This document contains sections for:

- Sprint planning and Task completion
- Core concepts used in project
- Flow of the Application.
- Demonstrating the product capabilities, appearance, and user interactions.
- Unique Selling Points of the Application
- Conclusions

The code for this project is available at https://github.com/pragathihebbarkm/LockedMeApp and this project is developed by Pragathi Hebbar.

Sprints planning and Task completion

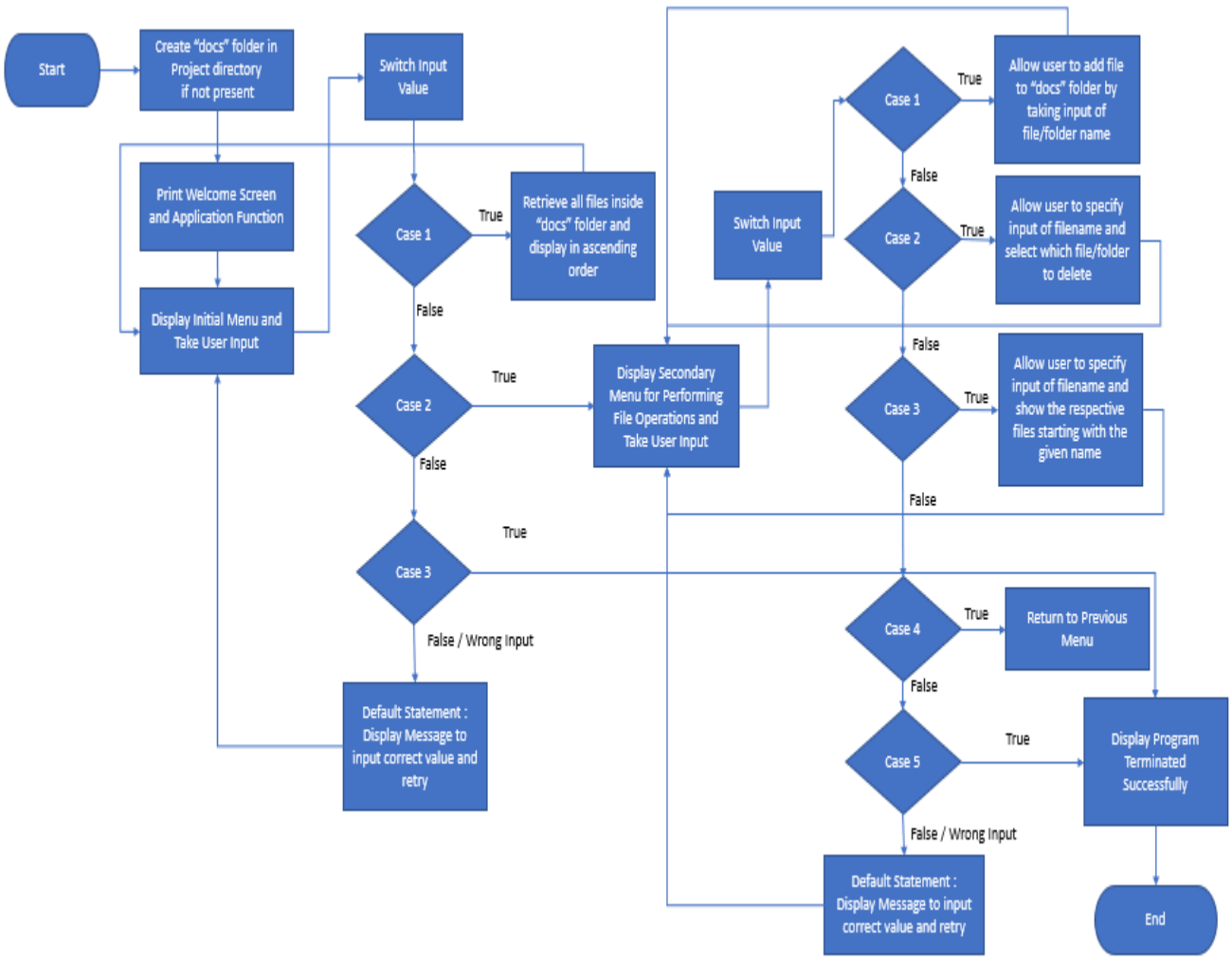
The project is planned to be completed in a single sprint. Tasks that are assumed to be completed in this sprint are :

- 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



Demonstrating the product capabilities, appearance, and user interactions

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

4. Writing a program in Java to handle Menu options selected by user (**DisplayOptions.java**)
5. Writing a program in Java to perform the File operations as specified by user (**FileHandleOperations.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 **LockedMeApp** 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 (**LockedMeApp.java**)

```
package com.myapp.lockedme;

public class LockedMeApp {

public static void main(String[] args) {

        // To create "docs" folder if not present in current folder structure

        FileHandleOps.createDocsFolderIfNotPresent("docs");

        Menu.displayWelcomeScreen("LockedMeApp", "Pragathi Hebbar K
M");

        DisplayOptions.handleWelcomeScreenInput();

        }

}
```

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

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

3.1. Displaying Welcome Screen

3.2. Displaying Initial Menu

3.3. Displaying Secondary Menu for File Operations available

Step 3.1: Writing method to display Welcome Screen

```
public static void displayWelcomeScreen(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"

                + " Retrieve all file names in the \"docs\" folder\n"

                + "Search, add, or delete files in \"docs\" folder.\n"

                + "\n**Please ensure you give the correct filename for searching or
        deleting files.**\n";

        System.out.println(companyDetails);

        System.out.println(appFunction);

}
```

Output

```
------------------------------------------------------------------
** Welcome to LockedMeApp.com.
** This application was developed by Pragathi Hebbar K M.
------------------------------------------------------------------

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

**Please ensure you give the correct filename for searching or deleting files
```

Step 3.2: Writing method to display Initial Menu

public static void displayMenuOptions() {

String menu = "\n\n****** Select any option number from below and press Enter ******\n\n"

+ "1) Retrieve all files inside \"docs\" folder\n"

+ "2) Display menu for performing File operations\n"

+ "3) Close the Application\n";

System.***out***.println(menu);

}

Output

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

1) Retrieve all files inside "docs" folder
2) Display menu for performing File operations
3) Close the Application
```

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

public static void displayFileHandleOptions() {

```
        String fileMenu = "\n\n****** Select any option number from below and
press Enter ******\n\n"

                + "1) Add a file to \"docs\" folder\n"

                + "2) Delete a file from \"docs\" folder\n"

                + "3) Search for a file from \"docs\" 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 "docs" folder
2) Delete a file from "docs" folder
3) Search for a file from "docs" folder
4) Show Previous Menu
5) Exit program
```

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

- Select your project and go to File -> New -> Class.
- Enter **DisplayOptions** in class name and click on “Finish.”
- **DisplayOptions** 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 {

                Menu.displayMenuOptions();

                int input = sc.nextInt();

                switch (input) {

                        case 1:

                                FileHandleOps.displayAllFiles("docs");

                                break;

                        case 2:

                                DisplayOptions.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 "docs" folder
2) Display menu for performing File operations
3) Close the Application

1
Displaying all files with directory structure in ascending order

|-- 5
|-- Demofile
|-- myfile1
|-- myfile2

Displaying all files in ascending order

5
Demofile
myfile1
myfile2
```

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 {

                                Menu.displayFileHandleOptions();
```

```
FileHandleOps.createDocsFolderIfNotPresent("docs");

int input = sc.nextInt();

switch (input) {

        case 1:

                // File Add

                System.out.println("Enter the name of
                the file to be added to the \"docs\"
                folder");

                String fileToAdd = sc.next();

                FileHandleOps.addNewFile(fileToAdd,
        sc);

                break;

        case 2:

                // File/Folder delete

                System.out.println("Enter the name of
        the file to be deleted from \"docs\" folder");

                String fileToDelete = sc.next();

                FileHandleOps.createDocsFolderIfNotP
        resent("docs");

                List<String> filesToDelete =
        FileHandleOps.displayFileLocations(fileToDele
        te, "docs");

                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) {
```

```
                FileHandleOps.deleteFileRecursi
            vely(filesToDelete.get(idx - 1));

            } else {

                // If idx == 0, delete all files
            displayed for the name

                for (String path : filesToDelete) {

                    FileHandleOps.deleteFile
                Recursively(path);

                }

            }

            break;

        case 3:

            // File/Folder Search

            System.out.println("Enter the name of
        the file to be searched from \"docs\" folder");

            String fileName = sc.next();

            FileHandleOps.createDocsFolderIfNotP
        resent("docs");

            FileHandleOps.displayFileLocations(file
        Name, "docs");

            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 "docs" folder
2) Display menu for performing File operations
3) Close the Application

2


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

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

3
Enter the name of the file to be searched from "docs" folder
myfile1


Found file at below location(s):
1: C:\JavaFSD\LockedMeApp\LockedMe\docs\myfile1


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

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

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

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

5.1. Creating “docs” folder in project if it’s not already present

5.2. Displaying all files in “docs” folder in ascending order and also with directory structure.

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

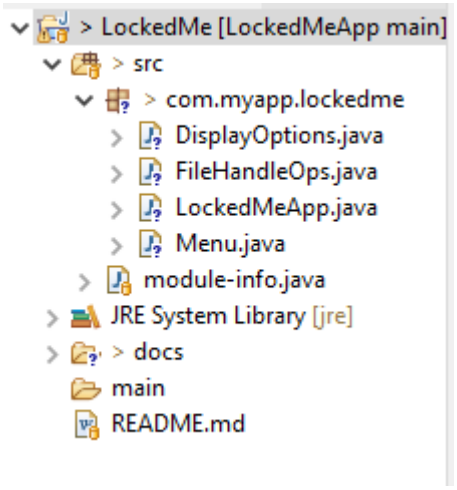
5.4. Search files as specified by user input in “docs” folder and it’s subfolders.

5.5. Deleting a file/folder from “docs” folder

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

```
public static void createDocsFolderIfNotPresent(String folderName) {
        File file = new File(folderName);
        // If file doesn't exist, create the docs folder
        if (!file.exists()) {
            file.mkdirs();
        }
}
```

Output



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

```
public static void displayAllFiles(String path) {
        FileHandleOps.createDocsFolderIfNotPresent("docs");
        // All required files and folders inside "docs" 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> fileListNames =
                FileHandleOps.listAllFilesInDirectory(path, 0, new
                ArrayList<String>());

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

                Collections.sort(fileListNames);

                fileListNames.stream().forEach(System.out::println);

        }


        public static List<String> listAllFilesInDirectory(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());

                                listAllFilesInDirectory(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 "docs" folder
2) Display menu for performing File operations
3) Close the Application

1
Displaying all files with directory structure in ascending order

|-- 5
|-- Demofile
|-- myfile1
|-- myfile2

Displaying all files in ascending order

5
Demofile
myfile1
myfile2
```

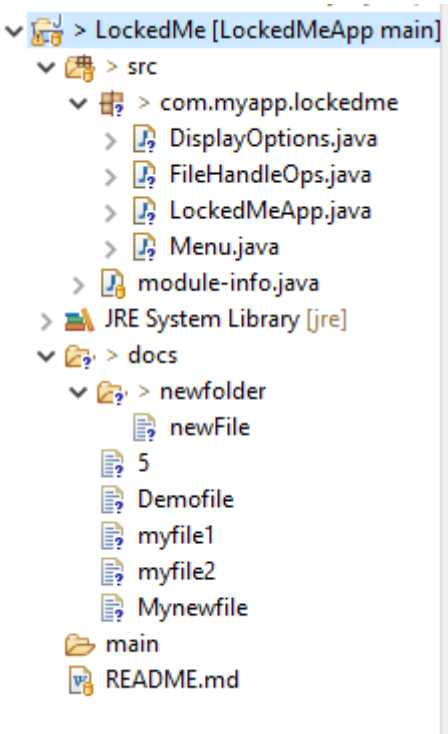
Step 5.3: Writing method to create a file/folder as specified by user input.

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

    FileHandleOps.createDocsFolderIfNotPresent("docs");

    Path pathToFile = Paths.get("./docs/" + 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 add new 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 "docs" folder
2) Delete a file from "docs" folder
3) Search for a file from "docs" folder
4) Show Previous Menu
5) Exit program

1
Enter the name of the file to be added to the "docs" folder
/newfolder/newFile
/newfolder/newFile created successfully
Would you like to add some content to the file? (Y/N)
y


Input content and press enter

new file created under new folder

Content written to file /newfolder/newFile
Content can be read using Notepad or Notepad++
```



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

```
public static List<String> displayFileLocations(String fileName, String path) {

        List<String> fileListNames = new ArrayList<>();

        FileHandleOps.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

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

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

3
Enter the name of the file to be searched from "docs" folder
myfile1


Found file at below location(s):
1: C:\JavaFSD\LockedMeApp\LockedMe\docs\myfile1
```

Step 5.5: Writing method to delete file/folder specified by user input in "docs" 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.

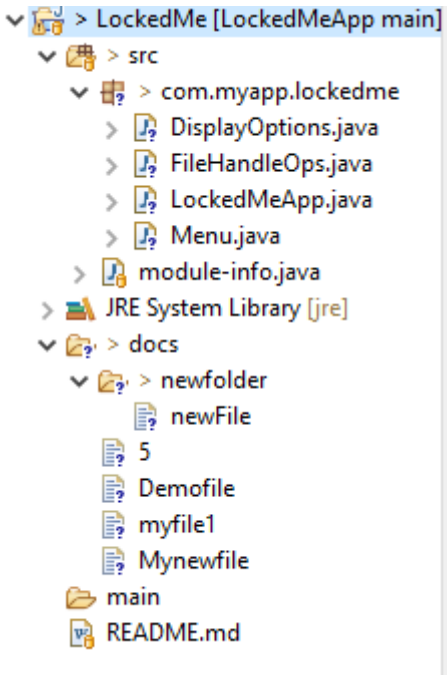
```
        public static void deleteFileRecursively(String path) {
                File currFile = new File(path);
                File[] files = currFile.listFiles();
                        if (files != null && files.length > 0) {
                                for (File file : files) {
                                        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”.

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

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

2
Enter the name of the file to be deleted from "docs" folder
myfile2


Found file at below location(s):
1: C:\JavaFSD\LockedMeApp\LockedMe\docs\myfile2

Select index of which file to delete?
(Enter 0 if you want to delete all elements)
1
myfile2 at C:\JavaFSD\LockedMeApp\LockedMe\docs deleted successfull
```



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

Conclusions

Further enhancements to this application can be made, some of the below points can be included :

- Checking user permissions, to check if user is allowed to delete the file or add the file at the specific location.
- Confirming from user if they really want to delete the selected directory/file if it's not empty.
- Retrieving files/folders by different criteria like Last Modified, Type, etc.
- Allowing user to append data to the file.

The code for this project is available at https://github.com/pragathihebbarkm/LockedMeApp .