COSC346 Assignment 2

Participants: Tiare horwood, Grace Park

<Object Oriented Elements>

Composition

Most of the class we created follows the ‘composition’ aspect of the object oriented principle. Classes like ‘appDeligate’ or ‘<something>ViewController’ inherits instance of other classes like NSObject or NSWindowController. This allowed us to reuse the code by their composition helpling us to develop faster and give easier view to future developers who might need access to code for further development.

Inheritance

All of the NSWindowControllers that are implemented in the system follows inheritance as object-oriented component. Functions like ViewDidLoad() overrides from the superclass which enables new objects to take on the properties of existing objects.

Polymorphism

Polymorphism is implemented as inheritance was implemented. Lot of the functions that are already contained inside the superclass were easily accessed and reused. Especially in windowControllers, methods like ‘soundPlayer?.currentTime’ were used which were already there to get correct behaviour for any soundPlayers.

Coupling and cohesion

Most of the system that are generated are not dependent on the other system. Changing the feature of a single window in future development will only affect the window that has been modified unless they have direct connection between them.

Abstraction

As we developed the feature, we tried our best to keep the names of variables in sensible manner. Names like ‘(something)Clicked’ allowed us to understand what actually happens when the action is called.

Encapsulation

To implement encapsulation aspect of the Object-oriented principle we used ‘fileprivate’ to hide values from unauthorised parties. Fileprivate was used in most of the windowControllers to implement buttons heading to previous or next file in the table. Since table was created in other class, they needed access to the main class that is containing the data for files in the table.

<Features>

We implemented import button to retrieve multiple selected files from local storage. When imported, it will be placed on the table which is located on the right of the main page. The table will show what kind of metadata it is along with the name of the file. For detailed metadata of the file, user can check it through ‘metadata’ button next to the ‘bookmark’ button. The description of the metadata changes real time and according to what file is currently being selected in table view.

User can double click to view the actual file. When user double clicks the cell inside the table, it will lead to extension windows which will be selected according to the metadata given opening along side with its relevant notes. For video, when you increase the size of the window

the video also should increase in size, this applies to image as well. The extensions focus on viewing what the actual file holds so it regulates editing in all kind of metadata.

Additionally, when reading from a media file with an incorrect extension, and error window will pop up explaining your file has the wrong extension and listing the correct ones a file should have.

While viewing the file through extension, user can jump to next metadata that uses same kind of extension (video to video or image to image) by clicking '>' and '<' button. '>' button will lead to next file while '<' leads to previous one. While viewing file with extension window, our system does not regulate users from viewing multiple files using multiple windows. This was because we considered it is more user friendly way of implementation. Also for user's easier navigation through other files. For all elements that are contained inside the window (like buttons or text fields etc) we added constraints so when user increases or decreases the window, all the elements on the window moves around in sensible manner.

For the Bookmark, users can add files displayed in the View table to the bookmark by right clicking on a chosen file and pressing the "Add to bookmark" menu item. The file is then automatically sorted into the correct media type folder. Double clicking the folders will either expand or collapse the folder. Double clicking on the files inside the folder will highlight the original file in the tableview. Helping locate it if there are too many media files. Right clicking on a file in bookmark, the option to remove files is available.

When export media files, only the things displayed in the text view will be exported. This mean files found through Searches and/or Filter can be exported instead of the whole library. When choosing a place to export the media files to, directories can be created to help with file management.

There are two types of searches. One that searches the full contents of the file, and the other searches only the metadata. As for filters, there are five options to filter by. All, Audio, Documents, Image and Video. Both filter and search can be used together to help speed up the search through media files.

Each media file comes attached with a note. Within the session, notes can be edited and saved. This stays true even when the library has been put through a filter or a search. Adding notes to a searched file or a filtered file will still be attached to the same file outside of being filtered and searched for.

Finally basic information about the system can be viewed through dropdown menu from the name of the system that is located on the top of the desktop. By clicking "About" button on the menu, user can view the contributors, version of the system and when it was released.

<Testing Code>

When testing the code, we have a seperate file filled with different types of JSON files in which we could test our code manually. We did the manually to get an idea of what it would be like to use our FileUI and what problems a user may face when trying to manipulate the media files.