LockedMe Project – Virtual Key for Repositories

This document contains sections for:

The code for this project is hosted at https://github.com/sitansusubudhi/LockedMe. The project is developed by Sitansu Subudhi.

Sprints planning and Task completion

The project is planned to be completed in 1 sprint (Calculated each sprint is 7 days). Tasks assumed to be completed in the sprint are:

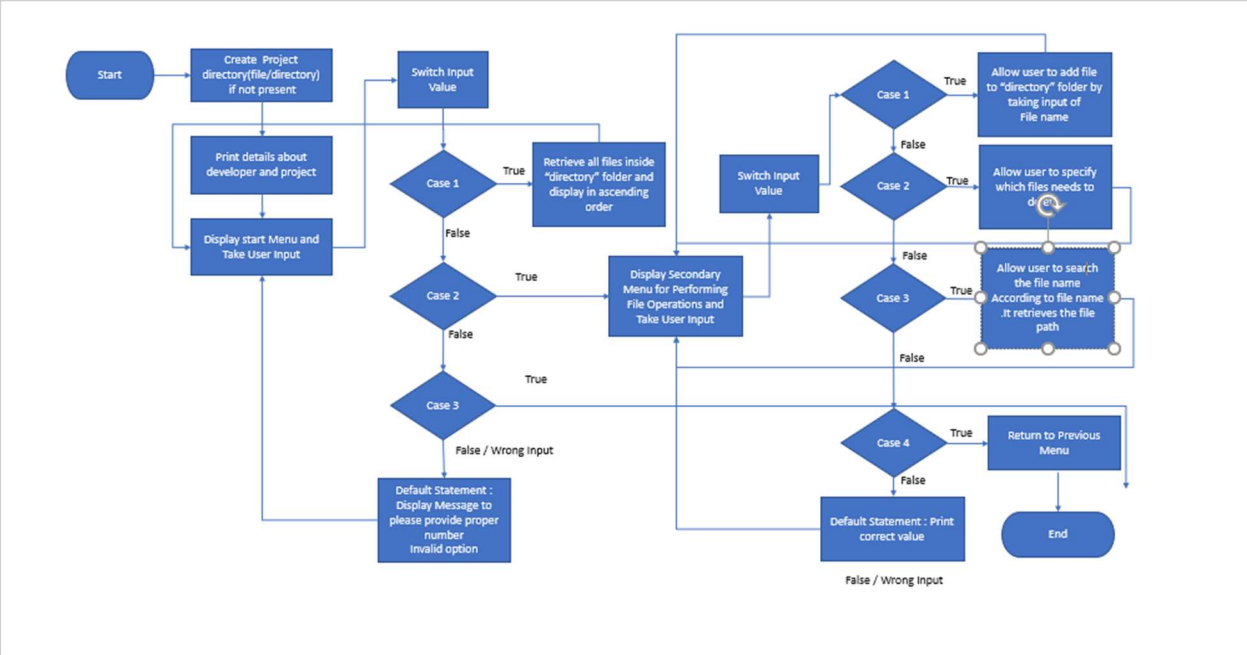
- Creating the flow of the application.
- Creating git repository virtualKey to track changes as development progresses.
- R&D for to understand the project and implemented.
- Testing the project with user input manually.
- Pushing code to GitHub Repository.
- Creating this specification document highlighting application capabilities, appearance, and user interactions.

Core concepts used in project

- File Handling
- Collection Framework & Sorting Algorithm.
- Basic java feature (switch -if else, util package etc.)
- OOPS concepts.
- Exception Handling.
- Flow Diagram.

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

Flow of the Application



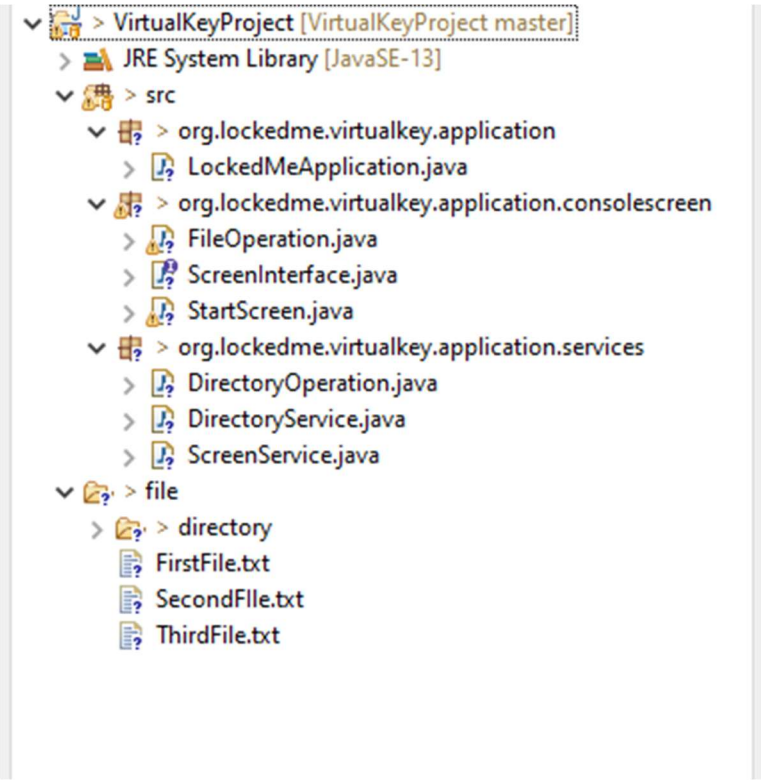
Demonstrating the product functionalities, appearance, and user interactions

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.
- Creates the packages for Application (Entry-point of program), Console-based interaction, and Services.
- Implement the project according to packages.

Step 2: File Structure



Step 3: Project Demonstration

3.1 Application name and the developer details.

output

```
Welcome to VirtualKey Project
Developer: Anupam Tiwari
Version: 1.0.0
About :It's starting phase product where client can organize the file in a such way like- Add, Delete, Search operation can be
performed.
```

Implementation logic-

```
    public void projectinfo() {
        System.out.println(welcomeText);
        System.out.println(developerText);
        System.out.println(projectVersion);
        System.out.println(about);
        System.out.println("\n");
        System.out.println("Please Choose below option 1->Show Listing of Files. 2->Show File Option Menu. 3->Quit the operation.");
        //System.out.println("\n");
        display();
    }

    @Override
    public void display() {
        System.out.println("Main Menu");
        for (String s : options) {
            System.out.println(s);
        }
```

3.2 If Folder is Empty output will be file not found.

Output

```
Please Choose below option 1->Show Listing of Files. 2->Show File Option Menu. 3->Quit the operation.
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
1
Listing Files in Ascending order :
        File not Found.
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
```

Implementation logic-

```
8
9     public static void PrintFiles() {
10        if (fileDirectory.fillFiles().isEmpty()) {
11            System.out.println("\tFile not Found.");
12        } else {
13            for (File file : DirectoryService.getFileDirectory().getFiles()) {
14                System.out.println("\t" + file.getName());
15            }
16        }
17    }
18
```

3.3 User can retrieve all files in Ascending order.

Output

```
Please Choose below option 1->Show Listing of Files. 2->Show File Option Menu. 3->Quit the operation.
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
1
Listing Files in Ascending order :
        bb.txt
        complete.txt
        FirstFile.txt
        same.txt
        SecondFIle.txt
        ThirdFile.txt
        vv.txt
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
```

Implementation logic-

```
8
9     public static void PrintFiles() {
10        if (fileDirectory.fillFiles().isEmpty()) {
11            System.out.println("\tFile not Found.");
12        } else {
13            for (File file : DirectoryService.getFileDirectory().getFiles()) {
14                System.out.println("\t" + file.getName());
15            }
16        }
17    }
18
19    public static DirectoryOperation getFileDirectory() {
20        return fileDirectory;
21    }
22
23    public static void setFileDirectory(DirectoryOperation fileDirectory) {
24        DirectoryService.fileDirectory = fileDirectory;
25    }
26
```

3.4 User can Add files.

Output1 (If File already exist It will not create any new file.)

```
Please Choose below option 1->Show Listing of Files. 2->Show File Option Menu. 3->Quit the operation.
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
2
File Options Menu
1. Add a File.
2. Delete A File.
3. Search A File.
4. Return to Menu.
1
Please Enter the Filename:
bb.txt
You are adding a file named: bb.txt
This File Already Exist
File Options Menu
1. Add a File.
2. Delete A File.
3. Search A File.
4. Return to Menu.
```

Output2 (create new file. Add Successfully)

```
Please Choose below option 1->Show Listing of Files. 2->Show File Option Menu. 3->Quit the operation.
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
2
File Options Menu
1. Add a File.
2. Delete A File.
3. Search A File.
4. Return to Menu.
1
Please Enter the Filename:
Anupam.txt
You are adding a file named: Anupam.txt
        File created: Anupam.txt successfully
File Options Menu
1. Add a File.
2. Delete A File.
3. Search A File.
4. Return to Menu.
```

Implementation logic-

```java
    public void AddFile() {
        System.out.println("Please Enter the Filename:");

        String fileName = this.getInputString();

        System.out.println("You are adding a file named: " + fileName);

        try {
            //Path path = FileSystems.getDefault().getPath(DirectoryOperation.name + fileName).toAbsolutePath();
            File file = new File(dir.getName() + fileName);

            if (file.createNewFile()) {
                System.out.println("\tFile created: " + file.getName()+" successfully");
                dir.getFiles().add(file);

            } else {
                System.out.println("This File Already Exist");
            }
        } catch (IOException e) {
            System.out.println("bla");
        }
    }
```

3.4 User can Delete specific files.

Output (File deleted successfully.)

```
Please Choose below option 1->Show Listing of Files. 2->Show File Option Menu. 3->Quit the operat
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
2
File Options Menu
1. Add a File.
2. Delete A File.
3. Search A File.
4. Return to Menu.
2
Please Enter the Filename:
Anupam.txt
You are deleting a file named: Anupam.txt
Deleted File: Anupam.txt
File Options Menu
1. Add a File.
2. Delete A File.
3. Search A File.
4. Return to Menu.
```

Implementation logic-

```
    }

    public void DeleteFile() {

        System.out.println("Please Enter the Filename:");

        String fileName = this.getInputString();

        System.out.println("You are deleting a file named: " + fileName);

        Path path = FileSystems.getDefault().getPath(DirectoryOperation.name + fileName).toAbsolutePath();
        File file = path.toFile();
        if (file.delete()) {
            System.out.println("Deleted File: " + file.getName());
            dir.getFiles().remove(file);
        } else {
            System.out.println("Failed to delete file:" + fileName + ", file was not found.");
        }
    }

```

3.4 User can Search specific files.

Output (File search for path and name successfully.)

```
Please Choose below option 1->Show Listing of Files. 2->Show File Option Menu. 3->Quit the operation.
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
2
File Options Menu
1. Add a File.
2. Delete A File.
3. Search A File.
4. Return to Menu.
3
Please Enter the Filename:
FirstFile.txt
        File name EXIST in directory: D:\Project\VirtualKeyProject\file\directory\FirstFile.txt
        Found FirstFile.txt
File Options Menu
1. Add a File.
2. Delete A File.
3. Search A File.
4. Return to Menu.
```

Implementation logic-

```
112
113     public void SearchFile() {
114
115         Boolean found = false;
116
117         System.out.println("Please Enter the Filename:");
118
119         String fileName = this.getInputString();
120         Path path = FileSystems.getDefault().getPath(DirectoryOperation.name + fileName).toAbsolutePath();
121         System.out.println("\tFile name EXIST in directory: " + path);
122
123         ArrayList<File> files = dir.getFiles();
124
125         for (int i = 0; i < files.size(); i++) {
126             if (files.get(i).getName().equals(fileName)) {
127                 System.out.println("\tFound " + fileName);
128                 found = true;
129             }
130         }
131         if (found == false) {
132             System.out.println("File not found.");
133         }
134     }
```

3.5 User After completion of all operations (Add, Search, Delete) can return to the main menu.

Output

```
Please Choose below option 1->Show Listing of Files. 2->Show File Option Menu. 3->Quit the operation.
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
2
File Options Menu
1. Add a File.
2. Delete A File.
3. Search A File.
4. Return to Menu.
4
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
```

Implementation logic-

```
    public void getUserInput() {
        int selectedOption;
        while ((selectedOption = this.getOption()) != 4) {
            this.showNavigateOption(selectedOption);
        }
    }

    @Override
    public void showNavigateOption(int option) {

        switch (option) {

        case 1: // Add File
            this.AddFile();

            this.display();
            break;
        case 2: // Delete File
            this.DeleteFile();

            this.display();
            break;
        case 3: // Search File
            this.SearchFile();
            this.display();
            break;
        case 4:
            // Go to Previous menu
            break;
```

3.6 User can quit the operation.

Output

```
Please Choose below option 1->Show Listing of Files. 2->Show File Option Menu. 3->Quit the operation.
Main Menu
1. Show Files Listing
2. Show File Options Menu
3. Quit the operation
3
Program exited successfully.
```

Implementation logic-

```
46    public void getUserInput() {
47        int selectedOption = 0;
48        while ((selectedOption = this.getOption()) != 4) {
49            this.showNavigateOption(selectedOption);
50        }
51    }
52
53    @Override
54    public void showNavigateOption(int option) {
55        switch (option) {
56
57        case 1: // Show All Files in Directory.
58            this.ShowFiles();
59
60            this.display();
61
62            break;
63
64        case 2: // Show File operation Options
65            ScreenService.setCurrentScreen(ScreenService.FileOptionsScreen);
66            ScreenService.getCurrentScreen().display();
67            ScreenService.getCurrentScreen().getUserInput();
68
69            this.display();
70            break;
71
72        case 3:
73            System.out.println("Program exited successfully.");
74            System.exit(1);
```

Step 4: Pushing the code to GitHub repository

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

  cd <folder path> (Eclipse workspace)

- Initialize repository using the following command:

  git init (Create repository)

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

  git add . (All Files)

git add -p (paragraph)

- 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 pull origin master (up-to-date)

  git push -u origin master

Unique Selling Points of the Application

1. Organize all Files in one place. All operation can control in one place. No needs to do it manually. It saves time.
2. This functionality can we use any enterprise project where file handling is used a lot like application related Education sector. Where we can perform operation on the data.

3. User is also provided the option to search content. Not only Filename but user can see the path of file.

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

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 ascending order.

7. 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.
- Handle the all-possible wrong input. Application will run steadily.
- Allowing user to append data to the file.