DESIGN PATTERNS
TERM PROJECT

Arda Atasoy 20180808045

STATEMENT OF WORK

In fact, the reason why I tried to establish a structure similar to a TV series-movie viewing service in this project is that I tried to implement this idea in a project in the first year and had problems on an oop basis. Therefore, the situations that required the use of the patterns I used in the project were the problems I encountered before and I didn't know anything about design patterns at that time.

The first of the problems I encountered was ensuring the singularity of the services that are responsible for the production of films or series for both or for each separately. The other problem was that the production of movies and TV series became modular. For example, TV series and movies should have been created by the order of a center, but should be provided by different services. In other words, it should not be presented as a different structure by overloading a method and increasing its parameters. Another stage was the consistency of the subscription system of the users. People should be able to receive all the news of the addition of all added series or movies, or just any of them, if desired, and also be able to receive notifications if a new episode is added to a series if they want. Trying to do this had messed things up a bit before and there was some spaghetti code situation.

HOW DID I SOLVE THIS PROBLEMS

I used three design patterns in the project. These are Singleton Design Pattern, Factory Design Pattern and Observer Design Pattern.

Only one of the MediaProducer class, which is responsible for the production process of the media, had to be produced and I solved this problem by using the Singleton Design Pattern.

Companies can order series or movies from the same company and do not care about which studio the content is produced by the company. I solved the ordering process through a tool with a command with Factory Design Pattern.

People who subscribe to the system are kept with objects derived from the User class, and if they want to be notified when a new movie or series is added to the system, they subscribe to the system via MediaCreator. The MediaObservable class helps to add-remove subscribers and to be notified.

UML CLASS DIAGRAMS

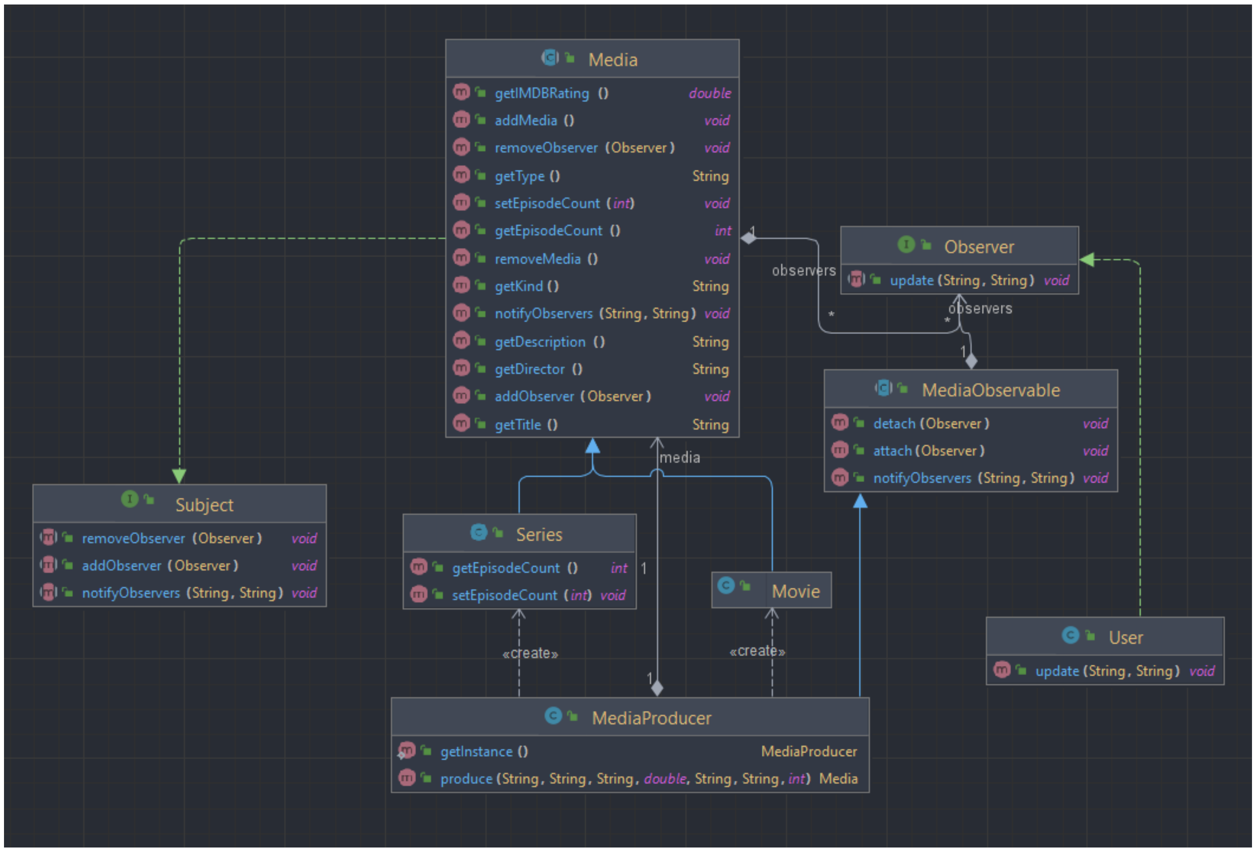
Media
getIMDBRating () double
addMedia () void
removeObserver (Observer) void
getType () String
setEpisodeCount (int) void
getEpisodeCount () int
removeMedia () void
getKind () String
notifyObservers (String, String) void
getDescription () String
getDirector () String
addObserver (Observer) void
getTitle () String
Observer
update (String, String) void
observers
observers
*
*
1
1
MediaObservable
detach (Observer) void
attach (Observer) void
notifyObservers (String, String) void
Subject
removeObserver (Observer) void
addObserver (Observer) void
notifyObservers (String, String) void
media
Series
getEpisodeCount () int
setEpisodeCount (int) void
1
Movie
«create»
«create»
1
User
update (String, String) void
MediaProducer
getInstance () MediaProducer
produce (String, String, String, double, String, String, int) Media

Diagram with dependicies and methods