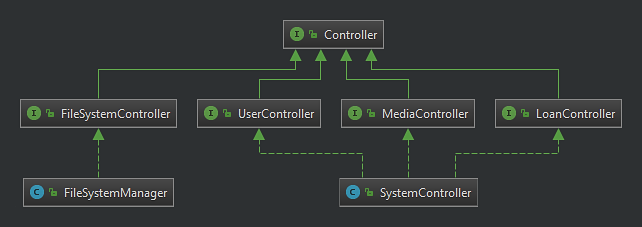
INGEGNERIA DEL SOFTWARE – ELABORATO

**Analisi dell’OPC**

In seguito verrà analizzato il programma Biblioteca SMARTin4t0r 3.0 per controllare se ogni package rispetta il principio **O** dell’insieme di principi S**O**LID, ovvero l’**OPC** (open-closed principle) secondo cui ogni modulo deve essere aperto all’estensione e chiuso alla modifica. Verranno trascurate le classi utility e in generale i package contenenti classi con funzionalità statiche e di uso singolo.

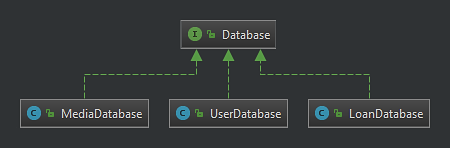
**L’analisi package per package**

Il **package** controller contiene le seguenti classi:



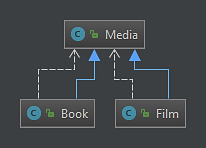
N.B.: per la struttura generale del programma è stato usato il pattern MVC.  
Come si può notare dallo schema UML, le varie interfacce più specifiche UserController, MediaController e LoanController estendono l’interfaccia più generica Controller, definendo metodi specifici per i vari controller che implementeranno tali interfacce.   
SystemController è la classe “controller” principale e, implementando tutte e tre le interfacce, contiene tutti i metodi relativi al controllo di utenti, media e prestiti. Durante il refactoring del programma sarà possibile spezzare in tre parti diverse la classe SystemController, dividendola in tre controller più specifici che implementeranno ognuno un’interfaccia diversa delle tre sopra descritte.  
Questo modulo **rispetta** il principio open-closed, in quanto le tre interfacce UserController, MediaController e LoanController contengono metodi di uso generico e sia “view” che “model” non fanno riferimento alla classe SystemController direttamente, bensì alle interfacce.

Il **package** database contiene le seguenti classi:



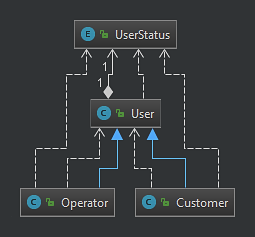
In questa prima versione del programma, l’interfaccia Database è ancora vuota. Per adesso, essa esiste solamente per limitare il tipo dei parametri che vengono passati al metodo generico di salvataggio dei database all’interno di SystemController, limitando il tipo a <D extends Database>.  
Molti metodi sono già comuni tra i vari database, ad esempio il metodo add() per aggiungere elementi ai database o il metodo fetch() per prendere un oggetto dal database. Per adesso tali metodi hanno nomi diversi a seconda del database (ad esempio addUser(), addMedia(), addLoan() per il metodo di aggiunta).  
Durante la fase di refactoring, perciò, si dovrebbero accomunare questi metodi (ad esempio trasformando i metodi add…() di prima in un semplice add()), aggiungendoli all’interfaccia e implementandoli in ogni database specifico.  
Questo modulo **non rispetta** il principio open-closed, in quanto l’interfaccia Database è ancora vuota.

Il **package** media contiene le seguenti classi:



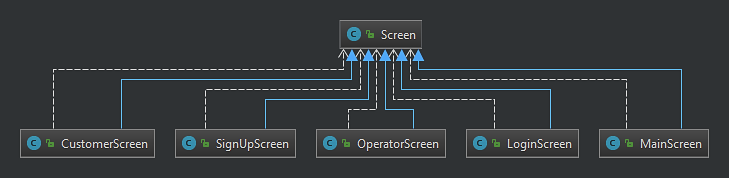
Nonostante non siano presenti interfacce, la classe Media è la classe padre di Book e Film. Singolarmente, queste due classi figlie sono chiuse alla modifica e il modulo intero è aperto all’estensione, in quanto una o più tipologie nuove di Media potranno essere create in futuro senza problemi (e senza dover modificare alcuna struttura dati all’interno del programma).  
Questo modulo perciò **rispetta** il principio open-closed.

Il **package** user contiene le seguenti classi:



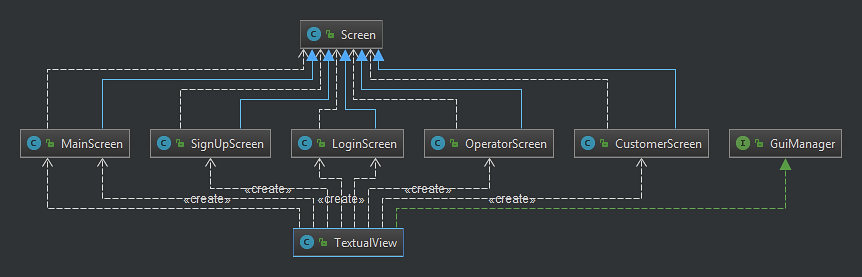
Questo modulo funziona in modo analogo al modulo precedente. User è la classe padre di Operator e Customer. Il modulo è facilmente estendibile e si possono aggiungere altre tipologie di User senza dover modificare alcun metodo o struttura dati. L’**enum** UserStatus richiede che, all’eventuale aggiunta di un’altra tipologia di User, venga aggiunta anche un’altra voce nell’enumerazione al fine di specificare lo status (la tipologia) dell’utente.  
In fase di refactoring, per migliorare l’estendibilità e la flessibilità, sarebbe preferibile eliminare l’**enum** e rendere il parametro privato userStatus (correntemente di tipo UserStatus) all’interno di User una semplice stringa, che verrà poi sovrascritta dalle classi figlie in modo del tutto indipendente, a seconda del tipo di utente che è stato definito.  
Questo modulo perciò **rispetta quasi del tutto** il principio open-closed.

Il **package** screens contiene le seguenti classi:



Tutti gli “schermi” (view testuali in questo caso) più specifici sono classi figlie di Screen, classe “schermo” generica. Gli “schermi” specifici sono chiusi alla modifica e, potendo aggiungere altre classi, altri “schermi” per altre funzioni future, senza dover apportare modifiche al codice, questo modulo è altamente estendibile.  
Esso **rispetta** quindi il principio open-closed.

Il **package** più generico gui (contenente anche il **package** screens di cui sopra) contiene le seguenti classi:



TextualView è la classe che gestisce l’interfaccia grafica testuale del programma. Nella prima versione del programma è stata anche abbozzata un’interfaccia grafica che, una volta completata, potrà facilmente essere integrata nel programma in modo del tutto indipendente e separato. Anch’essa implementerà l’interfaccia GuiManager che presenta al suo interno metodi di uso generico per classi view generiche, come ad esempio mainScreen(), signUpScreen(), ecc.  
Questo modulo perciò **rispetta** il principio open-closed.

**Riassunto finale**

|  |  |
| --- | --- |
| ***Modulo*** | ***Conformità a OPC*** |
| Controller | **✓** |
| Database | ✖ |
| Media | **✓** |
| User | **∼** |
| Screen | **✓** |