Pattern e principi di progettazione

## Pattern

I pattern utilizzati in progettazione sono i seguenti:

### Singleton

La classe Document è realizzata come un Singleton per presentare una via di accesso unica ai dati del modello “remoti”, cioè quelli non legati alla sessione corrente.

### Strategy

La classe Sessione, per fornire il metodo CalcolaValutazioneTotale(Recensione):float*,* utilizza,tramite composizione e delega, l’interfaccia ICalcoloValutazioneTotale, le cui concretizzazioni definiscono diversi algoritmi di calcolo alternativi. L’aggiunta o il cambiamento di un algoritmo di calcolo si può ottenere quindi semplicemente aggiungendo una classe che implementa quella interfaccia, o modificando una classe esistente, senza dover modificare la classe Sessione.

### Flyweight

Le istanze delle classi CalcoloValutazionePersonalizzata e CalcoloValutazioneNonPersonalizzata si ottengono tramite una factory CalcoloValutazioneTotaleFactory.

La classe CalcoloValutazioneNonPersonalizzata non ha stato e per questo può essere condivisa senza conflitti, quindi la factory ne mantiene una istanza sola (creata alla prima richiesta) e restituisce sempre quella.

La classe CalcoloValutazionePersonalizzata è invece specifica per un UtenteRegistrato, quindi le sue istanze vengono mantenute in un Dictionary<UtenteRegistrato, ICalcoloValutazioneTotale>. In questo modo ogni sessione dello stesso utente condividerà l’algoritmo di calcolo personalizzato, sia nel caso di più sessioni contemporanee, sia nel caso un utente faccia autenticazioni successive.

L’UtenteRegistrato è quindi lo stato intrinseco, la recensione da valutare è lo stato estrinseco.

### Model View Presenter

L’applicazione è stata organizzata secondo il meta-pattern Model View Presenter, in cui i Presenter si registrano agli eventi scatenati dal Model con una granularità abbastanza fine. Tutti i presenter infatti si registrano solo agli eventi del modello lanciati dalle istanze di loro competenza, ad esempio UtentePresenter con l’istanza di UtenteRegistrato corrente (con deregistrazione al Logout), oltre che al cambiamento della Sessione corrente. Inoltre grazie ai Presenter la MainForm non conosce nessuna classe di View.

### Template Method

Il presenter astratto VideogiochiPresenter ha il metodo *GetBindingSource():BindingSource* astratto, utilizzato all’interno del metodo BindData() concreto. Esso definisce la sorgente dei dati per il popolamento della lista dei videogiochi, ed è astratto per permettere alle classi derivate di ridefinirla con i propri criteri. la classe concreta VideogiochiNonRecensitiPresenter selezionerà solo i videogiochi senza recensione, mentre la classe VideogiochiRecensitiPresenter selezionerà solo quelli con la recensione.

## Principi di progettazione

Nella progettazione è stato molto utilizzato il principio di composizione e delega, ad esempio nel caso di UtenteRegistrato e Recensione, che delegano rispettivamente alle classi contenitore Preferenze e AspettiValutati la gestione delle liste.

Per il Single Responsibility Principle infatti ogni classe non dovrebbe avere più di una responsabilità.

Nella progettazione della autenticazione è stato seguito il principio della inversione delle dipendenze. La sessione non dipende direttamente da una classe concreta per fornire le funzionalità di autenticazione (anche qui SRP), ma dall’interfaccia IAutenticatore, che permette quindi una migliore estendibilità a metodi di autenticazione differenti.

Lo stesso principio è stato applicato grazie al pattern Strategy nelle classi del calcolo, e per la gestione della persistenza: il Document dipende dall’interfaccia IModelPersister, non dalla implementazione concreta (fittizia).

L’applicazione del principio di inversione delle dipendenze permette anche di rispettare pienamente l’Open-Closed, rendendo queste parti facili da estendere tramite l’aggiunta di nuove classi.