# 1 COSC346 - Assignment 1

- **Due Date**: 4pm, Monday 3rd September 2018
- **Weight**: This assignment is worth 20% of the final grade for the paper.
- **Marking**: You are permitted to do this assignment individually or in pairs, see below for the details.
- **What to submit**:
  - A complete Xcode project including the source code along with all the test cases via your GitHub assignment repository.
  - A report in PDF format. (See instructions below.)
- **How to submit**:
  - Commit and push your changes to your GitHub repository created for the assignment (if you're using branches we'll be marking the `master` branch).
  - On the due date/time we will automatically collect all the submissions from each of your GitHub assignment repositories.

## 1.1 Table of Contents



## 1.2 Learning Objectives

The aims of the assignment include the following learning objectives:

- To demonstrate programming skills using the Swift (version 4) programming language, and some of the Foundation classes.
- To employ in practice fundamental concepts in object-oriented programming such as polymorphism, inheritance, design patterns, coupling and cohesion.
- To reflect on the strengths and weaknesses of Swift and object-oriented programming techniques
- To deliver software that is well tested and documented.

### 1.3 Problem Description

In this assignment you will design, implement, test, and document a tool for managing a media library. For the core of the assignment, you must depend on nothing more than the Foundation framework.

We have a large collection of media of assorted types (images, video, music, text documents) and we want to develop a library to help manage this collection. Attached to each of these types of media is a set of metadata. For example, an image may have some metadata describing where the photo was taken; music files may have an associated album/artist; videos may be produced by a studio, and so on.

The library should have the following features:

- import/export data from files (see: File Import/Export, File Format, and Export Search Results to a File)
- search metadata for keywords (see: Searching)
- add/change/remove metadata keywords or values (Changing Keyword/Values)
- display a list of the files returned by the search (see: Searching)

Each of these features is described below. There are some other important notes that follow these descriptions.

We have provided you some skeleton code to get started. In the `protocols.swift` file you'll find the various protocols that you'll need to implement. Feel free to add your own protocols as you feel you need to, but **DO NOT MODIFY** the provided protocols – we'll be using them to test with.

You are strongly recommended to build your test cases as you write your code for the assignment. While testing code as you build it does slow down the rate at which code appears in your assignment, for many it will also slow down the rate at which buggy code appears in your assignment.

Keep in mind that you need to submit your testing code as well as your implementation of the media metadata collection manager. Take note of the University's policy on *Academic Integrity and Misconduct* below.

### 1.3.1 Academic Integrity and Academic Misconduct

Academic integrity means being honest in your studying and assessments. It is the basis for ethical decision-making and behaviour in an academic context. Academic integrity is informed by the values of honesty, trust, responsibility, fairness, respect and courage. Students are expected to be aware of, and act in accordance with, the University's Academic Integrity Policy.

Academic Misconduct, such as plagiarism or cheating, is a breach of Academic Integrity and is taken very seriously by the University. Types of misconduct include plagiarism, copying, unauthorised collaboration, taking unauthorised material into a test or exam, impersonation, and assisting someone else's misconduct. A more extensive list of the types of academic misconduct and associated processes and penalties is available in the University's Student Academic Misconduct Procedures.

It is your responsibility to be aware of and use acceptable academic practices when completing your assessments. To access the information in the Academic Integrity Policy and learn more, please visit the University's Academic Integrity website or ask at the Student Learning Centre or Library. If you have any questions, ask your lecturer.

## 1.4 Assignment Deliverables

Along with the source code, you must provide a report in PDF format. It is an important part of the assignment. Your report should indicate:

- the way in which object-oriented concepts were used in your design and implementation;
- how you tested your code;
- if you completed the assignment in a pair, you must explain the role taken by each member of your pair; and
- if you implemented any extensions, how many bonus marks (up to 3) you believe you should be awarded, and why. Note that the bonus marks can only be used to reach the maximum mark of 20.

The report usually should not need to be longer than one or two pages. You will lose marks if the report has obvious typos, spelling or grammatical errors.

## 1.5 Marking Guidelines

The assignment is worth 20% of your COSC346 mark and will be marked out of 100. There is not a strict marking scheme, as having one tends to disadvantage students but the marking principles will include the following considerations:

- Elegant (idiomatic) code will receive higher marks than ugly code (i.e., try to avoid ugly hacks, and feel free to fix hacks that you find in the skeleton code).
- Being unable to implement the library does not mean that you will fail the assignment, but you will need to clearly indicate what is and is not working, and why.
- Code that makes good use of object-oriented principles will receive higher marks than code that does not.
- You may improve on the skeleton code provided to you, but do not forget to document and justify what you did in the code and in the report. It is certainly not a requirement to redesign the skeleton code, however you may well find that it is not particularly idiomatic Swift, and you can improve upon it. *DO NOT* change the provided protocols as we will use them for testing your implementations.
- Uncommented code will receive a maximum mark of 15/20 (or 75%) for the assignment.
- Assignments that do not provide a testing framework will receive a maximum mark of 12/20 (or 60%). Your `main.swift` should use your testing code usefully and comprehensively. It should also be possible to use your code as a library that does not run tests, i.e., your testing should be cleanly separated from your implementation.
- An assignments that is submitted without a report will receive a maximum mark of 14/20 (or 70%).
- There are a maximum of 3 bonus marks available. You should make a case for how many bonus marks you deserve in your report.

We will also be looking at your git commit history so make sure you use appropriate sized commits with good commit messages.

### 1.6 Skeleton Code

This repository contains a skeleton program that has a command line interface. When you run this program, there will be a prompt > shown. At the prompt you can type various commands corresponding to the different features. The help text shows the list of commands that should be supported – at present only the `help` and `quit` commands are implemented. You'll also find additional examples of the commands in `cli.swift`.

### 1.7 File Import/Export

The import is goverend by the command `load <filename>` whereas export controlled by `save <filename>`. In both cases the filename can be specified as a path to a file. It should be able to respond to normal unix style paths (so ~ – an alias for the users home directory – should work as expected).

### 1.7.1 File Format

We will use JSON to describe our collection of metadata as it is a very common data exchange format. An example of some JSON data is as follows:

```
[
  {
    "name": "Paul",
    "office": "2.51",
    "languages":["python","swift"]
  },
  {
    "name": "Hamza",
    "office": "2.50",
    "languages":["java","swift"]
  }
]
```

In this example we have a `list` (the square brackets) of `dictionaries` (the curly brace) containing a set of key-value pairs. The keys must be `strings` but we can store all sorts of data as the values: `strings`, `ints`, `doubles`, `lists`, and `dictionaries`. In this example, each person has a list of programming languages. Other things to note, is that the key-values are separated by a colon and there is a comma at the end of the lines that continue the description of the object.

## 1.8 Metadata

The metadata will be described using the following pattern:

```
[
  {
    "fullpath": "/path/to/foobar.ext",
    "type": "image|video|document|audio",
    "metadata": {
      "key1": "value1",
      "key2": "value2",
      ...
    }
  },
  ...
]
```

For this assignment:

- the type will be one of those given strings (i.e. either an `image`, `video`, `document` or `audio`)
- the metadata key/value pairs will always be a string value
- each of these top-level dictionaries will always contain both a `fullpath` and a media `type`

### 1.8.1 Metadata Requirements

There are also some specific requirements for the different media types' metadata as shown in the list below. For example, an `image` file type *must* have `creator` and `resolution` metadata associated with it.

- image: creator, resolution,
- video: creator, resolution, runtime
- document: creator
- audio: creator, runtime

*Hint*: Swift (and most other languages) already have functions to serialise/deserialise JSON data **DO NOT WRITE THIS YOURSELF!**

## 1.9 Searching

You need to implement a simple search feature for scanning through the metadata attached to each file to find a term. This is provided by the command `list <term>`, where term is the thing we're looking for. If we use the above JSON data as a basis, then the resulting dictionary should look something like the following:

```
java -> [Hamza Instance]
python -> [Paul Instance]
swift -> [Paul Instance, Hamza Instance]
```

In the provided interface, we can interact with the index as follows:

```
> list python
0: Paul
> list swift
0: Paul
1: Hamza
```

*Note*: This is not designed to be a sophisticated search engine as there are many other things that need to be taken into account.

### 1.10 Changing Keyword/Values

You'll need to support the adding/changing/removing metadata items associated with a file. These options are supported via the following commands:

```
add <number> <key> <value> ...
set <number> <key> <value> ...
del <number> <key> ...
```

These commands rely on the result of a previous `list` or `search` command. The idea is that the `number` parameter refers to the index of the particular result in the list of results. This is better shown with an example:

```
> list
0: Paul
1: Hamza
> add 0 likes coffee
> search coffee
0: Paul
```

In this example, we first generate a list of all the items (Paul and Hamza instances) in our collection. We then want to add a new metadata element to the item "Paul". Finally, we perform a search for that new metadata item and show the result. The important thing to note about this is that the index has been updated with the freshly added metadata (the same changes should be reflected by the `set` and `del` commands too).

### 1.11 Export Search Results to a File

We should also export the last search's results to a file. This is performed by the `save-search` command, and otherwise behaves the same as the `save` command detailed above.

### 1.12 Important Notes

We should be able to import data from a file at any point and then be able to search for those items. We may want to add new metadata to a file — which should be reflected in the index and should be exported.

When you want to handle errors from your library throw an appropriate exception — don't print the error out to the terminal from within your library (do that from where you call that function).