***Virtual Key for Your Repositories***

Name: Warona Lekoba

Git Repo:

**This document comprises of:**

* [Sprint planning and Task completion](file:///C:\Users\TESLIN\Downloads\A%20SASANK%20Phase1%20-%20Virtual%20Key%20for%20Repositories.docx#Sprint_plan)
* [Core concepts used in project](file:///C:\Users\TESLIN\Downloads\A%20SASANK%20Phase1%20-%20Virtual%20Key%20for%20Repositories.docx#Core_concepts)
* [Flow of the Application](file:///C:\Users\TESLIN\Downloads\A%20SASANK%20Phase1%20-%20Virtual%20Key%20for%20Repositories.docx#Flow).
* [Demonstrating the product capabilities, appearance, and user interactions.](file:///C:\Users\TESLIN\Downloads\A%20SASANK%20Phase1%20-%20Virtual%20Key%20for%20Repositories.docx#Product_capability)
* [Conclusions](file:///C:\Users\TESLIN\Downloads\A%20SASANK%20Phase1%20-%20Virtual%20Key%20for%20Repositories.docx#Conclusions)

1. ***Sprint Planning***

This project will be completed in 2 sprints:

Sprint 1:

1. Decide Flow of the program
2. Design Classes and Methods
3. Design Exceptions
4. Create Project Structure
5. Code Data tier
6. Code Business tier interface

Sprint 2:

1. Code Business Tier Methods

a. List File Method

b. Create File Method

c. Search File Method

d. Delete File Method

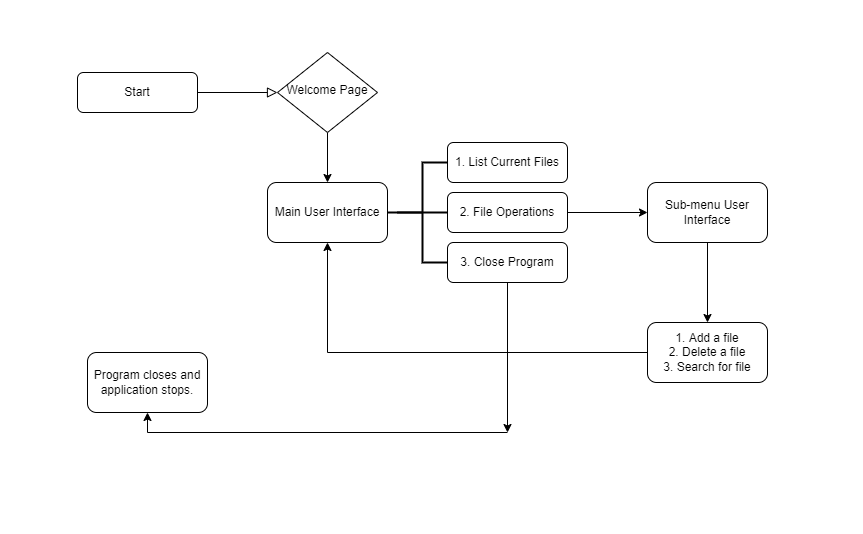
2. Code the level one menu

3. Code the level two menu

4. Call the Business Tier Methods in the Main File

5. Use String Formatting to display the output in a standard manner

1. ***Core Concepts used in Project***
2. Collections
3. Exception Handling
4. Array lists
5. **Sort Printwriter**
6. **Array FileIO operations**
7. **File handlers/functions**
8. ***Application Flow Chart***

******

https://app.diagrams.net/

1. [***Demonstrating the product capabilities, appearance, and user interactions.***](file:///C:\Users\TESLIN\Downloads\A%20SASANK%20Phase1%20-%20Virtual%20Key%20for%20Repositories.docx#Product_capability)

Source Code:

1. The source code below will contain the Welcome Screen, First Menu Options and Second Menu Options.

package com.LockedMe;

import java.util.Scanner;

import java.io.IOException;

public class LockedMeScreen {

public static void main(String[] args) throws IOException {

int ch=0, choice=0;

Scanner sc =new Scanner(System.in);

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

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

System.out.println("\t By: Lockers Pvt.Ltd ");

System.out.println("\t Developer: Warona Lekoba\n");

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

while(true)

{

System.out.println("Please select one of the following options :");

System.out.println("1. List Current Files");

System.out.println("2. File Operations");

System.out.println("3. Close Application");

try{

ch = sc.nextInt();

}

catch(Exception e)

{

System.out.println("Null Exception occurred");

}

switch(ch)

{

case 1: //List function feature to list all files in ascending order.

FileOperations.listFiles();

break;

case 2:

System.out.println("Please choose one of the following options :");

System.out.println("1. Add a File");

System.out.println("2. Delete a File");

System.out.println("3. Search for a File");

try{

choice = sc.nextInt();

}

catch(Exception e)

{

System.out.println("Null Exception occurred");

}

switch(choice)

{

case 1:

//Creation of a file takes place

System.out.println("Please input file name to be created: ");

String fileCreate = sc.next();

// Calling the function to create the file

FileOperations.createFile(fileCreate);

break;

case 2:

//deletion of the file

System.out.print("Please input file name to be deleted: ");

String fileDelete = sc.next();

// Calling the function to delete the file

FileOperations.deleteFile(fileDelete);

break;

case 3:

//Search for the file

System.out.println("Please input file name to be searched: ");

String fileSearch = sc.next();

// Calling the function to search the file

FileOperations.searchFile(fileSearch);

break;

default:

//In the case of unprecedented input execute this

System.out.println("\n Oops! Invalid Input, please repeat\n");

break;

}

break;

case 3:

//Voluntarily exiting the application

sc.close();

System.out.println("\n Thank you for using LockedMe! Until next time...");

System.exit(1);

break;

default:

//In the case of unprecedented input execute this

System.out.println("\n\n Oops! Invalid Input, Select within the range of 1-3\n");

break;

}

}

}

}

1. The source code below contains the file operations to be performed based on user selection.

package com.LockedMe;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.ArrayList;

public class FileOperations {

//sort file in ascending order

protected static String[] sort\_sub(String array[], int size){

String temp = "";

for(int i=0; i<size; i++){

for(int j=1; j<(size-i); j++){

if(array[j-1].compareToIgnoreCase(array[j])>0){

temp = array[j-1];

array[j-1]=array[j];

array[j]=temp;

}

}

}

return array;

}

//File listing function

protected static void listFiles() {

int fileCount = 0;

ArrayList<String> filenames = new ArrayList<String>();

File directoryPath = new File(System.getProperty("user.dir"));

File[] listOfFiles = directoryPath.listFiles();

fileCount = listOfFiles.length;

System.out.println("Files in ascending order: ");

for (int i = 0; i < fileCount; i++) {

if (listOfFiles[i].isFile()) {

filenames.add(listOfFiles[i].getName());

}

}

String[] str = new String[filenames.size()];

for (int i = 0; i < filenames.size(); i++) {

str[i] = filenames.get(i);

}

String[] sorted\_filenames = sort\_sub(str, str.length);

for(String currentFile: sorted\_filenames) {

System.out.println(currentFile);

}

}

//File delete function

protected static void deleteFile(String fileToBeDeleted) {

File file = new File( (System.getProperty("user.dir") ) + "\\" + fileToBeDeleted );

if(file.exists()) {

if ( file.delete() ) {

System.out.println("File deleted successfully!");

}

} else {

System.out.println("Sorry, File wasn't deleted (File Not Found)");

}

}

//File search function

protected static void searchFile(String fileToBeSearched) {

File file = new File( (System.getProperty("user.dir") ) + "\\" + fileToBeSearched );

//Check whether file whether file exists or not.

//If yes then display associated message

if(file.exists()) {

System.out.println("Yep! File found!");

} else {

System.out.println("Sorry, File not located (File Not Found)");

} PrintWriter pw;

try {

pw = new PrintWriter(fileToBeSearched); //may throw exception

pw.println("saved");

}

// providing the checked exception handler

catch (FileNotFoundException e) {

System.out.println(e);

}

}

//File creation function

protected static void createFile (String fileToBeCreated) {

File file = new File( (System.getProperty("user.dir") ) + "\\" + fileToBeCreated );

try {

if (file.createNewFile() ) {

System.out.println("File Created!");

} else {

System.out.println("File already exists:(");

}

} catch (IOException e) {

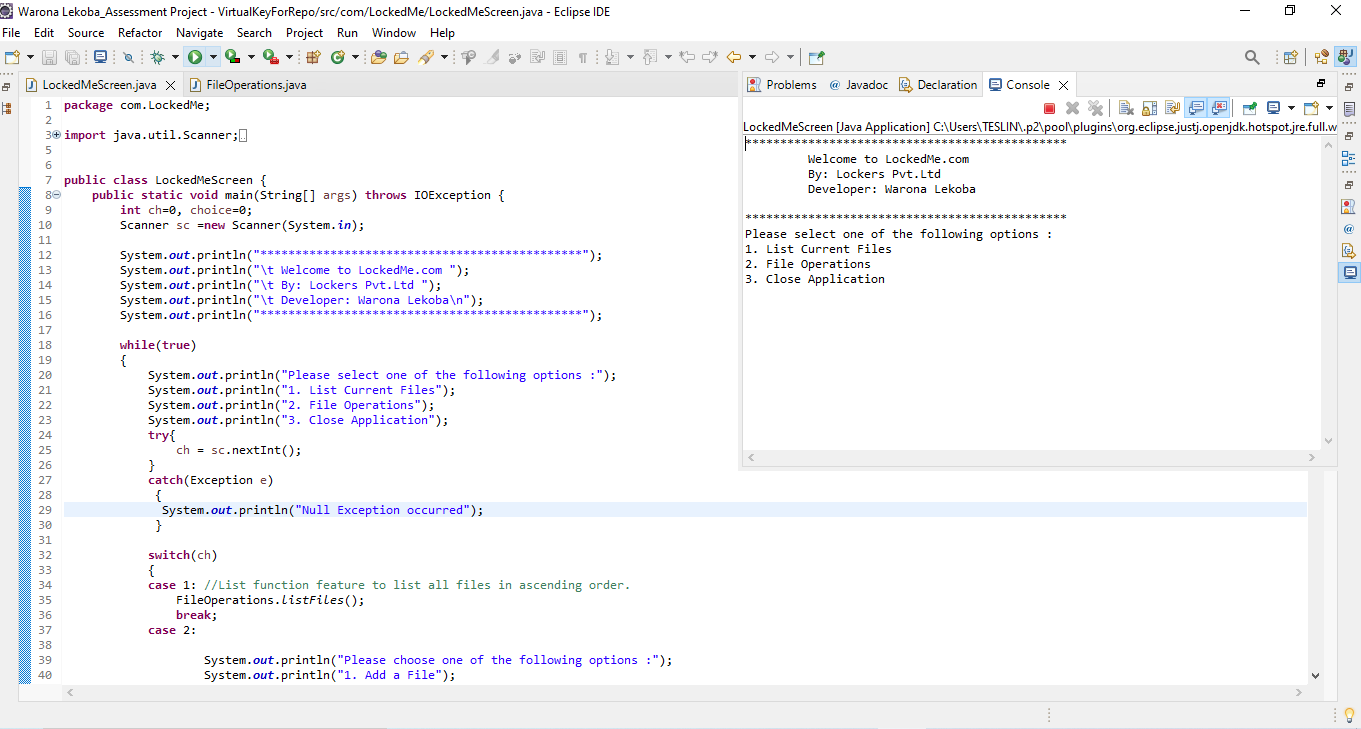
e.printStackTrace();

}

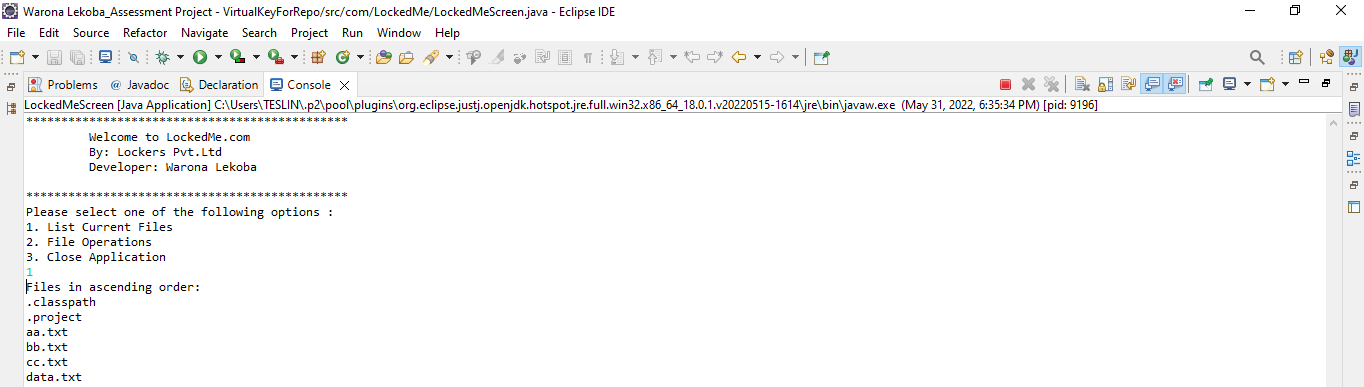
}

}

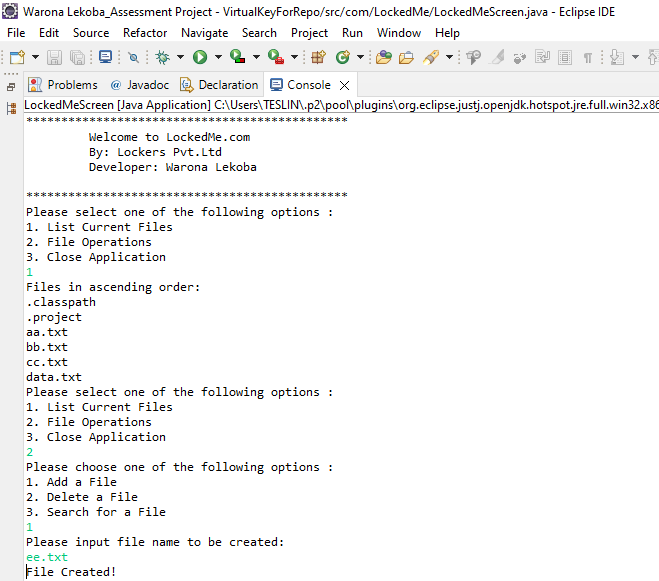
1. Screen Shots



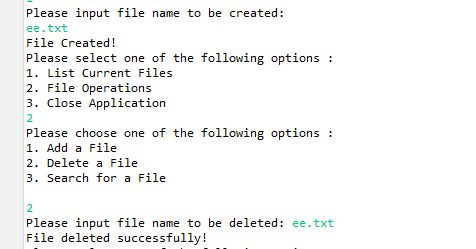
The screen shot below displays the user having selected option 1 (list current files) and the files in that folder in ascending order:



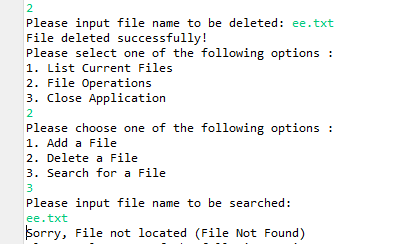
The screen shot below displays the user having selected option 2(perform file operations) and the list of file operations to be performed. The user has then chosen to create a fine “ee.txt” and the file have been created:



The screen shot below displays the user having selected to delete the file “ee.txt” from the folder:

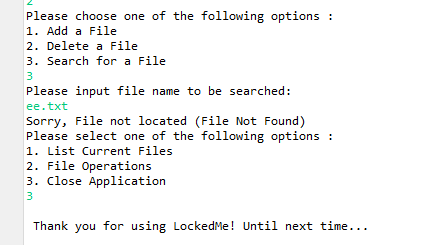


The screen shot below shows the user having selected to search if a file is in the folder or not:



The program then automatically returns to the main menu options after every sub-menu action is performed.

In the main menu options, the 3rd option is to close the program, as shown below:



1. ***Conclusion***

The file manager was created using the three-tier architecture. Special attention was paid to implement the pillars of the OOPM. Best practices were followed as and when possible. Rigorous testing was done to ensure that there are no spontaneous exits and all exceptions are handled. Some Exceptions are handled using Custom exception classes. The throws and throw keywords were used to handle exceptions.

Unique Selling Points:

● The application does not exit without giving the appropriate message

● The list of filenames is displayed in ascending order

● The file to be deleted is checked if it exists so that IOException is not raised

Enhancements:

● Providing a GUI.

● Tracking changes in files