COSC346 Media Library Manager Report

Baz Peden: 5471490
Harrison Ellerm: 4862241

OO concepts:

Inheritance:
We have multiple subclasses of a single superclass. A file can be of multiple types so we modeled this by having our ImageMultiMediaFile, VideoMultiMediaFile, DocumentMultiMediaFile and AudioMultiMediaFIle all be subclasses of MultiMediaFile. These subclasses provided new behaviour and capabilities such as methods to check that the correct metadata was being stored in the corresponding file type. This allowed us to have a class which was a more specialized representation of the object.

Polymorphism:
The add and removeMetadataWithKey methods in the MultiMediaCollection downcast files of type MMFile to type MultiMediaFiles. This is a type of polymorphism as the parent type comes from its inheritance hierarchy.

Protocols:
The MMFile, MMMetadata and MMCollection protocols enforced the polymorphic behaviour of their corresponding class implementations. This ensured that the classes contained the necessary variables and methods to capture the desired behaviours.

Access Control:
Variables that are to be used strictly within each class are private to ensure that data is only being changed when it should be. Functions such as the containsKey and delete are also private as they should only ever be used within the multiMediaFile class.

Design Patterns:
Importer, Exporter and commandLineParser are all classes making use of the Singleton design pattern. These classes are restricted to the instantiation of a class to one object which is shared globally.
The command structure has been used so that each CommandHandler encapsulates all the information needed to perform an action at an event at a later time.

Object Interconnections:
To minimize coupling we kept functionality as closely related to each object as we could. Functions within the multiMediaCollection class deal with the manipulation of MMFile instances as much as possible, while methods in the MMFile class deal with manipulation of MMMetadata. In this way we kept the flow of control as logically coherent as possible and minimized interdependencies between classes.

We tried to maximize cohesion by making sure each class had a clear purpose and as much logic as possible associated with each class existed within it. We also modularise each piece of logic into separate functions to break the code up so that it was easier to test, read and understand.

Testing:

Unit tests were created to ensure that the code was running as expected and with the least amount of bugs as possible. We focused on making sure our programme could handle many different cases and combinations of different Json files. This approach included testing using the four different file types and also the case when a file did not match any of the expected types and also when it partially matched. We ran unit tests on each commandHandler to make sure they were all working as expected.

Roles:

We worked together implementing a pair coding strategy the majority of the project. During the initial stages of the project we both collaborated on the planning and structural decisions about how to approach many of the problems for the project. Harry implemented the importing and exporting of Json files, unit testing and core class functionality. Baz worked on some of the commandHandler implementations, testing and initial search implementation.

Reasons for bonus marks:

We implemented the NSMMCollection protocol which ensured our collection implemented the rewriteMetadataToFile and removeMetadataFromFile methods. This gave our collection the functionality to be able to delete a specific piece of metadata from a specific file or overwrite a particular instance of metadata to a file.

The use of maps within the multiMediaFile and multiMediaCollection classes which allow for greater efficiency. Within the multiMediaFile we have a map which maps keys to an array of values. This allows us to retrieve all the values associated with a specific key with O(1) lookup and saves us traversing through all the key value pairs. An example where this is useful is when we want to delete a piece of metadata from a file and there are multiple values associated with a single key. The multiMediaCollection contains a value to file array map which allows us to find all the files that contain a specific value. This is useful when we are removing or replacing metadata from multiple files.

The subclassing of multimediaFile into multiple types allows us to quickly identify if a file we created has the required metadata key value pairs to be valid according to its type. This saves a messy switch statement with a multitude of case checking.

We were also able to figure out how to make use of a Bundle in our unit testing module, which allowed us to store json files conveniently within the actual project. In this way, the projects unit tests could be run on any machine, and have access to all the necessary files.