**Phase1 – Virtual Key for Repositories**

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/maheshsatu/Corejavamainproject.git**

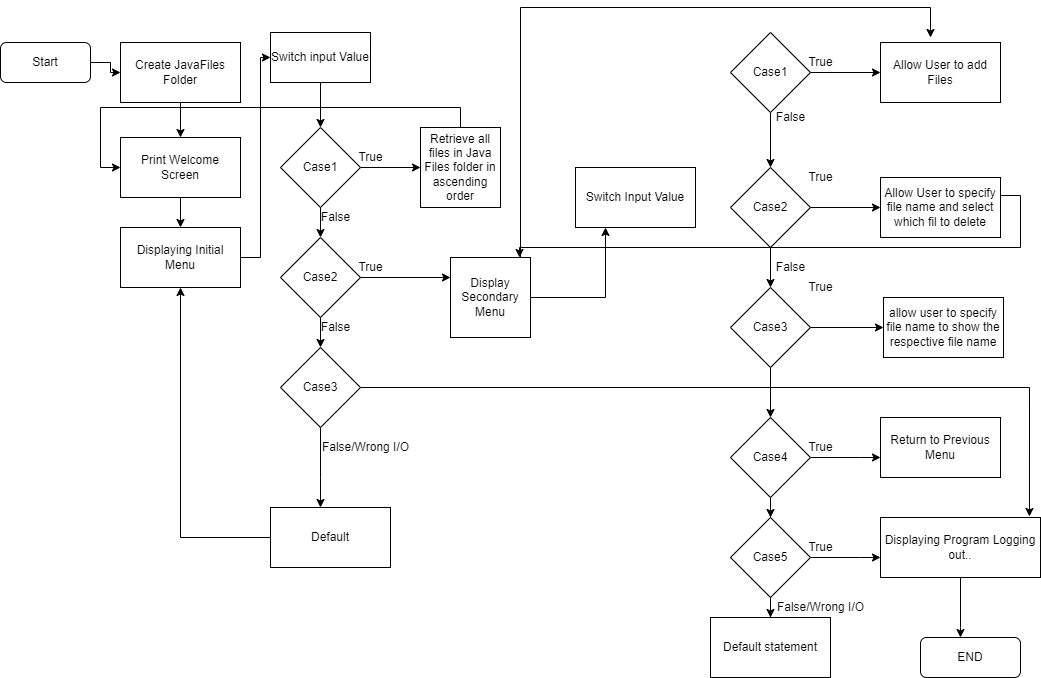
The project is developed by **SATULURISAIMAHESH**

## 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.
* sssssTesting 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 Chart:**



## 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 (**MainDef.java**)](#Step_2)
3. [Writing a program in Java to display Menu options available for the user (**Menu.java**)](#Step_3)
4. [Writing a program in Java to handle Menu options selected by user (**Controloptions.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.
* Type in any project name and click on “Finish.”
* Select your project and go to File -> New -> Class.
* Enter **MainDef** 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 (**M****ain.def.java)**

**package** com;

**public** **class** Maindef {

**public** **static** **void** main(String[] args) {

// Create "main" folder if not present in current folder structure

FileOperations.*createMainFolderIfNotPresent*("main");

Menu.*printWelcomeScreen*("LockedMe", "scared heart");

Controloptions.*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.”
* **package** com;
* **public** **class** Menu {
* **public** **static** **void** printWelcomeScreen(String appName, String developerName) {
* }
* **public** **static** **void** displayMenu() {
* String menu = "\n\n\*\*\*\*\*\* Select any option number from below 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);
* }
* **public** **static** **void** displayFileMenuOptions() {
* String fileMenu = "\n\n\*\*\*\*\*\* Select any option number from below and press Enter \*\*\*\*\*\*\n\n"
* + "1) Add a file to \"main\" folder\n" + "2) Delete a file from \"main\" folder\n"
* + "3) Search for a file from \"main\" folder\n" + "4) Show Previous Menu\n" + "5) Exit program\n";
* System.***out***.println(fileMenu);
* }
* }

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

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

package com;

import java.util.List;

import java.util.Scanner;

public class Controloptions {

public static void handleWelcomeScreenInput() {

boolean running = true;

Scanner sc = new Scanner(System.in);

do {

try {

Menu.displayMenu();

int input = sc.nextInt();

switch (input) {

case 1:

FileOperations.displayAllFiles("main");

break;

case 2:

Controloptions.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);

}

public static void handleFileMenuOptions() {

boolean running = true;

Scanner sc = new Scanner(System.in);

do {

try {

Menu.displayFileMenuOptions();

FileOperations.createMainFolderIfNotPresent("main");

int input = sc.nextInt();

switch (input) {

case 1:

// File Add

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

String fileToDelete = sc.next();

FileOperations.createMainFolderIfNotPresent("main");

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

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 \"main\" folder");

String fileName = sc.next();

FileOperations.createMainFolderIfNotPresent("main");

FileOperations.displayFileLocations(fileName, "main");

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);

}

}

## **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.”

**package com;**

**import java.io.File;**

**import java.io.IOException;**

**import java.nio.file.Files;**

**import java.nio.file.Path;**

**import java.nio.file.Paths;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.Collections;**

**import java.util.List;**

**import java.util.Scanner;**

**import java.util.stream.Collectors;**

**import java.util.stream.IntStream;**

**public class FileOperations {**

**public static void createMainFolderIfNotPresent(String folderName) {**

**File file = new File(folderName);**

**// If file doesn't exist, create the main folder**

**if (!file.exists()) {**

**file.mkdirs();**

**}**

**}**

**public static void displayAllFiles(String path) {**

**FileOperations.createMainFolderIfNotPresent("main");**

**// All required files and folders inside "main" 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;**

**}**

**public static void createFile(String fileToAdd, Scanner sc) {**

**FileOperations.createMainFolderIfNotPresent("main");**

**Path pathToFile = Paths.get("./main/" + 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 create file " + fileToAdd);**

**System.out.println(e.getClass().getName());**

**}**

**}**

**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\*\*\*\*\* 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);**

**}**

**}**

**}**

**}**

**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 {**

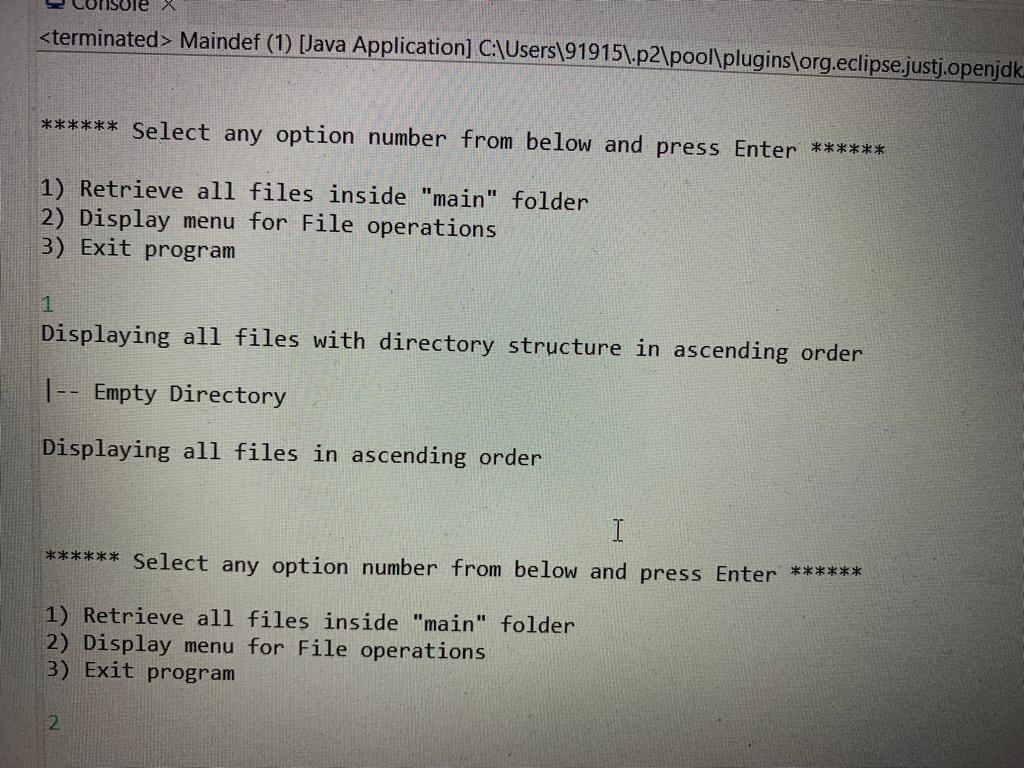
**System.out.println("Failed to delete " + currFileName);**

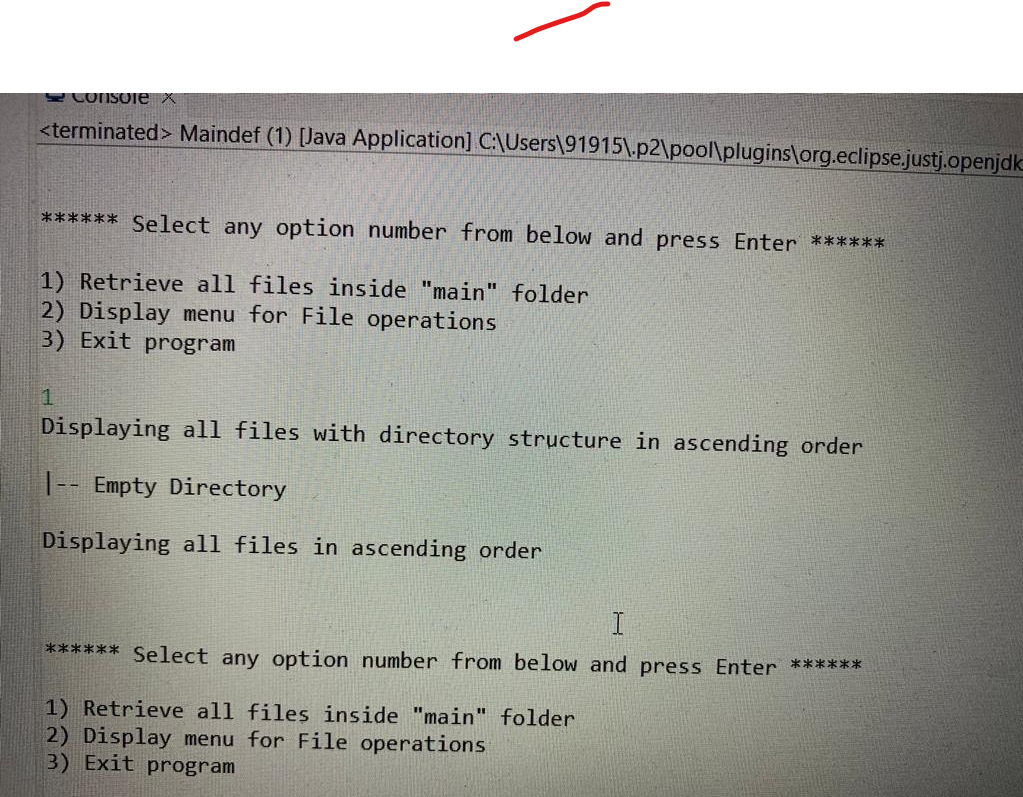
**}**

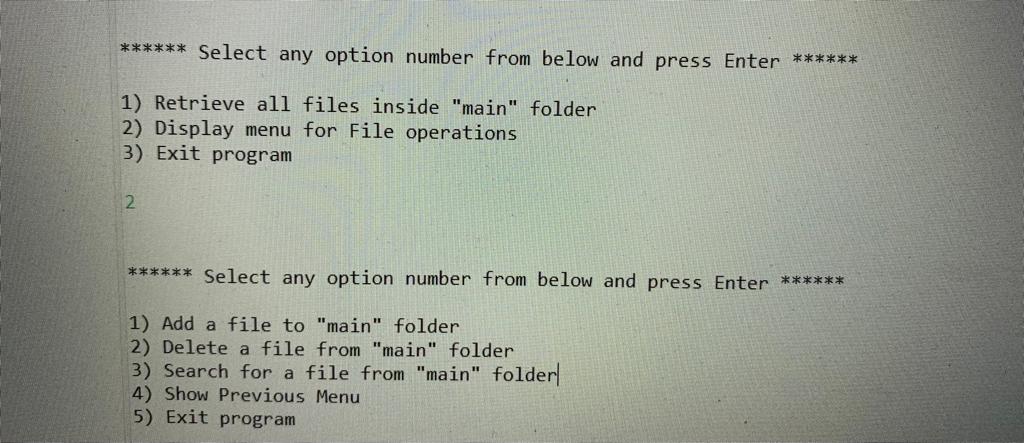
**}**

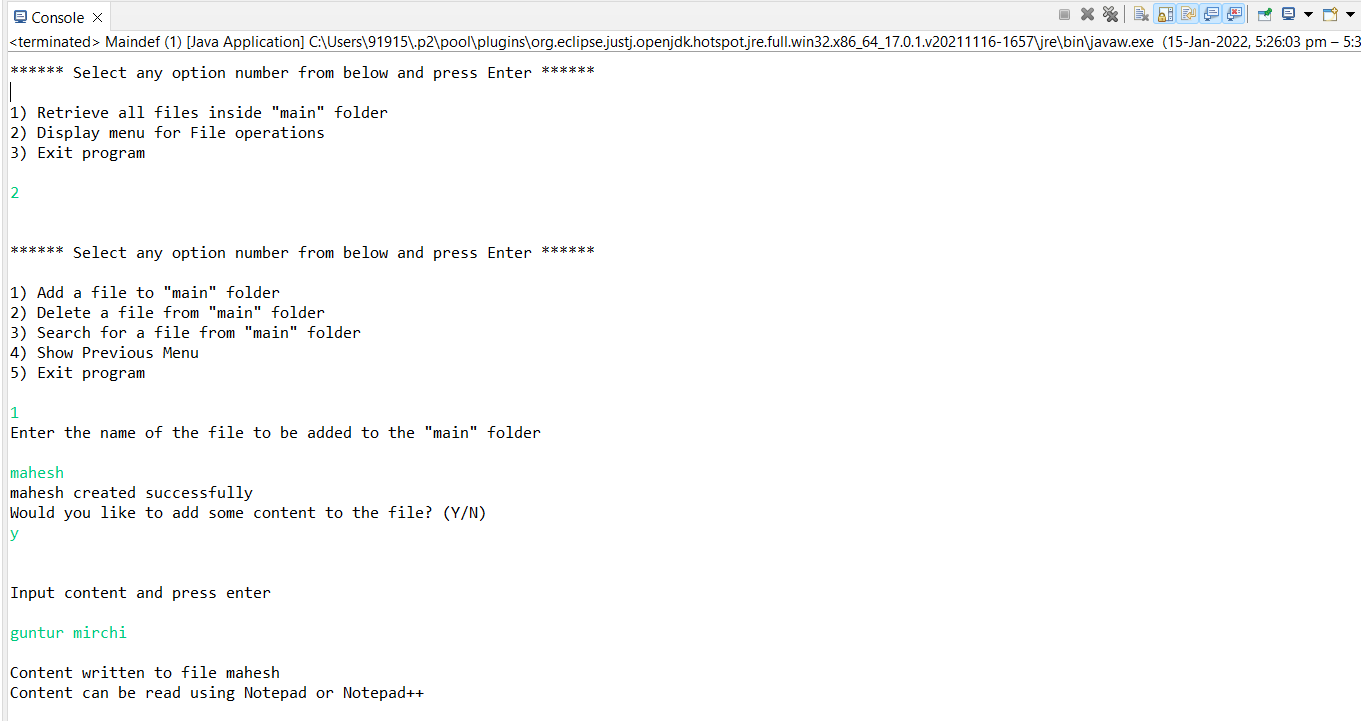
**}**

**Output’s:**

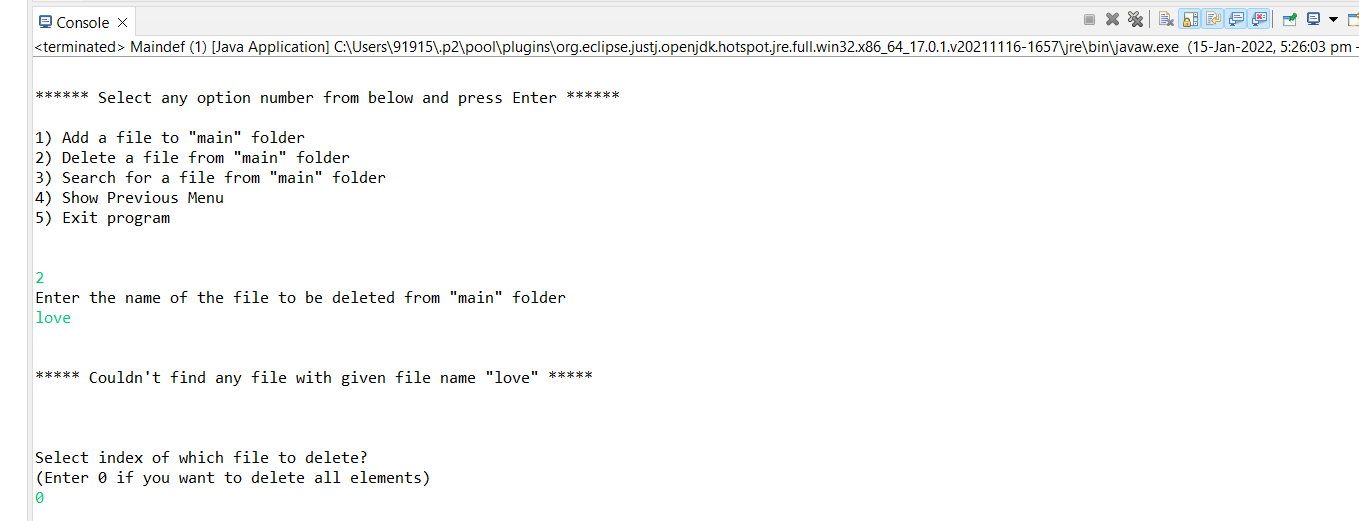
****

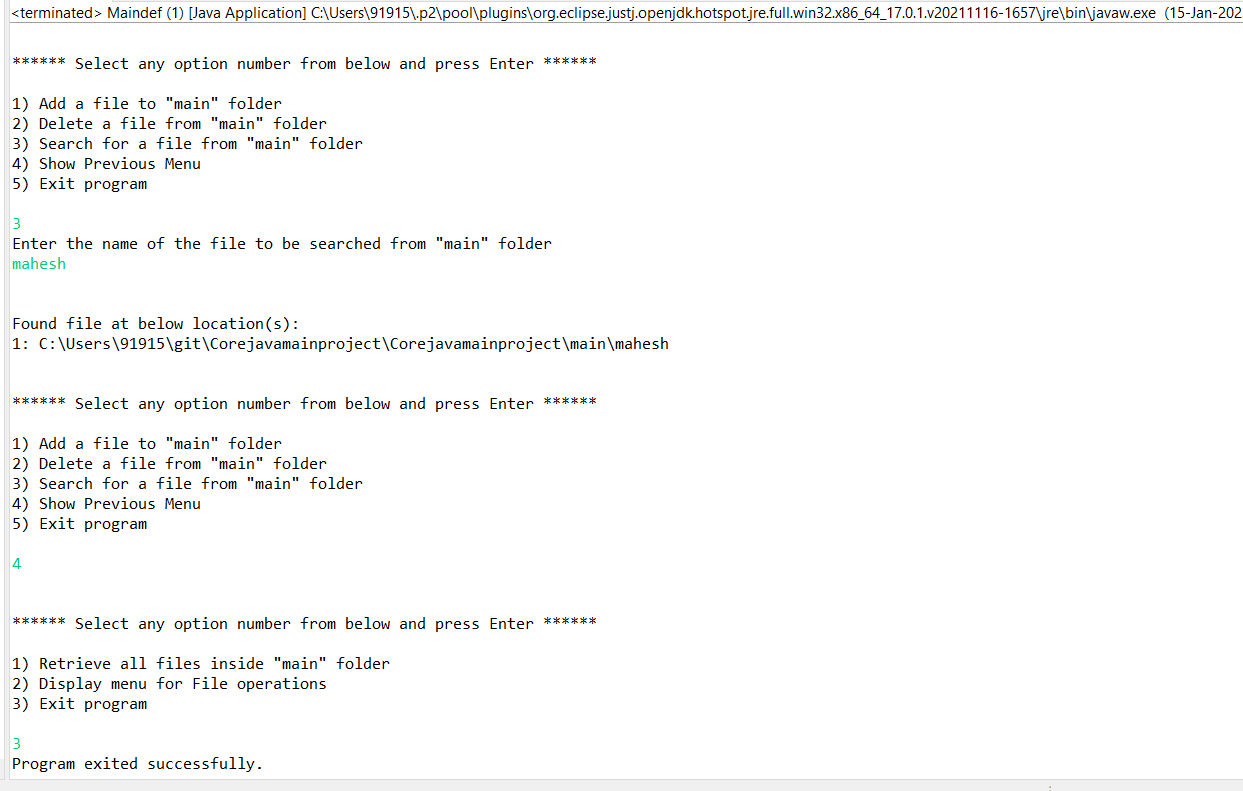


****

* 

****



**­­­­**

## **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.