**PROJECT: VIRTUAL KEY FOR YOUR REPOSITORY**

IN ASSOCIATION WITH SIMPLILEARN

**Company Lockers Pvt. Ltd.**

**Developer: Amit Rakte**

**Project Explanation:**

This application will be helpful to retrieve the files or folders by using command line interfaces and can be easy to access those files without any complexity. It will be beneficial for all the databases which we want to add, delete or search.

**Sprint 1:**

The project is planned to be completed within 1 Sprint along with following tasks:

1. Creating the flow of the application.
2. Clarify the specification and requirements.
3. Initializing git repository to track changes as development progresses.
4. Writing the Java program to fulfill the requirements of the project.

**Sprint 2:**

The project is planned to be completed within 2 Sprint along with following tasks:

1. Checking Case sensitive properties of file handling while adding new file.
2. Checking Case sensitive properties of file handling while deleting or searching the existing file.
3. Testing the Java program with different kinds of User input Pushing code to GitHub.
4. Creating this specification document highlighting application capabilities, appearance, and user interactions

**Concepts:**

1. Exception Handling.
2. File Handling.
3. Switch case.
4. If-else, for, for-each, while Loops.
5. Has-A relationship.
6. Constructors.
7. Collection.

**Software Requirements:**

1. Eclipse SE
2. GIT
3. GITHUB
4. Scrum

**Implementation :**

1. **Directory Path:**

package Project\_VirtualKeys;

import java.util.ArrayList;

import java.util.Collections;

import java.io.File;

import java.nio.file.FileSystems;

import java.nio.file.Path;

public class Directories {

public static final String name = "C:\\Users\\Ameet\\Desktop\\Project\\Files\\";

//TODO: Possibly use a HashMap? Too complex for now.

private ArrayList<File> files = new ArrayList<File>();

Path path = FileSystems.getDefault().getPath(name).toAbsolutePath();

File Dfiles = path.toFile();

public String getName() {

return name;

}

public void print() {

System.out.println("Existing Files: ");

files.forEach(f -> System.out.println(f));

}

public ArrayList<File> fillFiles() {

File[] directoryFiles = Dfiles.listFiles();

files.clear();

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

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

files.add(directoryFiles[i]);

}

}

Collections.sort(files);

return files;

}

public ArrayList<File> getFiles() {

fillFiles();

return files;

}

}

1. **Welcome Screen(Main Menu):**

package Project\_VirtualKeys;

import java.util.\*;

import Project\_VirtualKeys.Service\_Directory;

import Project\_VirtualKeys.Service\_screen;

public class WelcomeScreen implements Screen

{

private String welcomeText = " Project : Virtual Key for Your Repositories";

private String developerText = " Developer: Amit Rakte";

private ArrayList<String> options = new ArrayList<>();

public WelcomeScreen()

{

options.add("1. Show Files");

options.add("2. Show File Options Menu");

options.add("3. Quit");

}

public void introWS()

{

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

System.out.println(welcomeText);

System.out.println(developerText);

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

Show();

}

public void Show()

{

System.out.println("Main Menu");

for (String s : options)

{

System.out.println(s);

}

}

private void Quit()

{

System.out.println("Thank you ! Have a nice day ahead..!!");

}

public void GetUserInput()

{

int selectedOption = 0;

while ((selectedOption = this.getOption()) != 4)

{

this.NavigateOption(selectedOption);

}

}

public void NavigateOption(int option)

{

switch(option)

{

case 1: // Show Files in Directory

this.ShowFiles();

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

this.Show();

System.out.println("=======================");

break;

case 2: // Show File Options menu

Service\_screen.setCurrentScreen(Service\_screen.FileOptionsScreen);

Service\_screen.getCurrentScreen().Show();

Service\_screen.getCurrentScreen().GetUserInput();

System.out.println("=======================");

this.Show();

break;

1. **exitProgram():**

case 3://Quit Program

this.Quit();

break;

default:

System.out.println("Please enter valid selection-");

break;

}

}

public void ShowFiles()

{

System.out.println("List of Files: ");

Service\_Directory.PrintFiles();

}

private int getOption()

{

Scanner in = new Scanner(System.in);

int returnOption = 0;

try

{

returnOption = in.nextInt();

}

catch (InputMismatchException ex)

{

}

return returnOption;

}

}

1. **Files Operation(Second Menu):**

package Project\_VirtualKeys;

import java.io.File;

import java.io.IOException;

import java.nio.file.FileSystems;

import java.nio.file.Path;

import java.util.ArrayList;

import java.util.InputMismatchException;

import java.util.Scanner;

import Project\_VirtualKeys.Directories;

import Project\_VirtualKeys.Service\_screen;

public class Options implements Screen {

private Directories dir = new Directories();

private ArrayList<String> options = new ArrayList<>();

public Options() {

options.add("1. Add a File");

options.add("2. Delete A File");

options.add("3. Search A File");

options.add("4. Return to Menu");

}

public void Show() {

System.out.println("File Options Menu");

for (String s : options) {

System.out.println(s);

}

}

public void GetUserInput() {

int selectedOption;

while ((selectedOption = this.getOption()) != 4) {

this.NavigateOption(selectedOption);

}

}

@Override

public void NavigateOption(int option) {

switch(option) {

case 1: // Add File

this.AddFile();

this.Show();

break;

case 2: // Delete File

this.DeleteFile();

this.Show();

break;

case 3: // Search File

this.SearchFile();

this.Show();

break;

case 4: // Return to Menu

Service\_screen.setCurrentScreen(Service\_screen.WelcomeScreen);

Service\_screen.getCurrentScreen().Show();

Service\_screen.getCurrentScreen().GetUserInput();

break;

default:

System.out.println("Please make sure that you've entered the right Option");

break;

}

}

**A. Adding the file to directory:**

public void AddFile() {

System.out.println("Please Enter the Filename:");

String fileName = this.getInputString();

System.out.println("You are adding a file named: " + fileName);

try {

Path path = FileSystems.getDefault().getPath(Directories.name + fileName).toAbsolutePath();

File file = new File(dir.getName() + fileName);

if (file.createNewFile()) {

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

dir.getFiles().add(file);

System.out.println("File Workspace :"+Directories.name+file);

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

} else {

System.out.println("This File Already Exits, no need to add another");

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

}

}catch (IOException e){

System.out.println(e);

}

}

**B. Deleting the file from directory:**

public void DeleteFile() {

System.out.println("Please Enter the Filename:");

String fileName = this.getInputString();

System.out.println("You are deleting a file named: " + fileName);

//To Delete the file:

Path path = FileSystems.getDefault().getPath(Directories.name + fileName).toAbsolutePath();

File file = path.toFile();

if (file.delete()) {

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

dir.getFiles().remove(file);

System.out.println("File Workspace :"+Directories.name+file);

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

} else {

System.out.println("Failed to delete file:" + fileName + ", file was not found.");

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

}

}

**C. Searching the file in directory:**

public void SearchFile() {

Boolean found = false;

System.out.println("Please Enter the Filename:");

String fileName = this.getInputString();

System.out.println("You are searching for a file named: " + fileName);

//TODO Fix it so ArrayList obtains files

//Finished TODO

ArrayList<File> files = dir.getFiles();

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

if(files.get(i).getName().equals(fileName)) {

System.out.println("Found " + fileName);

found = true;

System.out.println("File Workspace :"+Directories.name+fileName);

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

}

}

if (found == false) {

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

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

}

}

private String getInputString() {

Scanner in = new Scanner(System.in);

return(in.nextLine());

}

private int getOption() {

Scanner in = new Scanner(System.in);

int returnOption = 0;

try {

returnOption = in.nextInt();

}

catch (InputMismatchException ex) {

System.out.println("Invalid input");

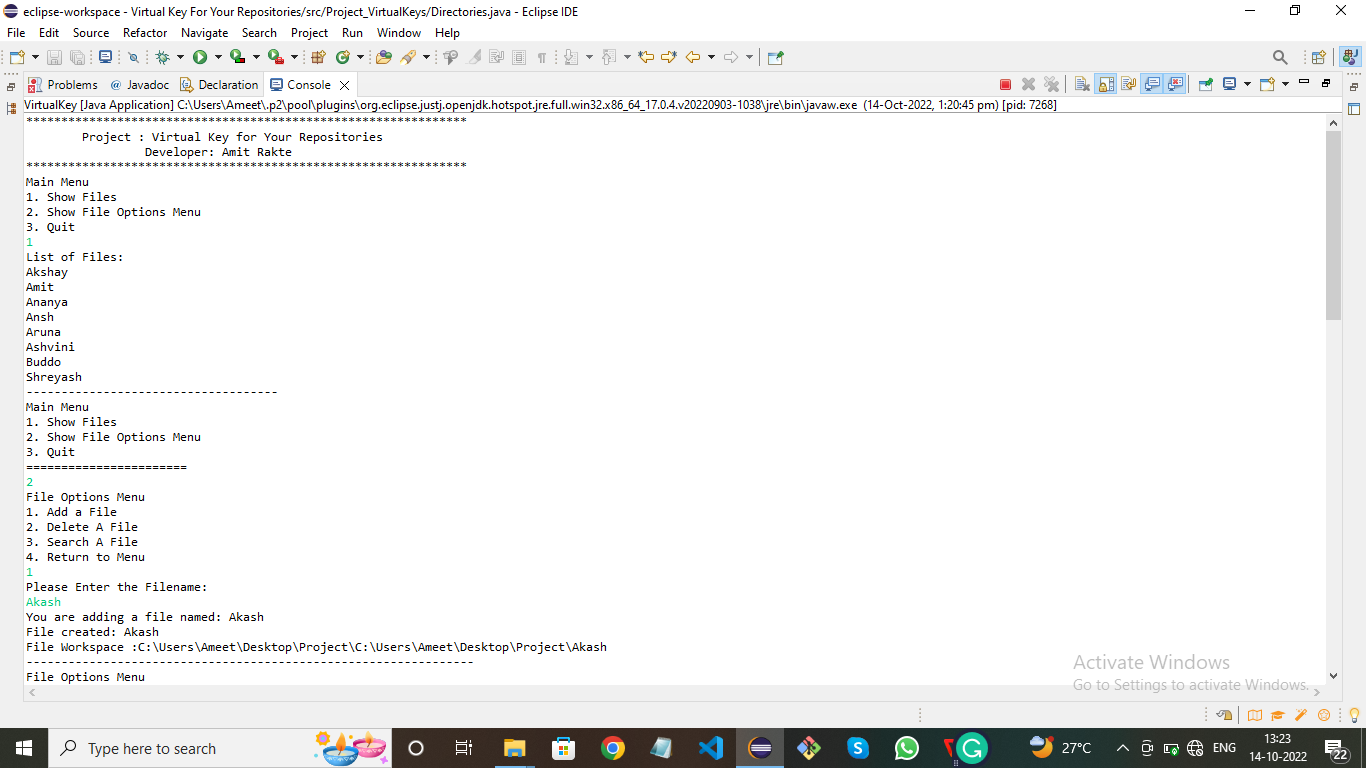
}

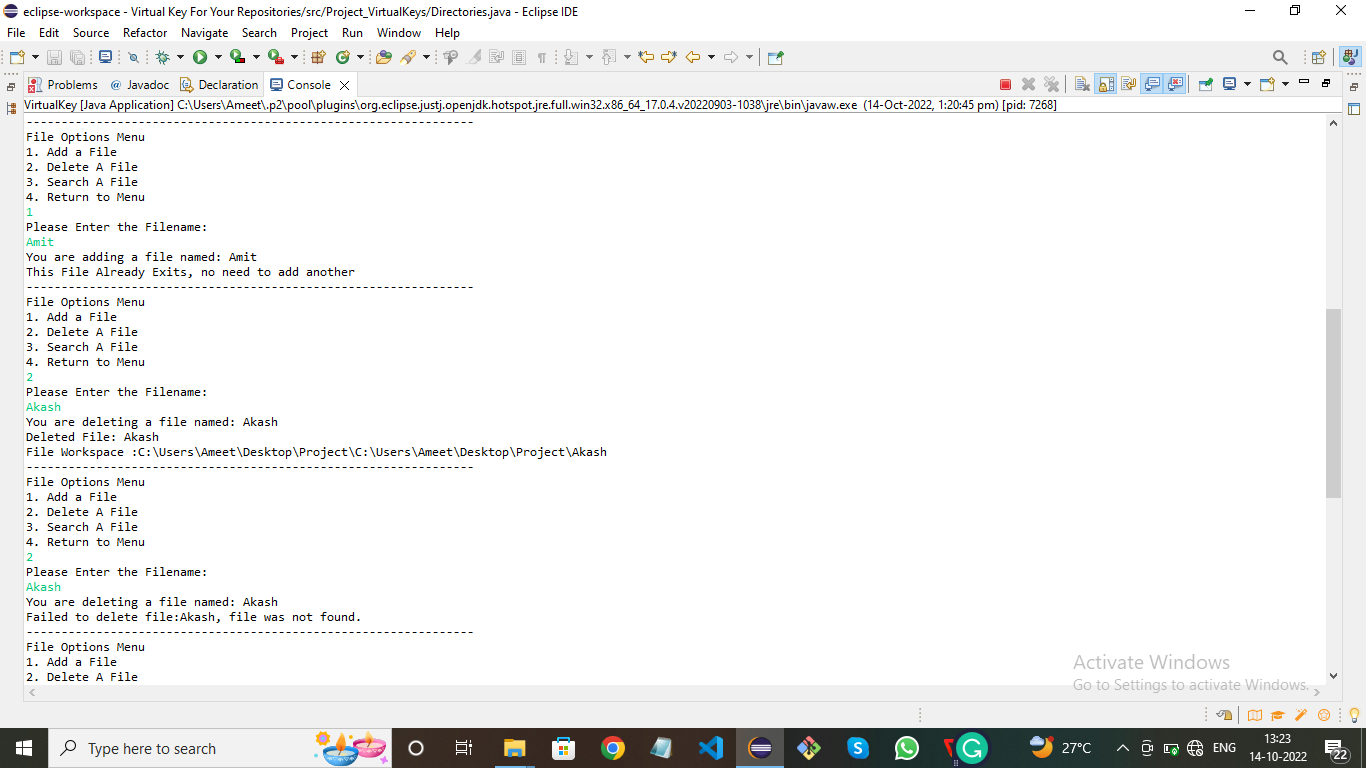
return returnOption;

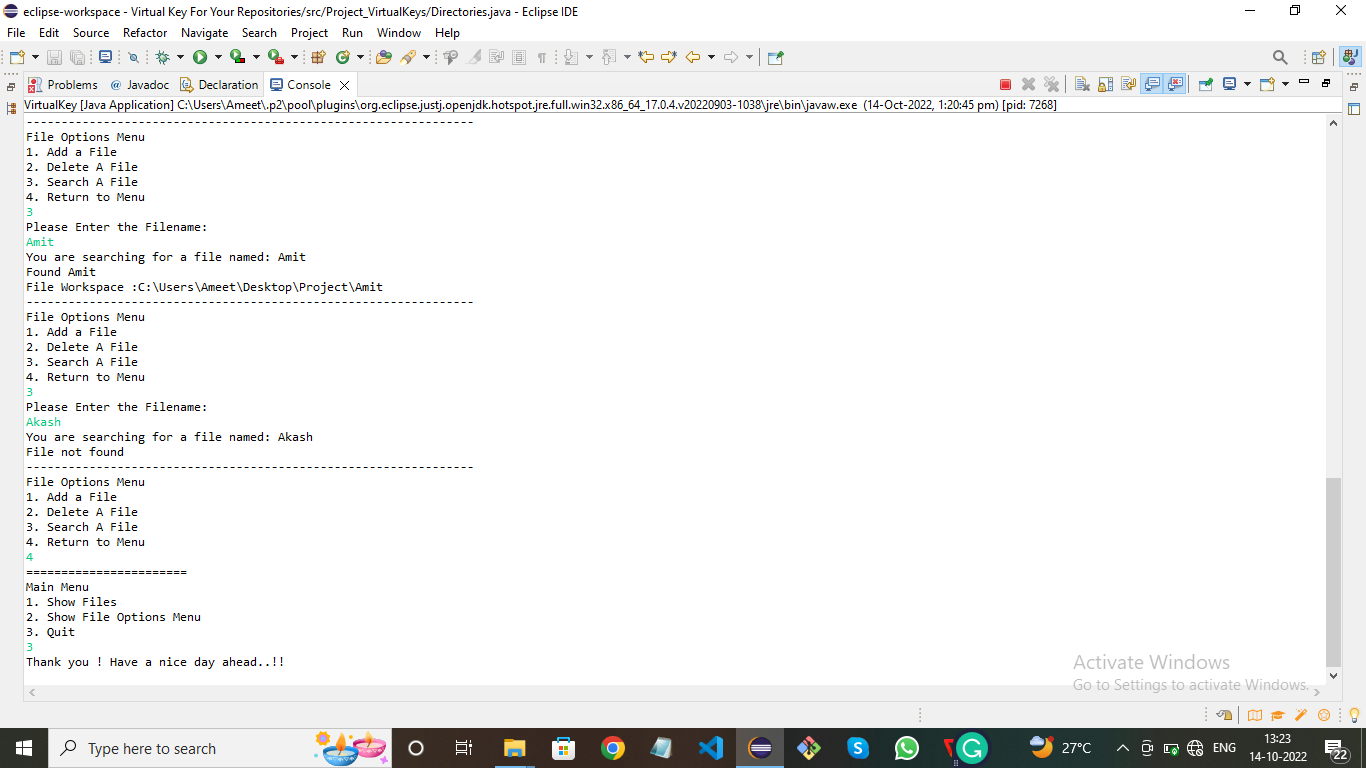
}

}

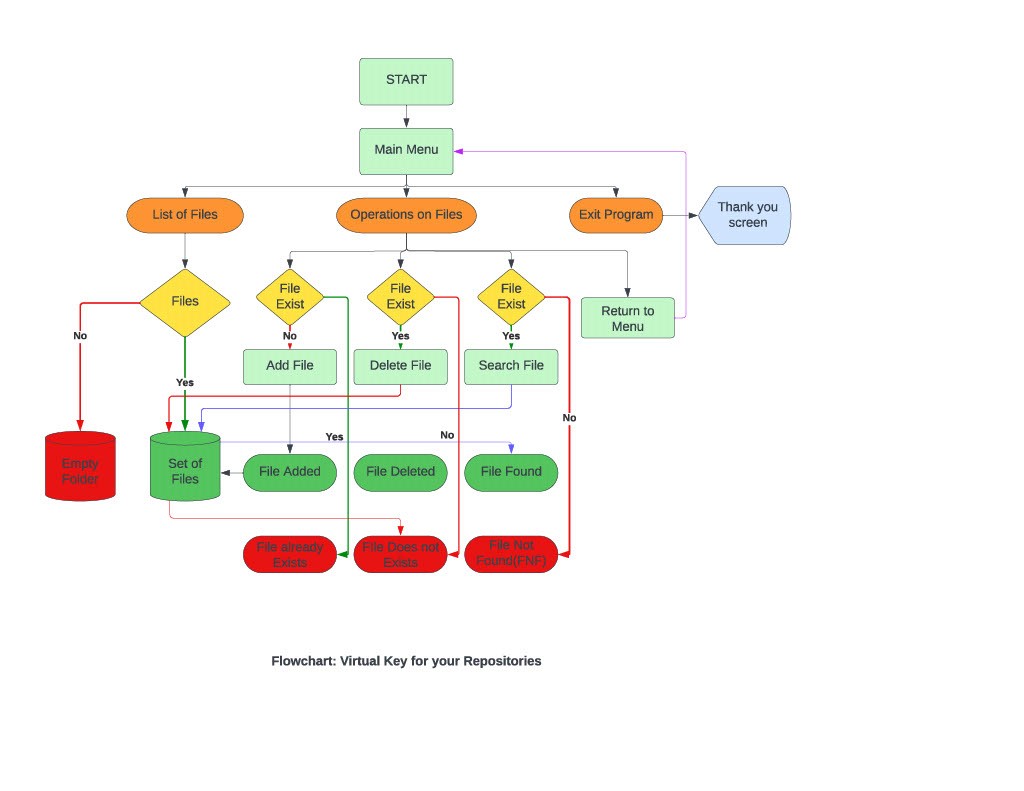
**Output Screenshots:**

****

****

****

**Flowchart:**



## **Unique Selling Points of the Application:**

1. This application can be run using command line as well as regular file creation.
2. It carefully balances what the user wants with what the files/folders does well or what it can going to operate on that files/folders.
3. We can enhance the project for giving the options for directories as comfort for the user. Also it will permit to delete the files by selection.
4. The application is designed to keep on running and taking user inputs even after exceptions occur.
5. To terminate the application, appropriate option needs to be selected.
6. The application can take any file/folder name as input. Even if the user wants to create nested folder structure, user can specify the relative path, and the application takes care of creating the required folder structure.
7. The application doesn’t restrict user to specify the exact filename to search/delete file/folder. They can specify the starting input, and the program searches all files/folder starting with the value and displays it.
8. The application also allows user to delete folders which are not empty. The user is able to switch between options or return to previous menu even after any required operation like adding, searching, deleting or retrieving of files is performed.
9. When the option to retrieve files in ascending order is selected, user is displayed with two options of viewing the files. Ascending order of folders first which have files sorted in them,Ascending order of all files and folders inside the “main” folder.The application is designed with modular in mind. Even if one wants to update the path, they can change it through the source code. Application has been developed keeping in mind that there should be very less “hard-coding” of data.

**Conclusion:**

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

1. Easy to check if user is allowed to Add the file, Delete the file and Search the file which are present in the workspace or not.
2. Allowing user to append data to the file. If the file is already there then the process will be stopped and get to the conclusion.
3. Asking user to verify if they really want to delete the selected directory if it’s not empty.
4. Retrieving files/folders by different criteria like Last Modified, Type, etc.
5. File handling and operations concepts are mostly used in overall the project. So it was lot helpful to experience the file handling concept practically.