OVERVIEW

Providing a library to compares two (http/https) api response efficiently for a large given inputs of api endpoints which resides inside two given files. It should able to handle any kind of exception while doing the comparison of api response. It should be able to compare xml and Json api responses.

GOALS

Design goals involves with major entities mentioned below

- Dealing with huge files, we have to take care of it that it is not causing too much memory pressure.
- Comparing two responses which could be xml,json or any other and comparison should be very accurate.
- Handle exception at various levels like corruption , api endpoint latency , invalid api endpoint etc.
- Testability measures like each core methods and code flow is testable via test data of positive and negatives inputs.

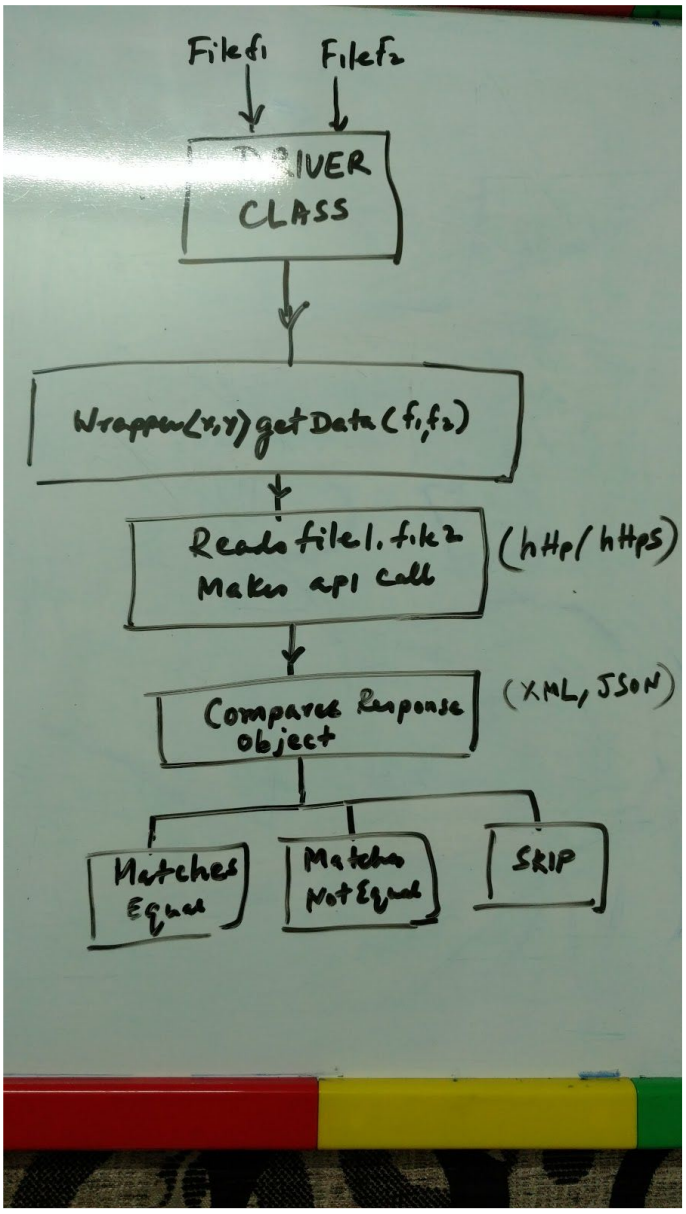
TERMINOLOGIES

<X,Y> : Type of the response object which will get compared

File streaming : An iterative algorithm which process through each line without keeping reference to them.

Serialisation : Translate object state in a format which can be stored in memory or file

PROCESS FLOW



With the intention of comparing two api responses whose endpoints are residing inside two big files. We will define a interface having the following method signature

```
public interface IComparator<X,Y> {

   boolean compare(X x,Y y);

   Wrapper<X,Y> getData(File f1,File f2) throws IOException;

}
```

boolean compare(X x,Y y);

This boolean compare method has be overridden by the implementing class which will be called by passing the two serialised api response objects which needs to be compared.Internally it will compare the byte array of type casted api response object and return true if compared return code is equal to 0 and false otherwise.

Wrapper<X,Y> getData(File f1,File f2)

The implementing class has to override this method getData in order to process the file inputs and do the response comparison.Here the getData method will receive two large files as inputs which will have api endpoints as it’s file contents. This will check the validity of the files before proceeding. As we are dealing with very high file size contents , loading the entire file would tax the system. So we would follow the approach of streaming through the file. We would leverage this with Apache Commons io LineIterator to achieve this. As the entire file will not be loaded into the memory, it’s consumption will or order close to ~150 Mb.