**Phase 1 Project Specification Flowchart Sprints Final Submission.docx**

***This file is written with open-source LibreOffice Writer.***

***February 2023***

**Table of Contents**

[Project Specification 2](#__RefHeading___Toc1793_213307066)

[Background 2](#__RefHeading___Toc1795_213307066)

[Git-hub repository link 2](#__RefHeading___Toc1797_213307066)

[Function Specification 2](#__RefHeading___Toc1799_213307066)

[Sprints Planning 3](#__RefHeading___Toc1808_213307066)

[Sprint1 3](#__RefHeading___Toc1810_213307066)

[Sprint2 3](#__RefHeading___Toc1812_213307066)

[Project code structures in Eclipse IDE 4](#__RefHeading___Toc1802_213307066)

[Java Core concepts used in the project 5](#__RefHeading___Toc1804_213307066)

[Inheritance 5](#__RefHeading___Toc1806_213307066)

[Encapsulation 5](#__RefHeading___Toc1814_213307066)

[“Do.. while” loop control for handling invalid inputs , 5](#__RefHeading___Toc1816_213307066)

[“Switch...case x” performs flow control of user choice in Menu 6](#__RefHeading___Toc1818_213307066)

[“Collection ArrayList” is used for filename processing, sorting, etc.. 6](#__RefHeading___Toc1820_213307066)

[“If {} else “ 7](#__RefHeading___Toc1822_213307066)

[Scalable code structures in class levels 7](#__RefHeading___Toc1824_213307066)

[Code re-use 8](#__RefHeading___Toc1826_213307066)

[Exception handling 9](#__RefHeading___Toc1828_213307066)

[Pre-defined class(es) are used 9](#__RefHeading___Toc1830_213307066)

[Simplicity and Informative User Interface: 9](#__RefHeading___Toc1832_213307066)

[Conclusion on enhancing the application and defining the Unique Selling Points 10](#__RefHeading___Toc1834_213307066)

[Flowchart 11](#__RefHeading___Toc1836_213307066)

# **Project Specification**

# Background

Prototype application is required for a budget approval in Company Lockers Pvt. Ltd.

The application uses the Command Line Interface (CLI) and console to interact with the end user.

Developed by Steven Feng Situ ( [stevenfeng.situ@vodafone.com](mailto:stevenfeng.situ@vodafone.com))

Java FSD Phase 1 end project

February 2023

# Git-hub repository link

**https://github.com/SFS0001/Java-FSD-training-projects/tree/main/Phase%201%20Project**

# **Function Specification**

Following functions are defined based on the requirements.

**1. Welcome Main Menu**

**1.1 List filenames in ascending order**

**1.2 Business-level operation, Sub-Menu**

**1.2.1 Add a file with user specified name** user filename input and file creation are non-case sensitive

**1.2.2 Delete a file with user specified name**

case-sensitive for input and deletion, if File-not-found, return an error message

**1.2.3 Search a file with user specified name**

exact match search, display search result, success or not-found

**1.2.4 Return to Welcome Main Menu – [1]**

**1.3 Exit and terminate the application**

# Sprints Planning

## Sprint1

Produce product function specification according to requirements

Document features and user-flow.

Works are documented in “**Specification FlowChart Sprints.docx”**

Install and Setup Git locally, GitHub version control remote repository and fully integrated between developing local/Server repository

Install and Setup Eclipse IDE

Welcome Main Menu (CLI/Console) and sub-menu of Business Operation, programming

Main flow control are established:

source codes :

**/src/com/WelcomeStartMenu.java**

**/src/com/BusinessOperation.java**

## Sprint2

- Develop codes for three business operations and return to Welcome menu, which forms the sub-menu of Business Operations Menu.

-- add a file

-- delete a file (case-sensitive)

-- search a file (case-sensitive)

-- close current context and exit to main context

-- close the application via Welcome menu

-- Demonstrate Java Core concepts and algorithms in application codes

ensuring all requirements are covered

This will be covered in the chapter of “**Core concepts used in the project”.**

-Welcome menu,

-- Application name and developer details

-- details of user interface , user interaction info.

-- accept user input to select one of the options

-- inputs out of range scenarios handing

- 1st user choice, return current file names in ascending order.

Root file can be empty or file or folder in it

- 2nd user choice,

a) add a file to existing directory list,

ignore the case sensitivity of filenames

b) Delete a file

case sensitive, exact match

Return message if FNF ( File not found)

(add-on feature: ignore existing directories , operating in files only )

c) Search a file

case sensitive on the file to retrieve the correct filenames

display the result on successful search

display not-success if that happened

Add-on features:   
 a. ignore existing directories , operating for files only

b. Display file last modification date and time, implying file creation time in this context

d) option to return back to main menu

Add-on feature, for all Operations, pause the flow for user to check the execution results,

let user to press ‘Enter’ to continue.

-3rd user choice,

close application

requirements, using Exceptions, Collections ,Sorting for sourcing code optimization and performance

Exception handling is integrated as part of the code, for example, file deletion , creation. Collection ArrayList is used for facilitate the sorting and searching of filenames.

Code is optimized by flow control and function categories, repeat method is called via inheritance for code re-use.

# Project code structures in Eclipse IDE

**Source-code files Function**

src/com/WelcomeStartMenu.java ---------- Welcome Menu

/BusinessOperation.java ---------- Business Operation Menu

/DeleteFile.java ---------- Delete a user specified file (case sensitive)

/ListFiles.java ---------- List files in ascending order

/SearchFile.java ---------- Search user specified file (case sensitive)

/AddFile.java ---------- Adding a user specified file (case in-sensitive)

FileStorage/dir-a test sub-directory   
 /dir-brc sub-directory   
 /aa1.txt test file   
 /... ....

# Java Core concepts used in the project

## Inheritance

Sub-class DeleteFile extends super-class SearchFile, to re-use the case-sensitive search method in super-class

**Code:**

*// subclass DeleteFile extends Super-class SearchFile for re-use the case-sensitive search method*

***public class DeleteFile extends SearchFile {***

## Encapsulation

**For security reason, Method deleteFile is protected, restricting calls from other package to delete a file.**

Code:

*// subclass DeleteFile extends Super-class SearchFile for re-use the case-sensitive search method*

***public******class*** *DeleteFile* ***extends*** *SearchFile {*

*//limiting access to deleteFile from other packages with protected method*

***protected void deleteFile () {***

*System.****out****.println("\nPlease enter the file name to delete: ");*

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

## **“Do.. while” loop** control for handling invalid inputs ,

***do*** *{*

*System.****out****.println("Please enter your choice [1, 2,or 3]: ");*

*userChoice = sc.nextInt();*

***if*** *( userChoice!=1 && userChoice!=2 && userChoice!=3 )*

*{ System.****out****.println("\nYou entered ["+userChoice+ "] is out of range, please enter again.\n");*

*};*

*}****while*** *(userChoice!=1 && userChoice!=2 && userChoice!=3);*

## **“Switch...case x” performs flow control of user choice in Menu**

***switch****(userChoice) {*

***case*** *1:*

*//List filenames ascending order*

*// Call public class ListFiles for Option 1,*

*ListFiles pFilenames=* ***new*** *ListFiles();*

*pFilenames.printFilename();*

*// here should return to main menu*

*pressEnterToContinue();*

***break****;*

***case*** *2:*

*//Business operations*

*//do menu 2 list 2.1, 2.2, 2.3*

*BusinessOperation operations=* ***new*** *BusinessOperation();*

*operations.operationMenu();*

***break****;*

***case*** *3:*

*// Exit application , choice 3*

*//exiting the application*

*System.****out****.println("......\nGood Bye, thanks for using!");*

*System.exit(0);*

***break****;*

## **“Collection ArrayList” is used for filename processing, sorting, etc..**

***Code*** *// define an collections ArrayList arFilename for filenames, prepare for sorting*

*ArrayList<String> arFilename=* ***new*** *ArrayList<String>();*

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

*{*

***if*** *(listFiles[i].isFile()) // exclude directory, only files allowed to be added to ArrayList*

*{*

*arFilename.add(listFiles[i].getName()); //get the filename by getName without absolute path*

*}*

*}*

*//Sorting ascending with Collections class sort*

*Collections.sort(arFilename);*

## **“If {} else “**

**condition flow control is used for dealing with “empty file directory” scenario:**

*//Printout the list with the help of Object i*

***if*** *(arFilename.size()==0) // If there is no file, print no-file information*

*{*

*System.****out****.println("\nThere is no file in this directory.");*

*}*

***else***

*{*

***for*** *(Object i : arFilename) // use Object i to printout ArrayList arFilename in vertical ascending order*

*{*

*System.****out****.println(i.toString());*

*}*

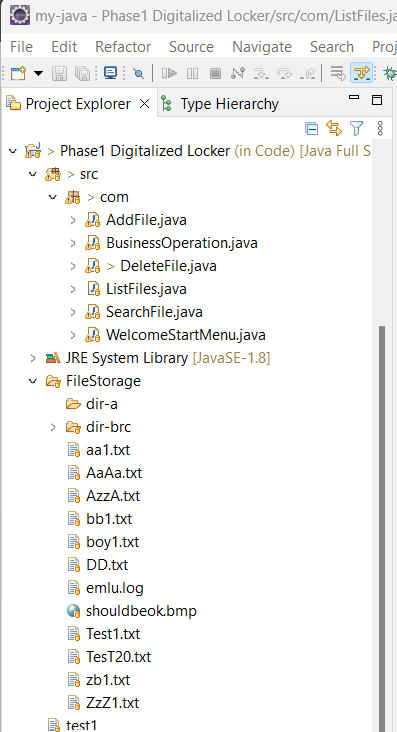
*}*

*System.****out****.println("-------------------------------------------------------------------"); //print footer*

*}*

## Scalable code structures in class levels

future add-on function can be easily expanded without change the main flow-control structure:

****

## Code re-use

**Search-File method of Search-File class is re-used for Delete-File class, which first locates the user specified file, prior to perform deletion.**

***public******boolean*** *CaseSensitiveSearch(String fileName) {*

***boolean*** *fileFound =* ***false****;*

*//filename with full path*

*String filePath="C:\\Users\\Daniel\\Documents\\Feng\\JavaFullStack\\Training\\Java Full Stack Developer Program-Steven Feng Situ\\Phase 1 Project\\Code\\FileStorage";*

*String filePathAndName="C:\\Users\\Daniel\\Documents\\Feng\\JavaFullStack\\Training\\Java Full Stack Developer Program-Steven Feng Situ\\Phase 1 Project\\Code\\FileStorage\\"+fileName;*

*File fP =* ***new*** *File(filePath);*

*File fPname =* ***new*** *File(filePathAndName);*

*//define Array listFiles storing list of file path and name*

*File[] listFiles = fP.listFiles();*

*// define an collections ArrayList arFilename for filenames, prepare for filename match test*

*ArrayList<String> arFilename1=* ***new*** *ArrayList<String>();*

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

*{* ***if*** *(listFiles[i].isFile()) // exclude directory, only files allowed to be added to ArrayList*

*{ arFilename1.add(listFiles[i].getName()); //get the filename without absolute path*

*}*

*}*

*// define and initiate file finding result flag, fFound*

***int*** *fFound =0;*

***for*** *(Object i : arFilename1) // use Object i to test ArrayList arFilename with user specified filename*

*{* ***if*** *(fileName.equals(i.toString()))*

*{*

*fileFound =* ***true****;*

*}*

*};*

***return*** *fileFound;*

*}*

**In deleteFile class,**

***/ / Inheritance from super class SearchFile for re-use the method CaseSensitiveSearch***

*// SearchFile sf = new SearchFile();*

***if*** *(CaseSensitiveSearch(fileName))*

*{*

*// delete the file if exact match true*

***try*** *{*

***if*** *(fPname.delete())*

*{ System.****out****.println("\n"+fileName+" is deleted.");*

*}* ***else***

*{ System.****out****.println("\n"+fileName+" exists, but cannot be deleted.");*

*}*

*}* ***catch*** *(Exception e )*

*{ e.printStackTrace();*

*}*

*}*

***else***

*{*

*System.****out****.println("\n"+fileName+" is not found, no deletion!");*

*};*

*}*

## Exception handling

**(try.. catch) is used to capture any exception during file deletion:**

*// delete the file if exact match true*

***try*** *{*

***if*** *(fPname.delete())*

*{ System.****out****.println("\n"+fileName+" is deleted.");*

*}* ***else***

*{ System.****out****.println("\n"+fileName+" exists, but cannot be deleted.");*

*}*

*}* ***catch*** *(Exception e )*

***{ e.printStackTrace();***

## Pre-defined class(es) are used

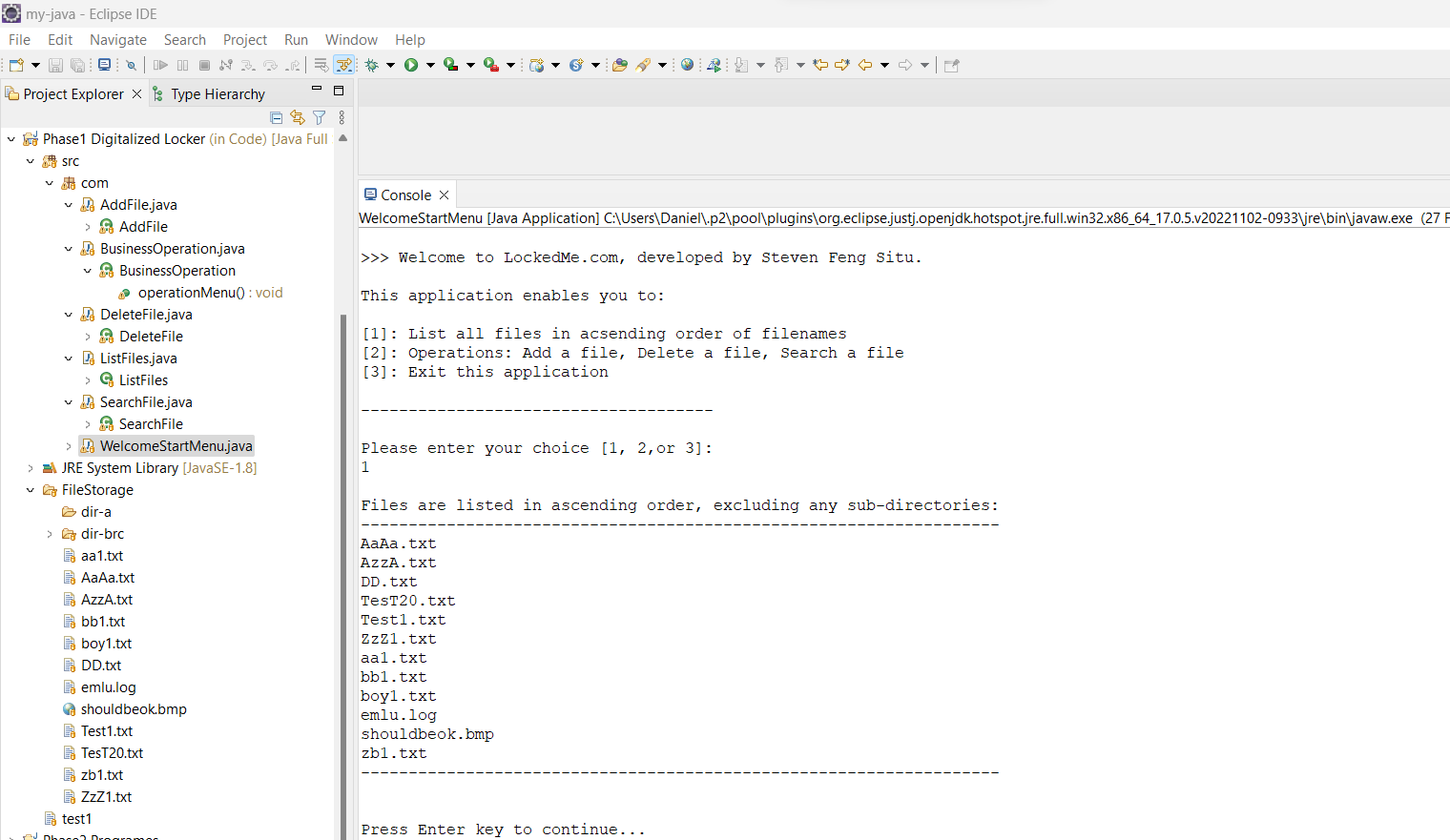
**import** java.io.File;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.text.SimpleDateFormat;

## Simplicity and Informative User Interface:

****

# Conclusion on enhancing the application and defining the Unique Selling Points

**- Simplified and informative User Interface**

**- Secured with Java native encapsulation**

**- Maximize code re-using inside**

**- Clear and scalable code structure for future function/feature addition**

**Continue in next page for Flow Chart >>>**

# Flowchart

Welcome Landing Menu

Option1, List filenames in ascending order

Option2, Business Operation...

Option3, Exit the application

Read Option

Option in [1,2,3]?

Err:retry

No

Yes

Option=1,   
 List filenames

Yes

No

List filenames ascending

Option=2,   
 BS.Operation...

Yes

No

Option=3,   
 Exit application

Exit application

BS Operations Menu

Opr1, Add file

Opr2, Delete file

Opr3, Search file

Opr4, Return to Welcome Menu

Opr=1,   
 Add file

Yes

Add file Operation

No

Opr=2,   
 Delete file

Read Op.

Delete file Op.

Yes

Opr=3,   
 Search file

No

Search file Op.

Yes

Opr=4,   
 Return to Welcome

Yes

No

No

No

Yes

No