**LockedMe.com Project Source Code**

**By Binu Thankachan**

|  |  |
| --- | --- |
| **Version # and Author** | **Version History** |
| 1.0 – Binu Thankachan | August 11, 2021 – Created document with project source code |
|  |  |
|  |  |

Contents

[Project Hierarchy 3](#_Toc79578083)

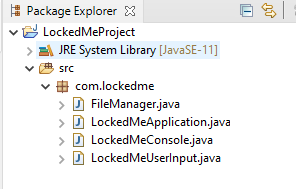
[Class LockedMeApplication.java Source Code 4](#_Toc79578084)

[Class LockedMeConsole.java Source Code 9](#_Toc79578085)

[Class LockedMeUserInput.java Source Code 12](#_Toc79578086)

[Class FileManager.java Source Code 15](#_Toc79578087)

# Project Hierarchy



# Class LockedMeApplication.java Source Code

**package** com.lockedme;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** java.util.List;

/\*\*

\*

\* The LockedMeApplication class is the main entry point for the application

\* **@author** Binu Thankachan

\*

\*

\*/

**public** **class** LockedMeApplication {

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

// instantiating LockedMeUserInput object which encapsulates user input handling

LockedMeUserInput lockedMeUserInput = **new** LockedMeUserInput();

// variable to hold user's processing selection

**char** choice = ' ';

// variable to hold the directory path object

File directoryPath = **null**;

// variable to hold the file name to process

String fileName = "";

// variable to hold the # of lines to add to a file

**int** numLinesToAddToFile = 0;

// collection variable to hold the lines to write to a file

List<String> fileLines;

// variable holding a sentinal value determining when to continue in or exit loop

**boolean** doFileProcessing = **true**;

**while**(doFileProcessing) {

// LockedMeConsole static method to display main menu for the application

LockedMeConsole.*displayMenu*();

// LockedMeConsole static method to display a prompt for the user to enter their file processing choice

LockedMeConsole.*displayEnterChoicePrompt*();

// LockedMeUserInput method to display a prompt to get the user's file processing choice from console input

choice = lockedMeUserInput.getFileProcessingChoiceFromUser();

// switch/case statement to process each file processing choice

**switch**(choice) {

// case 1 is for listing all files in a directory

**case** '1':

// call LockedMeConsole static method to prompt user for directory path

LockedMeConsole.*displayEnterDirectoryPrompt*();

// call LockedMeUserInput method to get directory path from console user input

directoryPath = lockedMeUserInput.getDirectoryPathFromUser();

// call FileManager static method to list all files in the directory

FileManager.*listAllFilesInDirectoryInASCOrder*(directoryPath);

**break**;

// case 2 is for adding a new file to a directory

**case** '2':

// call LockedMeConsole static method to prompt user for directory path

LockedMeConsole.*displayEnterDirectoryPrompt*();

// call LockedMeUserInput method to get directory path from console user input

directoryPath = lockedMeUserInput.getDirectoryPathFromUser();

// call LockedMeConsole static method to prompt user for file name to add

LockedMeConsole.*displayEnterFileNamePrompt*();

// call LockedMeUserInput method to get file name from console user input

fileName = lockedMeUserInput.getFileNameFromUser();

// call LockedMeConsole static method to prompt user for the number of lines to add to the new file that is being added

LockedMeConsole.*displayEnterNumFileContentLinesPrompt*();

// call LockedMeUserInput method to get the number of lines to add to the new file from console user input

numLinesToAddToFile = lockedMeUserInput.getNumLinesInFileFromUser();

// instantiate fileLines to an ArrayList to hold the lines to add to the file

fileLines = **new** ArrayList<>();

// if # of lines to add to file is > 0, then prompt user for each line and add it to the fileLines ArrayList

**if** (numLinesToAddToFile > 0) {

**for** (**int** i=1; i <= numLinesToAddToFile; i++) {

// call LockedMeConsole static method to prompt user for a line to add to the file

LockedMeConsole.*displayEnterFileContentPrompt*(i);

// call LockedMeUserInput method to get the line from console user input

String line = lockedMeUserInput.getLineToAddToFileFromUser();

// use collection add method to add the line to fileLines ArrayList

fileLines.add(line);

}

} **else** {

System.***out***.println("You entered an invalid # of lines. Terminating add file processing.");

**break**;

}

// call FileManager static method to add the new file with content to the directory specified by the user

FileManager.*addAFileToDirectory*(directoryPath,fileName,numLinesToAddToFile,fileLines);

**break**;

// case 3 is for searching for a file in a directory

**case** '3':

// call LockedMeConsole static method to prompt user for directory path

LockedMeConsole.*displayEnterDirectoryPrompt*();

// call LockedMeUserInput method to get directory path from console user input

directoryPath = lockedMeUserInput.getDirectoryPathFromUser();

// call LockedMeConsole static method to prompt user for file name to search for

LockedMeConsole.*displayEnterFileNamePrompt*();

// call LockedMeUserInput method to get file name from console user input

fileName = lockedMeUserInput.getFileNameFromUser();

// call FileManager static method to search for a file in a directory

FileManager.*searchForFileInDirectory*(directoryPath, fileName);

**break**;

// case 4 is for deleting a file in a directory

**case** '4':

// call LockedMeConsole static method to prompt user for directory path

LockedMeConsole.*displayEnterDirectoryPrompt*();

// call LockedMeUserInput method to get directory path from console user input

directoryPath = lockedMeUserInput.getDirectoryPathFromUser();

// call LockedMeConsole static method to prompt user for file name to search for

LockedMeConsole.*displayEnterFileNamePrompt*();

// call LockedMeUserInput method to get file name from console user input

fileName = lockedMeUserInput.getFileNameFromUser();

// call FileManager static method to delete a file from the directory specified

FileManager.*deleteFileFromDirectory*(directoryPath, fileName);

**break**;

// case 5 is for exiting file processing loop

**case** '5':

// set sentinal value for loop processing to false so that it exits the loop

doFileProcessing = **false**;

System.***out***.println("\n\t\* \* \* Exiting file processing. Goodbye. \* \* \*\n");

**break**;

**default**:

System.***out***.println("Invalid input. Try again.");

}

}

// close the scanner object

lockedMeUserInput.closeScanner();

}

}

# Class LockedMeConsole.java Source Code

**package** com.lockedme;

/\*\*

\*

\* The LockedMeConsle class encapsulates console user prompt handling for the LockedMeApplication

\* **@author** Binu Thankachan

\*

\*/

**public** **class** LockedMeConsole {

/\*\*

\* The displayMenu method displays the main menu for the LOCKEDME.COM application.

\* **@return** void

\*

\*/

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

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

System.***out***.println("\t\t\tLOCKEDME.COM by Binu Thankachan");

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

System.***out***.println("\nPlease select your file processing choice:\n");

System.***out***.println("1. List files in a directory in ascending order");

System.***out***.println("2. Add a file to a directory (case sensitive add)");

System.***out***.println("3. Search for a file in a directory (case sensitive search)");

System.***out***.println("4. Delete a file from a directory (case sensitive delete)");

System.***out***.println("5. Exit file processing");

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

}

/\*\*

\* The displayEnterChoicePrompt method displays the prompt for entering the user's menu choice.

\* **@return** void

\*/

**public** **static** **void** displayEnterChoicePrompt() {

System.***out***.println("Enter choice (1-5): ");

}

/\*\*

\* The displayEnterDirectoryPrompt method displays the prompt for entering the directory to process.

\* **@return** void

\*/

**public** **static** **void** displayEnterDirectoryPrompt() {

System.***out***.println("Enter directory file path: ");

}

/\*\*

\* The displayEnterFileNamePrompt method displays the prompt for entering the file to process.

\* **@return** void

\*/

**public** **static** **void** displayEnterFileNamePrompt() {

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

}

/\*\*

\* The displayEnterNumFileContentLinesPrompt method displays the prompt for entering the number of lines of content to add to a file.

\* **@return** void

\*/

**public** **static** **void** displayEnterNumFileContentLinesPrompt() {

System.***out***.println("Enter # of lines to add to file: ");

}

/\*\*

\* The displayEnterFileContentPrompt method displays the prompt for entering a line to add to a file.

\* **@param** lineNum (int)

\* **@return** void

\*/

**public** **static** **void** displayEnterFileContentPrompt(**int** lineNum) {

System.***out***.println("Enter line " + lineNum + ": ");

}

}

# Class LockedMeUserInput.java Source Code

**package** com.lockedme;

**import** java.io.File;

**import** java.util.Scanner;

/\*\*

\*

\* The LockedMeUserInput class encapsulates console user input handling for the LockedMeApplication

\* **@author** Binu Thankachan

\*

\*/

**public** **class** LockedMeUserInput {

**private** Scanner sc;

**private** **char** choice;

**private** File directoryPath;

**private** String fileName;

**private** **int** numLinesInFile;

LockedMeUserInput(){

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

}

/\*\*

\* The getFileProcessingChoiceFromUser method retrieves the user's selection from console input.

\* **@return** char

\*/

**public** **char** getFileProcessingChoiceFromUser() {

**this**.choice = sc.nextLine().charAt(0);

**return** **this**.choice;

}

/\*\*

\* The getDirectoryPathFromUser method retrieves the directory path to process from console input.

\* **@return** File

\*/

**public** File getDirectoryPathFromUser() {

String path = sc.nextLine();

// instantiates directoryPath File object

directoryPath = **new** File(path);

**return** **this**.directoryPath;

}

/\*\*

\* The getFileNameFromUser method retrieves the file name to process from console input.

\* **@return** String

\*/

**public** String getFileNameFromUser() {

**this**.fileName = sc.nextLine();

**return** **this**.fileName;

}

/\*\*

\* The getNumLinesInFileFromUser method retrieves the # of lines in a file from console input.

\* **@return** int

\*/

**public** **int** getNumLinesInFileFromUser() {

**if** (sc.hasNextInt()) {

**this**.numLinesInFile = sc.nextInt();

sc.nextLine();

} **else** {

// set numLinesInFile to -1 to signal invalid input

**this**.numLinesInFile = -1;

}

**return** **this**.numLinesInFile;

}

/\*\*

\* The getLineToAddToFileFromUser method retrieves the line to add to a file from console input.

\* **@return** String

\*/

**public** String getLineToAddToFileFromUser() {

String line = sc.nextLine();

**return** line;

}

/\*\*

\* The closeScanner method closes the class' scanner object if it is not null.

\* **@return** void

\*/

**public** **void** closeScanner() {

**if** (sc != **null**)

**this**.sc.close();

}

}

# Class FileManager.java Source Code

**package** com.lockedme;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.OutputStreamWriter;

**import** java.nio.file.DirectoryStream;

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

**import** java.nio.file.NoSuchFileException;

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

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

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

/\*\*

\* The FileManager class encapsulates file handling functions for the LockedMeApplication

\* **@author** Binu Thankachan

\*

\*/

**public** **class** FileManager {

**private** **static** List<String> *fileNames*;

/\*\*

\* The directoryExists method checks whether the input path exists or not.

\*

\* **@param** directoryPath (File)

\* **@return** boolean

\*/

**public** **static** **boolean** directoryExists(File directoryPath) {

// gets directory path

Path path = Paths.*get*(directoryPath.getAbsolutePath());

**boolean** exists = **true**;

// checks if directory exists

**if** (!Files.*exists*(path)) {

exists = **false**;

}

**return** exists;

}

**public** **static** **boolean** fileExists(File filePath) {

// gets file path

Path path = Paths.*get*(filePath.getAbsolutePath());

**boolean** exists = **true**;

// checks if file exists

**if** (!Files.*exists*(path)) {

exists = **false**;

}

**return** exists;

}

/\*\*

\* The listAllFilesInDirectoryInASCOrder method encapsulates the logic to list all files in a user specified directory in ascending order.

\*

\* **@param** directoryPath (File)

\* **@return** void

\*/

**public** **static** **void** listAllFilesInDirectoryInASCOrder(File directoryPath) {

*fileNames* = **new** ArrayList<>();

// check if directory exists before proceeding

**boolean** dirExists = *directoryExists*(directoryPath);

**if** (!dirExists) {

System.***out***.println("\n\n\t\t\* \* \* No such directory exists: " + directoryPath.getAbsolutePath() + " Terminating directory list processing. \* \* \*\n");

**return**;

}

// initialize a path object before passing it to Files.newDirectoryStream

Path path = Paths.*get*(directoryPath.getAbsolutePath());

// use Files.newDirectoryStream to stream the list of files in the directory

**try**(DirectoryStream<Path> stream = Files.*newDirectoryStream*(path, "\*.\*")){

// iterate through the stream and save each file to the fileNames ArrayList in lowercase format for sorting

**for** (Path entry: stream){

*fileNames*.add(entry.getFileName().toString());

}

}

// catch exceptions from most specific to least specific

**catch** (NoSuchFileException e) {

System.***out***.println("\n\n\t\t\* \* \* No such file found: " + e.getMessage() + " \* \* \*\n");

**return**;

}

**catch** (IOException e){

System.***out***.println("\n\n\t\t\* \* \* An I/O error occurred while retrieving file: " + e.getMessage() + " \* \* \*\n");

**return**;

}

**catch** (Exception e){

System.***out***.println("\n\n\t\t\* \* \* An error occurred while retrieving file: " + e.getMessage() + " \* \* \*\n");

**return**;

}

// call Collections class sort method to sort the ArrayList in ascending order

Collections.*sort*(*fileNames*);

System.***out***.println("\n\tList of files in the specified directory in case sensitive ascending order:\n");

// for loop to iterate through fileNames collection and display each file name

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

System.***out***.println("\t\t" + *fileNames*.get(i));

}

System.***out***.println("\n\n\t\t\* \* \* All files in directory successfully listed \* \* \*\n");

**return**;

}

/\*\*

\* The addAFileToDirectory method encapsulates logic to add a new file with content to a directory

\*

\* **@param** directoryPath (File)

\* **@param** fileName (String)

\* **@param** numLinesToAddToFile (int)

\* **@param** fileLines (List<String>)

\* **@return** void

\* **@throws** IOException

\*/

**public** **static** **void** addAFileToDirectory(File directoryPath,String fileName,**int** numLinesToAddToFile,List<String> fileLines) **throws** IOException {

// check if directory exists before proceeding

**boolean** dirExists = *directoryExists*(directoryPath);

**if** (!dirExists) {

System.***out***.println("\n\n\t\t\* \* \* No such directory exists: " + directoryPath.getAbsolutePath() + " Terminating add file processing. \* \* \*\n");

**return**;

}

// build path string variable with directory plus filename

String path = directoryPath.getAbsolutePath() + "/" + fileName;

// instantiate a fout File object using path

File fout = **new** File(path);

**try** {

**if** (fout.exists()) {

System.***out***.println("\n\n\t\t\* \* \* CANNOT CREATE FILE. THE FILE ALREADY EXISTS. \* \* \*\n");

**return**;

} **else** {

fout.getParentFile().mkdirs();

// create the new file in the directory

//fout.createNewFile();

fout.getCanonicalFile().createNewFile();

// create file output stream object instantiated using fout

FileOutputStream fos = **new** FileOutputStream(fout);

// create buffered writer object passing new output stream writer object instantiated using fos

BufferedWriter bw = **new** BufferedWriter(**new** OutputStreamWriter(fos));

// iterate through fileLines and write each line to the new file

**for** (**int** i = 0; i < numLinesToAddToFile; i++) {

String line = fileLines.get(i);

bw.write(line);

bw.newLine();

}

// close buffered writer object at the end

bw.close();

System.***out***.println("\n\n\t\t\* \* \* FILE CREATED SUCCESSFULLY \* \* \*\n");

**return**;

}

}

// catch exceptions in most specific to least specific hierarchy

**catch**(NoSuchFileException e)

{

System.***out***.println("\n\n\t\tNo such file exists");

}

**catch**(IOException e)

{

System.***out***.println("\n\n\t\tI/O error occurred.");

}

**catch**(Exception e)

{

System.***out***.println("\n\n\t\tAn error occurred during add file processing.");

}

**return**;

}

/\*\*

\* The searchForFileInDirectory method encapsulates logic to search for a file in a directory using case sensitivity.

\*

\* **@param** directoryPath (File)

\* **@param** fileName (String)

\* **@return** void

\*/

**public** **static** **void** searchForFileInDirectory(File directoryPath,String fileName) {

// check if directory exists before proceeding

**boolean** dirExists = *directoryExists*(directoryPath);

**if** (!dirExists) {

System.***out***.println("\n\n\t\t\* \* \* No such directory exists: " + directoryPath.getAbsolutePath() + " Terminating search file processing. \* \* \*\n");

**return**;

}

// build path string variable with directory plus filename

String path = directoryPath.getAbsolutePath() + "/" + fileName;

// instantiate a fout File object using path

File fout = **new** File(path);

**try** {

**if** (fout.exists() && fout.getCanonicalFile().getName().equals(fout.getName())) {

System.***out***.println("\n\n\t\t\* \* \* FILE FOUND \* \* \*\n");

} **else** {

System.***out***.println("\n\n\t\t\* \* \* FILE NOT FOUND \* \* \*\n");

}

}

**catch**(Exception e) {

System.***out***.println("\n\n\t\tAn error occurred while searching for the file. No file found.");

**return**;

}

**return**;

}

/\*\*

\* The deleteFileFromDirectory method encapsulates logic to delete a file from a directory.

\*

\* **@param** directoryPath (File)

\* **@param** fileName (String)

\* **@return** void

\*/

**public** **static** **void** deleteFileFromDirectory(File directoryPath,String fileName) {

// check if directory exists before proceeding

**boolean** dirExists = *directoryExists*(directoryPath);

**if** (!dirExists) {

System.***out***.println("\n\n\t\t\* \* \* No such directory exists: " + directoryPath.getAbsolutePath() + " Terminating delete file processing. \* \* \*\n");

**return**;

}

// build path string variable with directory plus filename

String path = directoryPath.getAbsolutePath() + "/" + fileName;

// instantiate a fout File object using path

File fout = **new** File(path);

**try** {

**if** (fout.exists() && fout.getCanonicalFile().getName().equals(fout.getName())) {

fout.delete();

System.***out***.println("\n\n\t\t\* \* \* FILE SUCCESSFULLY DELETED \* \* \*\n");

**return**;

} **else** {

System.***out***.println("\n\n\t\t\* \* \* FILE NOT FOUND \* \* \*\n");

**return**;

}

}

// catch exceptions in most specific to least specific hierarchy

**catch**(NoSuchFileException e)

{

System.***out***.println("\n\n\t\tNo such file exists");

}

**catch**(IOException e)

{

System.***out***.println("\n\n\t\tI/O error occurred.");

}

**catch**(Exception e)

{

System.***out***.println("\n\n\t\tAn error occurred during delete file processing.");

}

}

}