

Virtual key for Repositories

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 hosted at <https://github.com/surekhaitgithub/Newcodingboard.git>

The project is developed by [Duggasani Naga Surekha](#).

Sprints planning and Task completion

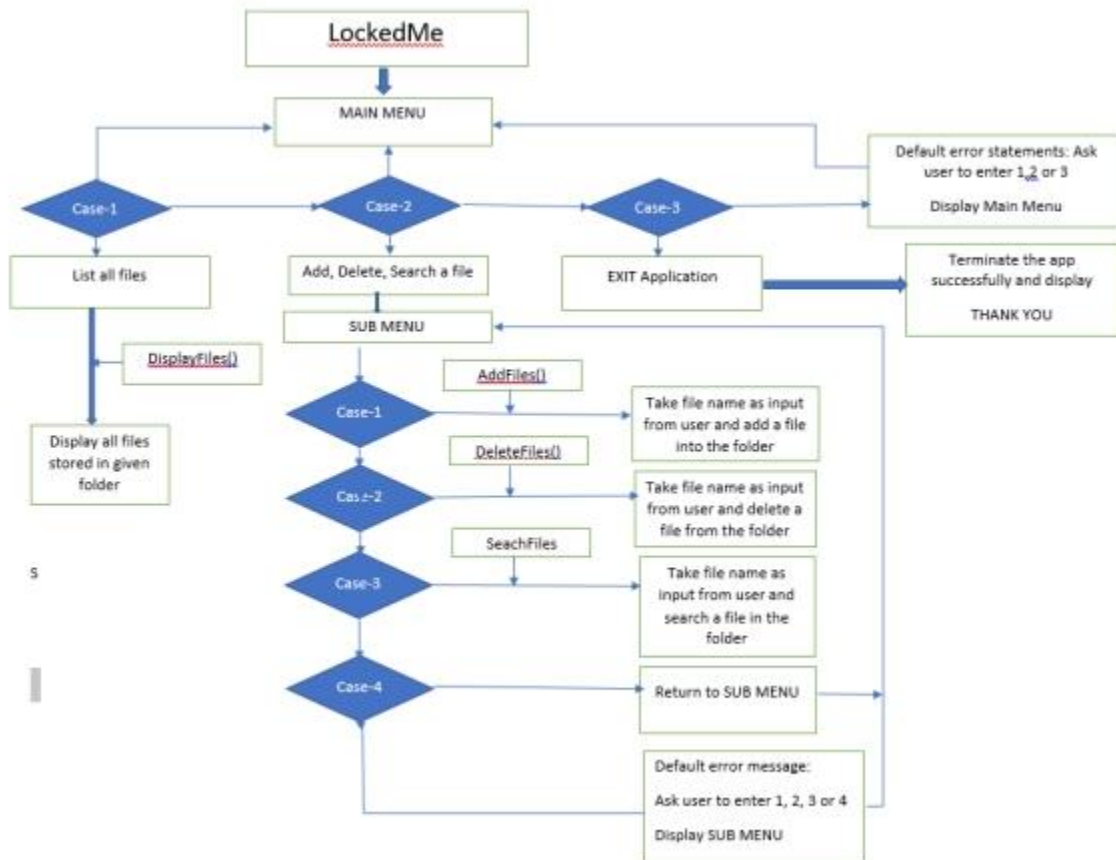
The project is planned to be completed in 2 sprint. Tasks assumed to be completed in the sprints are:

- 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

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.
- Create a project named as LockedMe , click on finish.
- Select your project and go to File -> New -> package.
- Create a package named as LockedMe , click on finish.
- Select your package and go to -> New -> class.
- Create 3 classes named Main.java, Switchcases .java, operations.java.

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

```
package LockedMe;
```

```
public class Main {
```

```
/*Enter your desired Directory path */
```

```
public static final String path = "C:\\Users\\My PC\\eclipse-workspace\\session8\\files";  
public static void main(String[] args) {
```

```
switchcases menu = new switchcases();
```

```
menu.introScreen();
```

```
menu.mainMenu();
```

```
}
```

```
}
```

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 -:

[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

```
package LockedMe;

import java.io.IOException;

import java.util.Scanner;

public class switchcases {

    private static final int a = 0;

    private static final int b = 0;

    private static final int c = 0;

    private static final int d = 0;

    Scanner scan = new Scanner(System.in);

    operation dao = new operation();

    public void introScreen() {

        System.out.println();

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

        System.out.println("          DEVELOPED BY Surekha          ");

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

        System.out.println("          LOCKEDME.COM          ");

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

        System.out.println("    File location: " + Main.path + "    ");

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

        System.out.println("\n\n");
```

```

}

public void exitScreen() {
System.out.println("*      THANK YOU      *");
}

```

OUTPUT:

```

***  LockedMe.com  ***
***Surekha ***

DIRECTORY : C:\Users\My PC\eclipse-workspace\session8\files|

```

Step 3.2: Writing method to display Initial Menu

```

public void mainMenuOptions() {
System.out.println("          MAIN MENU          ");
System.out.println("\n");
System.out.println(" Select any one of the option:- ");
System.out.println(" 1 - List All Files          ");
System.out.println(" 2 - Add or Delete or Search Files");
System.out.println(" 3 - Exit Application        ");
System.out.println("*****");
System.out.println("Enter your option:- ");

}

```

OUTPUT:

MAIN MENU

Select any one of the option:-

1 - List All Files

2 - Add or Delete or Search Files

3 - Exit Application

*****a

Enter your option:-

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

```
public void subMenuOptions() {  
    System.out.println("\n");  
    System.out.println("          SUB MENU          ");  
    System.out.println("\n");  
    System.out.println("| Select any one of the option: |");  
    System.out.println("  1 - Add a file              ");  
    System.out.println("  2 - Delete a file           ");  
    System.out.println("  3 - Search a file           ");  
    System.out.println("  4 - Go Back                 ");  
    System.out.println("\n");  
    System.out.println("Enter your option :- ");  
}
```

OUTPUT:

SUB MENU

```
| Select any one of the option: |  
  1 - Add a file  
  2 - Delete a file  
  3 - Search a file  
  4 - Go Back
```

Enter your option :-

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 void mainMenu() {  
  
    int choice = 0;  
    char decision = 0;  
    do {
```

```
mainMenuOptions();
```

```
try {  
    choice = Integer.parseInt(scan.nextLine());  
} catch (NumberFormatException e) {  
    System.out.println("Oops! option not found.");  
    mainMenu();  
}
```

```
switch (choice) {
```

```
case 1:
```

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

```
try {
```

```
dao.listAllFiles(Main.path);
```

```
}catch(NullPointerException e) {
```

```
System.out.println(e.getMessage());
```

```
}catch(IllegalArgumentException e) {
```

```
System.out.println(e.getMessage());
```

```
}catch(Exception e) {
```

```
System.out.println(e.getMessage());
```

```
}
```

```
System.out.println("\n*****\n");
```

```
break;
```

```
case 2:
```

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



```

subMenu();

break;

case 3:

System.out.println("\n Are you sure you want to exit ? ");
System.out.println(" (y) ==> yes (N) ==> no ");
decision scan.nextLine().toUpperCase().charAt(0);
if(decision == 'y') {
System.out.println("\n");
exitScreen();
System.exit(1);
}else if(decision == 'n') {
System.out.println("\n");
mainMenu();
}else {
System.out.println("\nInvalid Input \nValid Inputs :(y/n)\n");
mainMenu();
}

default:

System.out.println("\nInvalid Input \nValid Input Integers:(1-3)\n");
mainMenu();

}

}while(true);

```

```
}
```

OUTPUT:

MAIN MENU

Select any one of the option:-

1 - List All Files

2 - Add or Delete or Search Files

3 - Exit Application

*****a

Enter your option:-

2

SUB MENU

| Select any one of the option: |

1 - Add a file

2 - Delete a file

3 - Search a file

4 - Go Back

Enter your option :-

4

MAIN MENU

MAIN MENU

Select any one of the option:-

- 1 - List All Files
- 2 - Add or Delete or Search Files
- 3 - Exit Application

*****a

Enter your option:-

2

SUB MENU

- | Select any one of the option: |
- 1 - Add a file
 - 2 - Delete a file
 - 3 - Search a file
 - 4 - Go Back

Enter your option :-

4

MAIN MENU

```

Select any one of the option:-
1 - List All Files
2 - Add or Delete or Search Files
3 - Exit Application
*****a
Enter your option:-
3
|
Are you sure you want to exit ?
(Y) ==> Yes      (N) ==> No
Y

*      THANK YOU      *

```

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

```

public void subMenu() {
    String file = null;
    String fileName = null;
    int choice = 0;

    do {

        subMenuOptions();

        try {
            choice = Integer.parseInt(scan.nextLine());
        } catch (NumberFormatException e) {

```

```
System.out.println("Invalid Input \nValid Input Integers:(1-4)");  
subMenu();  
}
```

```
switch (choice) {  
    case 1:  
        System.out.println("\n==> Adding a File...");  
        System.out.println("Please enter a file name :- ");  
        file = scan.nextLine();  
        fileName = file.trim();  
        try {  
            dao.createNewFile(Main.path, fileName);  
        } catch (NullPointerException e) {  
            System.out.println(e.getMessage());  
        } catch (IOException e) {  
            System.out.println("Error occurred while adding file..");  
            System.out.println("Please try again...");  
        } catch (Exception e) {  
            System.out.println("Error occurred while adding file..");  
            System.out.println("Please try again...");  
        }  
        System.out.println("\n*****\n");  
        break;  
  
    case 2:  
        System.out.println("\n==> Deleting a File...");  
        System.out.println("Please enter a file name to Delete : ");
```

```
file = scan.nextLine();

fileName = file.trim();

try {
dao.deleteFile(Main.path, fileName);
}catch(NullPointerException e) {
System.out.println(e.getMessage());
}catch(IOException e) {
System.out.println("Error occurred while Deleting File..");
System.out.println("Please try again...");
}catch(Exception e) {
System.out.println("Error occurred while Deleting File..");
System.out.println("Please try again...");
}

System.out.println("\n*****\n");

break;

case 3:
System.out.println("\n==> Searching a File...");
System.out.println("Please enter a file name to Search : ");
file = scan.nextLine();
fileName = file.trim();

try {
dao.searchFile(Main.path, fileName);
}catch(NullPointerException e) {
System.out.println(e.getMessage());
}catch(IllegalArgumentException e) {
System.out.println(e.getMessage());
}catch(Exception e) {
```

```

System.out.println(e.getMessage());
}

System.out.println("\n*****\n");

break;

case 4: mainMenu();

break;

default:

System.out.println("Invalid Input \nValid Input Integers:(1-4)\n");

subMenu();

}

file = null;

fileName = null;

}while(true);

}

}

```

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 “main” folder in project if it’s not present

```
package LockedMe;

import java.io.File;

import java.io.IOException;

import java.util.Arrays;

import java.util.Set;

import java.util.TreeSet;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class operation {

    public void listAllFiles(String path) {

        if (path == null || path.isEmpty() || path.isBlank())

            throw new NullPointerException("Path cannot be Empty or null");

        File dir = new File(path);

        if(!dir.exists())

            throw new IllegalArgumentException("Path does not exist");

        if(dir.isFile())

            throw new IllegalArgumentException("The given path is a file. A directory is expected.");

        String [] files = dir.list();
```

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.)

```
if(files != null && files.length > 0) {

    Set<String>filesList = new TreeSet<String>(Arrays.asList(files));

    System.out.println("The Files in "+ dir.getAbsolutePath() + " are: \n");
```



```

for(String file1:filesList) {

System.out.println(file1);

}

System.out.println("\nTotal Number of files: "+ filesList.size());

}else {

System.out.println("Directory is Empty");

}

}

```

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

```

public void createNewFile(String path , String fileName) throws IOException {

if (path == null || path.isEmpty() || path.isBlank())

throw new NullPointerException("Path cannot be Empty or null");


if (fileName == null || fileName.isEmpty() || fileName.isBlank())

throw new NullPointerException("File Name cannot be Empty or null");


File newFile = new File(path + File.separator + fileName);


boolean createFile = newFile.createNewFile();


if (createFile) {

```

```
System.out.println("\nFile Successfully Created: " + newFile.getAbsolutePath());
```

```
}else if(!createFile) {
```

```
System.out.println("\nFile Already Exist.. Please try again." );
```

```
}
```

```
}
```

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

```
public void searchFile(String path , String fileName){

    if (path == null || path.isEmpty() || path.isBlank())
        throw new NullPointerException("Path cannot be Empty or null");
    if (fileName == null || fileName.isEmpty() || fileName.isBlank())
        throw new NullPointerException("File Name cannot be Empty or null");
    File dir = new File(path);
    if(!dir.exists())
        throw new IllegalArgumentException("Path does not exist");
    if(dir.isFile())
        throw new IllegalArgumentException("The given path is a file. A directory is expected.");
    String [] fileList = dir.list();
    boolean flag = false;
    Pattern pat = Pattern.compile(fileName);
```

```

if(fileList != null && fileList.length > 0) {

for(String file:fileList) {

Matcher mat = pat.matcher(file);

if(mat.matches()) {

System.out.println("File Found at location: " + dir.getAbsolutePath());

flag = true;

break;

}

}

}

if(flag == false)

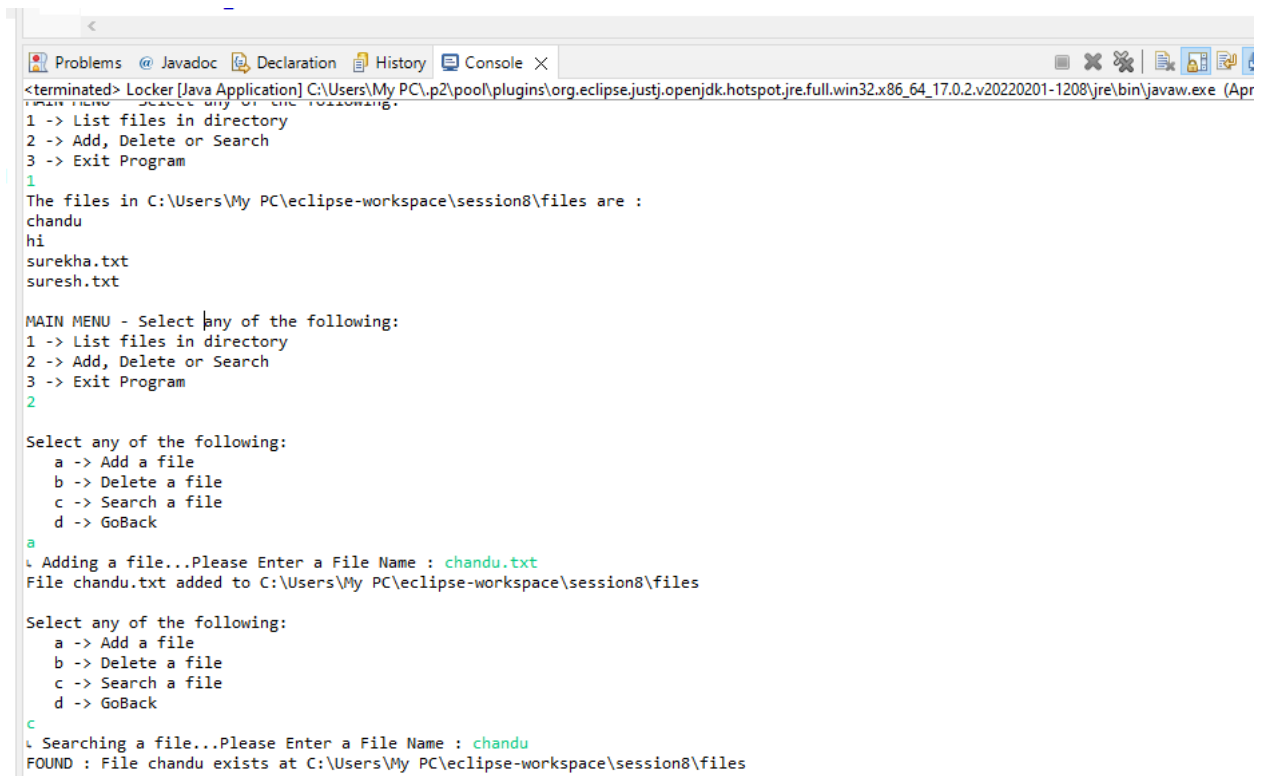
System.out.println("File Not Found.. Please try again.");

}

}

```

OUTPUT:



```

<terminated> Locker [Java Application] C:\Users\My PC\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20220201-1208\jre\bin\javaw.exe (Apr
MAIN MENU - Select any of the following:
1 -> List files in directory
2 -> Add, Delete or Search
3 -> Exit Program
1
The files in C:\Users\My PC\eclipse-workspace\session8\files are :
chandu
hi
surekha.txt
suresh.txt

MAIN MENU - Select any of the following:
1 -> List files in directory
2 -> Add, Delete or Search
3 -> Exit Program
2
Select any of the following:
a -> Add a file
b -> Delete a file
c -> Search a file
d -> GoBack
a
Adding a file...Please Enter a File Name : chandu.txt
File chandu.txt added to C:\Users\My PC\eclipse-workspace\session8\files

Select any of the following:
a -> Add a file
b -> Delete a file
c -> Search a file
d -> GoBack
c
Searching a file...Please Enter a File Name : chandu
FOUND : File chandu exists at C:\Users\My PC\eclipse-workspace\session8\files

```

Step 5.5: Writing method to delete a file as specified by user input in “main” folder and it’s subfolders.

```
public void deleteFile(String path , String fileName) throws IOException {  
    if (path == null || path.isEmpty() || path.isBlank())  
        throw new NullPointerException("Path cannot be Empty or null");  
    if (fileName == null || fileName.isEmpty() || fileName.isBlank())  
        throw new NullPointerException("File Name cannot be Empty or null");  
    File newFile = new File(path + File.separator + fileName);  
    boolean deleteFile = newFile.delete();  
    if (deleteFile) {  
        System.out.println("\nFile deleted Successfully");  
    }else {  
        System.out.println("\nFile Not Found.. Please try again." );  
    }  
}
```

```
Select any of the following:  
a -> Add a file  
b -> Delete a file  
c -> Search a file  
d -> GoBack  
b  
Deleting a file...Please Enter a File Name : suresh.txt  
File suresh.txt deleted from C:\Users\My PC\eclipse-workspace\session8\files
```

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.