Aidan Labourdette

Dr. Navabi

19 November 2021

M/W/F 11:15 – 12:05

**Honors Contract**

**Introduction**

I first came up with the idea of having a file size identifier when my PC was low on storage. I did not want to download one of the bloated and malicious-like computer cleaners currently available online, so I decided to create my own. Originally, the program did not have a GUI, instead it ran off the command line and would ask you for a file path manually. Deciding, that I could make it more powerful and more useful, I decided to make it more user friendly and easier to find the desired file. I thought that Java was the perfect language to use to create this program. Thanks to the Java Runtime Environment, the could would run on any operating system without any reworking of the code. Due to this feature, I could use this program on both my desktop computer to see which files took up the most space and delete them as well as on my MacBook. I also wanted to use some of the concept I learned throughout the semester. Some of these concepts include: JavaFX GUI, Comparable Interface, Multi-Threading and File-Reading.

**Requirements**

The requirements for this project are to go through all of the files defined in the given folder. Add them to an ArrayList and store that data in the app so that it can be displayed to the user and be changed when sorted. The app should be able to sort the files given based on the Comparable class chosen. The application should also be able to open the file chosen by the user so that they can edit or delete the file. The app should not freeze and should be enjoyable to use. The app should be able to handle errors and not crash. The user should also be able to choose the size of files to be stored in the ArrayList. The user should be able to know what they can and cannot do at a certain moment. They should also understand if the app is doing something, or if has crashed.

**Approach**

The approach I took was to build the back-end first. I understood that if I could try to find and add files to an ArrayList, then I could build a front-end and a GUI to suit that task. After I built the recursive methods of looking through all the files in the given directory, I tried it out on both Windows and MacOS. After confirming that it works and fixing bugs by using a black-box approach, I decided to work on a GUI that could allow the user to check the files in the given directory. After setting up a JavaFX project and importing the classes I wrote, I decided to base my front-end from Assignment 6. In that assignment, we built a JavaFX application that contains two pages. One where the user would input a class, and another where the user could check the number of students in that class. I decided that my first page should allow the user to add of the files to the ArrayList by running the back-end program using a button. Originally, I made it so that the user would have to manually enter the path to the folder; however, after looking through the JavaFX documentation, I found the FileChooser and FolderChooser classes. These two classes would natively open file explorer/finder and allow the user to define their desired path to their folder. Once I finished adding that feature, I started working on the second pane. The ListPane would allow the user to look at all their files, sorted by either Name, Path, Size, or Date. I started by designing the center of the pane where the user can see the actual details of the files found using the back-end task. At first, I tried using Labels instead of Texts because it would allow for a more pleasant viewing experience for the user; however, if a lot of files were displayed at once, the app would lag a lot due to the amount of rendering needed. Because of that problem, I decided to change it to Text, a simpler Node to render. After that, I decided to work on the sorting ComboBox. Thanks to Assignment 7, it was easy to use the ComboBox event handler and change it to a sorting Comparable instead of a Color. I decided to make it so that once the user changes the sorting method, the center pan would update and show the files in a new sorted order. After fixing a few bugs that came up, I understood that running the back-end task on the UI thread was not a good idea. Because of how the application was implemented, the back-end task would freeze the UI. I decided to multi-thread the back-end task, this decision removed the humongous load on the UI thread and stopped the freezing of the app. After making the application multi-threaded, I added a ComboBox that allowed the user to choose what the minimum size of each file needs to be before being added to the ArrayList of files. Thanks to this addition, the ListPane’s framerate drastically increased.

**Design**

The final design includes 9 classes. These classes are:

* **Comparators**
  + FileDateComparator
  + FileNameComparator
  + FilePathComparator
  + FileSizeComparator
* **Panes**
  + ListPane
  + SearchPane
* ContractApplication *(Main Class)*
* FileInformation
* SortingTypes

Each comparator takes in two variables and compares them returning -1 if the first item is less than the second item, 1 if vice versa, and 0 if they are equal. Although I added a third variable that allowed the user to choose between normal or reversed sorting, I never implemented it in the GUI and hence, never used. I then have two panes that are shown in the TabPane. The first of the two is the ListPane. This pane shows all the files found in the given directory with the parameters the user set. The second pane allows the user to choose a directory to search and set the parameters of the search. The ContractApplication class is the main class. It is run to open the application and contains the Application.start() method. It sets up the TabPane and adds the ListPane and the SearchPane to it. FileInformation is a class to store the information of each file that fits the criteria given by the user. Each FileInformation object contains a date, a fileName, and absoluteFilePath, a relativeFilePath, a fileSize, a lastModified variable, and a format variable to format the numbers into a string. The SearchPane creates the variables and adds it to the ArrayList while the ListPane reads the variables and display them in a readable fashion to the user. Finally, the SortingTypes class is an Enum class. I use that class for an easier time understanding my code when I am sorting the ArrayList in the ListPane class.

**UML Diagram:**

Text

UML Diagram of Project

[UML Diagram File](https://gofile.io/d/wuRiVk) – *Use yFile, IntelliJ IDEA, or Android Studios to open linked file*



Github Link: [link](https://github.com/Layendan/CSE_205_Honors_Contract)

**Simulation**

**Images of Application:**

The user starts on the main page where they can choose the path of the folder they wish to scan

Graphical user interface

Description automatically generated

Clicking the “Find Folder” button brings up the file explorer, allowing the user to choose the folder they wish to scan

A picture containing text, screenshot, monitor

Description automatically generated

Once the user clicks the “Start File Searching” button, the bottom changes to the total size of the files in that directory.

Graphical user interface

Description automatically generatedIf the user clicks the “Analyze Files” tab button, the application changes to show the files along with the information pertinent to that file.

Text

Description automatically generated

The user can then change the order of the files listed by clicking the “Sorting by:” button.

Graphical user interface, text

Description automatically generated

**Conclusion**

Problems I had with this project include preventing the freezing and crashing of the app when preforming demanding tasks as well as dealing with multi-threading and making the application thread-safe. Building a GUI that was easy to use but would not break the program and would not lag all the time was also a hard task that I had to overcome, at least partially. Some things that I would still want to improve in the app include reducing lag when displaying large amounts of files by creating more of smaller grids and loading/unloading them when in view or not or decrease the amount of event handlers and using instead one global event handler for clicking the path link. Sorting while on the ListPane also tends to freeze the GUI if the user is sorting many files, even though it is being multi-threaded. Most likely I am multi-threading it wrong, but it works and I do not necessarily want to change it at the moment. Other improvements I can add to the application later on would include a way to easily delete the files and be able to select multiple using a CheckBox. I would also want to add custom colors and visuals to each node to make them stand out more and create a more modern look to the application.