# VIEW

PER SEPARAZIONE MODELLO VISTA:

-tolto tutti i system.out.print da Fruitore e Fruitori e spostati in FruitoriView: in questo modo in Fruitore/i restano solo i dati (model) e i metodi che operano su di essi, mentre FruitoriView fa da view (li stampa) e anche da controller (?), fornendo al model l’input dell’utente. (fatto con tutte le classi)

Fare RisorseView con metodi uguali di FilmsView e LibriView per non avere codice duplicato?  
alcuni metodi hanno come parametro la classe sui quali vengono invocati, in modo che il metodo sia uguale ma stampi “libro” o “film” a seconda dell’invocazione.

Fruitore e Fruitori, Libro Libri Film Films, Prestito Prestiti, dovrebbero essere a posto

In tutti i metodi di StoricoController viene utilizzato il relativo model (Storico). La differenza sta che StoricoController interagisce con utente.

CLASSI DEL MODEL non dovrebbero importare il package *view*

MVC:

*Controller (logica di controllo)*:  
è la parte che fa da interfaccia tra il Model e il View. Si occupa di controllare gli eventi che scaturiscono dagli elementi nell’interfaccia grafica e chiamare gli opportuni metodi del Model, modificando i dati lì presenti e aggiornando la View.

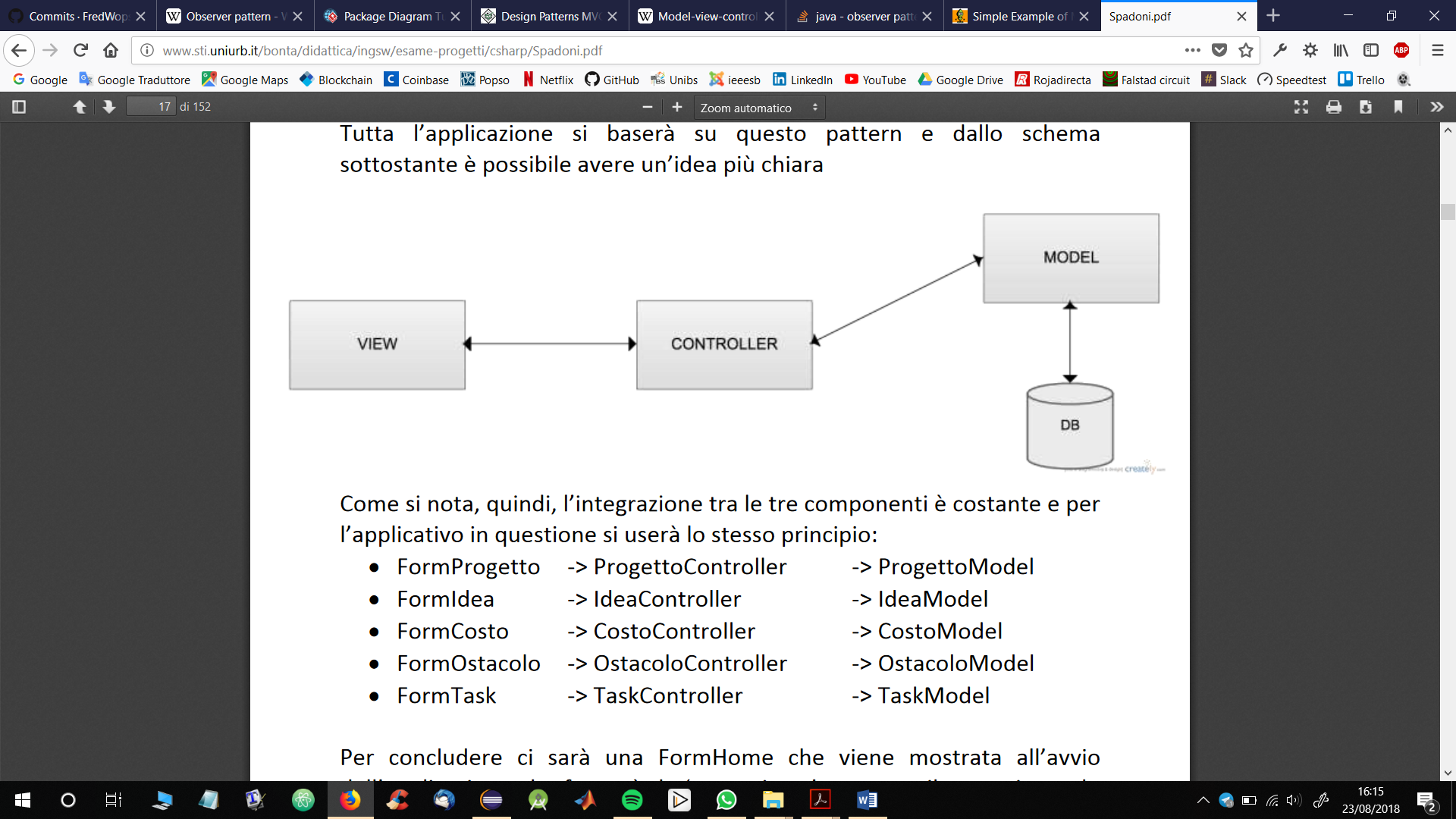
*Model*:

Rappresenta tutte le classi che hanno a che fare con il campo d'utilizzo e non con l'interazione dell'applicazione.

*View (logica di presentazione)*:

È l'interfaccia grafica dell'applicazione. Nel caso delle applicazioni a riga di comando si riduce alle istruzioni di stampa a video.

(da <http://www.astromoodle.altervista.org/download/pattern_part3.pdf>)

  
(idea su cosa scrivere nella relazione, da <http://www.sti.uniurb.it/bonta/didattica/ingsw/esame-progetti/csharp/Spadoni.pdf>)

Si potrà creare una specie di controller grasp per creare e gestire tutti i controller?  
MainController, credo rappresenti qualche pattern.

Si potrebbe fare un Controller Grasp (che funziona anche da FACADE: facade controller) che riceva tutti gli eventi di sistema dalla UI (ossia dal nostro main?) e che poi deleghi i compiti ai singoli controller MVC. Servirebbe per il punto 2). (vedi CONTROLLER GRASP)  
*“Facades are meant to be wrappers around complex functionality, their primary goal is hiding complexity of an underlying system. You can think of the Facade as a layer wrapping the complex functionality and providing simpler methods to interact with.   
A controller, on the other hand, is normally tied to a very specific piece of functionality in the system. Its main goal is to mediate the interaction between a view and a model, or in some cases just a model.  
It's more typical to have a facade interacting with several controllers, than vice versa.”*

Le nostre view sono statiche, non ha senso instanziare classi che stampino solo su console (?)  
Non vanno quindi passate come parametro ai controller. Oppure le istanzio lo stesso (come singleton probabilmente) in modo che se ci sarà un’interfaccia utente basterà sostituirla? Dovrà implementare un’interfaccia che comprenda tutti i metodi, in modo che non ci sarebbero problemi nella sostituzione?

sposto nel controller i metodi del model che richiedono interazione con la view: model non deve dipendere da VIEW.

Per esempio, addFilm (in films) richiedeva interazioni con la view. Ora delle interazioni si occupa il controller, Films si limita ad aggiungere il film creato.

Va fatto con tutto :C

filmController creato nel main, passato tra i vari menu fino ad arrivare a MenuAggiungiRisorsa, dove serve il controller per interagire con la view e creare il film.



Il controller (tramite la view) aggiorna i dati, e poi aggiorna quelli del model.

L’oggetto filmController viene passato tra i metodi e il suo model resta aggiornato, basta salvare l’archivio.

I dati restano salvati nel model del controller, che le salva sui file. Poi a inizio programma il main carica tali dati e li ridà al controller. Quindi poi al posto di passare il parametro Films passerò filmController?

filmController lavora su archivio e poi salva le sue modifiche (solo ai film) su file.

Quando filmArchivio modifica qualcosa, salva su file (che poi verrà caricato a inizio programma)

Se io creo archivio e lo passo come parametro al costruttore di FilmController, da filmController modifico l’archivio originale!

# MENU

Creata una classe per ogni menu, tolte tutte le parti di grafica e spostate nel package *view*.

Sono tutti Extract Class (del refactoring) ?

Anche per la grafica sono tutti extract method: i syso diventano metodi nella view (quindi anche move method)

Sono classi statiche, cioè non vengono istanziate ma hanno metodi statici.

Potrebbero diventare singleton? Così si possono istanziare ma solo una volta.

# EVENTI/INTERAZIONI SISTEMA (da casi d’uso) per SSD

* Verifica operatore
* Visualizza storico
* Visualizza fruitori
* Visualizza prestiti

Operatore

* Aggiungi risorsa (libro/film)
* Rimuovi risorsa (libro/film)
* Visualizza risorse(libri/film)
* Cerca una risorsa(libro/film)
* Login
* Registrazione
* Visualizza dati personali
* Rinnova iscrizione

Fruitore

* Termina prestiti
* Richiedi un prestito
* Rinnova prestito
* Visualizza prestiti in corso
* Cerca una risorsa(libro/film)

# REFACTOR

IN MenuRichiediPrestito c’era codice duplicato per le varie risorse: usando la superclasse Risorsa sono riuscito a ridurlo molto, lasciandolo solo per distinguere tra il numero massimo di risorse prestabile tra libri e film (si dovrebbe sistemare pure questo, un po’ dappertutto)

Storico da classe statica è diventato istanziabile: al costruttore si passano archivio ecc. e quando serve si può interrogare per avere informazioni storiche.  
In tutti i metodi di StoricoController viene utilizzato il relativo model (Storico). La differenza sta che StoricoController interagisce con utente.

# PATTERN

Singleton con parametri: <https://paolopatierno.wordpress.com/2011/07/06/pattern-singleton-con-parametri/>