Tiare Horwood [6115477]
Grace Park [8724170]

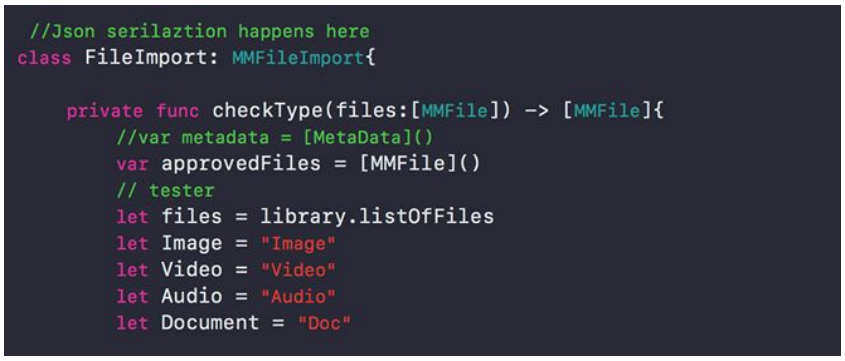
COSC346 Assignment 01

1. Object Oriented Conventions

Encapsulation

We kept our methods encapsulated into designated files along with any related data or code. An example, in our collection class we had functions that only dealt with the manipulation of objects of Collection() type. Another example, is when error checking, all errors are caught and passed through the MMClierror class instead of using print() statements mid methods to display what error was caught.s

Another concept of encapsulation we used was Information Hiding. For our code, we made sure all the variables assigned inside the class are being used by the class and are not accessible from outside by classing them as private as seen below.



```
 //Json serilaztion happens here
class FileImport: MMFileImport{

    private func checkType(files:[MMFile]) -> [MMFile]{
        //var metadata = [MetaData]()
        var approvedFiles = [MMFile]()
        // tester
        let files = library.listOfFiles
        let Image = "Image"
        let Video = "Video"
        let Audio = "Audio"
        let Document = "Doc"
```

Our checkType() is only used to check that imported MMFile types came paired with appropriate metaData key value pairs as a file are being encoded from a .json file to our Collection() type. Thus no other classes would need access to this function.

Inheritance

We do not use inheritance in our code.

Polymorphism

Because we do not use inheritance in our code, we also do not use polymorphism.

2. Testings

Most of the testings were done through importing a .json file and then testing our initial code against it. Giving the inevitable rise of errors coming up from freshly implemented code we tackled them accordingly:

- Syntax Errors
    - These were mostly dealt with by xCode giving explanations why the program would not compile.
- Logical Errors / Runtime Errors
    - Errors that interrupts the flow of logic, like index out of bounds errors or the unsuccessful casting of type String to Int, we wrote simple comment on the top of the function or class to check later. When we did get to around to dealing with them, we tried to break up the error test cases as much as possible to give the most useful feedback to the user explaining what went wrong. To further make sure we got as many fringe case errors, we asked a friend outside of computer science to cause errors in our code through prompt.
    - When dealing with logical errors where the code was not giving the desired output. We had to use the debugger feature, breakpoint, to slowly step through our code and point out the flaw in our logic. An example would be when we made the description of the MMFile initialise its value in init instead of the struct. Meaning that any mutatative functions like add would work but not be displayed when we called list to display our collection.

When checking the type of MMFile, we created both “trueImage” and “falseImage” for the type Image so when we pass the .json files we had a clear idea of what should pass and what should fail. To make sure that our checkType() function worked independently of other metaDatas in the file, we created files that had minimum required key value pair for it to pass checkType(). The final .json file we made was one of mixed true and false MMFiles to make sure that checkType() worked when passed them in one go.

3. Team Report

This was completed in as a pair. The work was evenly distributed between Grace Park and Tiare Horwood.