1 COSC346 - Assignment 1 Sample Solution

1.1 Table of Contents

- Overview
- Git Import Option 1
- Git Import Option 2
- Design
- Object Oriented Principles
- Design Patterns
- Testing
- Limitations

1.2 Overview

This repository contains a sample solution to the first assignment. There are a couple of different ways to get this working with your assignment 2 repos. If you feel that you want to fix any problems with the code feel free to do so, but keep in mind that it's not really the point of the second assignment.

I've given a brief overview of the different classes I've created.

The Importer is where a lot of the work is done. It is responsible for creating the instances of the File objects (on per type), as well as performing various checks to make sure that the file being read in exists and is valid.

Once the importer has finished loading, the resulting file list is returned to the collection. The collection is responsible for managing the interaction between the different components. The file list is then given to the indexer which adds the terms to its inverted index.

To list the files in the collection, the collection passes the search request to the indexer and returns the search results.

The add/set/del metadata happen via the collection. To add, the term/file are simpley added to the indexer and an entry for the metadata is added to the file. To delete, the term/file are removed from the indexer and the associated entry is removed from that file's list of metadata. An edit is a remove followed by an add.

To save, the various different objects are serialised to a struct (mirroring the JSON file format) and then written to the file.

1.2.1 Option 1

1. Download the zip file
2. Extract into your Assignment 2 repo
3. Add/Commit/Push these files.

This will work, but you'll lose the history.

1.2.2 Option 2

1. git clone the repo
2. git remote set-url asgn2
3. git push asgn2 master

This will preserve the history, but requires that you don't have *any* history in your repo.

1.3 Design

1.3.1 Object Oriented Principles

1. Composition (the collection is composed of Importer/Exporter and Indexer)
2. Inheritance (the different file types inherit from a base file class that does most of the work – specialisation)
3. Polymorphism (the collection only cares about MMFiles yet each are a different concrete instance, some methods in the collection/index)
4. Coupling (the indexer isn't dependent on the internals of the files)
5. Cohesion (the indexing has been separated from the collection, the import and export are separate from eachother and the collection)
6. Abstraction (the use of protocols)
7. Encapsulation (visibility is set to be as restrictive as possible where appropriate, each object has a clear set of responsibilities)

1.3.2 Design Patterns

The following design patterns make an appearence in the application.

1. Command (for handling the different commands)
2. MVC (the Metadata/File objects are the model, the view is the ResultSet and the controller is the collection/index)

3. Decorator (in the validation of metadata)
4. Factory (in creating the files from the JSON data)
5. Facade (the collection could be viewed as a facade for the index/import/export)

The following design patterns could be used in the project.

1. **Observer** used to notify the index that files/metadata/terms have been updated/changed
2. **Strategy** used to select the import/export based on serialisation methods
3. **Strategy** used to select the specific validator for checking that the file is valid (beyond the keyword validator at present)

1.4 Testing

Done via unit tests. The current code coverage metric is around 86% (which could be better) but it covers the majority of the applicaiton.

Each module has it's own test cases, for example, the Importer is tested separetely from the other classes in the system.

Have a look through the test cases (especially the MediaLibraryManagerTests.swift) as they contain examples of how to generate commands that you may find useful for Assignment 2)

1.5 Limitations

- The collection only searches for the *values* of the metadata, and not the keywords