# **LockedMe – Virtual Key for Repositories**

This document contains sections for:

* [Sprint planning](#_Sprints_planning_and)
* [Concepts used in project](#_Core_concepts_used)
* [Flowchart of application](#Flow)
* [Demonstration of product capabilities, appearance and user interaction](#_Demonstration_of_product)
* [Selling Points of application](#_Selling_Points_of)
* [Scope of advancement in application](#_Scope_of_advancement)

The code for this project is hosted at <https://github.com/kanika880/LockedMe.com> .

The project is developed by Kanika Gupta.

## **Sprints planning and Task completion**

The project is planned to be completed in 2 sprints. Tasks assumed to be completed in first 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.

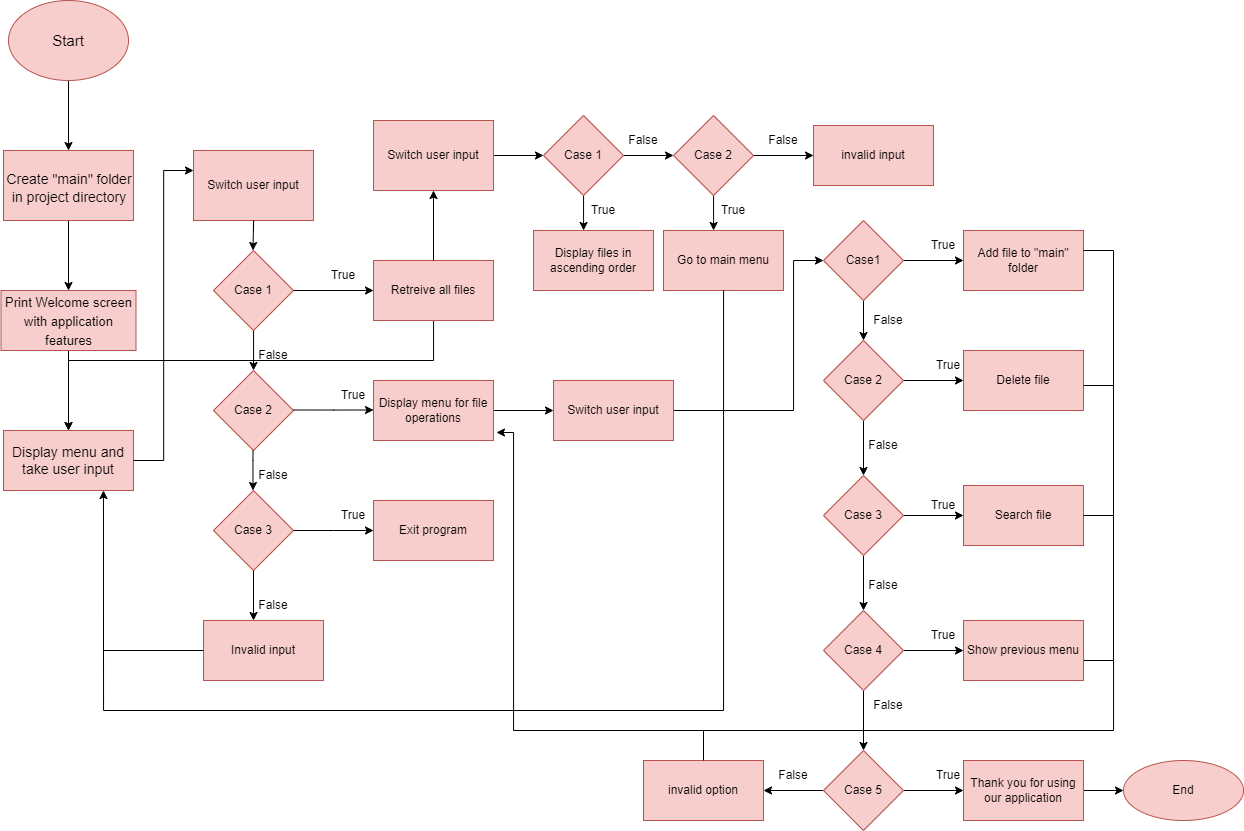
Tasks assumed to be completed in the second sprint are:

* 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, Exception Handling, Switch case, Streams API

## **Flow of the Application**



## **Demonstration of product capabilities, appearance, and user interactions**

The following sub-sections have been configured to highlight the project's appearance and user interactions in order to demonstrate the product's capabilities:

1. [Creating the project in Eclipse](#_Step_1:_Creating)
2. [Writing a Java program for the application's entry point (**LockedMeMain.java**)](#_Step_2:_Writing)
3. [Writing a Java program to display Menu options available for the user (**MenuOptions.java**).](#_Step_3:_Writing)
4. [Writing a JavaProgram to perform the File operations as specified by user (**FileOperations.java**)](#_Step_4:_Writing)
5. [Writing a program in Java to create file as specified by user (**FileAddOperations.java**)](#_Step_5:_Writing)
6. [Writing a program in Java to delete file as specified by user (**FileDeleteOperations.java**)](#_Step_6:_Writing)
7. [Writing a program in Java to search file as specified by user in “main” folder (**FileSearchOperations.java**)](#_Step_7:_Writing)
8. [Pushing the code to GitHub repository](#_Step_8:_Pushing)

### **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 **LockedMeMain** in any class name, and click on “Finish”.

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

**public** **class** LockedMeMain {

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

DisplayMenu.*welcomeScreen*();

DisplayMenu.*displayMenu*();

}

}

### **Step 3: Writing a Java program to display Menu options available for the user (DisplayMenu.java)**

* Select your project and go to File -> New -> Class.
* Enter **DisplayMenu** in class name and click on “Finish.”
* **DisplayMenu** consists methods for -:
  1. Display welcome screen.
  2. Display menu options and handle them.
  3. Display File menu options and handle them

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

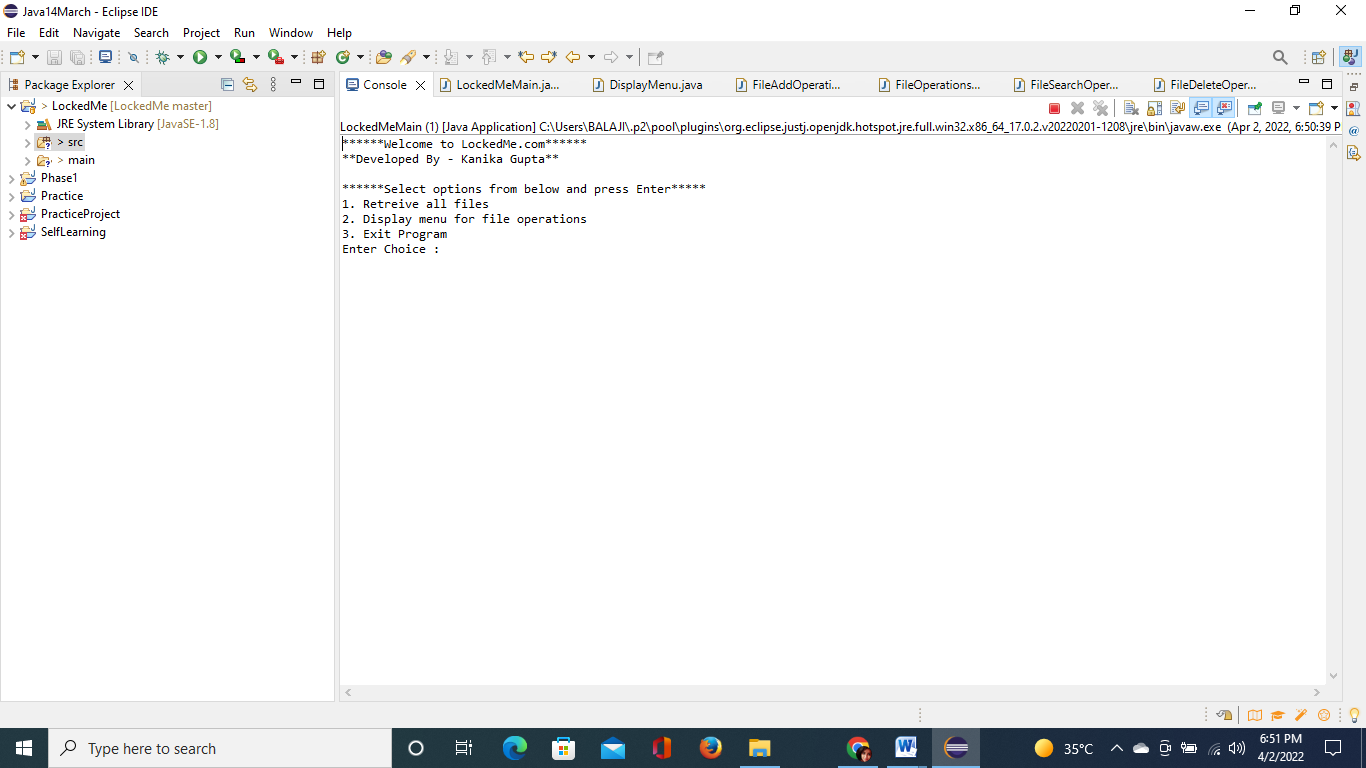
**public** **static** **void** welcomeScreen() {

System.***out***.println("\*\*\*\*\*\*Welcome to LockedMe.com\*\*\*\*\*\*");

System.***out***.println("\*\*Developed By - Kanika Gupta\*\*");

}

**Output:**



**Step 3.2:** Writing method to display menu options and to handle them.

**public** **static** **void** displayMenu() {

**do** {

System.***out***.println("\n\*\*\*\*\*\*Select options from below and press Enter\*\*\*\*\*");

System.***out***.println("1. Retreive all files");

System.***out***.println("2. Display menu for file operations");

System.***out***.println("3. Exit Program");

**try**{

System.***out***.print("Enter Choice : ");

*choice* = *sc*.nextInt();

}

**catch**(InputMismatchException e) {

System.***out***.println("Invalid Option. Please enter valid option.");

DisplayMenu.*displayMenu*();

}

**switch**(*choice*) {

**case** 1:

FileOperations.*display*();

**break**;

**case** 2:

DisplayMenu.*displayFileMenu*();

**break**;

**case** 3:

System.***out***.println("Thank you for using our application!");

**break**;

**default**:

System.***out***.println("Invalid Option. Please enter a valid option.");

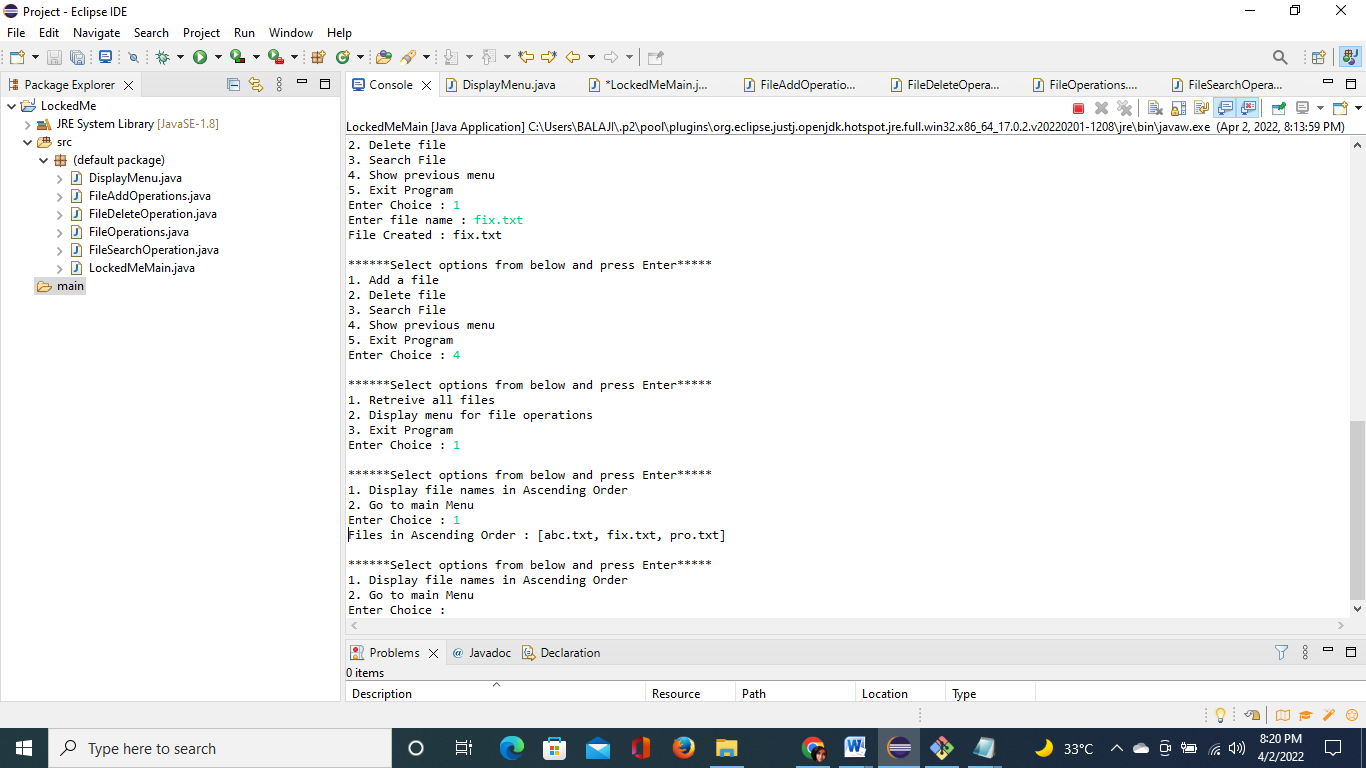
}

}

**while**( *choice*!=3 );

}

**Output:**



**Step 3.3:** Writing method to display menu for file operations and to handle them.

**public** **static** **void** displayFileMenu() {

System.***out***.println("\n\*\*\*\*\*\*Select options from below and press Enter\*\*\*\*\*");

System.***out***.println("1. Add a file");

System.***out***.println("2. Delete file");

System.***out***.println("3. Search File");

System.***out***.println("4. Show previous menu");

System.***out***.println("5. Exit Program");

**try** {

System.***out***.print("Enter Choice : ");

*choice* = *sc*.nextInt();

}**catch**(Exception e) {

System.***out***.println("Invalid Option. Please enter valid option.");

DisplayMenu.*displayFileMenu*();

}**switch**(*choice*) {

**case** 1:

FileAddOperations.*addingFile*();

**break**;

**case** 2:

FileDeleteOperation.*deletingFile*();

**break**;

**case** 3:

FileSearchOperation.*searchingFile*();

**break**;

**case** 4:

DisplayMenu.*displayMenu*();

**break**;

**case** 5:

System.***out***.println("Thank you for using our application!");

**break**;

**default**:

System.***out***.println("Invalid Option. Please enter a valid option.");

}

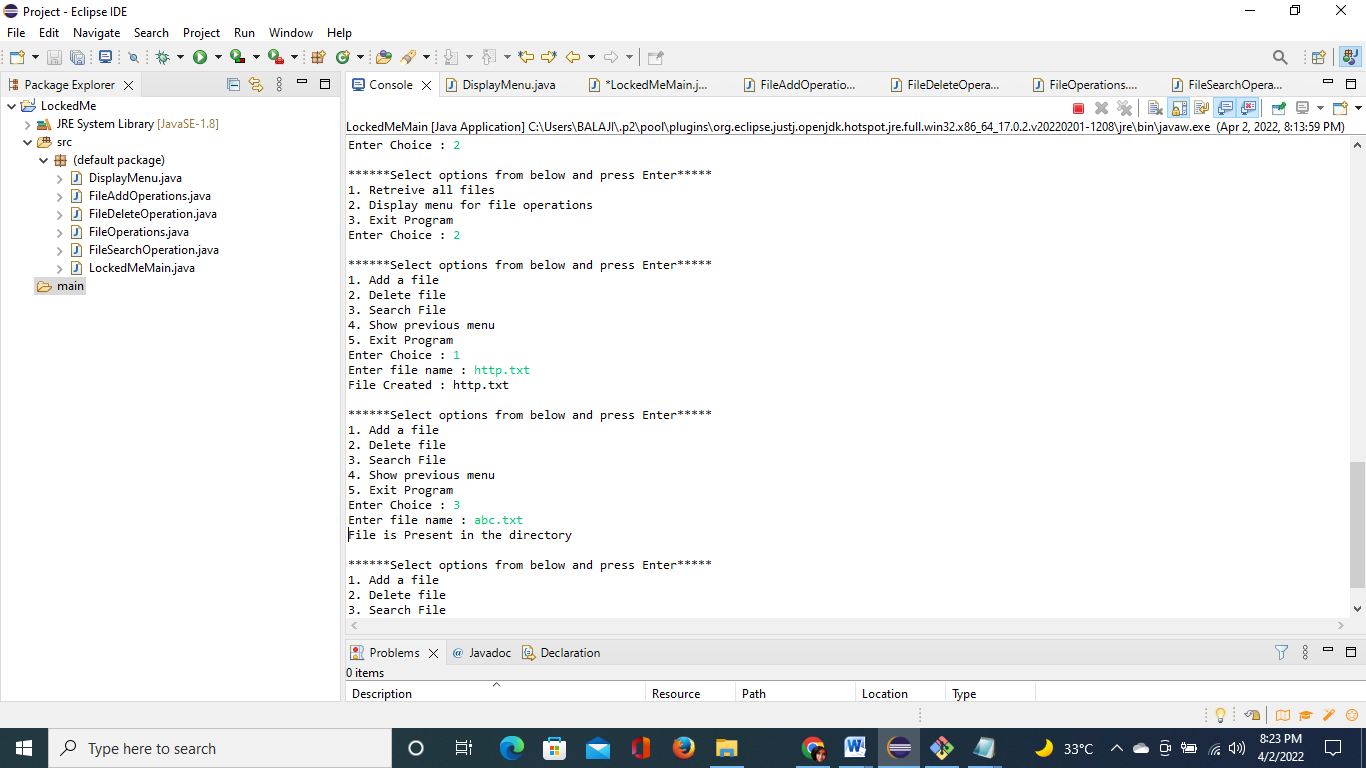
{

**while**( *choice*!=5 );

}

}

**Output:**



### **Step 4: Writing a JavaProgram 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
  2. Displaying all files in “main” folder in ascending order and also with directory structure.

**Step 4.1:** Writing method to create “main” folder in project if it’s not present

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

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

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

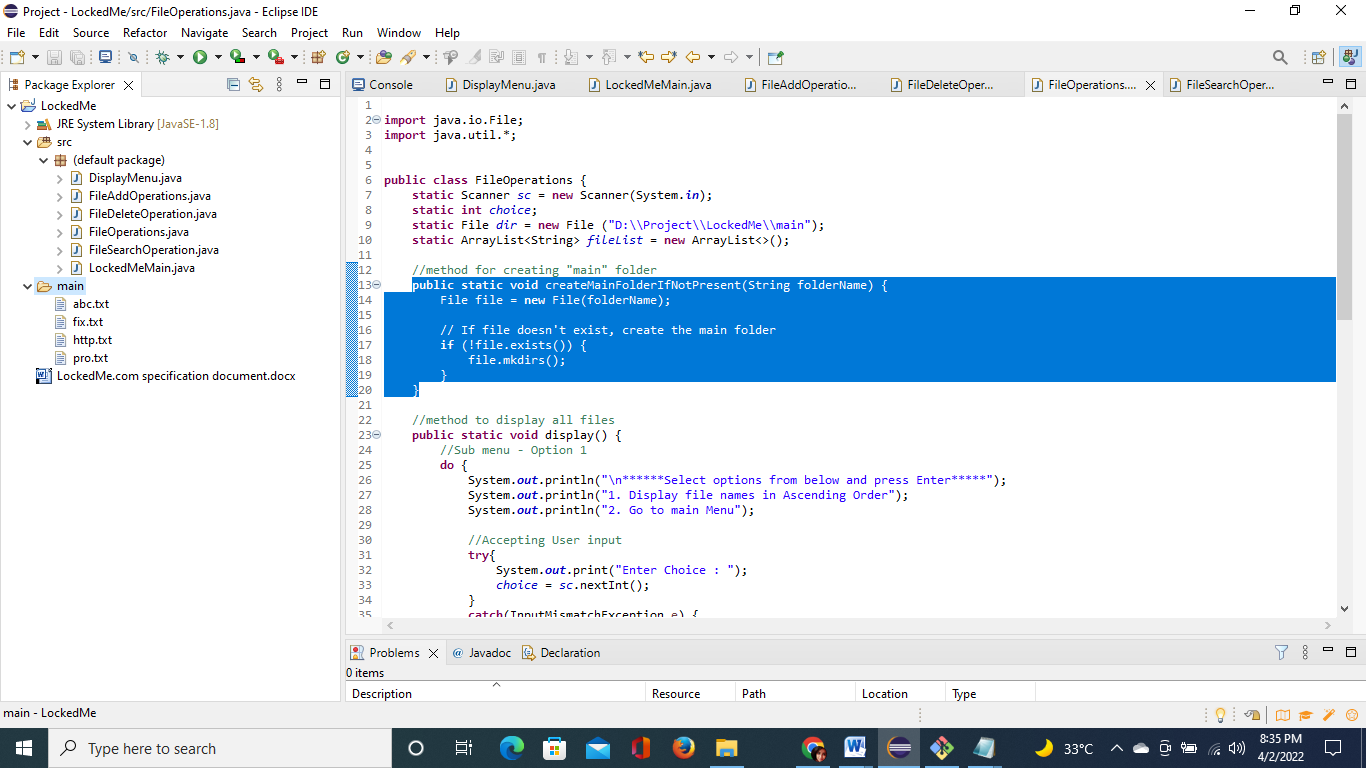
**if** (!file.exists()) {

file.mkdirs();

}

}

**Output:**



**Step 4.2:** Writing method to display all files in “main” folder in ascending order.

**public** **static** **void** display() {

//Sub menu - Option 1

**do** {

System.***out***.println("\n\*\*\*\*\*\*Select options from below and press Enter\*\*\*\*\*");

System.***out***.println("1. Display file names in Ascending Order");

System.***out***.println("2. Go to main Menu");

//Accepting User input

**try**{

System.***out***.print("Enter Choice : ");

*choice* = *sc*.nextInt();

}

**catch**(InputMismatchException e) {

System.***out***.println("Invalid Option. Please enter valid option.");

FileOperations.*display*();

}

//Performing User Functions

**switch**(*choice*) {

**case** 1:

FileOperations.*ascendingOrder*();

**break**;

**case** 2:

DisplayMenu.*displayMenu*();

**break**;

**default**:

System.***out***.println("Invalid Option. Please enter valid option.");

**break**;

}

}

**while**(*choice*!=2);

}

**public** **static** **void** ascendingOrder() {

String[] file = *dir*.list();

//Gets filenames from the directory and adds it to an arraylist, that is used to arrange the contents in ascending order

**if** (file == **null**) {

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

} **else** {

**for** (**int** i = 0; i< file.length; i++) {

String filename = file[i];

*fileList*.add(filename);

}

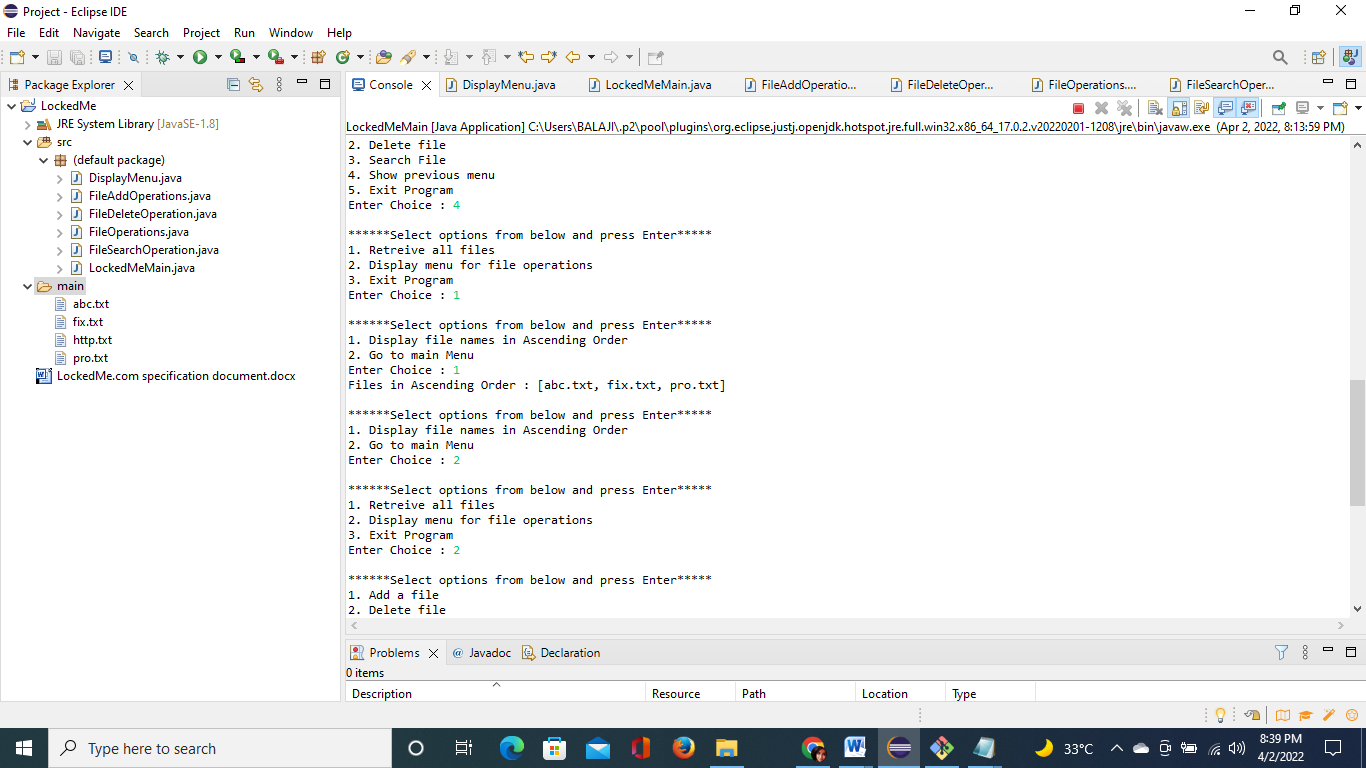
}

Collections.*sort*(*fileList*);

System.***out***.println("Files in Ascending Order : " + *fileList*);

}

**Output:**



### **Step 5: Writing a program in Java to create file as specified by user (FileAddOperations.java)**

**public** **static** **void** createFile(String fileName) {

File dir = **new** File ("D:\\Project\\LockedMe\\main");

**try** {

File file = **new** File(dir, fileName);

**if** (file.createNewFile()) {

System.***out***.println("File Created : " + file.getName());

DisplayMenu.*displayFileMenu*();

} **else** {

System.***out***.println("File already exists. Please enter another name");

FileAddOperations.*createFile*(fileName);

}

} **catch** (IOException e) {

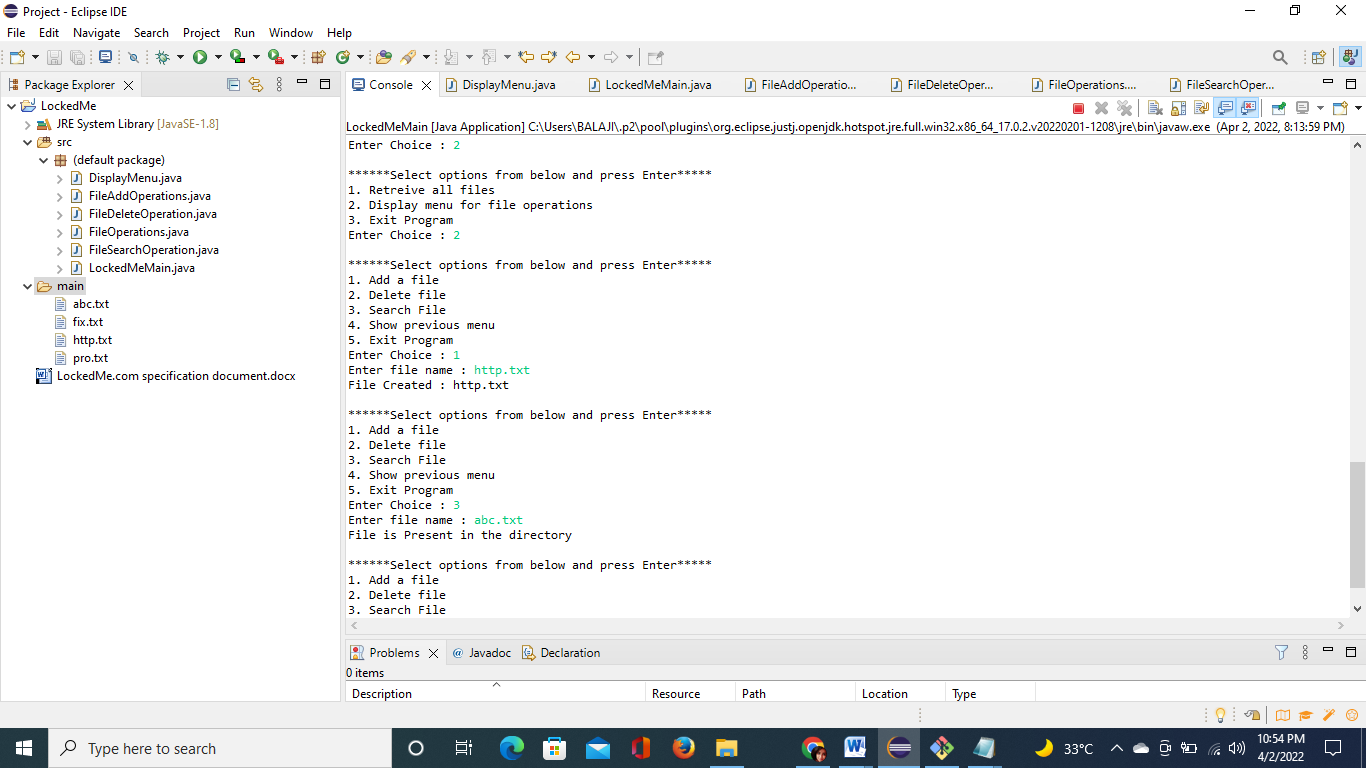
System.***out***.println("An error occurred.");

e.printStackTrace();

}

}

**Output:**



### **Step 6: Writing a program in Java to delete file as specified by user (FileDeleteOperations.java)**

**public** **class** FileSearchOperation {

**static** Scanner *sc* = **new** Scanner(System.***in***);

**static** File *dir* = **new** File ("D:\\Project\\LockedMe\\main");

**static** ArrayList<String> *fileList* = **new** ArrayList<>();

**static** String *fileName*;

//Accepts name of the file to be searched, calls function that will search the file

**public** **static** **void** searchingFile() {

//Accepting name of the file from the user

System.***out***.print("Enter file name : ");

*fileName* = *sc*.next();

FileSearchOperation.*searchList*();

FileSearchOperation.*search*(*fileName*);

}

//Gets the filenames from the directory and adds it to an arraylist, that is used to search the file

**public** **static** **void** searchList() {

String[] children = *dir*.list();

**if** (children == **null**) {

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

} **else** {

**for** (**int** i = 0; i< children.length; i++) {

String filename = children[i];

*fileList*.add(filename);

}

}

}

//Searching the arraylist to check if the file is present

**public** **static** **void** search(String fileName) {

**boolean** present = *fileList*.contains(fileName);

**if** (present) {

System.***out***.println("File is Present in the directory");

DisplayMenu.*displayFileMenu*();}

**else** {

System.***out***.println("File is not present in the directory.");

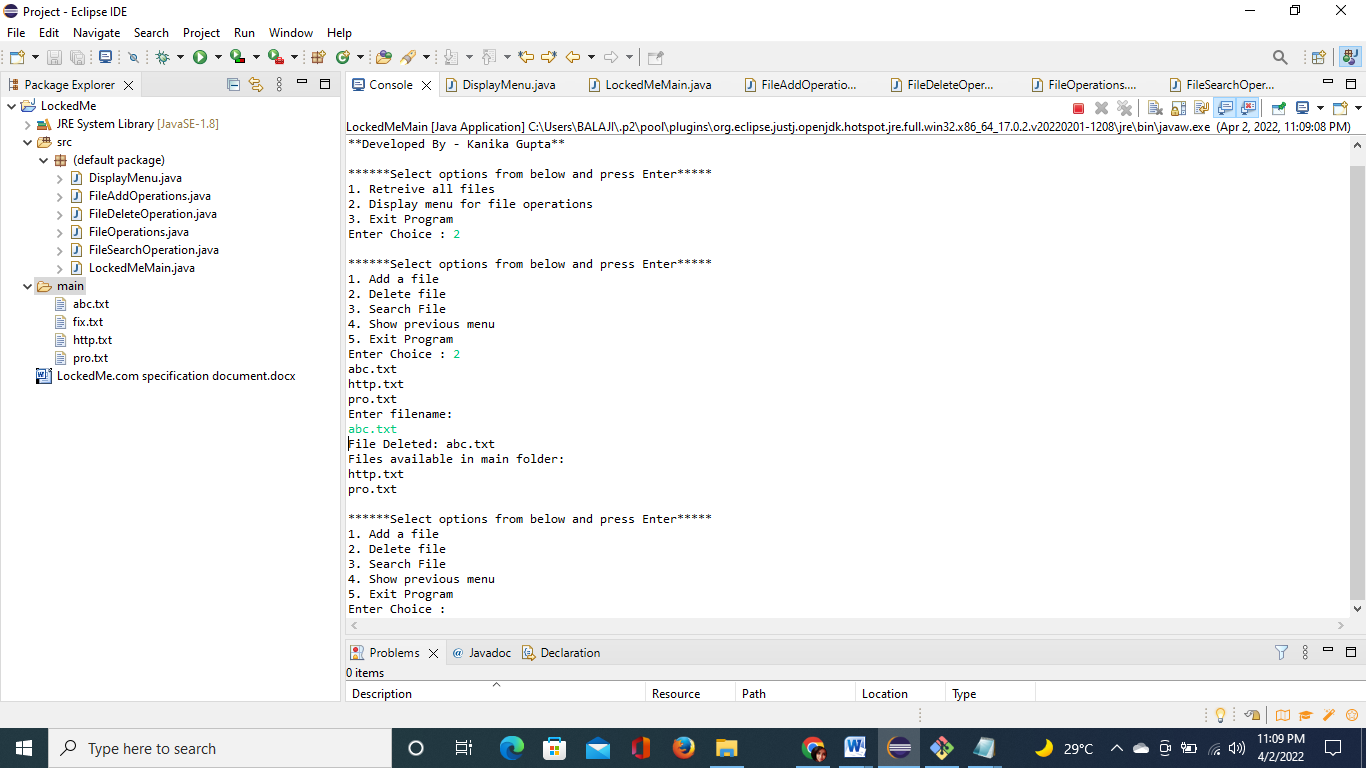
DisplayMenu.*displayFileMenu*();

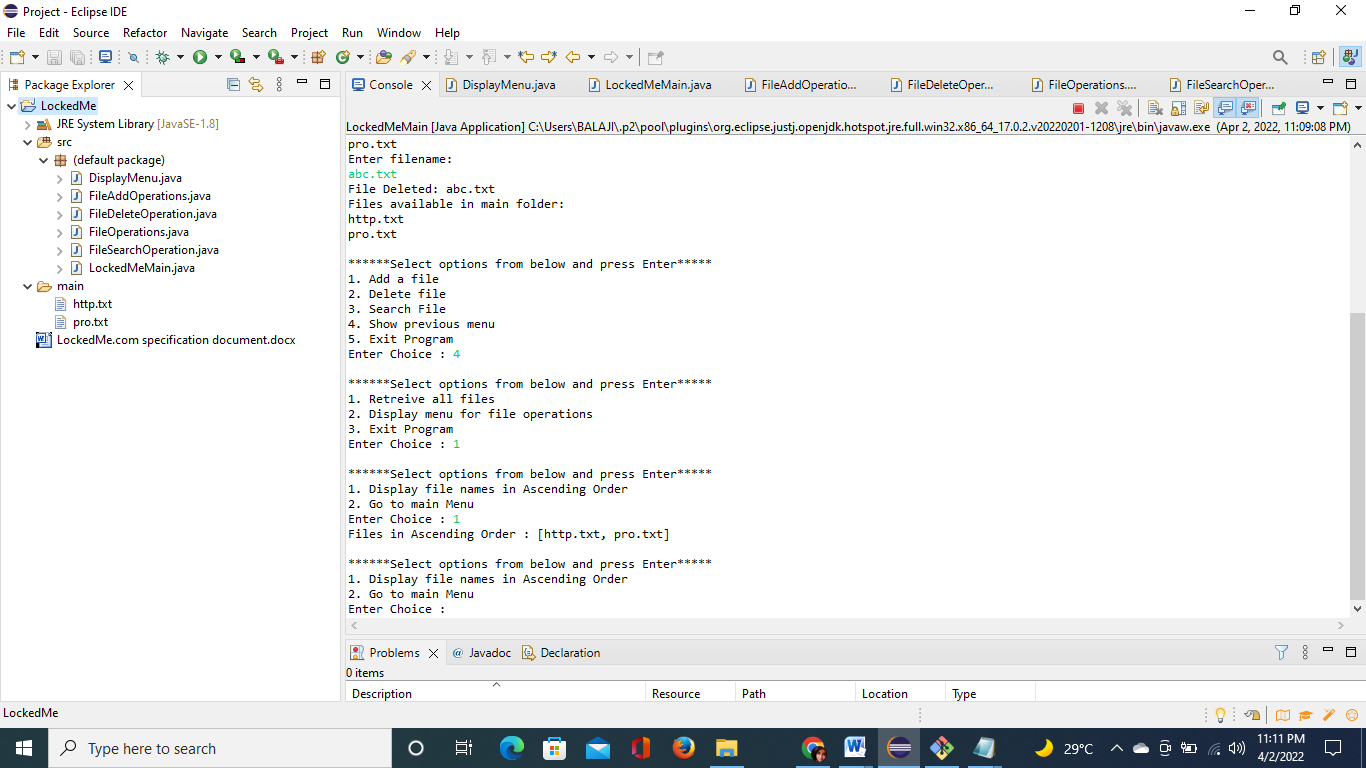
}

}

}

**Output:**





### **Step 7: Writing a program in Java to search file as specified by user in “main” folder (FileSearchOperations.java)**

**public** **class** FileDeleteOperation {

**static** Scanner *sc* = **new** Scanner(System.***in***);

**static** String *fileName*;

**static** File *dir* = **new** File ("D:\\Project\\LockedMe\\main");

**public** **static** **void** deletingFile() {

FileDeleteOperation.*displayAllFile*();

System.***out***.println("Enter filename:");

*fileName* = *sc*.next();

FileDeleteOperation.*fileDelete*(*fileName*);

}

**public** **static** **void** fileDelete(String fileName) {

File file = **new** File(*dir*, fileName);

**if** (file.delete()) {

System.***out***.println("File Deleted: " + file.getName());

FileDeleteOperation.*displayAllFile*();

DisplayMenu.*displayFileMenu*();

}**else** {

System.***out***.println("File not found");

DisplayMenu.*displayFileMenu*();

}

}

**public** **static** **void** displayAllFile() {

String[] children = *dir*.list();

**if** (children == **null**) {

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

} **else** {

**for** (**int** i = 0; i< children.length; i++) {

String filename = children[i];

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

}

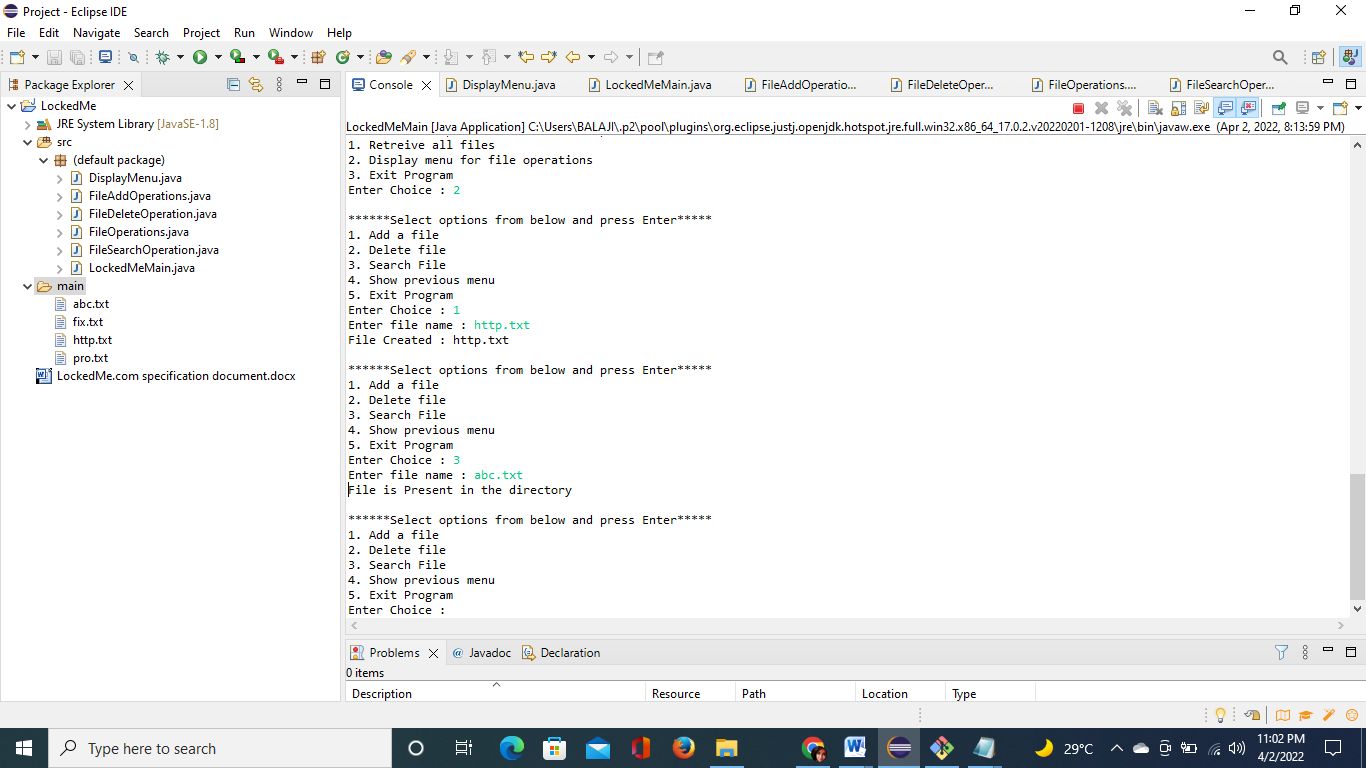
}

}

}

**Output:**

To verify if file is deleted on Eclipse, right click on Project and click “Refresh”.



### **Step 8: 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**

## **Selling Points of the Application**

1. Even if there are exceptions, the application is designed to keep running and accepting user input. The appropriate option must be selected to terminate the application.
2. The application can take any file/folder name as input.
3. Users can search files/folders by specifying the file/folder name.
4. The application also allows user to delete files/folders.
5. Even after performing required operations such as adding, searching, deleting, or retrieving files, the user can seamlessly switch between options or return to the previous menu.
6. Users can retrieve files in ascending order in form of array.
7. The application is designed to be modular. Even if you want to change the path, you can do so through the source code. The application was designed with the goal of having as little "hardcoding" of data as possible.

## **Scope of advancement in application**

Further advancements to the application can be made which may include:

* Allowing users to write content in newly created file.
* Allowing users to search files without entering complete file name.
* 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.