

# **LockedMe – 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 project capabilities, appearance, and user inputs](#)
- [Conclusion](#)

The code for this project is hosted at

<https://github.com/nazishkn67/SimplilearnProject>

The project is developed by Nazish Kamran.

## **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.
- 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**

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

```

graph TD
    Start([Start]) --> CreateMain[Create "main" folder in Project directory if not present]
    CreateMain --> PrintWelcome[Print Welcome Screen and Application Function]
    PrintWelcome --> DisplayMenu[Display Initial Menu and Take User Input]
    DisplayMenu --> SwitchInput1[Switch Input Value]
    SwitchInput1 --> Case1_1{Case 1}
    Case1_1 -- True --> RetrieveFiles[Retrieve all files inside "main" folder and display in ascending order]
    RetrieveFiles --> DisplayMenu
    Case1_1 -- False --> Case2_1{Case 2}
    Case2_1 -- True --> DisplaySecondaryMenu[Display Secondary Menu for Performing File Operations and Take User Input]
    DisplaySecondaryMenu --> SwitchInput2[Switch Input Value]
    SwitchInput2 --> Case1_2{Case 1}
    Case1_2 -- True --> AddFile[Allow user to add file to "main" folder by taking input of file/folder name]
    AddFile --> DisplayMenu
    Case1_2 -- False --> Case2_2{Case 2}
    Case2_2 -- True --> DeleteFile[Allow user to specify input of filename and select which file/folder to delete]
    DeleteFile --> DisplayMenu
    Case2_2 -- False --> Case3_2{Case 3}
    Case3_2 -- True --> ShowFiles[Allow user to specify input of filename and show the respective files starting with the given name]
    ShowFiles --> DisplayMenu
    Case3_2 -- False --> Case4_2{Case 4}
    Case4_2 -- True --> ReturnMenu[Return to Previous Menu]
    ReturnMenu --> DisplayMenu
    Case4_2 -- False --> Case5_2{Case 5}
    Case5_2 -- True --> ProgramTerminated[Display Program Terminated Successfully]
    ProgramTerminated --> End([End])
    Case5_2 -- False / Wrong Input --> DefaultStatement1[Default Statement : Display Message to input correct value and retry]
    DefaultStatement1 --> DisplayMenu
    Case3_1{Case 3} -- False / Wrong Input --> DefaultStatement2[Default Statement : Display Message to input correct value and retry]
    DefaultStatement2 --> DisplayMenu
  
```

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 \(MainPage.java\)](#)
- 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.
- Type in any project name and click on “Finish.”
- Select your project and go to File -> New -> Class.
- Enter **MainPage** 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 (**MainPage.java**)

```
1 package com.simplilearn;
2
3 public class MainPage {
4
5     public static void main(String[] args) {
6
7         // Create "main" folder if not present in current folder structure
8         FileOperations.createMainFolderIfNotPresent("main");
9
10        MenuOptions.printWelcomeScreen("LockedMe.com");
11
12        HandleOptions.handleWelcomeScreenInput();
13    }
14
15
16 }
17
```

## 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](#)

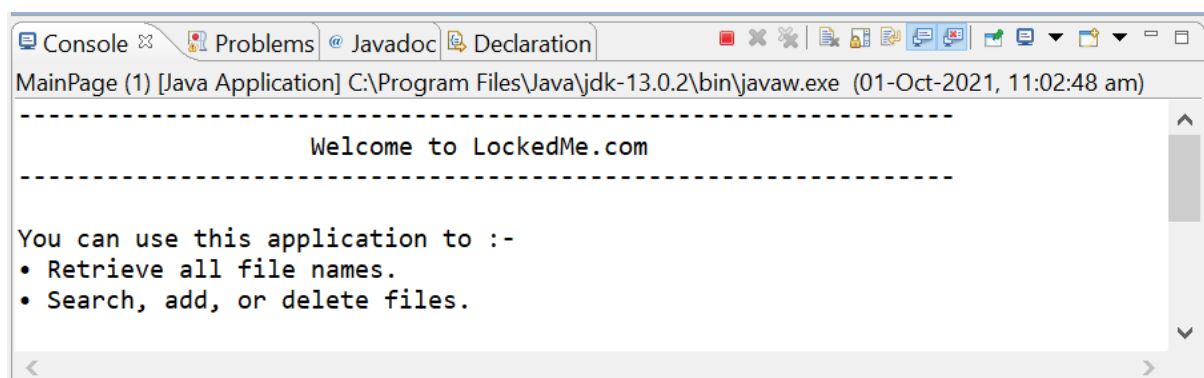
3.2. [Displaying Initial Menu](#)

3.3. [Displaying Secondary Menu for File Operations available](#)

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

```
4
5 public static void printWelcomeScreen(String appName) {
6     String companyDetails = String.format("-----\n"
7         + "                Welcome to %s \n"
8         + "-----\n", appName);
9     String appFunction = "You can use this application to :-\n"
10        + "• Retrieve all file names.\n"
11        + "• Search, add, or delete files.\n";
12     System.out.println(companyDetails);
13
14     System.out.println(appFunction);
15 }
16
```

#### Output:



```
Console Problems Javadoc Declaration
MainPage (1) [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (01-Oct-2021, 11:02:48 am)

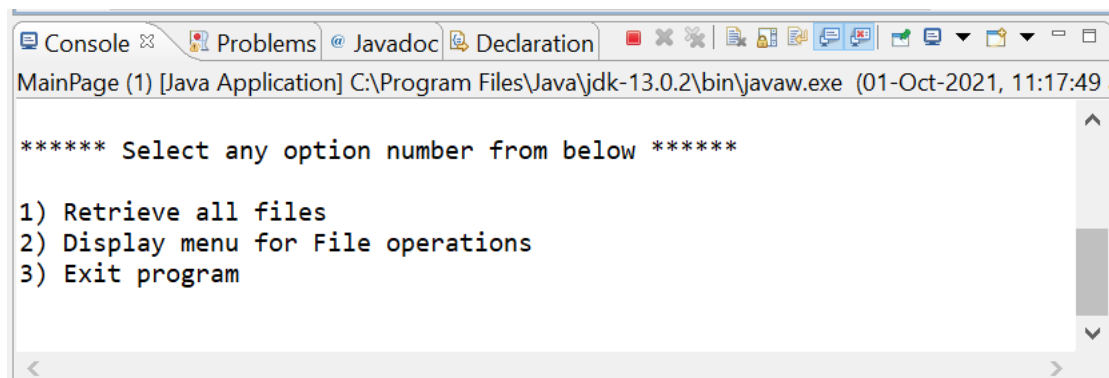
-----
Welcome to LockedMe.com
-----

You can use this application to :-
• Retrieve all file names.
• Search, add, or delete files.
```

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

```
16
17 public static void displayMenu() {
18     String menu = "\n\n***** Select any option number from below *****\n\n"
19         + "1) Retrieve all files\n" + "2) Display menu for File operations\n"
20         + "3) Exit program\n";
21     System.out.println(menu);
22
23 }
24
```

#### Output:



```
Console Problems Javadoc Declaration
MainPage (1) [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (01-Oct-2021, 11:17:49)

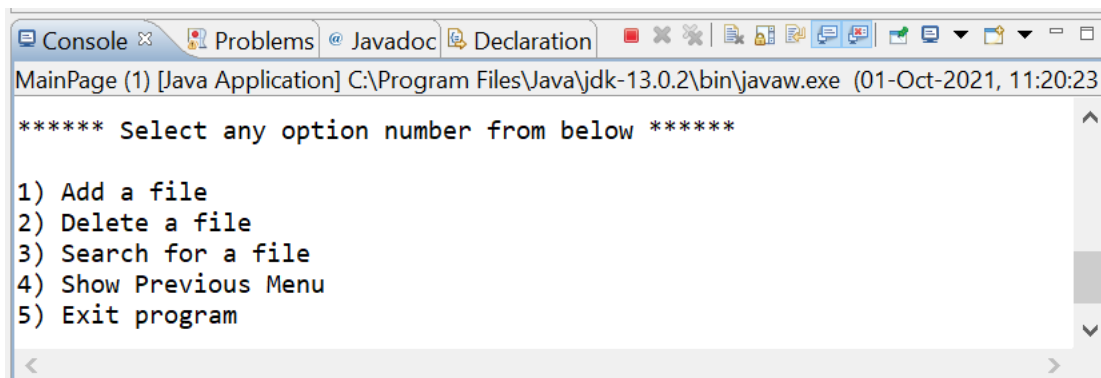
***** Select any option number from below *****

1) Retrieve all files
2) Display menu for File operations
3) Exit program
```

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

```
24
25 public static void displayFileMenuOptions() {
26     String fileMenu = "\n\n***** Select any option number from below *****\n\n"
27         + "1) Add a file\n" + "2) Delete a file\n"
28         + "3) Search for a file\n" + "4) Show Previous Menu\n" + "5) Exit program\n";
29
30     System.out.println(fileMenu);
31 }
32
```

#### Output:



```
Console x Problems @ Javadoc Declaration
MainPage (1) [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (01-Oct-2021, 11:20:23)
***** Select any option number from below *****
1) Add a file
2) Delete a file
3) Search for a file
4) Show Previous Menu
5) Exit program
```

### 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 static void handleWelcomeScreenInput() {
    boolean running = true;
    Scanner sc = new Scanner(System.in);
    do {
        try {
            MenuOptions.displayMenu();
            int input = sc.nextInt();

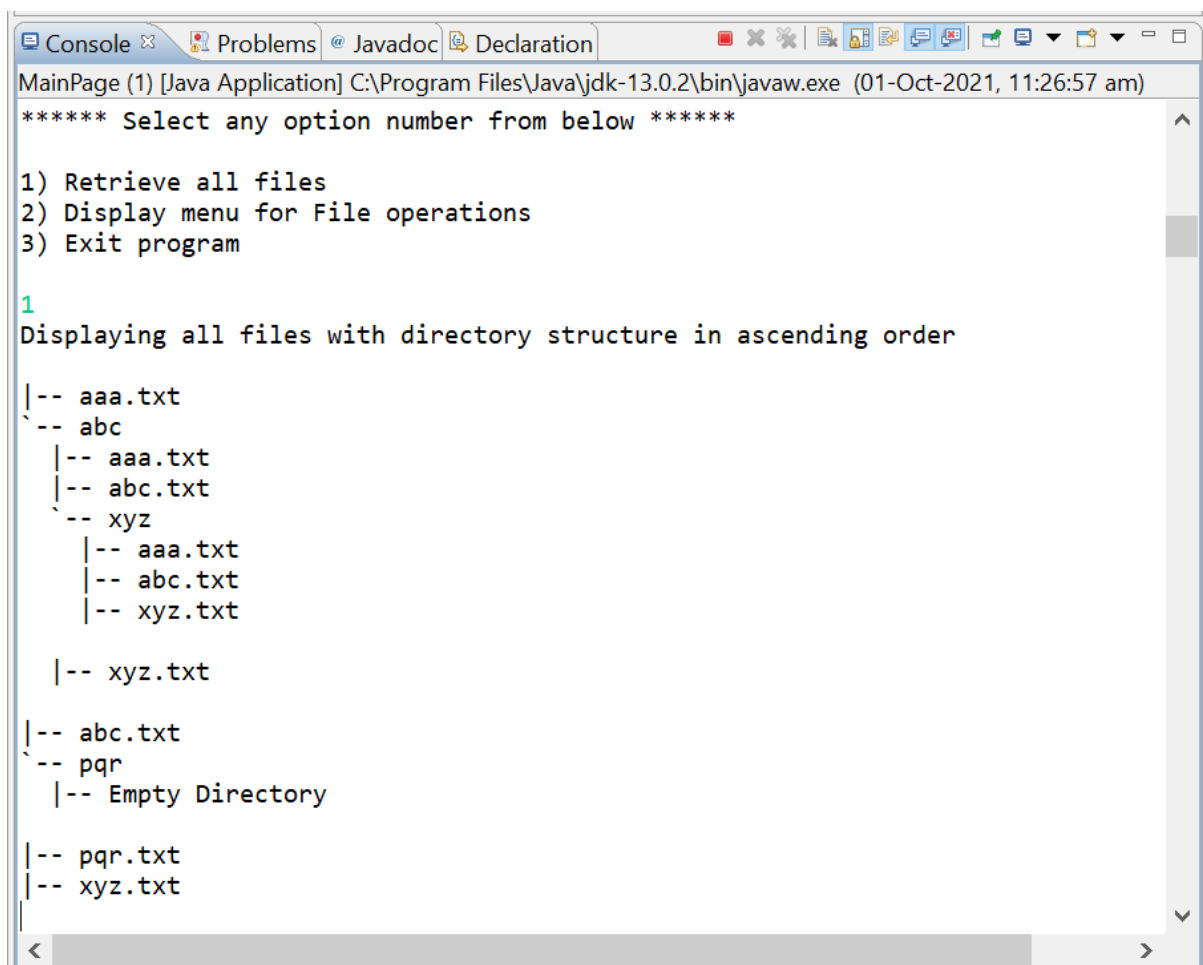
            switch (input) {
```

```

        case 1:
            FileOperations.displayAllFiles("main");
            break;
        case 2:
            HandleOptions.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:



The screenshot shows a Java IDE window with a console tab active. The console displays the output of a Java application. At the top, it says "\*\*\*\*\* Select any option number from below \*\*\*\*\*". Below this, a list of options is shown: "1) Retrieve all files", "2) Display menu for File operations", and "3) Exit program". The user has entered "1", and the application responds by displaying all files with directory structure in ascending order. The output is a tree-like structure of files and directories, including "aaa.txt", "abc", "xyz", and "Empty Directory".

```

MainPage (1) [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (01-Oct-2021, 11:26:57 am)
***** Select any option number from below *****

1) Retrieve all files
2) Display menu for File operations
3) Exit program

1
Displaying all files with directory structure in ascending order

|-- aaa.txt
|-- abc
    |-- aaa.txt
    |-- abc.txt
    |-- xyz
        |-- aaa.txt
        |-- abc.txt
        |-- xyz.txt
|-- xyz.txt
|-- abc.txt
|-- pqr
    |-- Empty Directory
|-- pqr.txt
|-- xyz.txt

```

## 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 {
            MenuOptions.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();

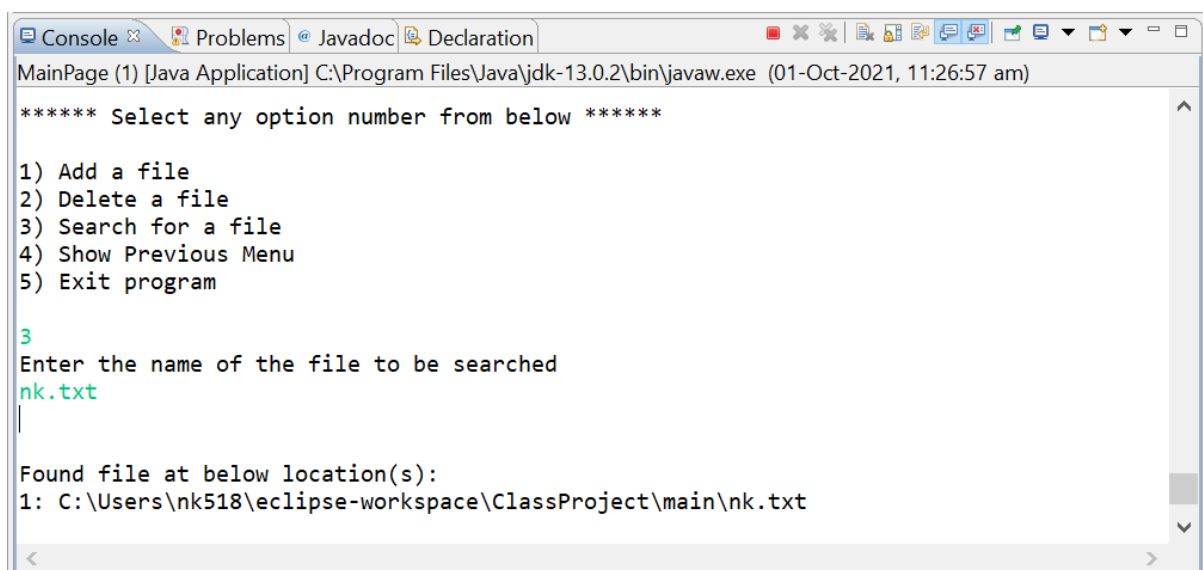
        FileOperations.createMainFolderIfNotPresent("main");
        FileOperations.displayFileLocations(fileName,
"main");

        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);
}

```

## Output:



The screenshot shows a Java IDE console window with the following output:

```

MainPage (1) [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (01-Oct-2021, 11:26:57 am)
***** Select any option number from below *****
1) Add a file
2) Delete a file
3) Search for a file
4) Show Previous Menu
5) Exit program

3
Enter the name of the file to be searched
nk.txt

Found file at below location(s):
1: C:\Users\nk518\eclipse-workspace\ClassProject\main\nk.txt

```



## 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.](#)

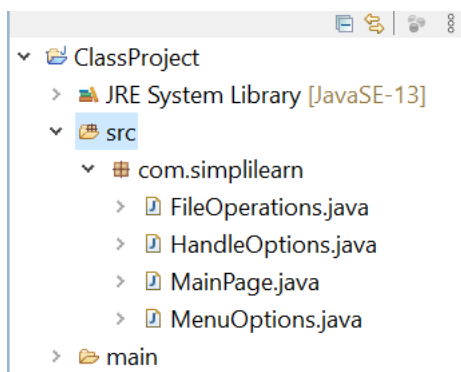
5.4.[Search files as specified by user input in “main” folder and it’s subfolders.](#)

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

```
17
18 public static void createMainFolderIfNotPresent(String folderName) {
19     File file = new File(folderName);
20
21     // If file doesn't exist, create the main folder
22     if (!file.exists()) {
23         file.mkdirs();
24     }
25 }
26
```

### 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.)

```
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> fileListNames =
FileOperations.listFilesInDirectory(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> 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);

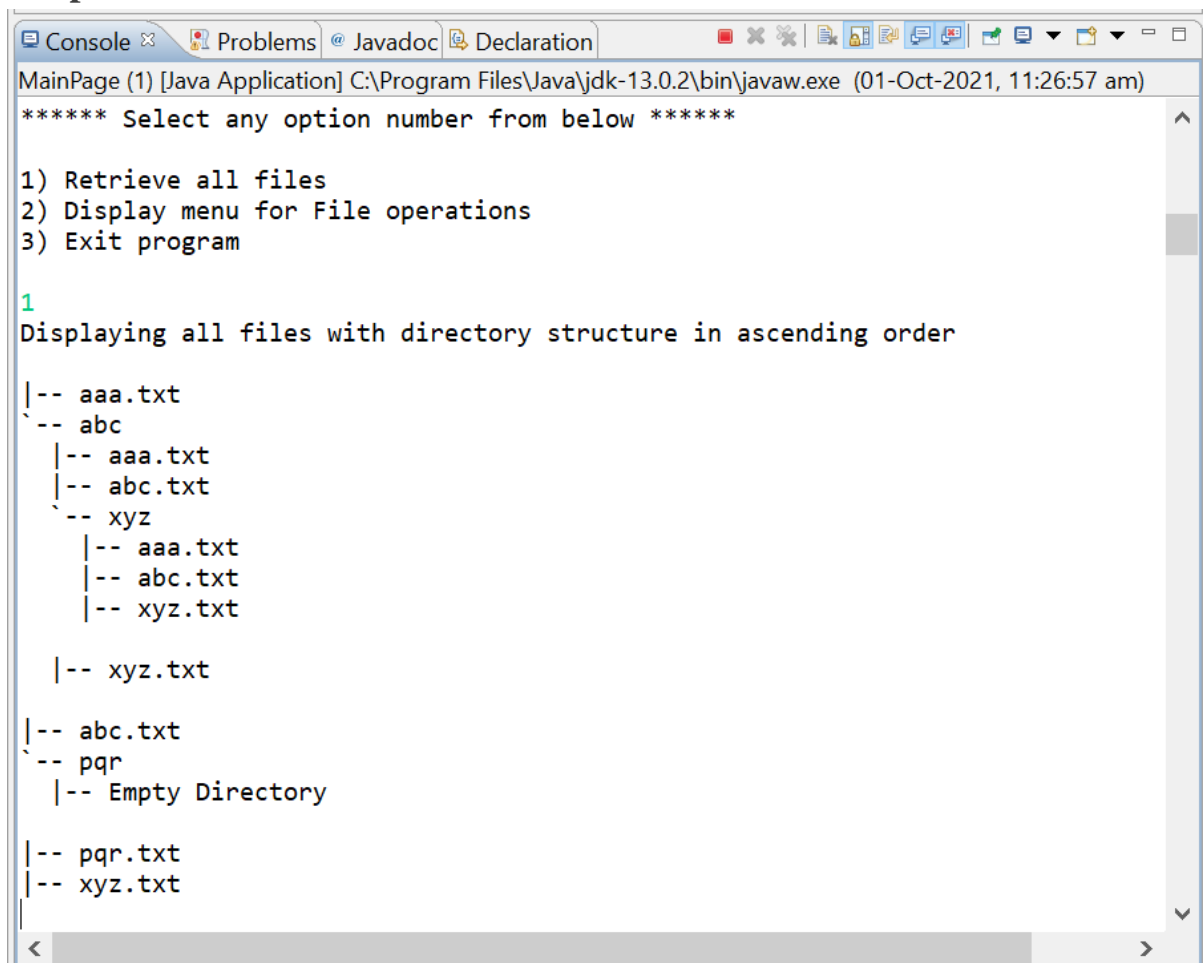
    if (files != null && files.length > 0) {
        for (File file : filesList) {

            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;
}
```

## Output:



The screenshot shows a Java IDE window with a console tab active. The console displays the output of a Java application. At the top, it says 'MainPage (1) [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (01-Oct-2021, 11:26:57 am)'. Below this, the program prompts the user to 'Select any option number from below'. The user has entered '1', and the program displays a directory structure of files and folders in ascending order. The output is as follows:

```
***** Select any option number from below *****

1) Retrieve all files
2) Display menu for File operations
3) Exit program

1
Displaying all files with directory structure in ascending order

|-- aaa.txt
|-- abc
    |-- aaa.txt
    |-- abc.txt
    |-- xyz
        |-- aaa.txt
        |-- abc.txt
        |-- xyz.txt
|-- xyz.txt

|-- abc.txt
|-- pqr
    |-- Empty Directory

|-- pqr.txt
|-- xyz.txt
```

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

```
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());
    }
}

```

## Output:

### Folders are automatically created along with file

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

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

1

Enter the name of the file to be added to the "main" folder

/testing/with/folder/creation/test\_file.txt

/testing/with/folder/creation/test\_file.txt created successfully

Would you like to add some content to the file? (Y/N)

Y

Input content and press enter

Checking if file content written in specified file.

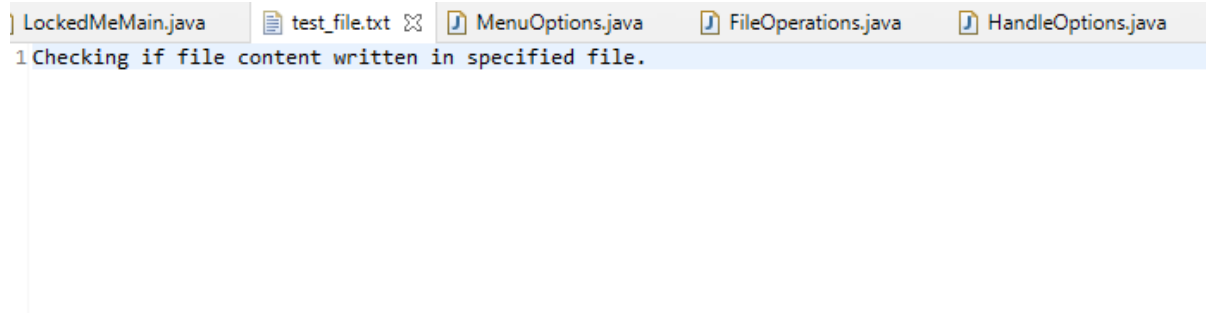
Content written to file /testing/with/folder/creation/test\_file.txt

Content can be read using Notepad or Notepad++

```

> src
> JRE System Library [jdk-12.0.2]
v main
  > abc
  > def
  v testing
    v with
      v folder
        v creation
          test_file.txt
          def.txt
          dkb.txt
          kjb.txt
          sks.txt

```



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

```
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> fileList = Arrays.asList(files);

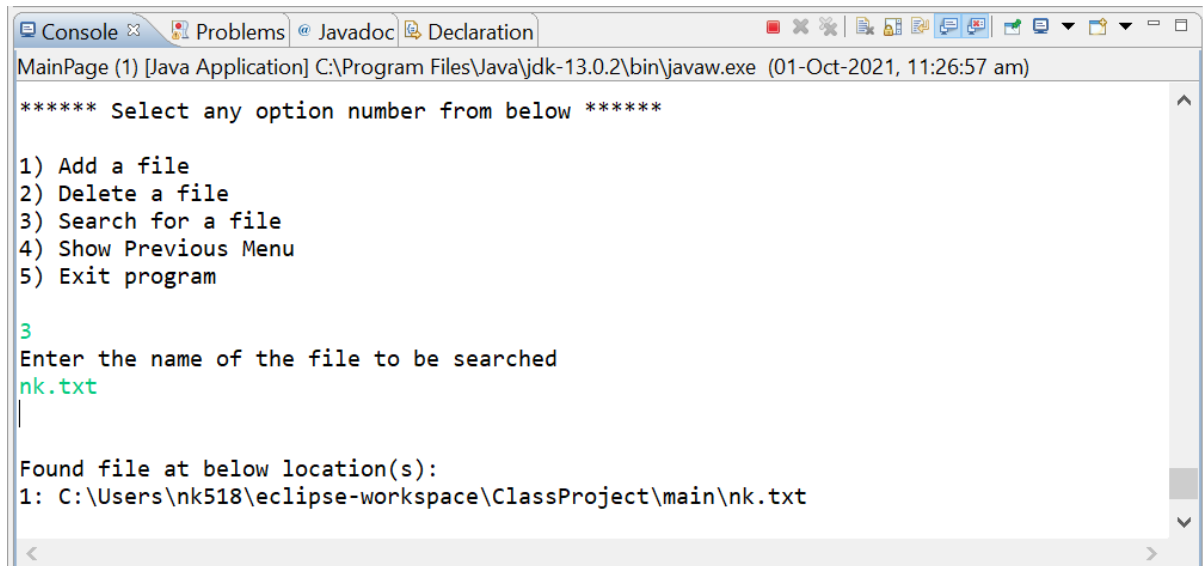
    if (files != null && files.length > 0) {
        for (File file : fileList) {

            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:

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

A screenshot of a Java application window titled 'MainPage (1) [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (01-Oct-2021, 11:26:57 am)'. The window has tabs for 'Console', 'Problems', 'Javadoc', and 'Declaration'. The 'Console' tab is active, displaying the following text: '\*\*\*\*\* Select any option number from below \*\*\*\*\*', a numbered list (1) Add a file, (2) Delete a file, (3) Search for a file, (4) Show Previous Menu, (5) Exit program, the number '3' (highlighted in green), 'Enter the name of the file to be searched', 'nk.txt' (highlighted in green), and 'Found file at below location(s):'. Below this, it lists '1: C:\Users\nk518\eclipse-workspace\ClassProject\main\nk.txt'.

```
***** Select any option number from below *****
1) Add a file
2) Delete a file
3) Search for a file
4) Show Previous Menu
5) Exit program

3
Enter the name of the file to be searched
nk.txt

Found file at below location(s):
1: C:\Users\nk518\eclipse-workspace\ClassProject\main\nk.txt
```

**Step 5.5:** Writing method to delete file/folder specified by user input in “main” 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);
            }
        }
    }
}
```

```

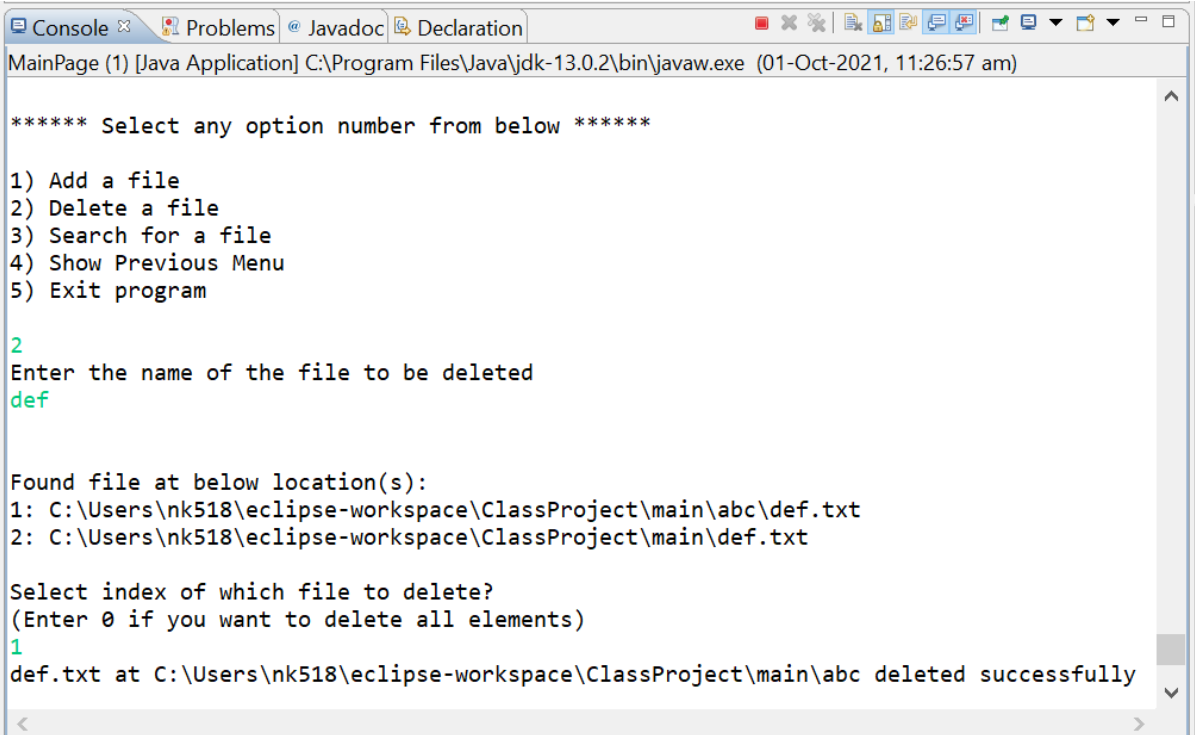
        }
    }

    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”.



```

MainPage (1) [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (01-Oct-2021, 11:26:57 am)

***** Select any option number from below *****
1) Add a file
2) Delete a file
3) Search for a file
4) Show Previous Menu
5) Exit program

2
Enter the name of the file to be deleted
def

Found file at below location(s):
1: C:\Users\nk518\eclipse-workspace\ClassProject\main\abc\def.txt
2: C:\Users\nk518\eclipse-workspace\ClassProject\main\def.txt

Select index of which file to delete?
(Enter 0 if you want to delete all elements)
1
def.txt at C:\Users\nk518\eclipse-workspace\ClassProject\main\abc deleted successfully

```

## 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**

## Conclusion

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.