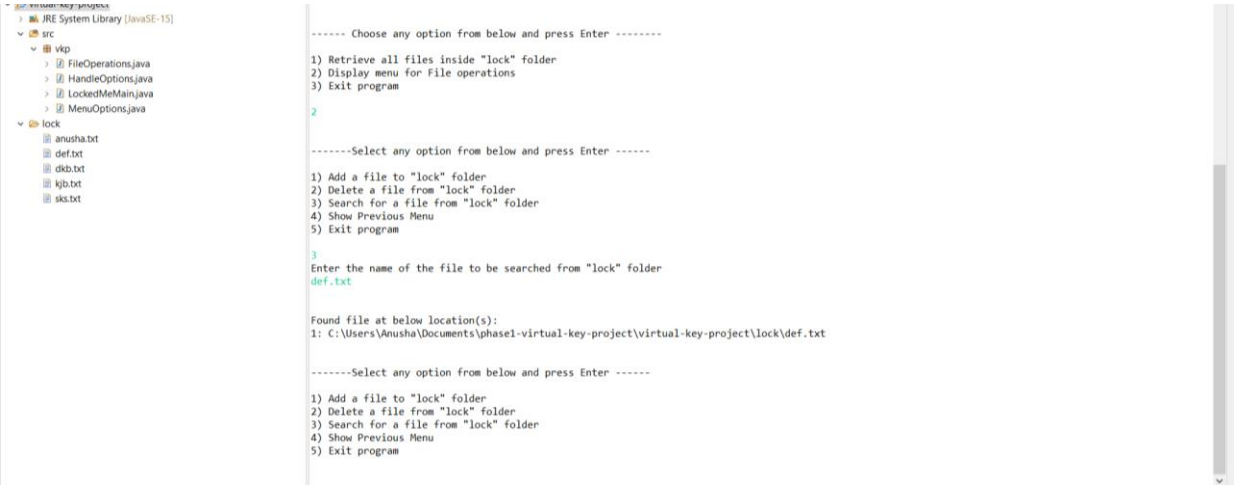


Step 4.2: Writing method to handle user input in Secondary Menu for File Operations

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

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

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

					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 \"lock\" folder");
                                        String fileName = sc.next();

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


                                        break;
                                case 4:
                                        // Go to Previous menu
                                        return;
                                case 5:
                                        // Exit
                                        System.out.println("Program executed 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:



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

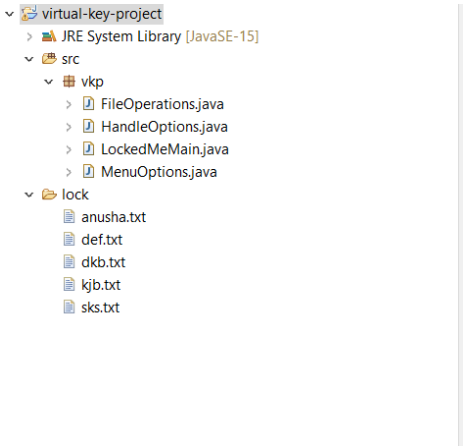
- Select your project and go to File -> New -> Class.
- Enter **FileOperations** in class name and click on "Finish."
- **FileOperations** consists methods for -:

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 "lock" 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 lock folder
	if (!file.exists()) {
		file.mkdirs();
	}
}
```

Output:

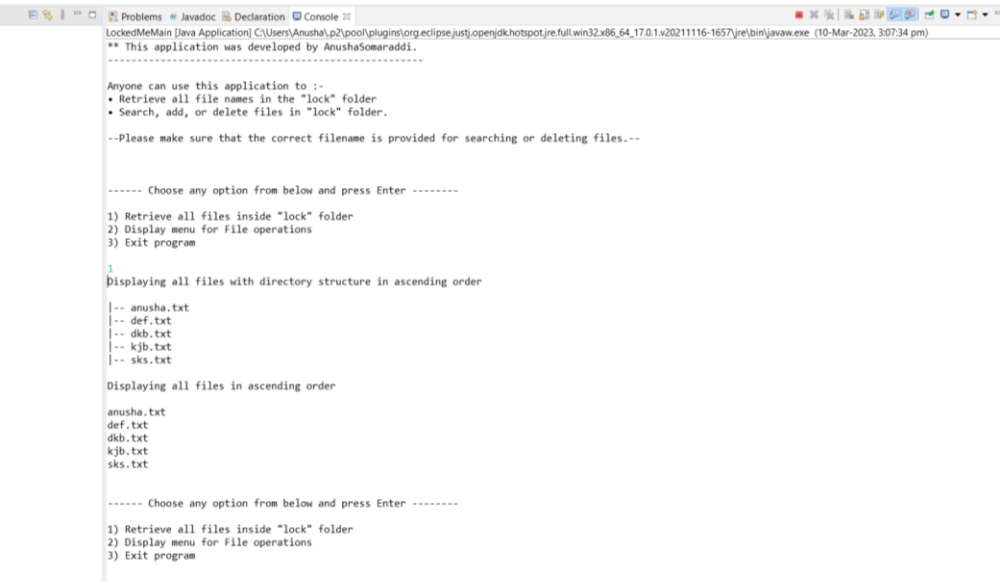


Step 5.2: Writing method to display all files in “lock” folder in ascending order and also with directory structure. (“`--" represents a directory. “|--” represents a file.)

```
public static void displayAllFiles(String path) {
		FileOperations.createMainFolderIfNotPresent("lock");
		// All required files and folders inside “lock" 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:

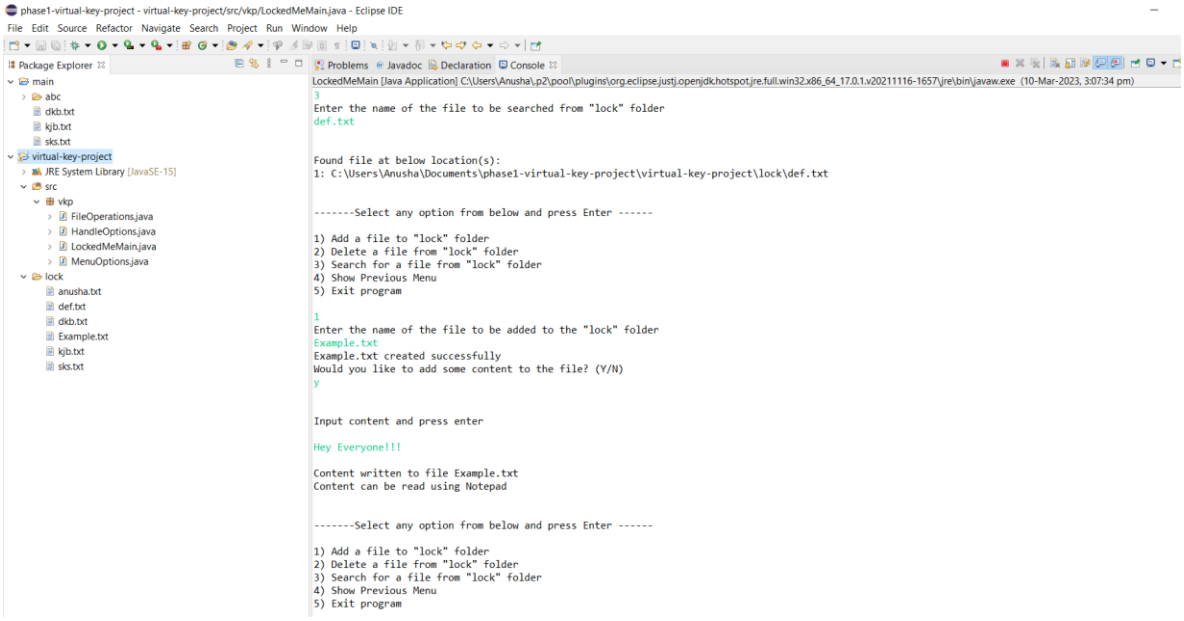


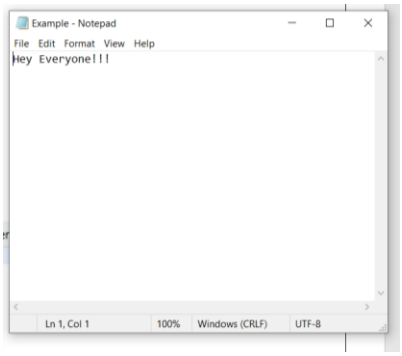
Step 5.3: Writing method to create a file/folder as specified by user input.

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

			System.out.println("Do 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\n Enter content and press enter\n");
				String content = sc.nextLine();
				Files.write(pathToFile, content.getBytes());
				System.out.println("\n Content written to file " + fileToAdd);
				System.out.println(" Content can be read using Notepad");
			}

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

```
        }
    }
```

Output:


phase1-virtual-key-project - virtual-key-project/src/vkp/LockedMeMain.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
main
abc
dkb.txt
kjb.txt
sks.txt
virtual-key-project
JRE System Library [JavaSE-15]
src
vkp
FileOperations.java
HandleOptions.java
LockedMeMain.java
MenuOptions.java
lock
anusha.txt
def.txt
dkb.txt
Example.txt
kjb.txt
sks.txt
Problems Javadoc Declaration Console
LockedMeMain [Java Application] C:\Users\Anusha\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe (10-Mar-2023, 3:07:34 pm)
3
Enter the name of the file to be searched from "lock" folder
def.txt
Found file at below location(s):
1: C:\Users\Anusha\Documents\phase1-virtual-key-project\virtual-key-project\lock\def.txt
-------Select any option from below and press Enter ------
1) Add a file to "lock" folder
2) Delete a file from "lock" folder
3) Search for a file from "lock" folder
4) Show Previous Menu
5) Exit program
1
Enter the name of the file to be added to the "lock" folder
Example.txt
Example.txt created successfully
Would you like to add some content to the file? (Y/N)
y
Input content and press enter
Hey Everyone!!!
Content written to file Example.txt
Content can be read using Notepad
-------Select any option from below and press Enter ------
1) Add a file to "lock" folder
2) Delete a file from "lock" folder
3) Search for a file from "lock" folder
4) Show Previous Menu
5) Exit program

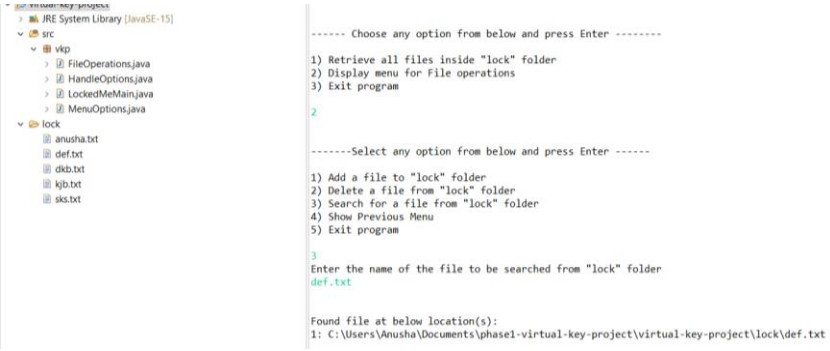

Example - Notepad
File Edit Format View Help
Hey Everyone!!!
Ln 1, Col 1
100%
Windows (CRLF)
UTF-8

Step 5.4: Writing method to search for all files as specified by user input in “lock” 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---- Unable to find any file with given file name
\"" + fileName + "\" ----\n\n");
		} else {
			System.out.println("\n\n Found the 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

				if (file.isDirectory()) {
					searchFileRecursively(file.getAbsolutePath(), fileName,
fileListNames);
				}
			}
```

```
        }
    }
```

Output:



Step 5.5: Writing method to delete file/folder specified by user input in “lock” 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:

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 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.

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

7. The application is designed with modularity in mind.

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.

GitHub- https://github.com/anushasomaraddi/Phase-1-Java-Fsd-Project.git
Author-Anusha Somaraddi.