**LockedMe – Project**

This document contains sections for:

* [Sprint planning and Task completion](#Sprint_plan)
* [Core concepts used in project](#Core_concepts)
* [Flow of the Application](#Flow).
* [Demonstrating the product capabilities, appearance, and user interactions.](#Product_capability)
* [Unique Selling Points of the Application](#USP)
* [Conclusions](#Conclusions)

The code for this project is hosted at <https://github.com/Gousia-syed/LockedMeProject>

The project is developed by **Gousia. Syed.**

## **Sprints planning and Task completion**

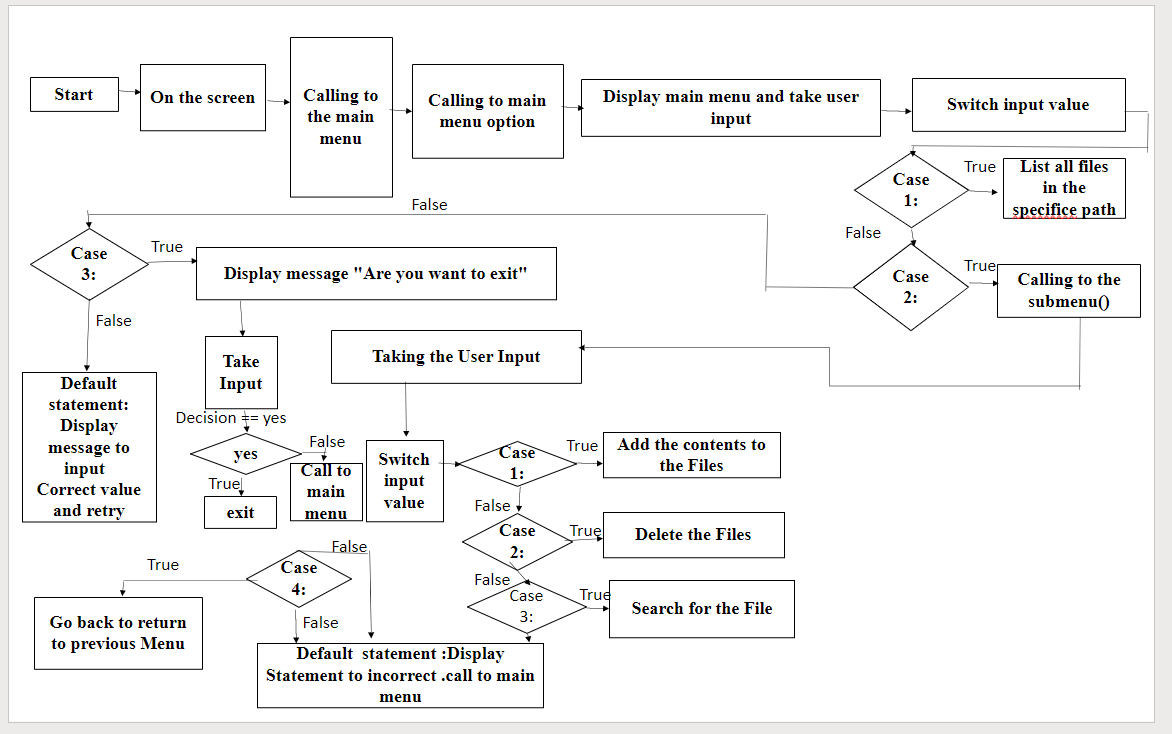
The project is planned to be completed in 2 sprints. Tasks assumed to be completed in the sprint are:

* 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.

## **Core concepts used in project**

Collections framework, File Handling, Searching, 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](#Step_1)
2. [Writing a program in Java for the entry point of the application (**LockedMe.java**)](#Step_2)
3. [Writing a program in Java to display Menu options available for the user (**MenuOptions.java**)](#Step_3)
4. [Writing a program in Java to handle Menu options selected by user (**HandleOptions.java**)](#Step_4)
5. [Writing a program in Java to perform the File operations as specified by user (**FileOperations.java**)](#Step_5)
6. [Pushing the code to GitHub repository](#Step_6)

## **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 gousia, click finish.
* Select your package and go to ->New->class.
* Create 3 classes named Main.java, objectives.java, project.java.

## **Step 2:** Writing a program in Java for the entry point of the application

**Code:**

## package gousia;

public class Main {

/\*Enter your desired Directory path \*/

public static final String path = "C:\\Users\\GOUSIA\\Desktop\\locked-me";

public static void main(String[] args) {

objectives menu = new objectives();

menu.introScreen();

menu.mainMenu();

}

## **Step 3:** Writing a program in Java to display Menu options available for the user

* Select your project and go to File -> New -> Class.
* Enter **MenuOptions** in class name and click on “Finish.”
* **MenuOptions** consists methods for -:
  1. [Displaying Welcome Screen](#Step_3_1)
  2. [Displaying Initial Menu](#Step_3_2)
  3. [Displaying Secondary Menu for File Operations available](#Step_3_3)

**Step 3.1: Writing method to display Welcome Screen**

**Code:**

package gousia;

import java.io.IOException;

import java.util.Scanner;

public class objectives {

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);

project dao = new project();

public void introScreen() {

System.out.println();

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

System.out.println("\* DEVELOPED BY GOUSIA SYED \*");

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");

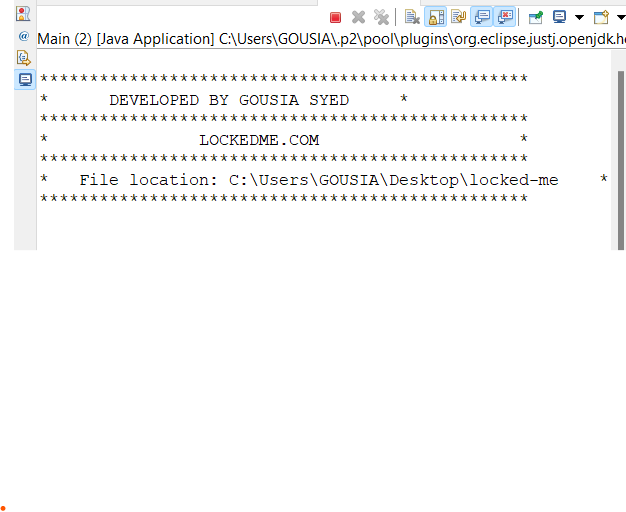
}

public void exitScreen() {

System.out.println("\* THANK YOU \*");

}

**Output:**



**Step 3.2: Writing method to display Initial Menu**

**Code:**

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");

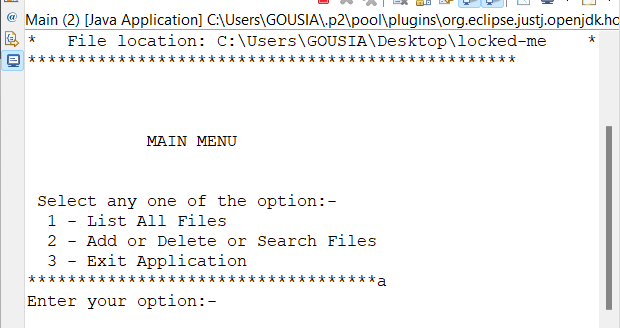
System.out.println(" 3 - Exit Application ");

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

System.out.println("Enter your option:- ");

}

**Output:**



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

**Code:**

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 ");

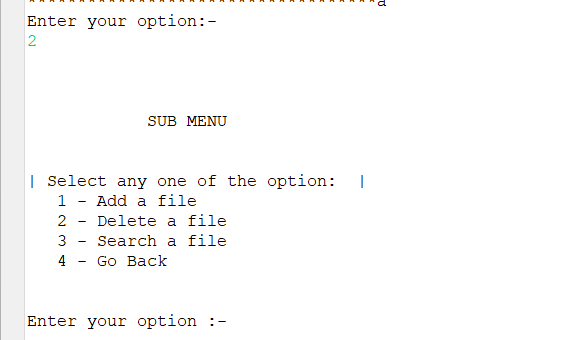
System.out.println(" 4 - Go Back ");

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

System.out.println("Enter your option :- ");

}

**Output:**



## **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 -:
  1. [Handling input selected by user in initial Menu](#Step_4_1)
  2. [Handling input selected by user in secondary Menu for File Operations](#Step_4_2)

**Step 4.1: Writing method to handle user input in initial Menu**

**Code:**

public void mainMenu() {

int choice = 0;

char decision = 0;

do {

mainMenuOptions();

try {

choice = Integer.parseInt(scan.nextLine());

} catch (NumberFormatException e) {

System.out.println("\nInvalid Input \nPlease select options between 1-3)\n");

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");

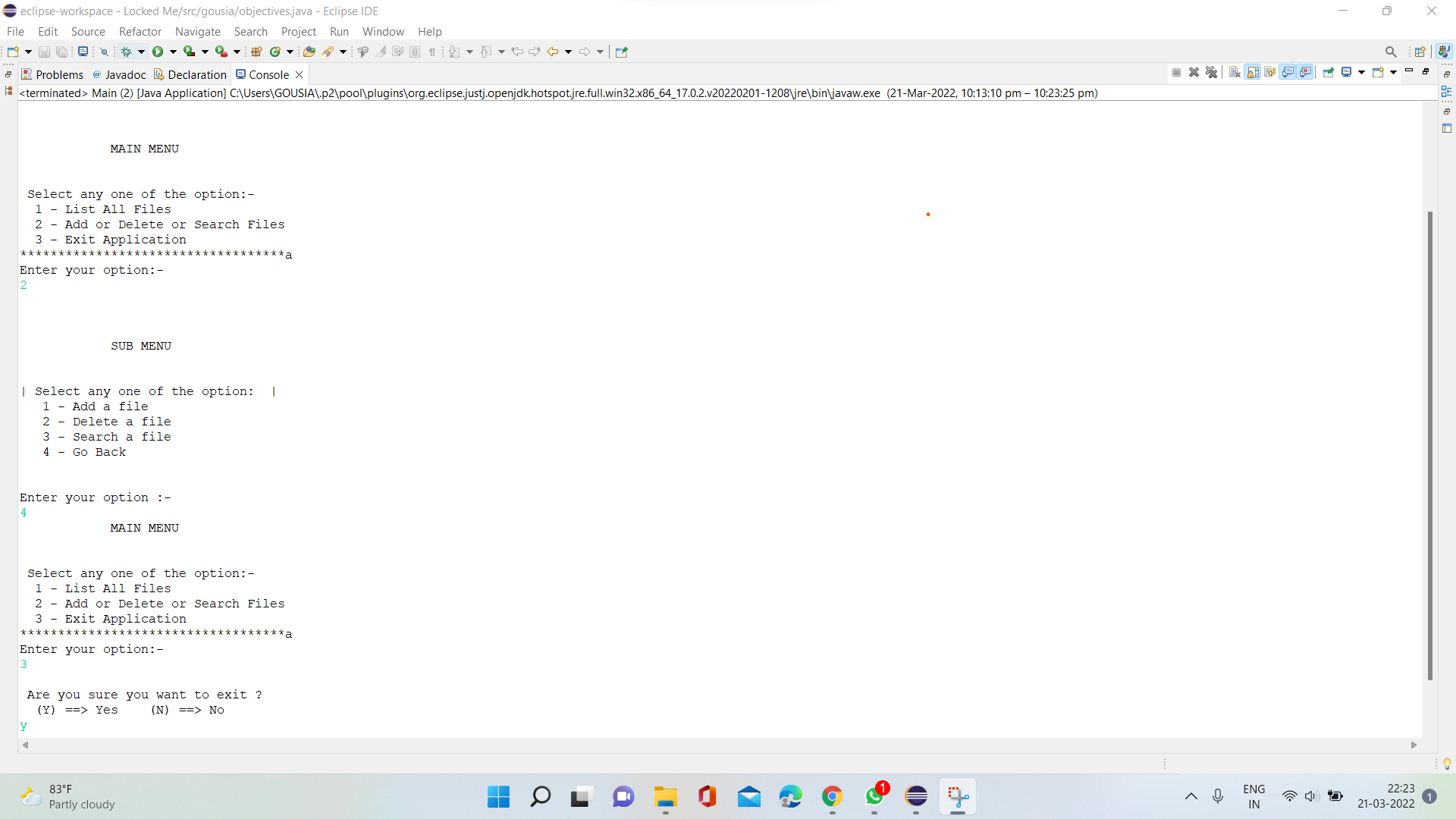
mainMenu();

}

}while(true);

}

**Output:**



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

**Code:**

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;

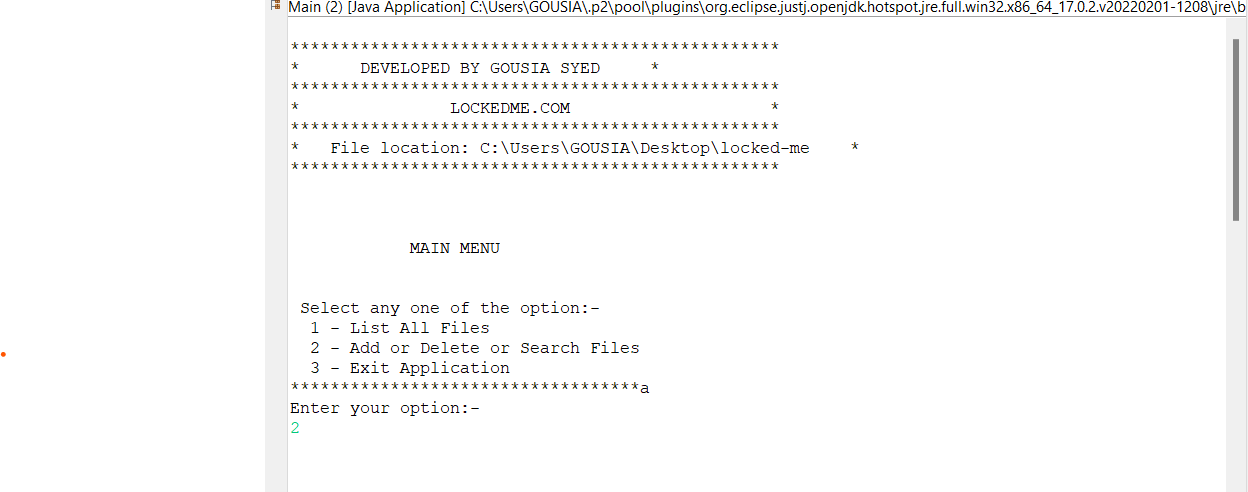
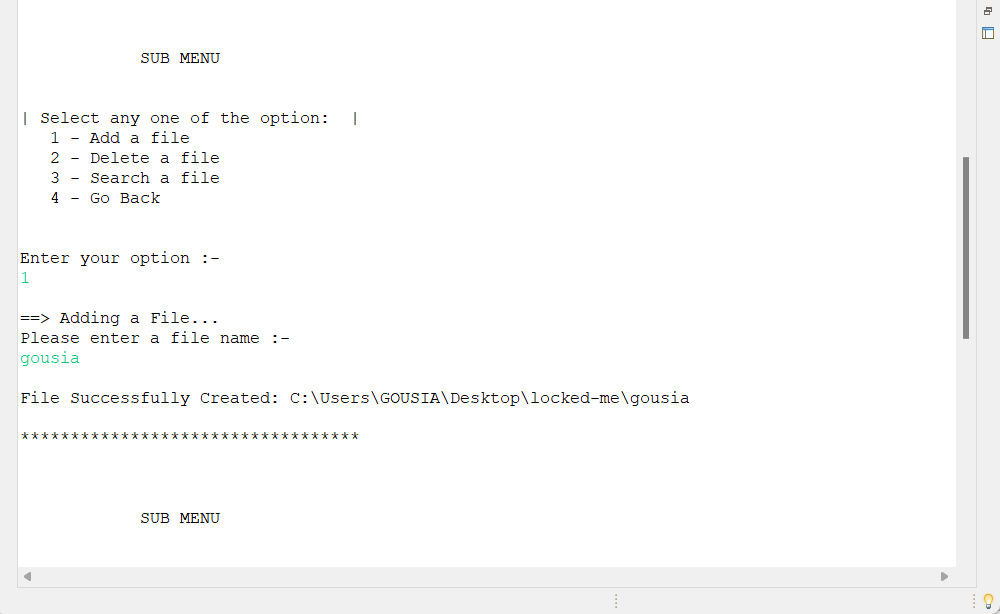
fileName = null;

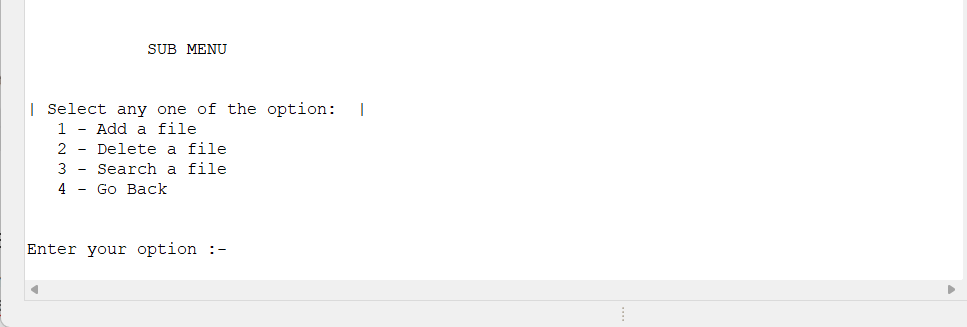
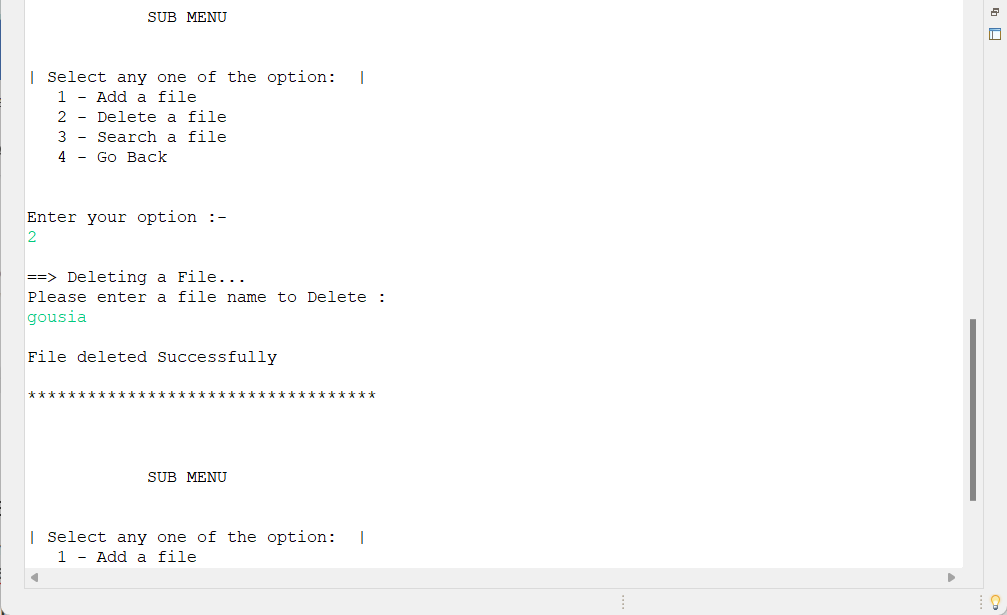
}while(true);

}

}

**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 -:
  1. [Creating “main” folder in project if it’s not already present](#Step_5_1)
  2. [Displaying all files in “main” folder in ascending order and also with directory structure.](#Step_5_2)
  3. [Creating a file/folder as specified by user input.](#Step_5_3)
  4. [Search files as specified by user input in “main” folder and it’s subfolders.](#Step_5_4)
  5. [Deleting a file/folder from “main” folder](#Step_5_5)

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

**Code:**

package gousia;

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 project {

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())

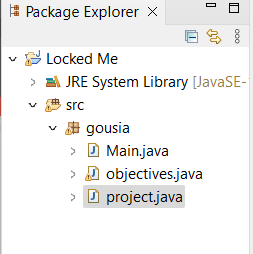
throw new IllegalArgumentException("Path does not exist");

if(dir.isFile())

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

String [] files = dir.list();

**Output:**



**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.)**

**Code:**

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

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());

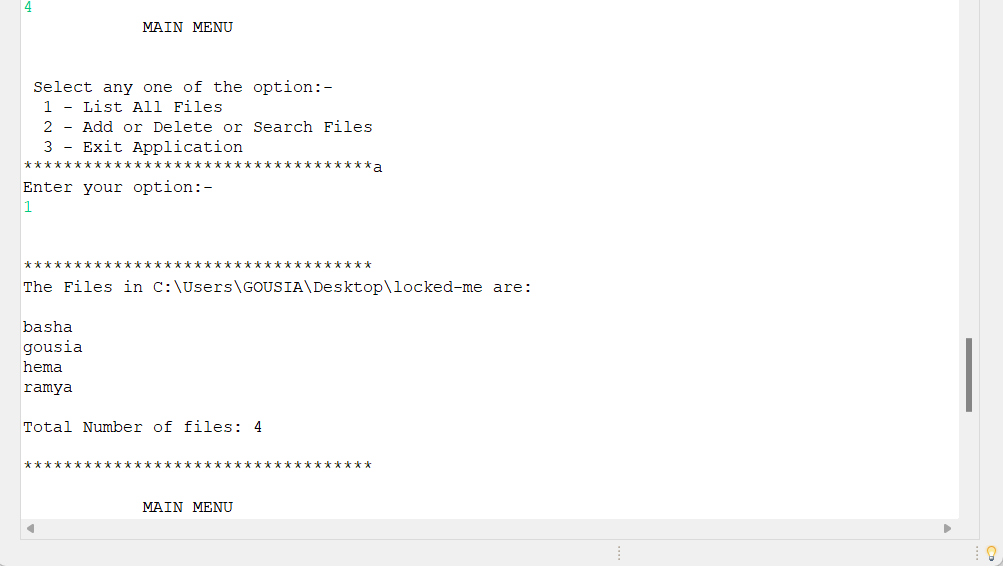
}else {

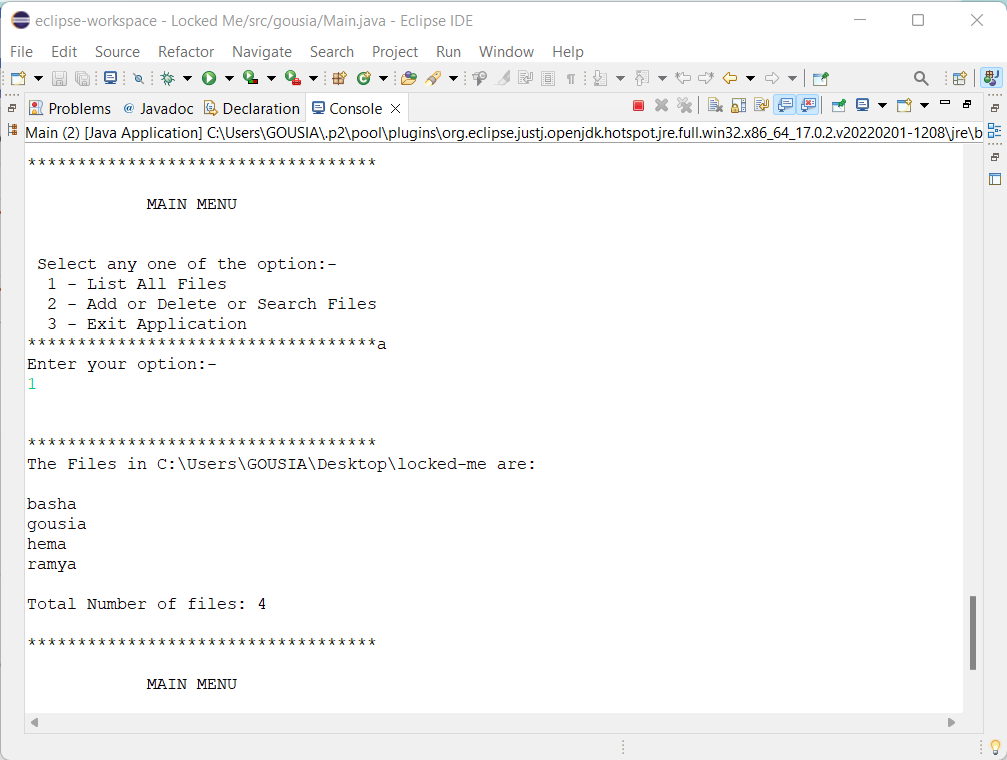
System.out.println("Directory is Empty");

}

}

**Output:**





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

**Code:**

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 {

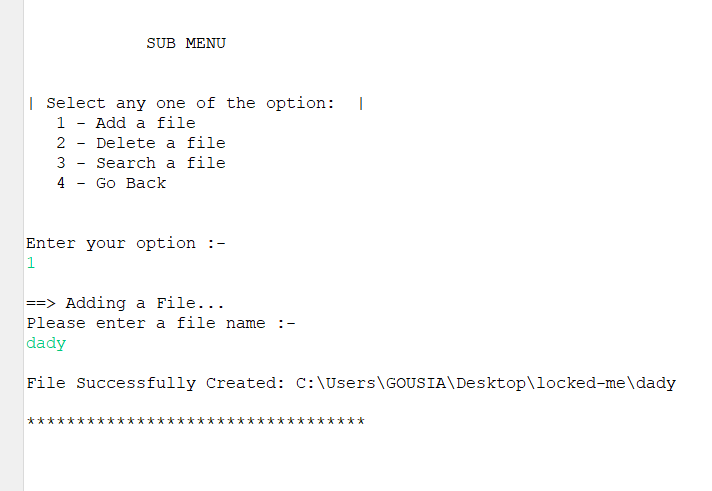
System.out.println("\nFile Not Found.. Please try again." );

}

}

**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.**

**Code:**

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)

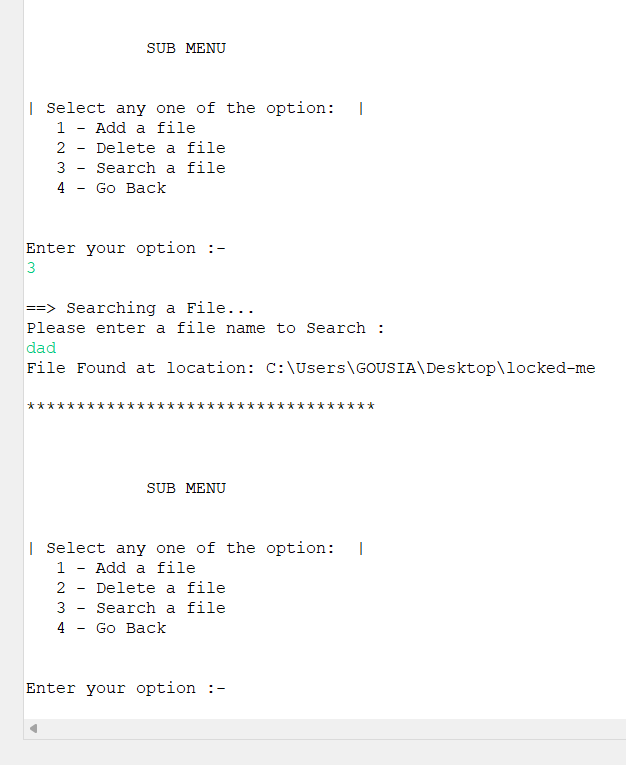
System.out.println("File Not Found.. Please try again.");

}

}

**Output:**

**All files starting with the user input are displayed along with index**



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

**Code:**

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 {

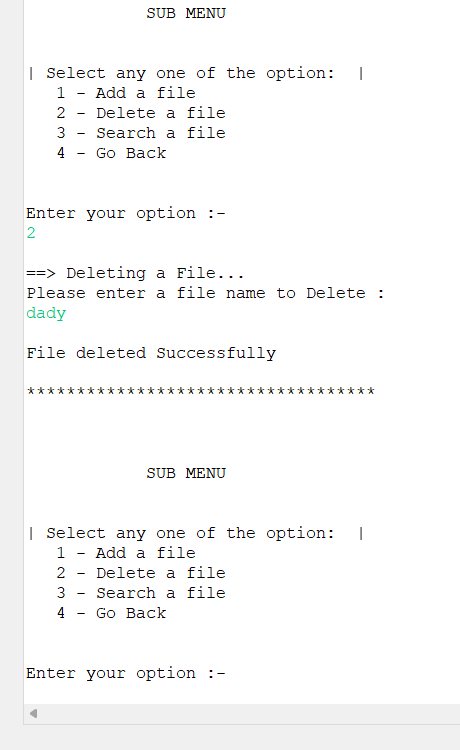
System.out.println("\nFile Not Found.. Please try again." );

}

}

**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>**

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

**git push -u origin master**

## **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.