Virtual Key for Your Repositories: LockMe.com Prototype

Developer: Nicholas Auyeung

Original Repository:
https://github.com/nicholas-auyeung/Virtual-Key-for-Your-Repositories

Upon cloning the provided ‘Virtual Key for Your Repositories’ repository from github the following program is given:

The program ‘LockMePrototype’ is a prototype file handler application created according to LockMe.com’s requested features and specifications…

Features

Core Business-level operations:

- Option to add a user specified file to the application
- Option to delete a user specified file from the application
- Option to search a user specified file from the application
- Navigation option to close the current execution context and return to the main context
- Option to close the application

Additional Business-level operations:

- Option to display the existing files in current directory in ascending order
- Option to switch between existing directories
- Option to add sub level directory to the main directory of the application

Sprints

Sprint 1:

- Plan out the flow of the application(flowchart) with the required business-level operations
- Come up with and implement additional business-level operations to enhance user experience without compromising main business logic

Sprint 2:

- Plan application architecture and custom exceptions

- Plan implementation of core business-level operations(data-structures and algorithms)

Sprint 3:

- Pseudocode planned architecture and implementation

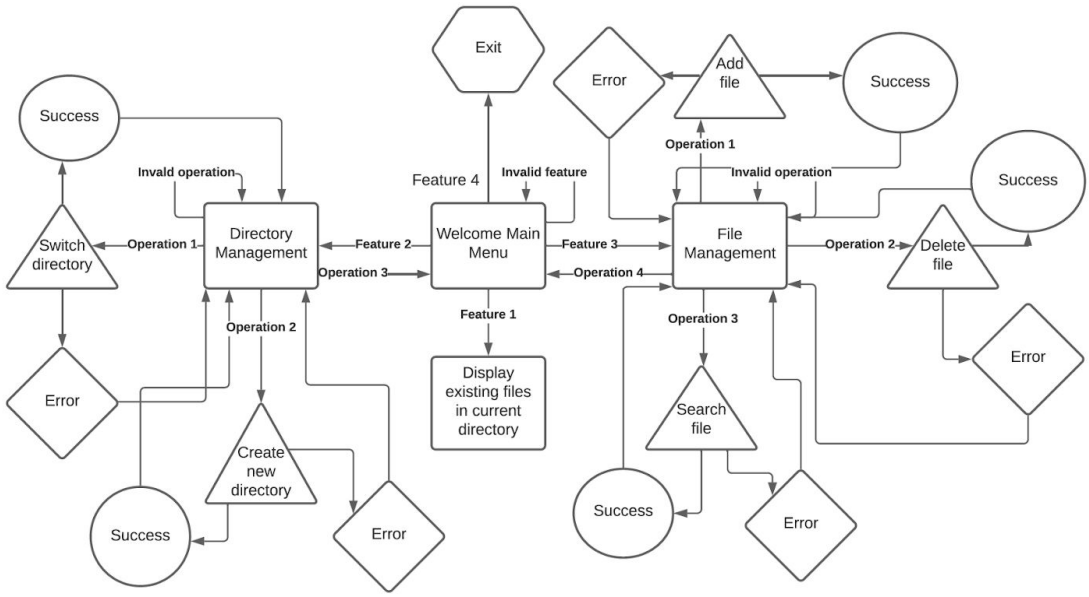
Sprint 4:

- Turn Pseudocode into working prototype of application

Sprint 5:

- Fix bugs and test working prototype
- Deploy

Flow Chart of LockMePrototype



Implementation

The application ‘LockMePrototype’ is separated into 4 different packages for ease of error correction and future expandability. These 4 packages are as follows…

com.prototype:

Contains:

- LockMePrototype.java

This package contains the main driver for the LockMePrototype application and brings every package together.

LockMeProtoype.java
The main driver runs off a while loop, prompting the user with a menu of features/operations to be executed until the user has decided to exit the application. The loop is immune to user mismatch inputs and will only end when the user decides to exit the application. Within the loop is a switch statement which executes the feature/operation that the user has inputted and throws the necessary exceptions per feature/operation.
**refer to appendix 'snippet of main loop and switch statement logic' for reference*

com.business.operation:

Contains:

- LockMeDirectoryHandler.java
- LockMeDirectoryOperations.java
- LockMeFileHandler.java
- LockMeFileHandlerOperations.java

This package contains all the business logic required for the application.
LockMeDirectoryOperations.java
LockMeFileOperations.java

These are both of the interfaces for the business operations in the application. 'LockMeDirectoryOperations.java' is the interface for the directory operations and 'LockMeFileOperations.java' for the file operations.

LockMeDirectoryOperations.java

This file contains 3 directory operations functions as required by the 'LockMeDirectoryOperations.java' interface. All 3 functions make use of the Java File Library for ease of access for directory creation, directory validation, and listing the files from a directory.

Functions

```
@Override
public void displayDirectory(String directoryName) throws NullDirectoryException{
    File dir = new File(directoryName);
    List<String> children = Arrays.asList(dir.list());
    if(children == null) {
        throw new NullDirectoryException("Directory is empty\n");
    }else {
        children.stream().sorted().forEach(System.out::println);
    }
}
```

public void displayDirectory(String directoryName) throws NullDirectoryException

This function displays the files from a directory in ascending order.

This is achieved by instantiating a File object with the passed in directory path and calling the object's File.list() method which returns a String array of the file names contained in the specified directory declared upon object creation. The String array of file names was cast into an ArrayList in order to allow for ease of sorting and data manipulation. The ArrayList is then streamed, sorted(provided by Java Stream library), and output to the console achieving the desired functionality. If the working directory is null, custom exception NullDirectoryException will be thrown.

```
@Override
public void makeDirectory(String directoryName) throws FailCreateDirectoryException {
    File f = new File(directoryName);
    if (f.mkdir()) {
        System.out.println("Directory creation successful\n");
    } else {
        throw new FailCreateDirectoryException("Directory creation failed\n");
    }

}
```

public void makeDirectory(String directoryName) throws FailCreateDirectoryException

This function creates a directory within the current application directory.

This is achieved by instantiating a File object with the passed in directory path which includes the desired name for the new directory appended to the current working directory and calls the object's File.mkdir() method which will create a directory with the appended name. If directory creation fails, custom exception FailCreateDirectoryException will be thrown.

```
@Override
public void validDirectory(String directoryName) throws IsNotDirectoryException, DirectoryNotFoundException {
    File f = new File(directoryName);
    if (f.exists()) {
        if (f.isDirectory()) {
            System.out.println("Directory exists\n");
        } else {
            throw new IsNotDirectoryException("This is not a directory\n");
        }
    } else {
        throw new DirectoryNotFoundException("Directory not found\n");
    }
}
```

public void validDirectory(String directoryName) throws IsNotDirectoryException, DirectoryNotFoundException

This function validates if the passed in directory name is a valid directory.

This is achieved by instantiating a File object with the passed in directory path which will then call the object's File.exists() method which will return true if the name matches with any file or directory, and following if true will call the File.isDirectory() method which will return true if passed in name is a directory. If the first condition, File.exists() fails, custom exception DirectoryNotFoundException is thrown and if second condition, File.isDirectory() fails, custom exception IsNotDirectoryException is thrown.

LockMeFileHandlerOperations.java

This file contains 3 file operations functions as required by the 'LockMeFileHandlerOperations.java' interface. All 3 functions make use of the Java File Library for ease of access for adding a file, deleting a file, and searching for a file in the working directory.

Functions

```
@Override
public void addFile(String fileName) throws InvalidFileNameException, FailCreateFileException {
    String filePath = pathName + "/" + fileName;
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(fileName);
    if(matcher.matches()) {
        try {
            File f = new File(filePath);
            if (f.createNewFile()) {
                System.out.println("File creation succeeded\n");
            } else {
                System.out.println("File already exists\n");
            }
        } catch (IOException e) {
            throw new FailCreateFileException("Error occured\n");
        }
    }else {
        throw new InvalidFileNameException("Invalid filename\n");
    }

}
```

public void addFile(String fileName) throws InvalidFileNameException, FailCreateFileException

This function adds a file into the working directory.

This is achieved by first validating the file name with the Java Pattern and Java Matcher libraries to create a pattern from a declared regex string and match it against the passed in file name. If the file name validation succeeds then a File object will be instantiated with the passed in file name appended to the current directory path which

will then call the object's File.createNewFile() and create the specified file, adding it to the working directory. If the file name fails to pass the regex test then custom exception InvalidFileNameException is thrown and if file fails to create, FailCreateFileException is thrown.

```
@Override
public void deleteFile(String fileName) throws InvalidFileNameException, FileDoesNotExistException {
    String filePath = pathName + "/" + fileName;
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(fileName);
    if(matcher.matches()) {
        File f = new File(filePath);
        if (f.exists()) {
            f.delete();
            System.out.println("File deleted\n");
        } else {
            throw new FileDoesNotExistException("File does not exist\n");
        }
    }else {
        throw new InvalidFileNameException("Invalid filename\n");
    }
}
```

public void deleteFile(String fileName) throws InvaldFileNameException, FileDoesNotExistException

This function deletes a file from the working directory.

This is achieved by first validating the file name with the Java Pattern and Java Matcher libraries to create a pattern from a declared regex string and match it against the passed in file name. If the file name validation succeeds then a File object will be instantiated with the passed in file name appended to the current directory path which will then call the object's File.exists() method to validate that the file exists and following if it is passed, will call the File.delete() method to delete the file. If the file name fails to pass the regex test then the custom exception InvalidFileNameException is thrown and if the file does not exist the custom exception FileDoesNotExistException will be thrown.

```
@Override
public void searchFile(String fileName) throws InvalidFileNameException, NullDirectoryException, FileDoesNotExistException{
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(fileName);
    if(matcher.matches()) {
        File f = new File(pathName);
        System.out.println(pathName);
        List<String> children = Arrays.asList(f.list());
        if (children == null) {
            throw new NullDirectoryException("Directory is empty\n");
        } else {
            if(children.contains(fileName)) {
                System.out.println("File " + fileName + " is found\n");
            }else {
                throw new FileDoesNotExistException("File is not found\n");
            }
        }
    }else {
        throw new InvalidFileNameException("Invalid filename\n");
    }
}
```

public void searchFile(String fileName) throws InvalidFileNameException, NullDirectoryException, FileDoesNotExistException

This function searches for a file from the working directory.

This is achieved by first validating the file name with the Java Pattern and Java Matcher libraries to create a pattern from a declared regex string and match it against the passed in file name. If the file name validation succeeds then a File object will be instantiated with the pathname variable passed in during instantiation of the LockMeFileHandler object created in the main driver. The File object File.list() is then called to return a String array of the files in the directory and cast into an ArrayList for ease of searching. The ArrayList.Contains method from the Java ArrayList Library is called to see if the file name exists in the ArrayList of files. If the file name fails to pass the regex test then the custom exception InvalidFileNameException is thrown, if the list of directory files is returned as null then the custom exception NullDirectoryException is thrown, and if ArrayList.Contains returns false the custom exception FileDoesNotExistException is thrown.

com.application.operations:

Contains:

- ApplicationOperations.java
- ExitApplication.java

This package contains all of the generic application logic.

ExitApplication.java

This is an interface for generic application exiting operations.

ApplicationOperations.java

This file contains 1 exit function as required by the ExitApplication interface.

Functions

```
@Override
public void exitApp() {
    System.out.println("Exiting Application...");
    System.exit(0);
}
```

public void exitApp()

This function exits the application.

This is achieved by calling the Java lang library and calling the System class containing the System.exit() method which terminates the java virtual machine.

com.exceptions:

Contains:

- DirectoryNotFoundException.java
- FailCreateDirectoryException.java
- FailCreateFileException.java
- FileDoesNotExistException.java
- InvalidFileNameException.java
- IsNotDirectoryException.java
- NullDirectoryException.java

DirectoryNotFoundException.java
FailCreateDirectoryException.java
FailCreateFileException.java
FileDoesNotExistException.java
InvalidFileNameException.java
IsNotDirectoryException.java
NullDirectoryException.java

These are all custom exception classes which inherit the Java Exception base class and simply store the exception message.

Usage

Once run, the console will welcome and prompt the user for the feature that they want to perform.

```
************************************

Welcome to LockMe.com Prototype

************************************
Developer: Nicholas Auyeung

-----------------------------------
Main Menu
-----------------------------------

Current directory: /Users/nicholasauyeung/eclipse-workspace/LockMePrototype

Enter the following feature you would like to access:
(1) Display existing files in current directory
(2) Directory management
(3) File management
(4) Exit application
```

Enter the corresponding integer beside the feature you would like to select
Feature 1 (Display existing files in current directory)
*upon entering '1' the list of files in current working directory will output

```
1
.classpath
.git
.gitignore
.project
.settings
bin
src
```

Feature 2(Directory Management)
*upon entering '2' you will be taken to the 'Directory Management' sub-menu

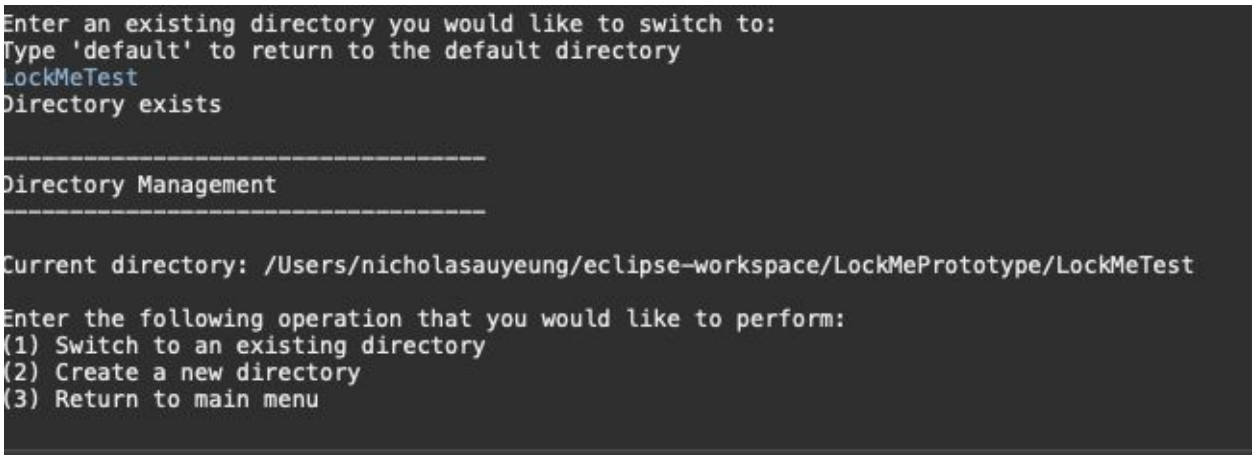
```
-----------------------------------
Directory Management
-----------------------------------

Current directory: /Users/nicholasauyeung/eclipse-workspace/LockMePrototype

Enter the following operation that you would like to perform:
(1) Switch to an existing directory
(2) Create a new directory
(3) Return to main menu
```

Enter the corresponding integer beside the operation you would like to select

Operation 1(Switch to an existing directory)
*upon entering '1' you will be prompted to enter a directory name

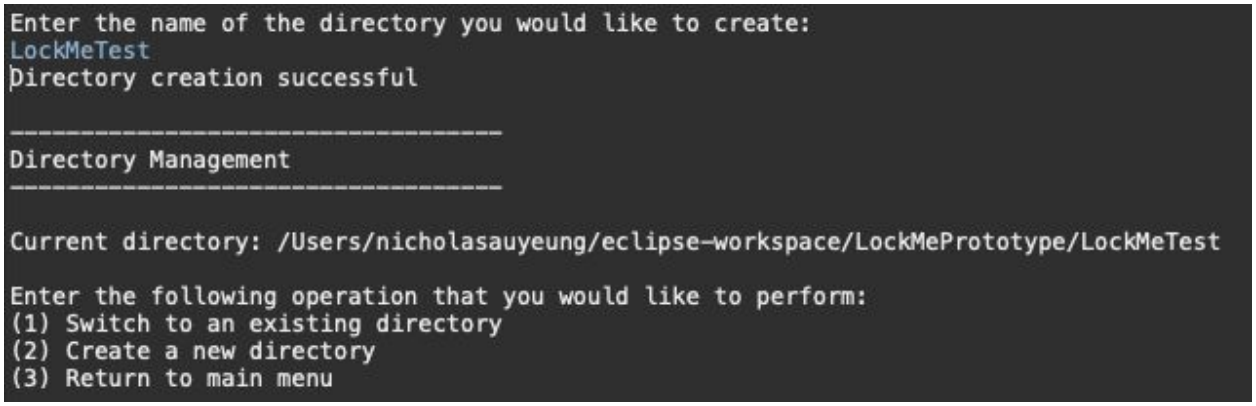
```
Enter an existing directory you would like to switch to:
Type 'default' to return to the default directory
```

If you would like to switch to the default directory enter ‘default’ otherwise enter a valid existing directory name
*if successful you will see the ‘Directory exists’ message on the console and the current directory will be switched to the stated directory, otherwise you will see an error message.



Operation 2(Create a new directory)
*upon entering ‘2’ you will be prompted to enter a directory name

Enter a valid directory name to create
*If successful you will see the ‘Directory creation successful’ message on the console and the current directory switched to the new directory, otherwise you will see an error message.



Operation 3(Return to main menu)
*upon entering ‘3’ you will be returned to the main menu and see ‘Returning to main menu…”

```
3
Returing to main menu...
```

Feature 3(File Management)
*upon entering ‘3’ you will be taken to the ‘File Management’ sub-menu

```
---------------------------------
File Management
---------------------------------
Current directory: /Users/nicholasauyeung/eclipse-workspace/LockMePrototype/LockMeTest

Enter the following operation that you would like to perform:
(1) Add a file
(2) Delete a file
(3) Search for a file
(4) Return to main menu
```

Operation 1(Add a file)

```
Enter the name of the file you would like to create:
```

Enter a valid file name to add
*If successful you will see the ‘File creation succeeded’ message on the console and the file will be added to your current working directory.

```
Enter the name of the file you would like to create:
LockMeTestFile.txt
File creation succeeded
```

Operation 2(Delete a file)

```
Enter the name of the file you would like to delete:
```

Enter a valid file name to delete
*If successful you will see the ‘File deleted’ message on the console and the file will be deleted from your current working directory.

```
Enter the name of the file you would like to delete:
LockMeTestFile.txt
File deleted
```

Operation 3(Search for a file)

```
Enter the name of the file you would like to search for:
```

Enter a valid file name to search for
*If successful you will see the ‘File file name is found’ message on the console

```
Enter the name of the file you would like to search for:
LockMeTestFile.txt
File LockMeTestFile.txt is found
```

Exiting

When finished with all operations, return to the main menu and enter ‘4’ to exit application.

```
(4) Exit application
4
Exiting Application...
```

UPSs(Unique Selling Points)

This application is a preliminary file system currently in prototype as a file handler displaying 3 core file handling operations, adding, deleting, and searching for files. LockMePrototype has robust user input validation allowing the user for error without the application crashing. This application also prints the current directory regardless of which feature the user is accessing to allow for awareness when modifying the files in the directory. LockMePrototype is a proper beginning to LockMe.com’s digitalization.

Enhancing the application

As stated above, this application is a file handler and can be expanded to have a file system architecture and functionality in the future. In the future to enhance the current state of the application, additional file handling operations such as writing and reading, advanced directory structure, file allocation, and dates handled features can be implemented.

Appendix

snippet of main loop and switch statement logic

```
while (featureValid == false) {
    try {
        System.out.println("\n-------------------------------------");
        System.out.println("Main Menu");
        System.out.println("-------------------------------------\n");
        System.out.println("Current directory: " + directoryName + "\n");
        System.out.println("Enter the following feature you would like to access: ");
        System.out.println("(1) Display existing files in current directory");
        System.out.println("(2) Directory management");
        System.out.println("(3) File management");
        System.out.println("(4) Exit application");
        operationValid = false;
        operation = scan.nextInt();
        switch (operation) {
        case 1:
            try {
                directoryHandler.displayDirectory(directoryName);
            } catch (NullDirectoryException e1) {

            }
            break;
        case 2:
            while (operationValid == false) {
                try {
                    System.out.println("-------------------------------------");
                    System.out.println("Directory Management");
                    System.out.println("-------------------------------------\n");

                    System.out.println("Current directory: " + directoryName + "\n");
                    System.out.println("Enter the following operation that you would like to perform: ");
                    System.out.println("(1) Switch to an existing directory");
                    System.out.println("(2) Create a new directory");
                    System.out.println("(3) Return to main menu");
                    operation = scan.nextInt();
                    switch (operation) {
                    case 1:
                        System.out.println("Enter an existing directory you would like to switch to: ");
                        System.out.println("Type 'default' to return to the default directory");
                        input = scan.next();
                        try {
                            if(input.compareTo("default") == 0) {
                                directoryName = defaultDirectory;
                                fileHandler.setPathName(directoryName);
                            }else {
                                directoryHandler.validDirectory(directoryName + "/" + input);
                                directoryName = defaultDirectory + "/" + input;
                                fileHandler.setPathName(directoryName);
                            }
                        } catch (IsNotDirectoryException| DirectoryNotFoundException e) {
                            System.out.println(e.getMessage());
                        }
                        break;
```