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 prototype of the application will be then presented to the relevant stakeholders for the 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: Create, Search and Delete.
- 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 goal of the company is to deliver a high-end quality product as early as possible.

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 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

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 it.
- The submission of your GitHub repository link is mandatory. In order to track your task, you need to share the link of the repository. You can add a section in 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)

GitHub REPO:
https://github.com/catiafsantos/PhaseIProjectDelivery.git

- All relevant information, such as this file, printscreens, the code, etc... is provided on this REPO. I will not touch the file after 24th of January of 2022 for grading reasons.

Project Name: PhaseIProjectDelivery
Project Developer: Cátia Santos (catia.santos3@vodafone.com)
Company Name: Vodafone

Sprint Planning:

The following information is not accurate, the project and the tasks did not take the time that is being mentioned bellow, the following is being provided as data to implement the Agile knowledge acquired during the course.

- Hereby the Sprint Tasks and estimates in story points, let's assume that the average is 10 SPs per sprint:

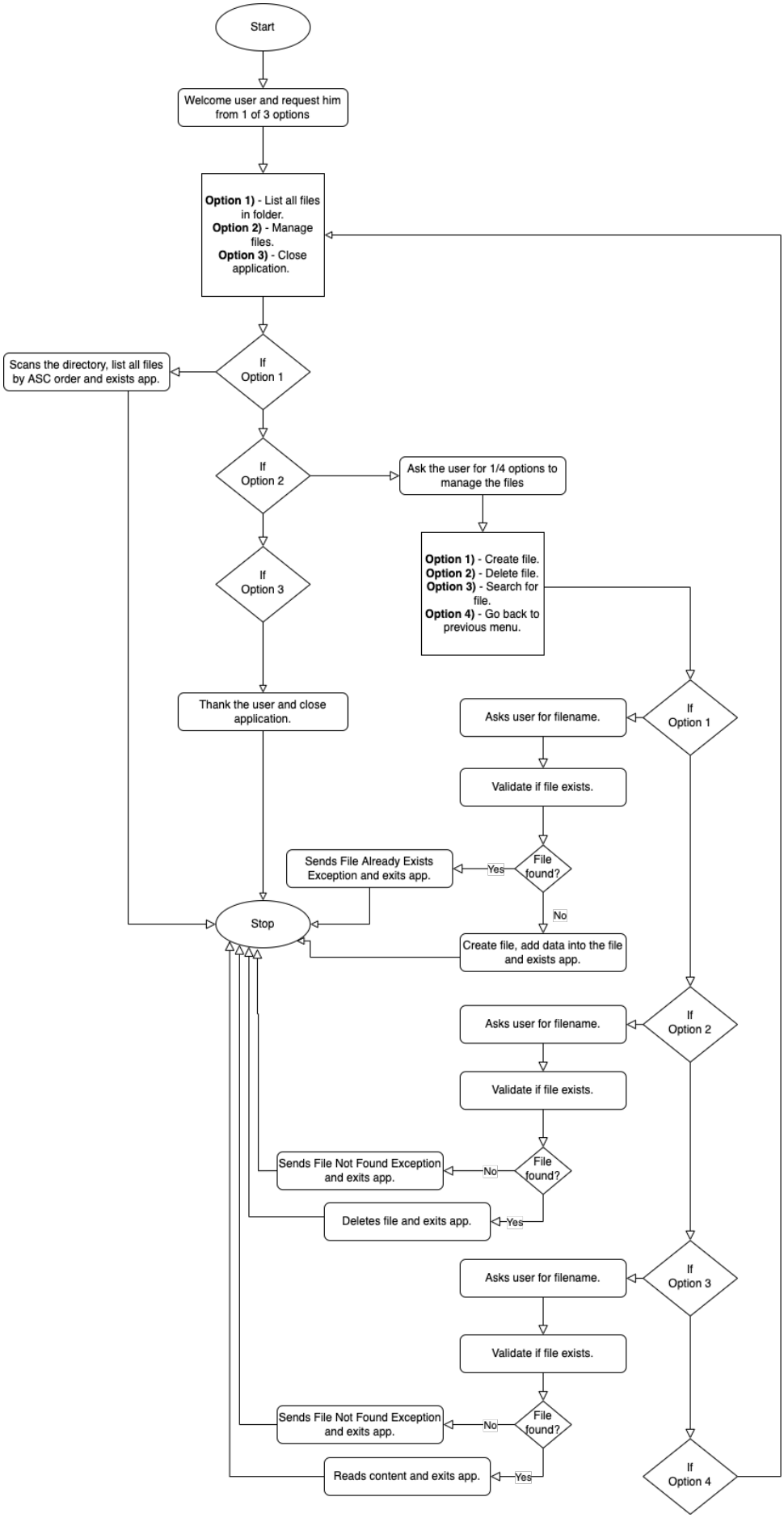
Sprint	Task Code + Title	Simple Task Description	Estimation (SP)
1	TASK-134 – Env Prep	Download the required tools, such as IDE, Java, JDK, Git, GitHub etc. Install the software	1
1	TASK-156 – Project Design	Design the application flowcharts, algorithms and behaviors and document them.	2
1	TASK-904 – Menu creation	Create console Menu with user input for 3 options, one with 3 sub-options Sub-options need to allow for user to input filename.	2
1	TASK-905 – Menu testing	Test the menu options with user input and console logging.	1

1	TASK-855 – Project Setup	Create the project in java Create the directory of the files Setup the Repository in the GitHub and link local with remote.	1
1	TASK-1440 – Create File	Create the code to add a file according with the filename provided in the user input. Write data into the file previously created. Return exception or successful message according with behavior.	3
2	TASK-1449 – Delete File	Create the code to delete a file according with the filename provided in the user input. Return exception or successful message according with behavior.	3
2	TASK-1169 – Search File	Create the code to search for a file according with the filename provided in the user input. Return exception or successful message according with behavior. If file is found read its contents to the user.	4
2	TASK-1169 – List Files	Create the code to list all files in the directory by ASC order.	3
3	TASK-1181 – Exit Application	Create the code to allow user to exit application.	2
3	TASK-1187 – Create main code	Create the code to join the user options with the methods previously created.	2
3	TASK-1194 – Testing	Test the code functionalities - test all inputs and options.	1
3	TASK-1200 – Documentation	Document the test behaviors with examples. Document the algorithms used.	1

So, the conclusion is that it will take around 2 full sprints and 6 more SPs (story points) which is roughly half more sprint, therefore 2/5 sprints.

Flowchart:

- Flowchart was designed used diagram.io online.


Start
Welcome user and request him from 1 of 3 options
Option 1) - List all files in folder.
Option 2) - Manage files.
Option 3) - Close application.
If Option 1
Scans the directory, list all files by ASC order and exists app.
If Option 2
Ask the user for 1/4 options to manage the files
If Option 3
Option 1) - Create file.
Option 2) - Delete file.
Option 3) - Search for file.
Option 4) - Go back to previous menu.
Thank the user and close application.
Stop
If Option 1
Asks user for filename.
Validate if file exists.
File found?
Yes
Sends File Already Exists Exception and exits app.
No
Create file, add data into the file and exists app.
If Option 2
Asks user for filename.
Validate if file exists.
File found?
No
Sends File Not Found Exception and exits app.
Yes
Deletes file and exits app.
If Option 3
Asks user for filename.
Validate if file exists.
File found?
No
Sends File Not Found Exception and exits app.
Yes
Reads content and exits app.
If Option 4

Technologies/Tools used:

The following technologies were used on the creation of this project:

- Java 11 – OpenJDK Temurin 11 + SDK 11
- Git version 2.31.1
- GitHub
- Mac OS Monterey Version 12.1
- IDE – IntelliJ IDEA 2021.2 Community Edition
- Microsoft Word for Max OS X – Version 16.57
- Diagram.io Online - https://app.diagrams.net/

Algorithms used:

The following algorithms were written for this project:

I decided to divide the project into two classes, the fileHandling() one that contains:

- **createFile()** – This algorithm was created to add a file as per user request.
  - Uses the filename provided by the user on the mainMenu() class
    - Uses a if/else statement with file.exists() to identify if the file already exists and if so, fails saying that the file already exists and cannot be created, exiting the application.
    - If the file does not exist, a file is created using Files.write() with the filename provided by the user.
  - The filename is case insensitive as requested by the description of the project.
  - To get the file the Paths.get() method is being used.
  - Additionally, I added the functionality of writing something hardcoded onto the file using the FileWriter() class.

- **deleteFile()** – This algorithm was created to delete a file as per user request.
  - Uses the filename provided by the user on the mainMenu() class
    - If the file does not exist, an exception is thrown using a try()/catch() saying that the file requested does not exist therefore cannot be deleted, exiting the application.
    - If the file exists, the file gets deleted using Files.delete() and a success message is shown, exiting the application.
  - To get the file the Paths.get() method is being used.

- **searchFile()** – This algorithm was created to search for a file as per user request.
  - Uses the filename provided by the user on the mainMenu() class
    - If the file does not exist, an exception is thrown using a try()/catch() saying that the file request does not exist therefore cannot be retrieved, exiting the application.
    - If the file exists, the content of the file is shown using the readFile() method.
  - To get the file the Paths.get() method is being used.

- **readFile()** – This algorithm was created to read a file when the searchFile() method is successful.
  - Uses the filename provided by the user on the mainMenu() class.
  - No exceptions for file not found were added since this method is always being called by searchFile() one that already does that validation.
  - To get the file the Paths.get() method is being used.

- **orderFiles()** – This algorithm allows to get a list of all available files in the directory by ascending order.
  - Uses Files.list() with the .sorted() option to sort the files by ASC order, which is the default.
    - Uses the .forEach() to create a loop and print the file path and name.

And the mainMenu() class that contains:

- **main() –** The main class where the program goes when it gets triggered and that allows to access the other logic.
  - Gives the user some information about the application and the options available to choose from.
  - Scans the user chosen option and uses it on the switch() statement.
    - The switch() conditional statement goes through the first 3 options, and continues the program by calling the other methods at each case.
      - In case of 1 being selected, it shows a statement saying the option was chosen and triggers the orderFiles() method.
      - In case of 2 being selected, it shows a statement saying that the option was chosen and triggers the extraMenuOptions() method, allowing to access the remaining file handling options.
      - In case of 3 being selected, it shows a statement saying that the option was chosen, and it goes out of the program, exiting the application with a System.exit(0).
      - In case of any other option being chosen, it shows a statement saying that the option was invalid and exiting the application.

- **extraMenuOptions() –** This algorithm allows to access the remaining file handling options.
  - Gives the user more information about the additional file handling options available.
  - Scans the user chosen option and uses it on the switch() statement.
    - The switch() conditional statement goes through the first 3 options, and continues the program by calling the other methods at each case.
      - In case of 1 being selected, asks the user to input the filename and changes it to lower cases to be case

insensitive, and uses that filename on the createFile() method.
- In case of 2 being selected, asks the user to input the filename, and uses that filename on the deleteFile() method.
- In case of 3 being selected, asks the user to input the filename, and uses that filename on the readFile() method.
- In case of 4 being selected, it goes back into the main() method.
- In case of any other option being chosen, it shows a statement saying that the option was invalid and exiting the application.

Additionally, there are common things between the two of them, such as:

- In all places where is necessary to send a message input to the console the System.out.println() method was used.
- In all places where is necessary to get user input from the console the Scanner() method was used.
- All file handling methods use try()/catch() and throws for IOException.
- All methods/classes that belong to Java existing collections were imported for use inside the written algorithms, such as Files(), FileWriter() and Paths().

Examples of the application running can be found into the GitHub Repo on the Documentation and Examples Folder.

END