COMP 303 Neyir Erdeser
Assignment 3 260736004

Part 1:
I used the Singleton design pattern. To achieve it, I changed the default constructor to private and included a static variable INSTANCE in the Library class. The two new attributes name and emailID are null at instantiation. To ensure that name is not null when the client is using the INSTANCE, I made INSTANCE private and included a getter method with the precondition of name not being null. This means that the client first needs to use the setter method for name to change the name, and only after that can they have access to the only library object. There are no restrictions about the emailID, meaning that they will not be required to fill the information, but I have choose to not let the client delete existing information, so once an emailID is set, it can be changed but it cannot be set back to null.

Part 2:
I used the Flyweight design pattern. Instead of creating both a new interface and a factory class, I opted to using the Watchable interface for this, since that is the interface that both Movie and TVShow classes implement. Then, I created a factory class ContentFactory, manages the creation of Watchables, namely Movies and TVShows. Factory has two hash maps that it uses as cache for movies and tv shows separately, which allows client to create one tv show and one movie that share the same name. the two methods getMovie and getTVShow in the factory class, checks the appropriate cache to see if a watchable with the same name already exists, and returns that in that case. If it does not yet exists, it calls the appropriate now-protected constructor to create a new one, add the newly created watchable into the cache, and return it to the client.

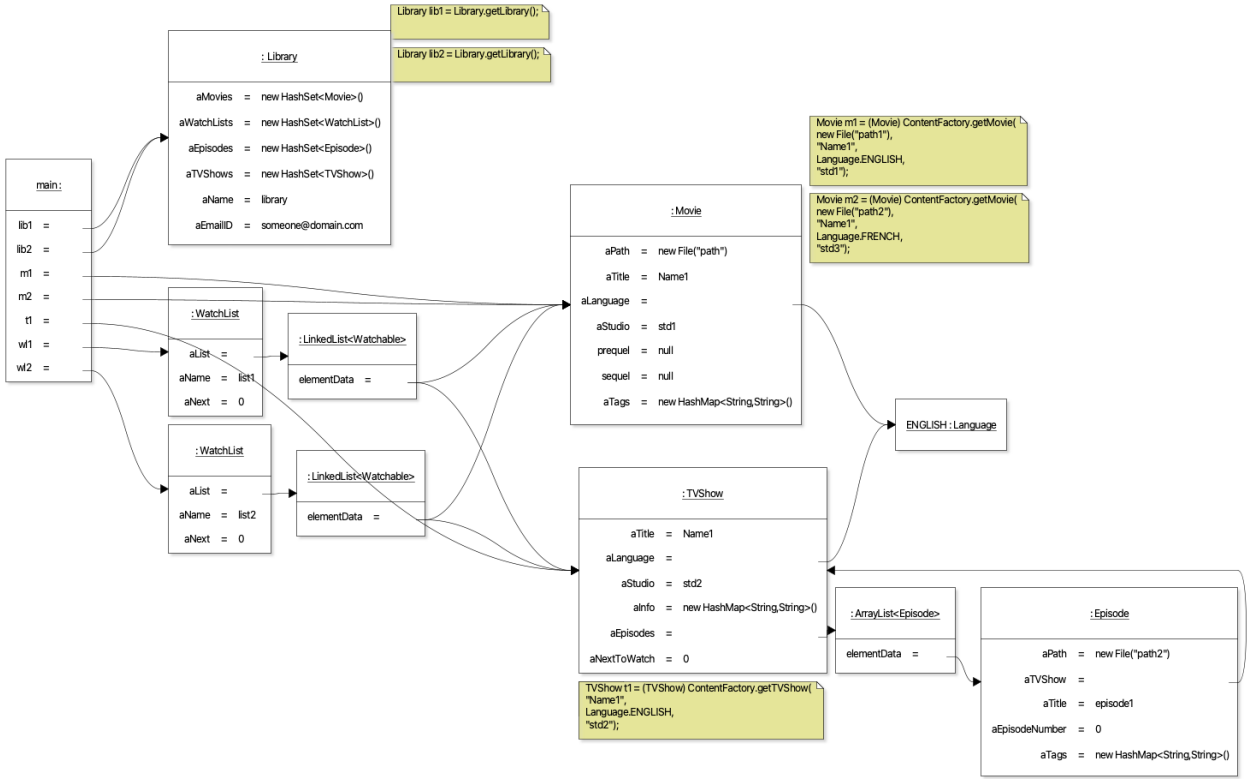
Part 3:
In WatchList class iterator() was already overridden in the base line. What I did was to override the equals() method from Java.Lang, and after doing the preliminary checks, utilize the overridden iterator method to go through the content of both watchlists one by one and ensure that they are indeed equal. In the case we are comparing the Watchable objects with the default .equals(), meaning we're checking whether they are the same object with same address, disregarding the content of them.

- I have done minor changes to the baseline, this is the one big-ish change: in Library;

```
public void addWatchList(WatchList pList) {
   assert pList != null;
   aWatchLists.add(pList);
   for (Watchable watchable : pList) {
      if(watchable instanceof Movie) addMovie((Movie) watchable);
      if(watchable instanceof TVShow) addTVShow((TVShow) watchable);
      if(watchable instanceof Episode) aEpisodes.add((Episode) watchable);
   }
}
```

the method was defaulting all items in the watchlist as a movie, where a watchlist can contain movies, tvshows and episodes, so it resulted in exceptions when trying to add a watchlist to library.

Neyir Erdeser
260736004



An object diagram, displaying most of my Driver class, focusing on featuring the assignments requirements. Some aspects are ignored for readibility