LockedMe – Virtual Key for Repositories

This document contains the following

- Project and developer details
- Sprint planning and tasks completion
- Algorithms and Flowchart of the application
- Core concepts used in the project
- Links to the GitHub repository
- Demonstration of product capabilities, appearance, and user interactions
- Unique Selling Points of the application
- Conclusion

Project and Developer details:

Project objective:

As a Full Stack Developer, complete the features of the application by planning the development in terms of sprints and then push the source code to the GitHub repository. As this is a prototyped application, the user interaction will be via a command line.

Background of the problem statement:

Company Lockers Pvt. Ltd. hired you as a Full Stack Developer. They aim to digitize their products and chose LockedMe.com as their first project to start with. You're asked to develop a prototype of the application. The application prototype will then be presented to the relevant stakeholders for budget approval. Your manager has set up a meeting where you're asked to present the following in the next 15 working days (3 weeks):

- Specification document - Product's capabilities, appearance, and user interactions
- Number and duration of sprints required
- Setting up Git and GitHub account to store and track your enhancements of the prototype
- Java concepts being used in the project
- Data Structures where sorting and searching techniques are used.
- Generic features and three operations:
    - Retrieving the file names in an ascending order
    - 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

The flow and features of the application:

- Plan more than two sprints to complete the application
- Document the flow of the application and prepare a flow chart
- List the core concepts and algorithms being used to complete this application
- Code to display the welcome screen. It should display:
    - Application name and the developer details
    - The details of the user interface such as options displaying the user interaction information
    - Features to accept the user input to select one of the options listed
- The first option should return the current file names in ascending order. The root directory can be either empty or contain few files or folders in it
- The second option should return the details of the user interface such as options displaying the following:
    - Add a file to the existing directory list
        - You can ignore the case sensitivity of the file names
    - Delete a user-specified file from the existing directory list
        - You can add the case sensitivity on the file name in order to ensure that the right file is deleted from the directory list
        - Return a message if FNF (File not found)
    - Search a user-specified file from the main directory
        - You can add the case sensitivity on the file name to retrieve the correct file
        - Display the result upon the successful operation
        - Display the result upon unsuccessful operation
    - Option to navigate back to the main context
- There should be a third option to close the application
- Implement the appropriate concepts such as exceptions, collections, and sorting techniques for source code optimization and increased performance

You must use the following:

- Eclipse/IntelliJ: An IDE to code for the application
- Java: A programming language to develop the prototype
- Git: To connect and push files from the local system to GitHub
- GitHub: To store the application code and track its versions
- Scrum: An efficient agile framework to deliver the product incrementally
- Search and Sort techniques: Data structures used for the project
- Specification document: Any open-source document or Google Docs

The following requirements should be met:

- The source code should be pushed to your GitHub repository. You need to document the steps and write the algorithms in them.
- The submission of your GitHub repository link is mandatory. In order to track your task, you need to share the link to the repository. You can add a section to your document.
- Document the step-by-step process starting from sprint planning to the product release.
- Application should not close, exit, or throw an exception if the user specifies an invalid input.
- You need to submit the final specification document which includes:
    - Project and developer details
    - Sprints planned and the tasks achieved in them
    - Algorithms and flowcharts of the application
    - Core concepts used in the project
    - Links to the GitHub repository to verify the project completion
    - Your conclusion on enhancing the application and defining the USPs (Unique Selling Points)

Developer Details:

Mahendra Kumar Singh

Mahendrakumarsingh9893@gmail.com

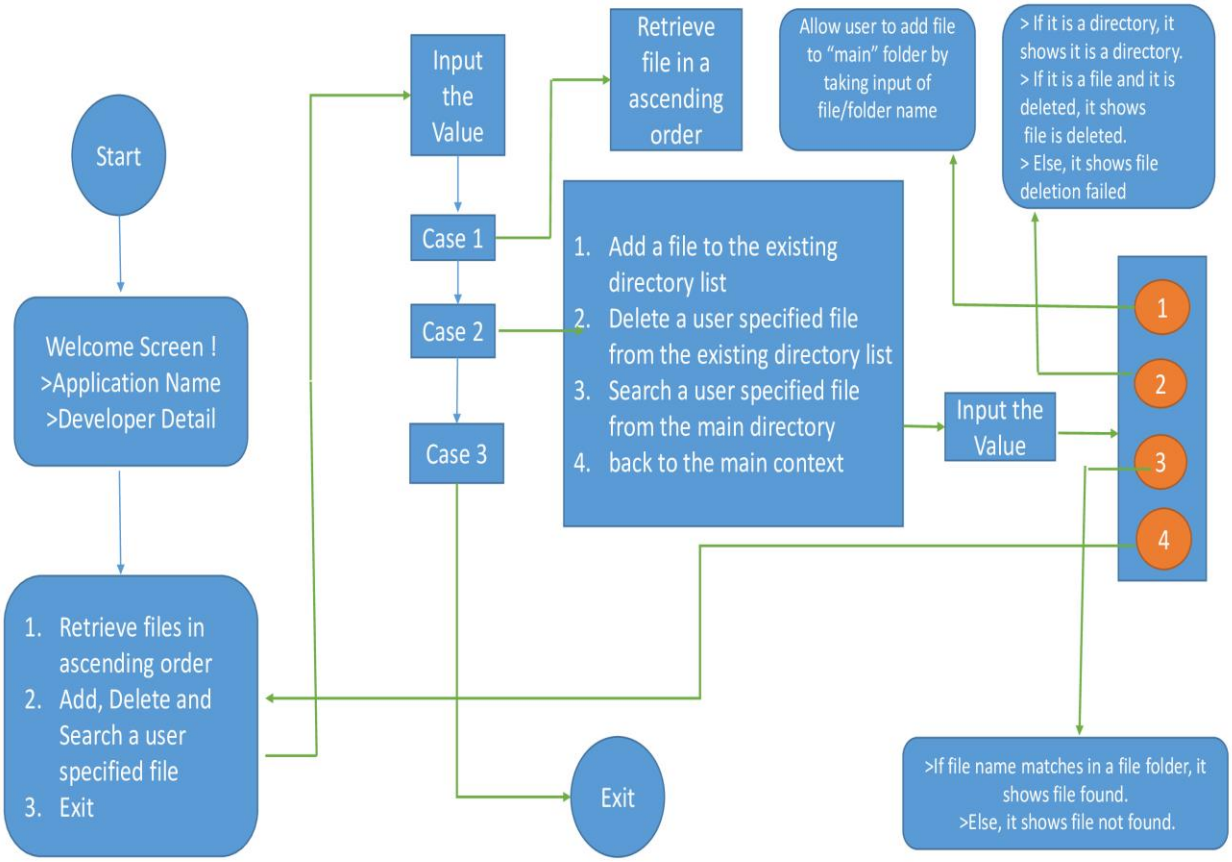
Spring planning and Task completion

Tasks assumed to be completed in the sprint are:

- Creating the flow of the application
- Initializing git repository to track changes as development progresses.
- Writing the Java program to fulfill the requirements of the project.
- Testing the Java program with different kinds of User input
- Pushing code to GitHub.
- Creating this specification document highlighting application capabilities, appearance, and user interactions.

Algorithms and Flowchart of the application:

Flowchart:



Flowchat.pdf

Algorithms:

1. Retrieving files in ascending order:

1. START
2. Prompt path, file and assign all the files in the folder to the files list.
3. Sort the list of files.
4. If(files.length)>0 then print all the files using for each loop.
5. Else print that the root directory is empty.
6. END

2. Adding a user-specified file to the application:

1. START
2. Prompt for the name of the file to be added with the path.
3. Prompt for content to write onto the file using File output stream.
4. Convert content to bytes.
5. Write the content onto the file and close using fos.write(); and fos.close();.
6. Print file is added to the directory.
7. END

3. Deleting a user-specified file from the application:

1. START
2. Prompt for the name of the file to be deleted with the path.
3. If(file.isDirectory()), then print it as a directory.
4. Else if(file.delete()), then print file is deleted successfully.
5. Else print file deletion failed.
6. END

4. Searching a user-specified file from the application

1. START
2. Prompt for directory path and file name.
3. Assign all the files in that directory path to the files list.
4. Declare flag = false.
5. Compare the user-specified file with all the files in the directory using for loop. If matched, flag = true.
6. If flag = true, print file is found.
7. Else print file is not found.

Core concepts used in the project:

- File Handling
- Exception Handling
- Encapsulation
- Packages

Links to the GitHub repository:

https://github.com/Mahendra1272/simplilearn2023

Demonstration of product capabilities, appearance, and user interactions:

Step 1: Creating a new project in Eclipse

1. Creating a project in Eclipse:
   - Open Eclipse IDE.
   - Go to File → New → Project → Java Project → Next.
   - Give the project name as 'MyProject', uncheck 'Create module-info.java file' and click Finish.
   - Right-click on the 'MyProject' project in project explorer → New → Class.
   - Create another class as 'Main' and check 'public static void main(String[] args)' and click on Finish

Step 2: Writing a program in Java for the entry point of the application (**Main.java**)

```
package mypackage;
```

```
public class Main {

	public static void main(String[] args) {


		FileOperations.createMainFolderIfNotPresent("main");

		MenuOptions.printWelcomeScreen
("Java Project
" , "Mahendra Kumar Singh");

		HandleOptions.handleWelcomeScreenInput();
	}


}
```

Step 3: Writing a program in Java to display Menu options available for the user (**MenuOptions.java**)

- Select your project and go to File -> New -> Class.
- Enter **MenuOptions** in class name and click on “Finish.”
- **MenuOptions** consists methods for -:

3.1. Displaying Welcome Screen

3.2. Displaying Initial Menu

Displaying Secondary Menu for File Operations available

Step 3.1: Writing method to display Welcome Screen

```
public class MenuOptions {

	public static void printWelcomeScreen(String appName, String developerName) {
		String companyDetails =
String.format("*************************************************\n"
				+ "** Welcome to %s \n" + "** This application was
developed by %s.\n"
				+
"*************************************************\n", appName, developerName);
		String appFunction = "You can use this application to :-\n"
				+ "• Retrieve all file names in the \"main\" folder\n"
				+ "• Search, add, or delete files in \"main\" folder.\n"
				+ "\n**Please be careful to ensure the correct filename is
provided for searching or deleting files.**\n";
		System.out.println(companyDetails);
```

```
            System.out.println(appFunction);
      }
```

Output:

Step 3.2: Writing method to display Initial Menu

```
public static void displayMenu() {
            String menu = "\n\n****** Select any option number from below and press
Enter ******\n\n"
                        + "1) Retrieve all files inside \"main\" folder\n" + "2)
Display menu for File operations\n"
                        + "3) Exit program\n";
            System.out.println(menu);

      }
```

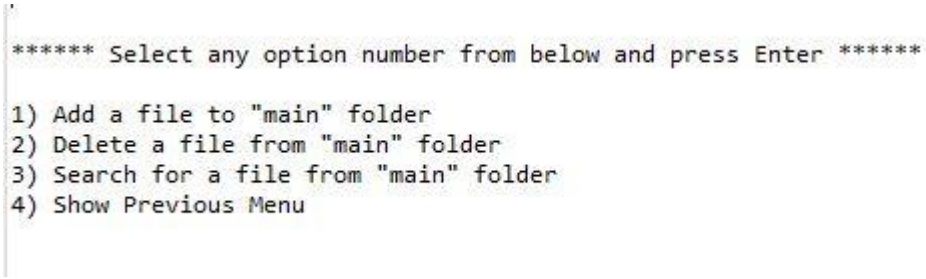
Output:

```
****** Select any option number from below and press Enter ******

1) Retrieve all files inside "main" folder
2) Display menu for File operations
3) Exit program
```

Step 3.3: Writing method to display Secondary Menu for File Operations

```
public static void displayFileMenuOptions() {
            String fileMenu = "\n\n****** Select any option number from below and
press Enter ******\n\n"
                        + "1) Add a file to \"main\" folder\n" + "2) Delete a file
from \"main\" folder\n"
                        + "3) Search for a file from \"main\" folder\n" + "4) Show
Previous Menu\n";

            System.out.println(fileMenu);
      }
```

Output:



Step 4: Writing a program in Java to handle Menu options selected by user (**HandleOptions.java**)

- Select your project and go to File -> New -> Class.
- Enter **HandleOptions** in class name and click on “Finish.”
- **HandleOptions** consists methods for -:

4.1. Handling input selected by user in initial Menu

Handling input selected by user in secondary Menu for File Operations

Step 4.1: Writing method to handle user input in initial Menu

```
public static void handleWelcomeScreenInput() {
		boolean running = true;
		Scanner sc = new Scanner(System.in);
		do {
			try {
				MenuOptions.displayMenu();
				int input = sc.nextInt();

				switch (input) {
				case 1:
					FileOperations.displayAllFiles("main");
					break;
				case 2:
					HandleOptions.handleFileMenuOptions();
					break;
				case 3:
					System.out.println("Program exited successfully.");
					running = false;
					sc.close();
```

```
                                        System.exit(0);
                                        break;
                                default:
                                        System.out.println("Please select a valid option from
above.");
                                }
                        } catch (Exception e) {
                                System.out.println(e.getClass().getName());
                                handleWelcomeScreenInput();
                        }
                } while (running == true);
        }
```

Output:

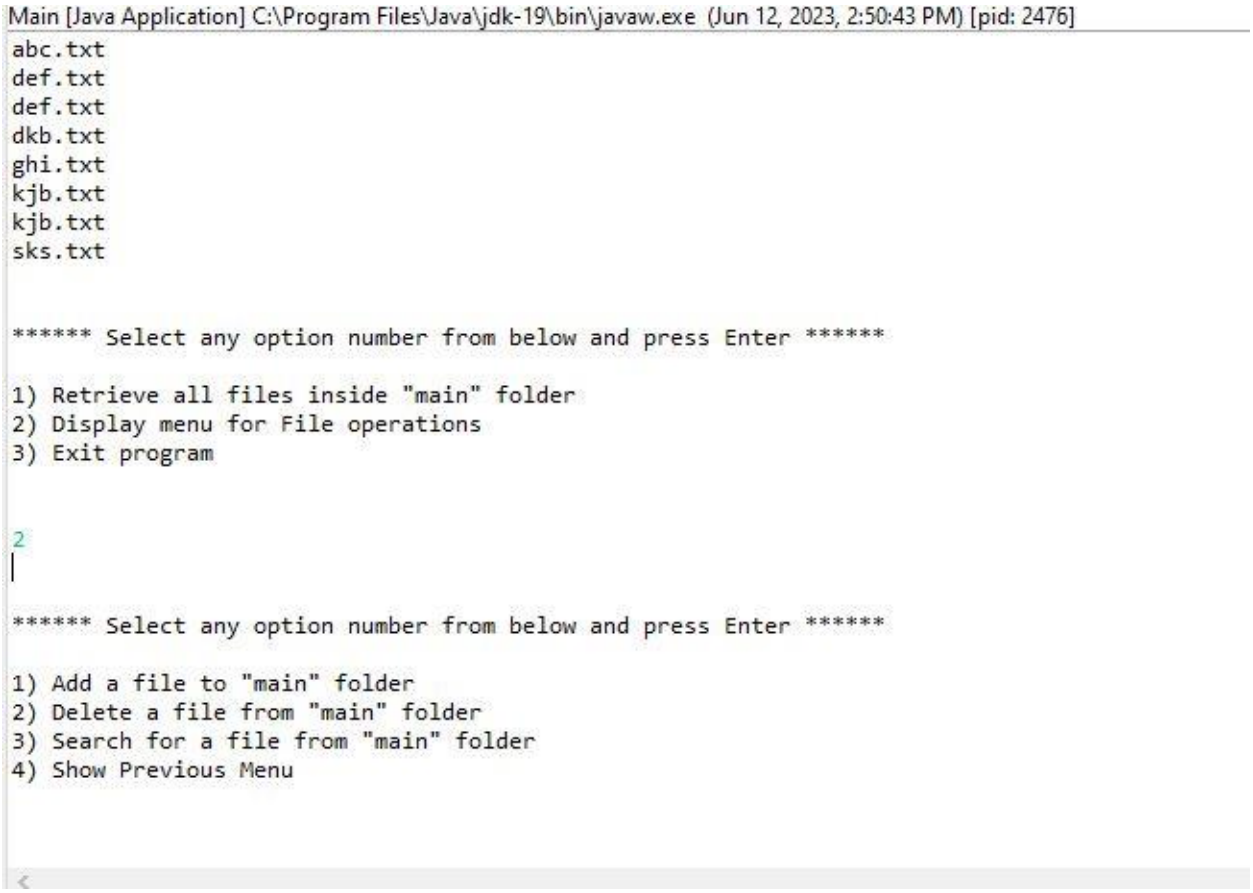
```
Displaying all files with directory structure in ascending order

`-- abc
  `-- aabc
    |-- ghi.txt

  |-- abc.txt
  |-- def.txt
  |-- kjb.txt

|-- def.txt
|-- dkb.txt
|-- kjb.txt
|-- sks.txt

Displaying all files in ascending order

aabc
abc
abc.txt
def.txt
def.txt
dkb.txt
ghi.txt
kjb.txt
kjb.txt
sks.txt
```

Step 4.2: Writing method to handle user input in Secondary Menu for File Operations

```
public static void handleFileMenuOptions() {
		boolean running = true;
		Scanner sc = new Scanner(System.in);
		do {
			try {
				MenuOptions.displayFileMenuOptions();
				FileOperations.createMainFolderIfNotPresent("main");

				int input = sc.nextInt();
				switch (input) {
				case 1:

					System.out.println("Enter the name of the file to be
added to the \"main\" folder");
					String fileToAdd = sc.next();

					FileOperations.createFile(fileToAdd, sc);

					break;
				case 2:

					System.out.println("Enter the name of the file to be
deleted from \"main\" folder");
					String fileToDelete = sc.next();
```

```
                                FileOperations.createMainFolderIfNotPresent("main");
                                List<String> filesToDelete =
FileOperations.displayFileLocations(fileToDelete, "main");

                                String deletionPrompt = "\nSelect index of which
file to delete?"
                                            + "\n(Enter 0 if you want to delete all
elements)";
                                System.out.println(deletionPrompt);

                                int idx = sc.nextInt();

                                if (idx != 0) {

        FileOperations.deleteFileRecursively(filesToDelete.get(idx - 1));
                                } else {


                                      for (String path : filesToDelete) {

        FileOperations.deleteFileRecursively(path);
                                      }
                                }


                                break;
                          case 3:

                                System.out.println("Enter the name of the file to be
searched from \"main\" folder");
                                String fileName = sc.next();

                                FileOperations.createMainFolderIfNotPresent("main");
                                FileOperations.displayFileLocations(fileName,
"main");


                                break;
                          case 4:

                                return;

                          default:
                                System.out.println("Please select a valid option
from above.");
                          }
                    } catch (Exception e) {
```

Output:



Step 5: Writing a program in Java to perform the File operations as specified by user (**FileOperations.java**)

- Select your project and go to File -> New -> Class.
- Enter **FileOperations** in class name and click on “Finish.”
- **FileOperations** consists methods for -:

5.1. Creating “main” folder in project if it’s not already present
5.2. Displaying all files in “main” folder in ascending order and also with directory structure.
5.3. Creating a file/folder as specified by user input.
5.4. Search files as specified by user input in “main” folder and it’s subfolders.

Deleting a file/folder from “main” folder

Step 5.1: Writing method to create “main” folder in project if it’s not present

```
public class FileOperations  {

	public static void createMainFolderIfNotPresent(String folderName) {
		File file = new File(folderName);

		if (!file.exists()) {
			file.mkdirs();
		}
	}
```

Step 5.2: Writing method to display all files in “main” folder in ascending order and also with directory structure. (“`--" represents a directory. “|--” represents a file.)

```
public static void displayAllFiles(String path) {
		FileOperations.createMainFolderIfNotPresent("main");
		// All required files and folders inside "main" folder relative to current
		// folder
		System.out.println("Displaying all files with directory structure in ascending
order\n");

		// listFilesInDirectory displays files along with folder structure
		List<String> filesListNames = FileOperations.listFilesInDirectory(path, 0, new
ArrayList<String>());

		System.out.println("Displaying all files in ascending order\n");
		Collections.sort(filesListNames);

		filesListNames.stream().forEach(System.out::println);
	}

	public static List<String> listFilesInDirectory(String path, int indentationCount,
List<String> fileListNames) {
		File dir = new File(path);
		File[] files = dir.listFiles();
		List<File> filesList = Arrays.asList(files);

		Collections.sort(filesList);

		if (files != null && files.length > 0) {
			for (File file : filesList) {

				System.out.print(" ".repeat(indentationCount * 2));

				if (file.isDirectory()) {
					System.out.println("`-- " + file.getName());

					// Recursively indent and display the files
					fileListNames.add(file.getName());
					listFilesInDirectory(file.getAbsolutePath(), indentationCount
+ 1, fileListNames);
				} else {
```

```
                        System.out.println("|-- " + file.getName());
                        fileListNames.add(file.getName());
                    }
                }
            } else {
                System.out.print(" ".repeat(indentationCount * 2));
                System.out.println("|-- Empty Directory");
            }
            System.out.println();
            return fileListNames;
        }
```

Output:

```
Displaying all files with directory structure in ascending order

|-- 2
|-- 4
`-- abc
  `-- aabc
    |-- ghi.txt

  |-- abc.txt
  |-- def.txt
  |-- kjb.txt

|-- def.txt
|-- dkb.txt
|-- kjb.txt
|-- sks.txt

Displaying all files in ascending order

2
4
aabc
abc
abc.txt
def.txt
def.txt
dkb.txt
ghi.txt
kjb.txt
kjb.txt
sks.txt
```

Step 5.3: Writing method to create a file/folder as specified by user input.

```
public static void createFile(String fileToAdd, Scanner sc) {
		FileOperations.createMainFolderIfNotPresent("main");
		Path pathToFile = Paths.get("./main/" + fileToAdd);
		try {
			Files.createDirectories(pathToFile.getParent());
			Files.createFile(pathToFile);
			System.out.println(fileToAdd + " created successfully");

			System.out.println("Would you like to add some content to the file?
(Y/N)");
			String choice = sc.next().toLowerCase();

			sc.nextLine();
			if (choice.equals("y")) {
				System.out.println("\n\nInput content and press enter\n");
				String content = sc.nextLine();
				Files.write(pathToFile, content.getBytes());
				System.out.println("\nContent written to file " + fileToAdd);
				System.out.println("Content can be read using Notepad or
Notepad++");
			}

		} catch (IOException e) {
			System.out.println("Failed to create file " + fileToAdd);
			System.out.println(e.getClass().getName());
		}
	}
```

Output:

Folders are automatically created along with file

```
****** Select any option number from below and press Enter ******

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu
5) Exit program

1
Enter the name of the file to be added to the "main" folder
/testing/with/folder/creation/test_file.txt
/testing/with/folder/creation/test_file.txt created successfully
Would you like to add some content to the file? (Y/N)
Y


Input content and press enter

Checking if file content written in specified file.

Content written to file /testing/with/folder/creation/test_file.txt
Content can be read using Notepad or Notepad++
```

Step 5.4: Writing method to search for all files as specified by user input in “main” folder and it’s subfolders

```
public static void handleFileMenuOptions() {
		boolean running = true;
		Scanner sc = new Scanner(System.in);
		do {
			try {
				MenuOptions.displayFileMenuOptions();
				FileOperations.createMainFolderIfNotPresent("main");

				int input = sc.nextInt();
				switch (input) {
				case 1:
					// File Add
					System.out.println("Enter the name of the file to be
added to the \"main\" folder");
					String fileToAdd = sc.next();

					FileOperations.createFile(fileToAdd, sc);

					break;
				case 2:
					// File/Folder delete
					System.out.println("Enter the name of the file to be
deleted from \"main\" folder");
					String fileToDelete = sc.next();

					FileOperations.createMainFolderIfNotPresent("main");
					List<String> filesToDelete =
FileOperations.displayFileLocations(fileToDelete, "main");
```

```
                                    String deletionPrompt = "\nSelect index of which
file to delete?"
                                                + "\n(Enter 0 if you want to delete all
elements)";
                                    System.out.println(deletionPrompt);

                                    int idx = sc.nextInt();

                                    if (idx != 0) {

        FileOperations.deleteFileRecursively(filesToDelete.get(idx - 1));
                                    } else {

                                          // If idx == 0, delete all files displayed
for the name
                                          for (String path : filesToDelete) {

        FileOperations.deleteFileRecursively(path);
                                          }
                                    }


                                    break;
                              case 3:
                                    // File/Folder Search
                                    System.out.println("Enter the name of the file to be
searched from \"main\" folder");
                                    String fileName = sc.next();

                                    FileOperations.createMainFolderIfNotPresent("main");
                                    FileOperations.displayFileLocations(fileName,
"main");


                                    break;
                              case 4:
                                    // Go to Previous menu
                                    return;

                              default:
                                    System.out.println("Please select a valid option
from above.");
                              }
                        } catch (Exception e) {
                              System.out.println(e.getClass().getName());
                              handleFileMenuOptions();
                        }
                  } while (running == true);
            }
}
```

Output:

```
****** Select any option number from below and press Enter ******

1) Add a file to "main" folder
2) Delete a file from "main" folder
3) Search for a file from "main" folder
4) Show Previous Menu

1
Enter the name of the file to be added to the "main" folder
```

2. Pushing the code to GitHub Repository:

- Open Git Bash and navigate to the folder where you have created your files.
  cd Simplilearn\APRIL 2023\Day05
- Initialize the repository using the below command
  git init
- Add all the files to your git repository using the below command
  git add .
- Commit the changes using the below command
  git commit -m "first commit"
- Add the URL for the remote repository where your local repository will be pushed
  git remote add origin https://github.com/Mahendra1272/simplilearn2023
- Push the files to the folder you initially created using below command
  git push -u origin master

Conclusion:

The application has been developed according to all the required features mentioned in the project description. Further enhancements to the application can be made by the following:

- Asking if the user wants to delete the file if is not empty.
- Allowing user to retrieve files sorted by Data modified or Type.