Corso di Programmazione ad Oggetti – a.a. 2013/2014
Corso di Laurea in Ingegneria e Scienze Informatiche – Università di Bologna

Relazione relativa alla realizzazione di jSnail Motion Pictures: un sistema per l'organizzazione e la gestione di file video

Componenti del gruppo: Antony Chiossi, Michele Buzzoni, Luca Accorsi

Analisi del problema: data la scarsa presenza di movie managers online è stata progettata una applicazione per la gestione elettronica di una collezione di video presenti in un computer. Le funzionalità previste sono molteplici. Si richiede di poter:

- aggiungere allo storage dell'applicativo video singoli oppure data una directory di partenza analizzare tutte le sub-directories ed aggiungere automaticamente i file video individuati (controllando che l'estensione rientri in un determinato set di estensioni gestite vedi help dell'applicazione);
- memorizzare in modo permanente, tra una sessione e l'altra, le informazioni relative ai video;
- controllare che il video una volta aggiunto nell'applicazione corrisponda effettivamente ad un file contenuto nel file system ed in caso negativo poter effettuare il "riallineamento";
- ricercare un video contenuto nello storage del programma in base a diverse proprietà;
- cercare online informazioni su di un determinato video e se trovate memorizzarle per la consultazione offline;
- modificare le informazioni del video (regista, generi, ecc..) eventualmente marcandolo in vari modi per poterlo individuare più velocemente;
- organizzare i video in diversi modi (rimuovendoli dall'application storage o dal file system, muovendoli oppure copiandoli);
- visualizzare il video con il player di default del sistema operativo oppure con un embedded player (per ragioni di tempo) molto elementare;

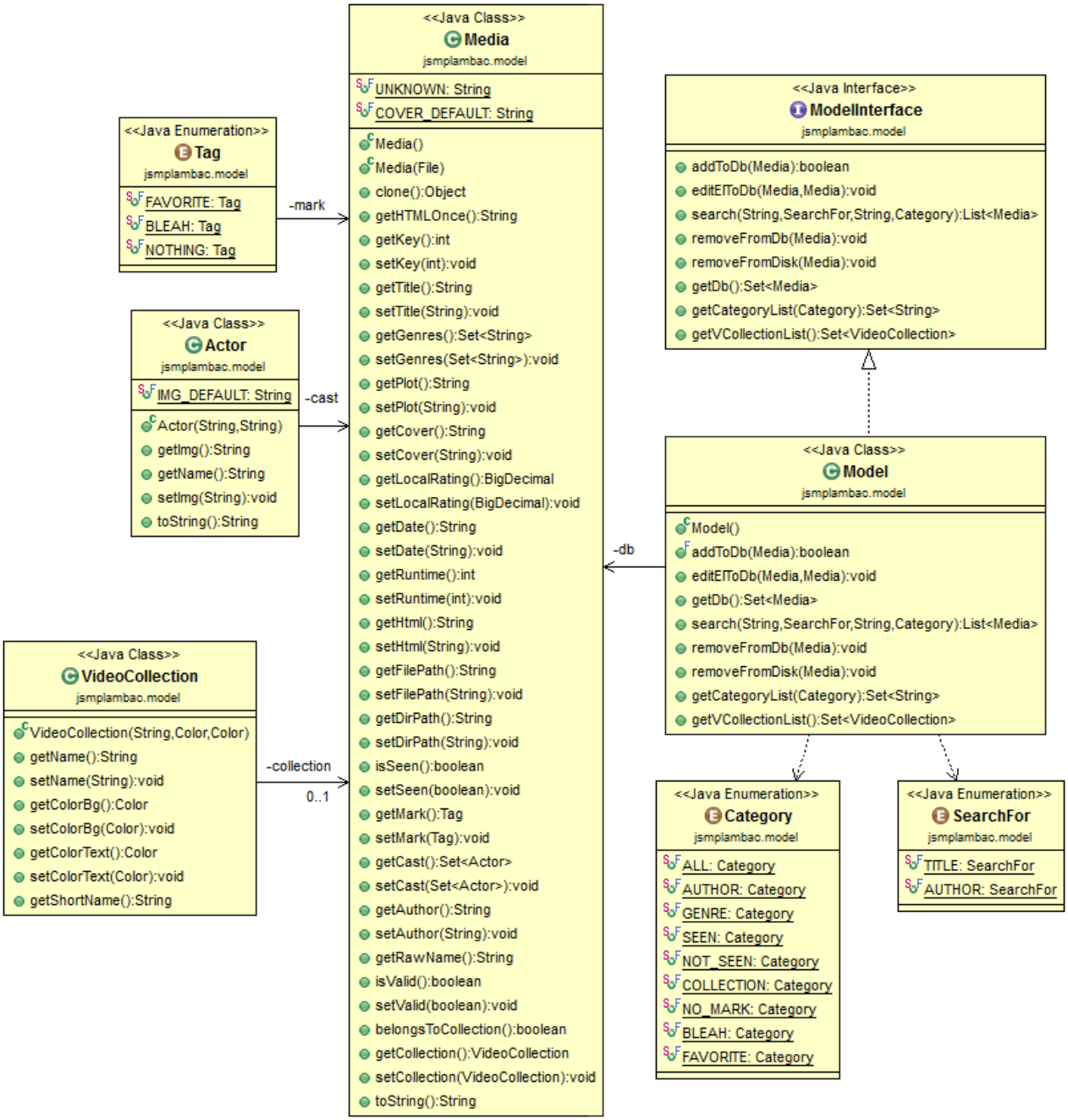
Dati i requisiti appena descritti lo sviluppo dell'applicazione viene effettuato utilizzando un'interfaccia grafica basata sulle librerie *SWING* che si presenta fondamentalmente come una lista di video sui quali è possibile effettuare right-click per svolgere le azioni sopra citate ed una main toolbar dalla quale è possibile effettuare quelle azioni generali che non interessano il singolo video.

Progettazione architetturale: lo sviluppo del sistema è stato effettuato utilizzando in maniera pervasiva il pattern *MVC* (*Model – View – Controller*) il quale si basa su una netta distinzione delle responsabilità tra i tre "componenti". Il *Model* si occupa della gestione delle strutture dati necessarie e dell'implementazione delle operazioni su di esse. È la parte centrale e fondamentale dell'applicazione che però rimane invisibile all'utente. La *View* fornisce una modalità di interazione visuale, intuitiva ed immediata con l'applicazione da parte dell'utilizzatore. È anch'essa una componente fondamentale in quanto la semplicità d'uso di un applicativo influisce in modo molto rilevante sul suo effettivo utilizzo e diffusione. Il *Controller* lega i primi due componenti. Si occupa di trasformare gli eventi ricevuti dalla *View* in operazioni effettuate sul *Model* aggiornando eventualmente la *View* dopo una modifica. Il disaccoppiamento garantito dall'utilizzo di questo pattern consente facile manutenzione e la possibilità di "sostituire" i vari componenti senza dover stravolgere il codice.

Segue ora il diagramma *UML* delle parti più rilevanti: (se non si riescono a capire gli schemi, le immagini sono allegate a parte!)

UML relativo al pattern *MVC*

Model: diagramma *UML* delle principali classi relative al *Model*.



Descrizione degli aspetti principali

ModelInterface e Model: interfaccia ed implementazione dell'insieme dei dati usati dal sistema (secondo il pattern *MVC*). Il *Model* implementa l'interfaccia **Serializable** per garantire il salvataggio su disco dei dati e quindi la loro persistenza.

Media: classe che modella un generico *Media* ovvero un generico file video su cui è basato l'intero programma. La classe oltre ad implementare l'interfaccia **Serializable** per la persistenza dei dati implementa anche l'interfaccia **Clonable** per garantire la clonazione di questi oggetti affinché si possano eseguire determinate operazioni da alcune classi del programma.

Actor: classe che modella un generico attore. Esso è caratterizzato da nome e immagine e viene

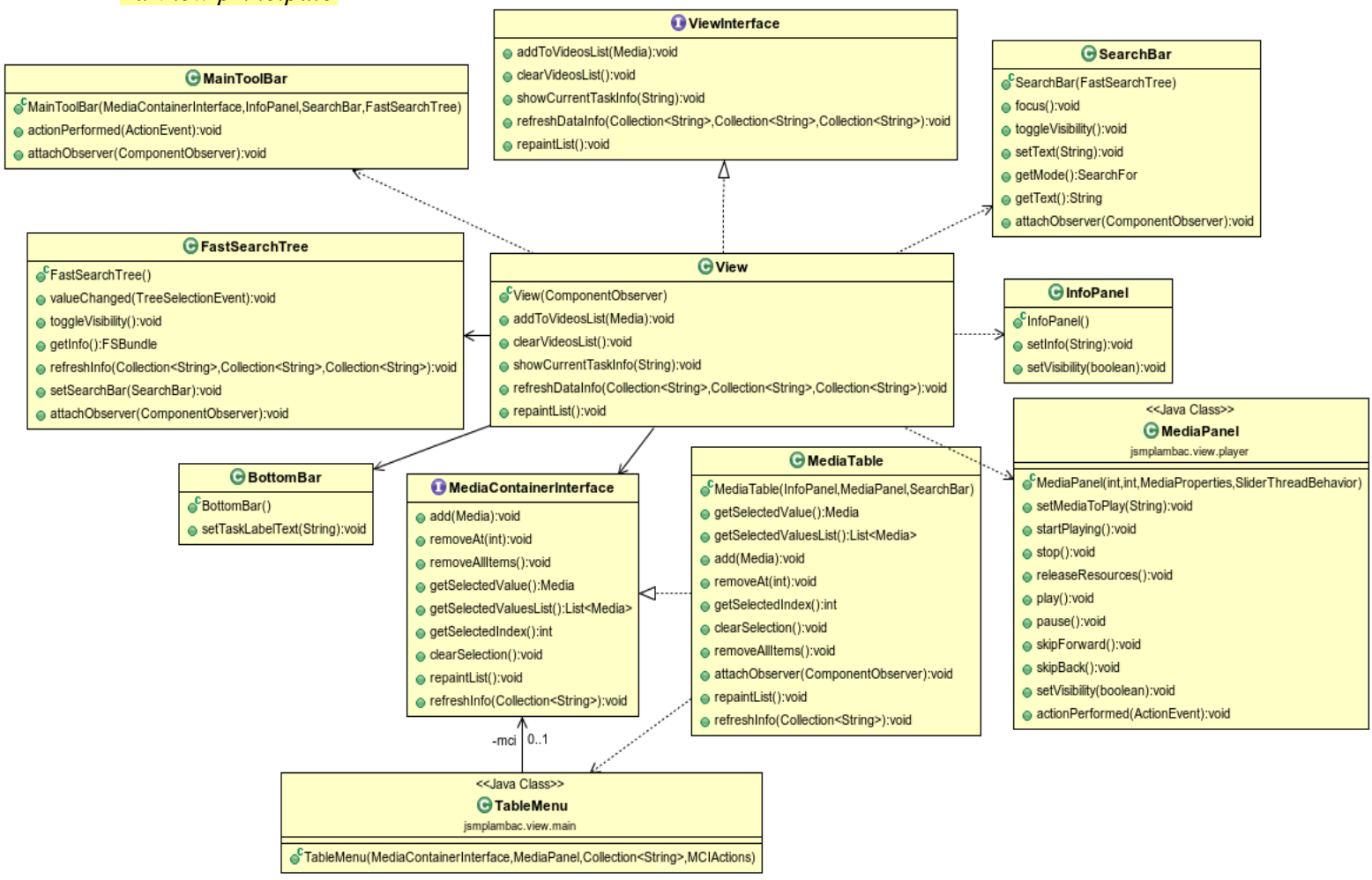
usato per la costruzione del cast di attori contenuti in *Media*. Anche essa implementa l'interfaccia **Serializable**.
VideoCollection: classe che modella il campo collection in *Media.* E' costituita da un nome, un colore per lo sfondo e un colore per il testo. Anche questa classe implementa **Serializable**.

Category e **SearchFor**: enumerazioni che vengono usate rispettivamente per le categorie per la ricerca ad albero e la barra di ricerca veloce. Vengono utilizzate all'interno del *Model* per personalizzare la ricerca.

Tag: altra enumerazione che definisce le opzioni che può avere il campo *mark* in *Media*.

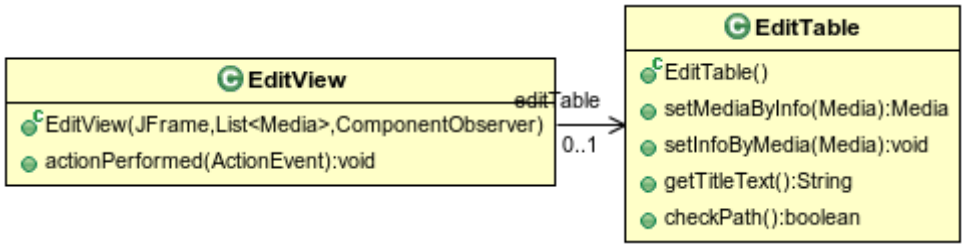
View: diagramma *UML* delle principali classi relative alla *View*.

La View principale



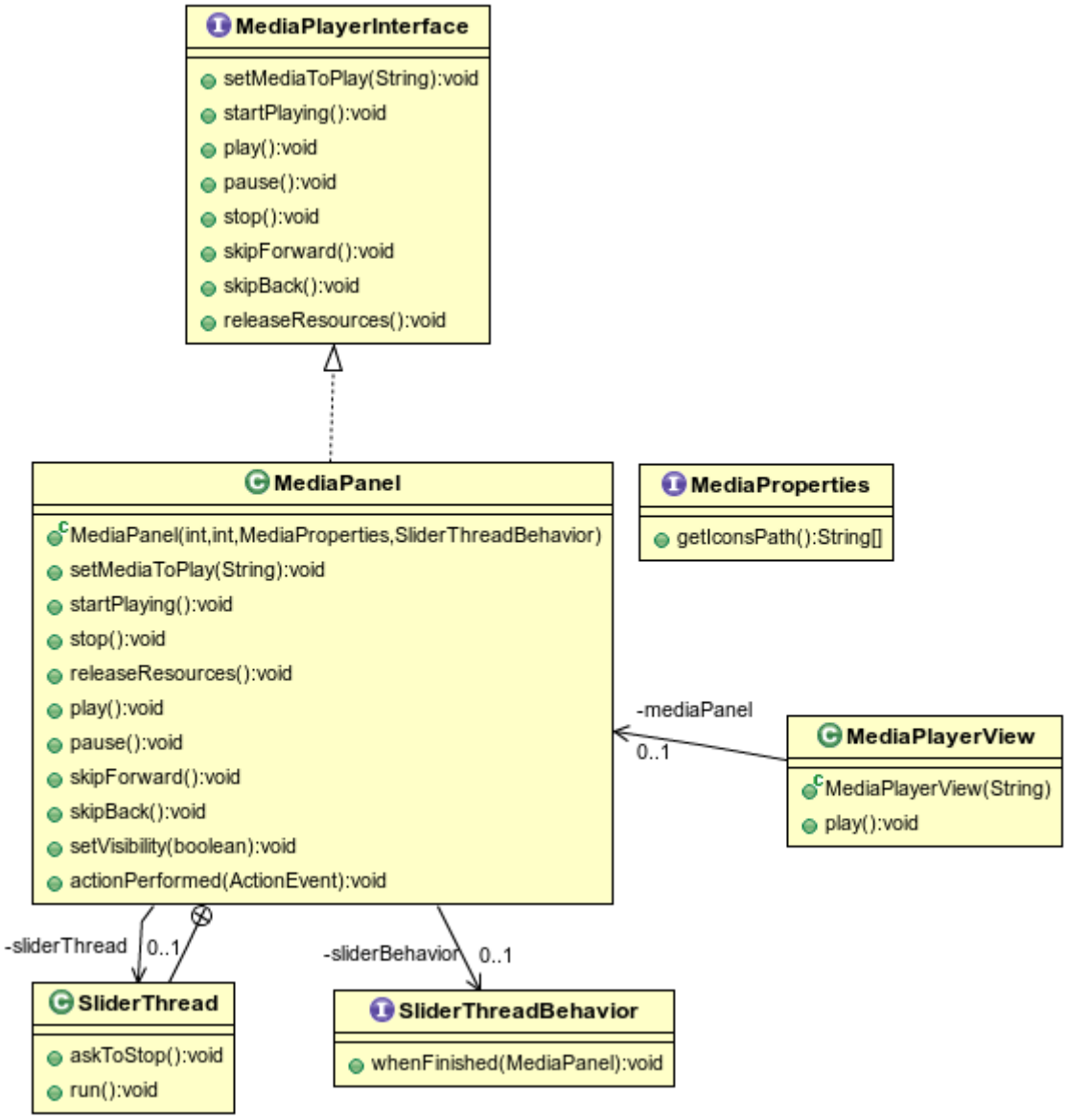
Il diagramma mette in evidenza le varie classi di cui si compone la view principale. Le frecce (con punta aperta) tratteggiate indicano che tali classi che non sono campi ma variabili locali. Per pulizia di presentazione questo diagramma non mostra tutte le relazioni che i vari componenti hanno tra loro.

La Edit View

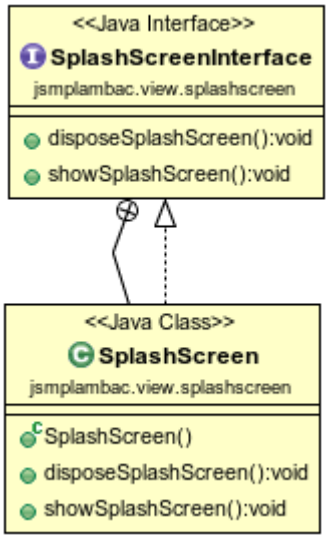


La EditView è quella schermata che permette all'utente di modificare alcune proprietà di una lista di Video (Media).

Il Media Player



Per completezza: lo SplashScreen



Descrizione degli aspetti principali:

- dalla signature dei metodi di molte classi sopra presentate si nota che contengono il metodo *attachObserver* (proveniente dall'interfaccia *ObservableComponent* (vedi organizzazione in package **jsmplambac.view.main** qua sotto)) che permette il collegamento di un oggetto *ComponentObserver*. Nel nostro caso viene collegato il *Controller* in quanto queste classi hanno la necessità di interagire con esso e quindi indirettamente con il *Model;*
- l'interfaccia grafica con la quale l'utente può interagire si compone fondamentalmente di tre finestre distinte: la view principale, la edit view e un embedded media player molto semplice;
- la view principale è una composizione di molte classi ciascuna delle quali è una estensione ad-hoc di un componente SWING esistente. Una breve descrizione di ciascuna di esse è presente nella sezione organizzazione in package;
- la edit view è accessibile all'utente attraverso il menu a tendina che compare cliccando col destro su un insieme di video. Questo è descritto dalla classe *TableMenu* della quale si ha un oggetto in *MediaTable*. I listener per le azioni di TableMenu sono presenti nel package **jsmplambac.listeners** classe *MCIActions* e sono descritti attraverso un'insieme di classi innestate non statiche;
- l'embedded media player e la *View* principale (la quale mette a disposizione anche la funzionalità di preview) utilizzano per la visualizzazione dei video la classe *MediaPanel* la quale incapsula un oggetto della classe *EmbeddedMediaPlayerComponent* delle librerie *vlcj*.

Controller: di cui lo schema UML è sopra riportato.
Descrizione degli aspetti principali
La classe *Controller* implementa l'interfaccia *ComponentObserver* la quale definisce le operazioni necessarie alla comunicazione tra *View* e *Controller* e di conseguenza *View* e *Model*.

Organizzazione in package: descrizione dei package e delle classi dei package principali.

- **jsmplambac.control** contiene i sorgenti che incapsulano il comportamento del controller. Esso ha il compito di inizializzare l'applicazione, ricevere gli eventi dalla *View* modificando il *Model* ed riaggiornando la prima.
- **jsmplambac.listeners** contiene i sorgenti che definiscono il comportamento di alcuni listener (come ad esempio quelli riguardanti il context menu che compare right-cliccando su un insieme di video) utilizzati nella parte grafica dell'applicazione. Sono stati organizzati in un package a parte per pulizia di codice.
- **jsmplambac.logger** contiene i sorgenti che incapsulano il comportamento del logger (realizzato tramite pattern *singleton*).

- **jsmplambac.main** entry point dell'applicazione.
- **jsmplambac.model** contiene i sorgenti che implementano la parte del *Model;*
- **jsmplambac.onlineinfo** contiene le classi che gestiscono le informazioni provenienti dal *Web;*
- **jsmplambac.task** contiene i sorgenti che definiscono i background task. Questo package è una "estensione" del *Controller.*

BackgroundTask Interface	Interfaccia che descrive il contratto che un compito a lungo termine deve soddisfare.
Abstract BackgroundTask	Implementazione di *BackgroundTaskInterface* del quale definisce tutto tranne il metodo *run.*
AAVDaemon	AutoAddVideoDaemon, estensione di *AbstractBackgroundTask* che definisce l'operazione di ricerca ed aggiunta automatica dei file video data una root directory fino alle foglie.
CopyDaemon	Estensione di *AbstractBackgroundTask.* Definisce l'operazione di copia di un video presente nello storage.
MoveDaemon	Estensione di *AbstractBackgroundTask.* Definisce l'operazione di movimento di un video presente nello storage.
Connectivity Daemon	Estensione di *AbstractBackgroundTask.* Esso si occupa di controllare lo stato della connessione. È implementato tramite pattern *singleton* in modo da essere accessibile da qualsiasi punto dell'applicazione.
BulkRemoveTask	Estensione di *AbstractBackgroundTask.* Definisce l'operazione di eliminazione di un insieme di video presenti nello storage.
Background TasksManager	Definisce le modalità di esecuzione di un oggetto che implementa *BackgroundTaskInterface.* È l'executor dei vari task. I task vengono organizzati in una coda ed eseguiti sequenzialmente (questo non è valido per *ConnectivityDaemon* il quale gira in background sempre). È implementato tramite pattern *singleton*.

- **jsmplambac.utilities** contiene classi di utilità. Al momento solamente una classe (modulo) contenente metodi statici per l'identificazione dei file video gestiti.
- **jsmplambac.view.edit** contiene i sorgenti relativi alla edit view, la schermata che permette all'utente di modificare alcune proprietà di un video memorizzato nello storage.
- **jsmplambac.view.main** contiene i sorgenti relativi alla view principale che mostra all'utente la lista dei video e le varie azioni che è possibile compiere.

ViewInterface	Interfaccia che descrive le azioni che una view deve essere in grado di svolgere. È la parte *View* del pattern *MVC*.
View	Implementazione di *ViewInterface* tramite *JFrame*. Principale componente con il quale l'utente interagisce. Si compone di una toolbar, una barra di ricerca, una tabella, pannelli per la ricerca rapida, la visualizzazione di informazioni e la preview di un determinato video scelto dalla tabella ed una bottom bar per alcune informazioni utili durante l'esecuzione dell'applicazione.
MediaContainer Interface	Interfaccia che descrive il contratto di un generico contenitore di video.
MediaTable	Implementazione di *MediaContainerInterface* attraverso JTable.
TableMenu	Descrive il popup menu che si ottiene cliccando col destro su uno o più video selezionati nell'implementazione di *MediaContainerInterface*. La definizione

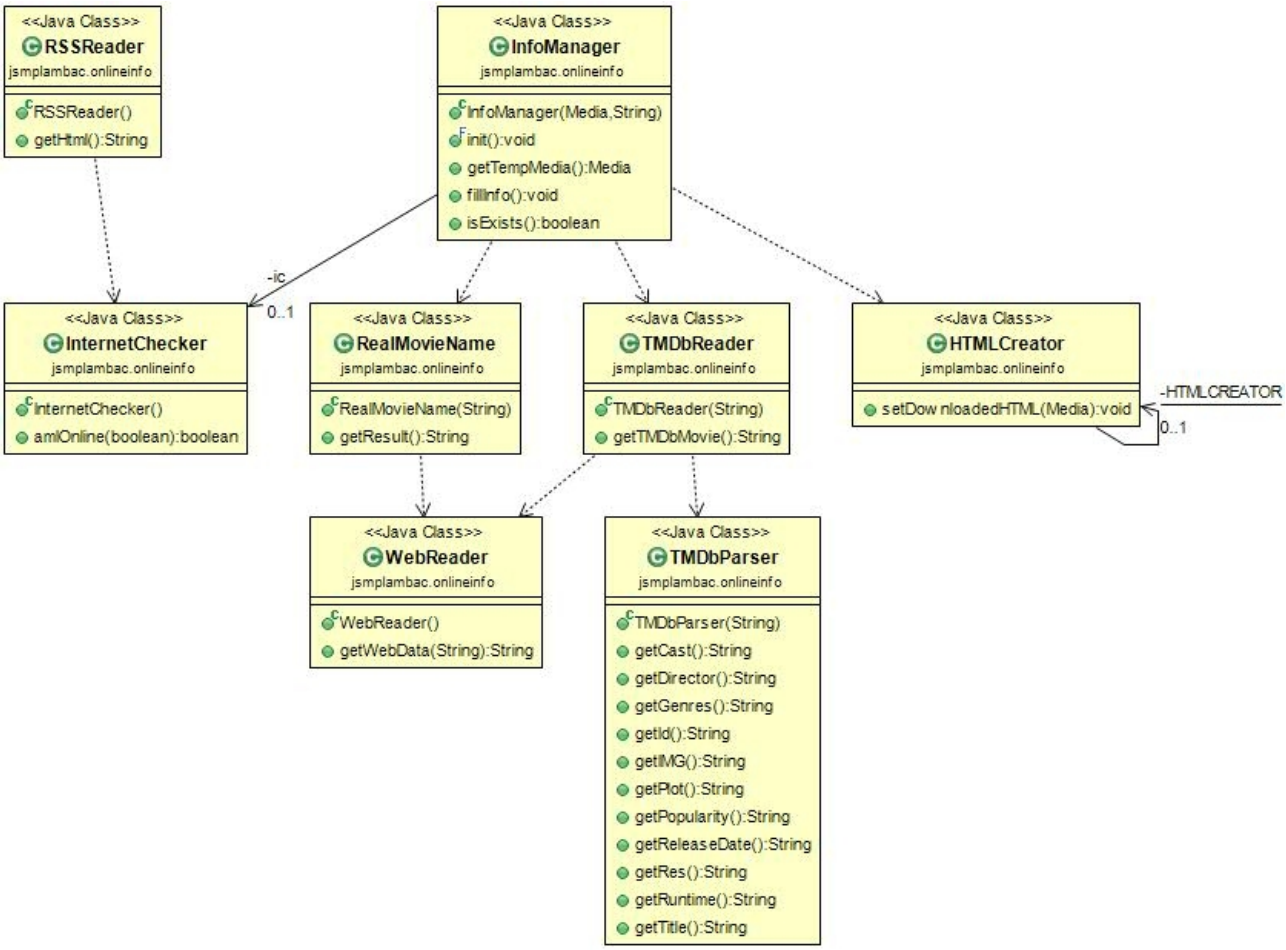
	dei listener associati ai vari menuItem è nel package **jsmplambac.listeners** nella classe *MCIActions*.
MediaTableTextRenderer	Definisce come debba essere disegnata la colonna "Name" della *MediaTable.*
MediaTableRaterRenderer	Definisce come debba essere disegnata la colonna "Rating" della *MediaTable.*
MediaTableCellBackgroundRender	Definisce come debba essere disegnata la colonna riguardante le collezioni.
AbstractMediaTableTagRenderer	Definisce come debbano essere disegnate le colonne "Mark" e "Seen" della *MediaTable.*
SearchBar	Descrive il layout ed il comportamento della barra di ricerca, la quale permette all'utente di ricercare i video all'interno dello storage per titolo o autore.
MainToolBar	Descrive il layout ed il comportamento della toolbar principale, la quale contiene alcune azioni generali (non sul singolo film).
InfoPanel	Descrive il layout ed il comportamento della pannello delle informazioni. Esso mostra all'utente i dati presi dal web/locali attraverso un *JTextPane*.
FastSearchTree	Descrive il layout ed il comportamento del pannello per la ricerca rapida. Esso permette all'utente di accedere con un click ai video secondo predeterminate proprietà. Quando gli oggetti di *FastSearchTree* e *SearchBar* sono "attivi" contemporaneamente i risultati vengono intersecati.
BottomBar	Descrive il layout ed il comportamento della barra inferiore, la quale permette di visualizzare alcune utili informazioni come: cosa sta succedendo nell'applicazione, lo stato della connettività, ecc.
LogObserver	Interfaccia che definisce il contratto che deve soddisfare un componente che osserva le modifiche al log del package **jsmplambac.logger** e le notifica all'utente.
ComponentObserver	Interfaccia che definisce le operazioni che il *Controller* del pattern MVC può ricevere dalla *View*.
ObservableComponent	Interfaccia che permette ad un componente della *View* di "agganciare" un riferimento al *Controller.* Le classi *ObservableComponent* e *ComponentObserver* mettono insieme il meccanismo del pattern *Observer* dove il *Controller* è il listener e i vari componenti sono i sources.

- **jsmplambac.view.factory** contiene i sorgenti delle classi che implementano la logica di factory per i componenti delle diverse parti grafiche dell'applicazione.
- **jsmplambac.view.player** contiene i sorgenti delle classi che implementano il riproduttore multimediale basato sulle librerie *vlcj*.
- **jsmplambac.view.splashscreen** contiene i sorgenti delle classi che definiscono lo splashscreen visualizzato all'avvio dell'applicazione durante il caricamento delle risorse.

Suddivisione del lavoro

Parte di Antony Chiossi

La parte di applicazione da me sviluppata è composta principalmente da classi che si occupano di reperire e gestire qualsiasi informazione non locale. La classe primaria è *InfoManager* la quale gestisce richieste eterogenee. Queste differenti richieste sono dovute dal fatto che l'utente interagendo con il programma può effettuare operazioni che richiedono o meno relazioni con il modello.



I package da me sviluppati sono:

- **jsmplambac.onlineinfo:** come detto in precedenza questo package acquisisce informazioni da siti web di terzi ed elabora le informazioni ottenute per renderle compatibili ed integrabili nella nostra applicazione.

    - **InfoManager:** sostanzialmente si occupa, una volta ricevuto in input un *Media* e un nome di un film (il quale può anche essere slegato dal *Media*), di clonare il *Media* e settare le informazioni (reperite da due classi: *TMDbReader*, *TMDbParser*) riguardanti l'input in un *Media* temporaneo. Così facendo si è in grado sia di agire direttamente sul *Media* originale, grazie ad un metodo che copia le informazioni nel *Media* esistente, sia indirettamente in quanto è possibile accedere al *Media* temporaneo. Quest'ultima caratteristica per esempio è usata in *EditView* e *MainToolBar* dove è necessario reperire informazioni che potrebbero o devono essere volatili, in quanto l'utente potrebbe decidere di non associare le informazioni acquisite ad alcun *Media* oppure in caso di ricerca di un film da *Online info* queste non devono essere associate a nessun *Media.*

- **TMDbReader:** è una classe finalizzata alla lettura (tramite un *WebReader*) di più pagine web contenenti informazioni riguardanti il film dato in input. Una volta acquisite tutte le informazioni è in grado di selezionarle (tramite *TMDbParser*) ed aggregarle in una stringa che sarà di facile lettura per l'utilizzatore finale.

- **TMDbParser:** è una classe dedicata all'estrapolazione, da una stringa in formato JSON (letta da *TMDbReader),* delle informazioni riguardanti i *Media* utili al programma.

- **RealMovieName:** ha come fine quello di cercare il nome del film per poi determinare se questo esiste o meno. Per fare ciò pulisce il nome da parole non utili ai fini della ricerca e tenta di trovare un nome che assomigli il più possibile a quello fornito in input. Se ciò accade lo ritorna, altrimenti restituise una versione semplificata del titolo.

- **HTMLCreator:** implementato tramite pattern *singleton* si occupa, preso un *Media* in input, di settare a quest'ultimo il suo campo html che servirà per avere una visione grafica e schematica delle sue informazioni nel *InfoPanel*.

- **RSSReader:** si occupa di leggere feed RSS, contenenti informazioni sui film attualmente in proiezione, da un sito di terzi per poi fornirne una versione abbellita graficamente nel *InfoPanel*.

- **1** – *PanelInfo*: utilizzato per mostrare graficamente il campo di qualsiasi *Media*.

- **2** – *News*: bottone che se premuto mostra nel *PanelInfo* i film che sono attualmente nelle sale cinematografiche. Per ora solo in provincia di Bologna.

- **3** – *Online Info*: se premuto chiede tramite input il nome di un film del quale cercherà le

informazioni e le mostrerà nel *PanelInfo*.

- **4** – *Edit Info*: quando premuto comparirà una nuova finestra che mostrerà ogni campo del *Media*. E' disponibile anche un bottone per effettuare il download delle informazioni da web, queste verranno poi immesse negli opportuni campi della nuova finestra. Una volta completata questa operazione l'utente potrà decidere di modificare e salvare le informazioni oppure di chiudere la finestra, passare ad un altro film, ripristinare le vecchie informazioni senza alterare il *Media* originale.

- **5** – *AutoUp info*: se premuto effettua un auto-aggiornamento del/dei *Media* selezionati. Molto utile nel caso in cui l'utente possegga molti film, infatti selezionati tutti i film (CTRL+A) e cliccato con il tasto destro del mouse su un film qualsiasi si è in grado di reperire e salvare le informazioni di ciascun *Media* in poco tempo.

- **6** – *Double click*: premendo due volte con il click sinistro del mouse su una riga della tabella si aprirà il *PanelInfo* con le informazioni riguardanti il film al suo interno.

Per tutte queste funzioni è ovviamente necessaria la connessione ad Internet.

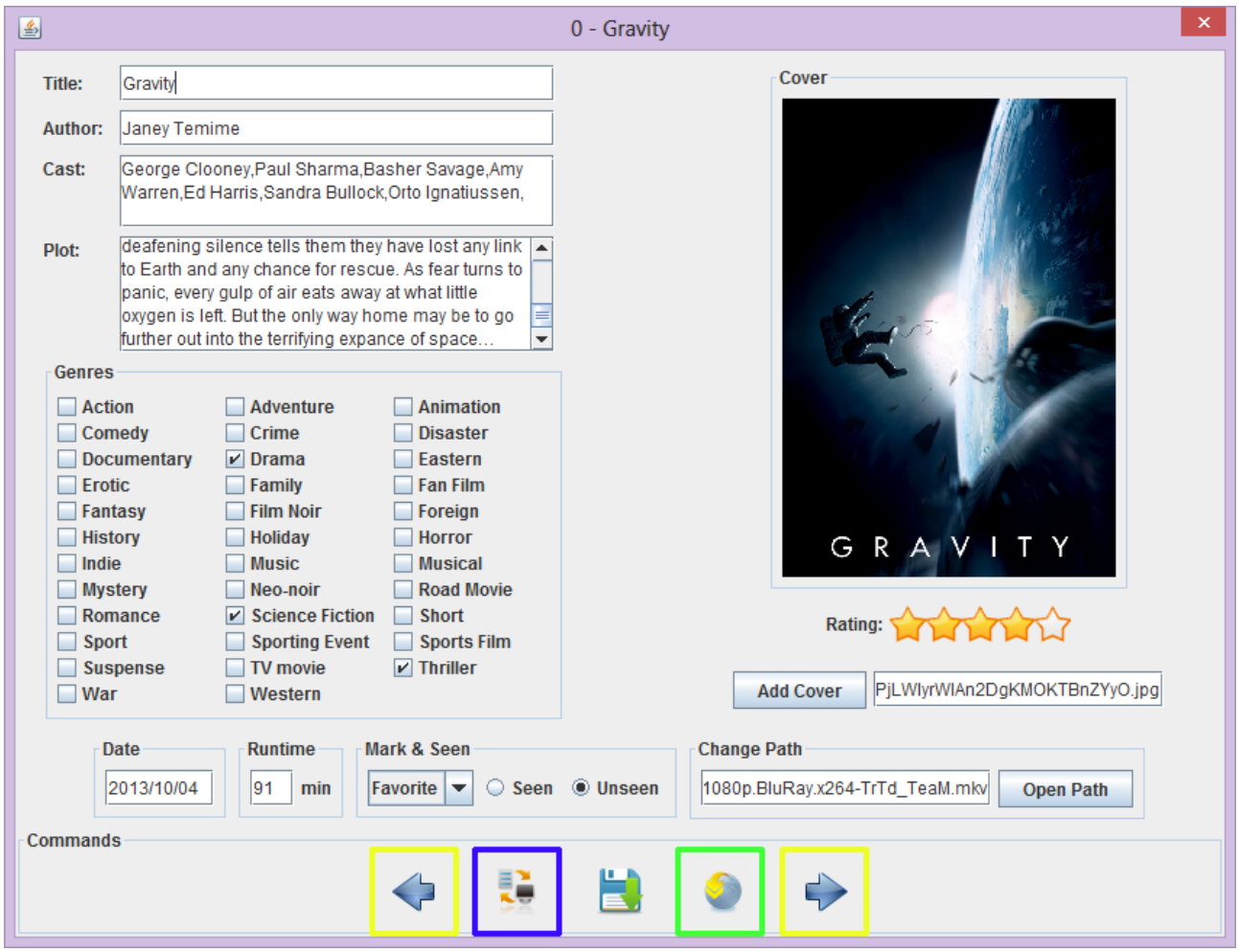
Librerie esterne utilizzate:
- jdome
- rome

Spunto da questo articolo.

Parte di Michele Buzzoni

La parte che ho dovuto sviluppare io si concentra principalmente sulla creazione del *Model*, della struttura dati e quindi di tutte le operazioni su di essa.
I package da me sviluppati sono:

- **jsmplambac.model:** come spiegato prima, questo è il package del *Model* relativo al pattern *MVC*. La classe *Media* definisce un generico file video. Essa è composta da 2 costruttori, uno generico senza parametri che crea un video "vuoto" ovvero con tutti i campi di default mentre un altro che prende come parametro un file dal quale ottiene alcune informazioni come il percorso e il suo nome. La classe è costituita principalmente da metodi getter e setter per il reperimento dei campi privati molto usati all'interno di altre classi. La classe *Media* contiene anche un campo collection, oggetto della classe *VideoCollection*, che identifica a che collezione fa parte (come trilogie, sequeal, ecc...). Se il video non appartiene a nessuna collection la nostra scelta implementativa è stata quella di riempire questo campo con una collection con nome nullo.
  La classe *Model* invece ospita la struttura dati: questo punto è stato più volte discusso all'interno del gruppo. Le opzioni da noi prese in considerazione sono state il *Set* e la *Map*. Utilizzando un Set ci siamo resi conto che le operazioni di modifica sul Media risultavano difficili da implementare. Abbiamo quindi optato per una mappa , una HashMap<Integer,Media> in cui la chiave è un intero progressivo che aumenta ogni volta che viene aggiunto un Media. Nel Media è anche presente un campo *key* che contiene la sua chiave in modo da rendere tutte le operazioni più veloci. Nel *Model* sono contenute le operazioni principali sul "*database*" come l'aggiunta, la rimozione, la modifica e la ricerca di media.

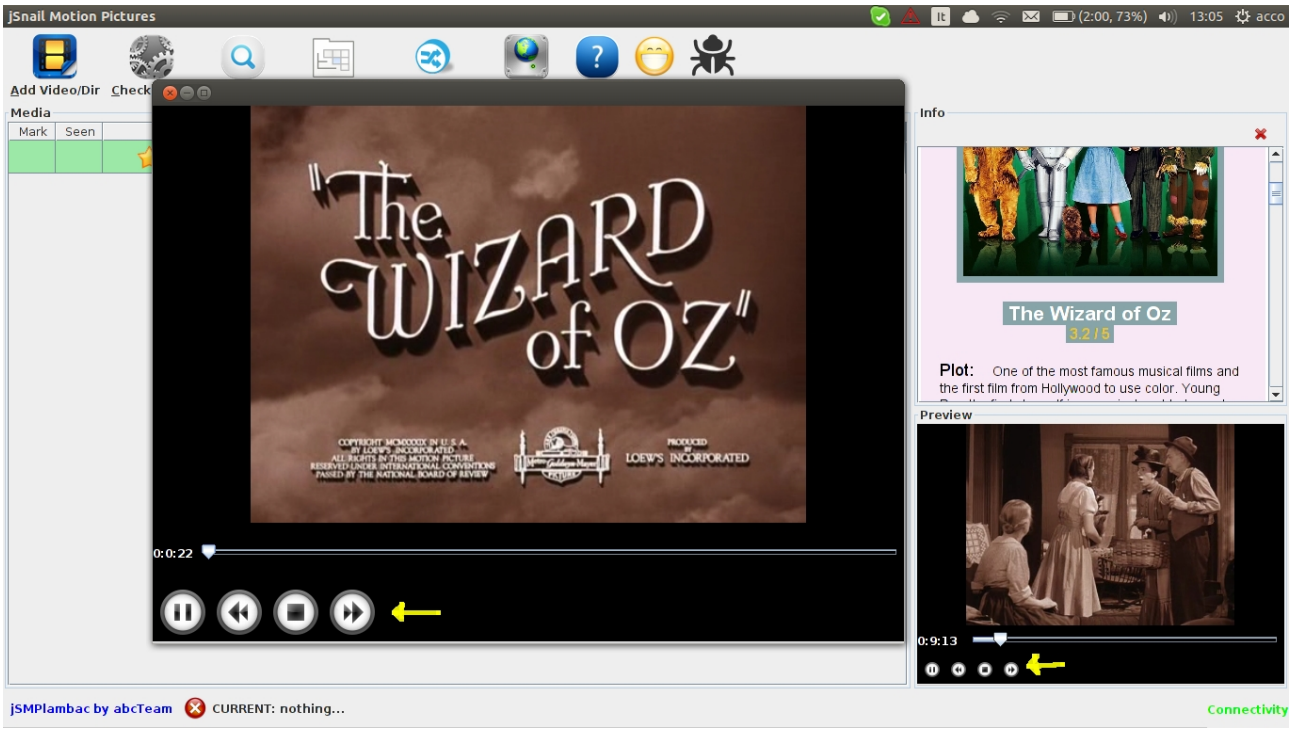


- **jsmplambac.view.edit:** Questo package si occupa della creazione e gestione della finestra di modifica di un media. E' composto da 2 classi: *EditView*, estensione di JDialog, il quale a sua volta è formata da 2 pannelli. Uno di questi è istanza della classe *EditTable* che estende appunto un Jpanel e si occupa del settaggio e reperimento delle informazioni dal media. In questa classe infatti vi sono 2 metodi molto importanti: *setMediaByInfo* che si occupa di prendere le informazioni da ogni JComponent per poi inserirli in un Media. *setInfoByMedia* invece si occupa di settare ogni JComponent con i rispettivi campi del media corrente.
  Questa finestra ha diverse funzionalità:
  - permette la modifica di più media. Nel caso il numero di media selezionati fosse maggiore di uno, attraverso i tasti direzionali è possibile scorrerli circolarmente per poterli modificare utilizzando una unica finestra.
  - Permette il ripristino delle impostazioni. Se non siamo soddisfatti delle modifiche apportate è possibile ripristinare le ultime informazioni salvate attraverso il pulsante "restore".
  - Permette la ricerca online delle informazioni. Attraverso l'apposito pulsante e il campo del titolo è possibile reperire le informazioni da Internet e tutti i campi vengono automaticamente aggiornati.

Parte di Luca Accorsi:
come definito all'inizio del percorso, la parte che ho dovuto sviluppare è stata principalmente l'interfaccia grafica per l'interazione con l'utente. In particolare i package da me sviluppati sono stati:

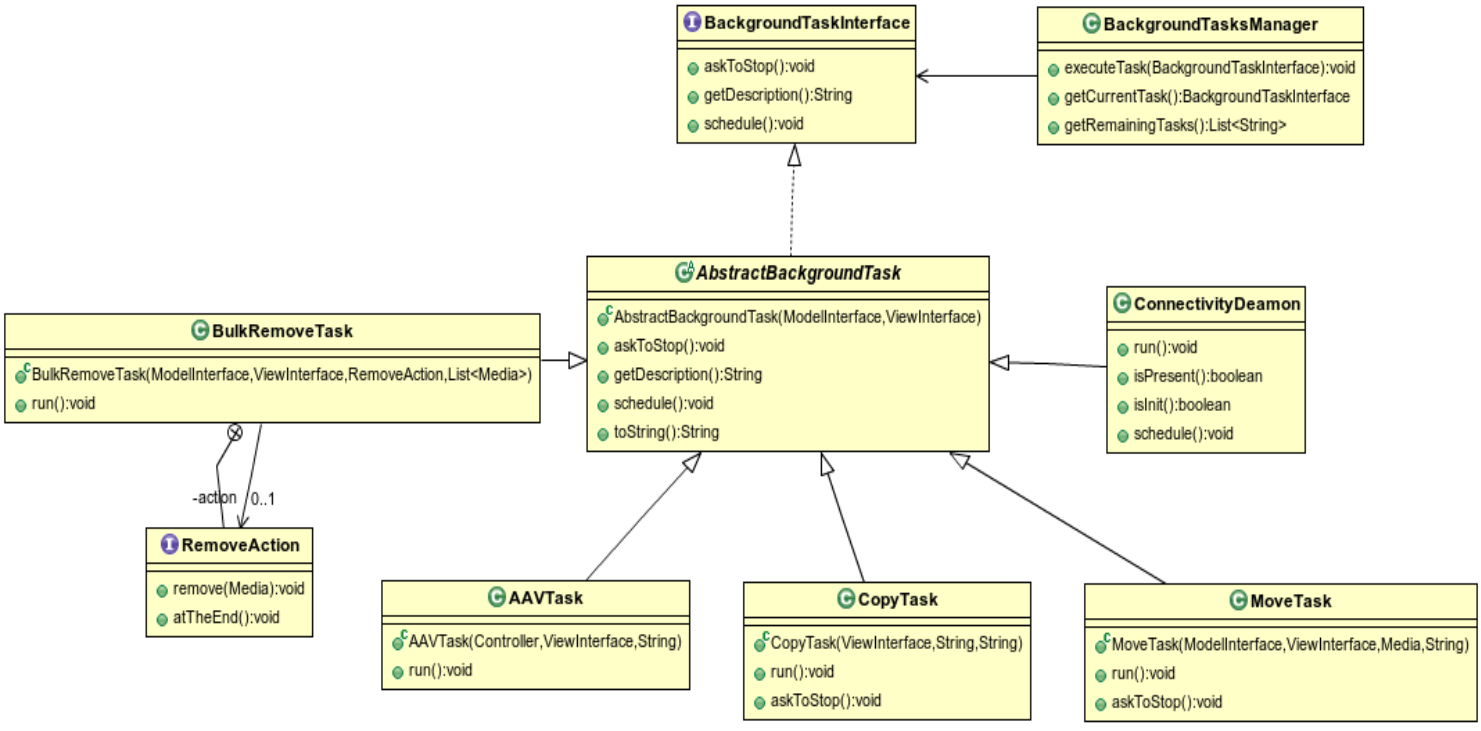
- **jsmplambac.view.main**: di cui il diagramma semplificato è sopra riportato. La classe fondamentale per il pattern *MVC* è l'interfaccia *ViewInterface* la quale mette a disposizione del controller i metodi necessari per l'aggiornamento della *View* a seguito delle modifiche sul *Model*. La classe concreta che implementa questi metodi è descritta da *View,* un'estensione di *JFrame*. Come è possibile notare dal Class Diagram *View* è una composizione di molte classi ciascuna delle quali modella un oggetto che incapsula il suo aspetto e le azioni che l'utente può eseguire su di esso. Per rendere l'interfaccia meno statica e più accattivante si sono venute a formare relazioni tra i vari componenti della *View* principale (ad esempio se voglio far nascondere il pannello di ricerca dalla toolbar principale un modo per farlo è facendo in modo che quest'ultima ne abbia un riferimento) che non illustrerò nel diagramma UML. Per quanto riguarda i *pattern* utilizzati in questo package si ha: *Template Method* utilizzato nella classe *AbstractMediaTableRenderer* nella quale il metodo *getTableCellRendererComponent* utilizza il metodo astratto *getIcon* che permette quindi la definizione di diverse icone a seconda del contesto. Possiamo vedere un'applicazione di tale renderer per il rendering delle prime due colonne della tabella le quali visualizzano effettivamente immagini diverse. Si ha poi il l'implementazione del pattern *Observer* per quanto riguarda le notifiche di eccezioni, da noi ritenute degne di essere riferite all'utente, avvenute nel programma. In questo caso abbiamo il *Logger* come sorgente al quale è stato agganciato un oggetto della classe *NotificationPopup* (del package **jsmplambac.factory)**, istanziata nel Main per essere subito pronta, che a fronte di messaggi di errore aggiunti notificherà graficamente all'utente.

- **jsmplambac.view.player**: di cui il diagramma è sopra riportato. Il componente principale di questo package è *MediaPanel,* un'implementazione di *MediaPlayerInterface* realizzata attraverso *JPanel. MediaPanel* permette la visualizzazione di un file video attraverso la libreria *vlcj*. Esso contiene all'interno una classe innestata non statica *SliderThread* la quale ha il compito di far avanzare il componente JSlider a seconda dell'avanzamento del video. Il comportamento dello *SliderThread* varia a seconda che sia impiegato nel pannello della preview o nell'embedded player. Questa differenza di comportamento viene definita attraverso un oggetto dell'interfaccia *SliderThreadBehavior,* quindi attraverso il pattern *Strategy*. Questo pattern viene utilizzato anche per la definizione delle icone attraverso l'implementazione dell'interfaccia *MediaProperties* che, nel caso di Embedded player avranno una dimensione diversa rispetto al pannello per la preview incastonato nella *View* principale (come mostra l'immagine sottostante);



Il pannello è lo stesso, cambiano le icone.

- **jsmplambac.task**:



I compiti a lungo termine, cioè che hanno una durata anche notevole, sono stati organizzati in un package a parte e vengono descritti dall'interfaccia *BackgroundTaskInterface* (la quale estende *Runnable*). La classe che implementa direttamente questa interfaccia è *AbstractBackgroundTask* la quale definisce tutti i metodi al di fuori di *run* che varia da task a task. La descrizione delle varie specializzazione è stata effettuata in **organizzazione in package**. Come si può notare dall'UML, sia

CopyTask che MoveTask ridefiniscono il metodo *askToStop* in quanto, per ora, le operazioni di copia e movimento siccome effettuate attraverso API di Java non è possibile interromperle durante l'esecuzione. I vari task vengono gestiti da *BackgroundTaskManager* il quale implementato tramite pattern *Singleton* si occupa di mantenerli in una coda ed eseguirli appena possibile. La gestione sequenziale dei thread è stata pensata soprattutto per semplificare l'implementazione della possibilità di arrestare i vari task durante la loro esecuzione. La classe *BulkRemoveTask* definisce alcune operazioni attraverso un oggetto che implementa l'interfaccia *RemoveAction,* si ha quindi l'utilizzo del pattern *Strategy*. Una estensione di *AbstractBackgroundTask* un po anomala è *ConnectivityDaemon* il quale, come suggerisce il nome, è un task che non viene schedulato come gli altri ma una volta creato rimane sempre attivo (fino alla richiesta di chiusura dell'applicazione). Si può notare come *schedule* sia stato ridefinito in modo da lanciare una *UnsupportedOperationException* se invocato. *ConnectivityDaemon* è implementato tramite pattern *Singleton* ed creato con *lazy inizialization* solamente se l'utente richiede una funzionalità che abbia a che fare con la connessione ad Internet.

Librerie esterne utilizzate:

- *vlcj (*capricasoftware*):* utilizzando l'installazione di vlc presente sul computer e le librerie *jna* e *platform* mette a disposizione classi e metodi per la realizzazione di riproduttori multimediali anche complessi;
- *rater:* modella il componente SWING per il rating.

Testing:

l'applicazione è stata testata in modo intensivo ed anche in scenari estremi e poco plausibili prendendo in considerazione diversi aspetti:

Il numero di film:

il programma è stato testato con un numero di video molto elevato (>100.000) per stimare la reattività dell'interfaccia utente a fronte di un carico di operazioni interne molto gravoso. Possiamo notare che:

- i tempi di inserimento aumentano in modo sensibile dopo i 50 mila film in quanto probabilmente aumenta la percentuale di collisione poiché si fa uso di una *HashMap*;
- durante l'inserimento, cancellazione, ecc (qualsiasi operazione che ha a che fare con il modello) si sono dovute disabilitare alcune operazione (ricerca, modifica proprietà, ecc) in quanto altrimenti si andava a modificare concorrentemente la struttura dati che modella lo storage;
- se si vuole analizzare una cartella contenente molti film il tempo chiaramente dipende dal numero di film presenti (e dalla *nota 1*);
- le operazioni che vanno ad aggiornare l'interfaccia grafica (ricerca, cancellazione, modifica, ecc) risultano essere abbastanza veloci (circa qualche secondo).

Input dell'utente:

il programma prevede la possibilità di personalizzare le informazioni relative ad un film attraverso componenti nei quali l'utente può scrivere liberamente. Per far fronte ad eventuali incongruenza tra i dati inseriti e quelli che ci vorrebbero sono stati effettuati opportuni controlli.

Problematiche possibili rilevate:

- **relative alla connessione Internet:**

se non si dispone di una connessione sufficientemente veloce è possibile che l'applicazione rilevi una mancanza di connessione oppure si potrebbe avere un rallentamento nel caricamento delle immagini prese da Internet.

Per motivi di tempo non è stato possibile effettuare il salvataggio della locandina di ogni film ciò comporta l'impossibilità di visualizzarla se la connessione non è disponibile.

- **relative al caricamento dei dati nei componenti *SWING:***

i componenti SWING per caricare un grosso quantitativo di dati impiegano diverso tempo (vedi funzione Random News), questo potrebbe influenzare la reattività della interfaccia grafica in quanto il caricamento delle info è comunque delegato all'EDT.

- **SO:**

si è rilevato che in Ubuntu 13.10 non viene rilevata la pressione del tasto CTRL. Quando si tenta di selezionare tutti i film della tabella con la pressione di CTRL+A viene erroneamente aperta la funzionalità di ricerca e gli items vengono deselezionati.

Un utilizzo medio dell'applicativo non comporta comunque particolari problemi per quanto riguarda reattività, velocità e prestazioni.

Note finali

Il processo di sviluppo è stato piuttosto lineare. In alcune occasioni si sono dovute prendere scelte diverse dall'iniziale fase di analisi. In particolare il progetto iniziale prevedeva di modellare i video in video singoli e in collezioni di video. Questo si sarebbe dovuto implementare attraverso il pattern *Composite* dove una entità *Video* avrebbe modellato il video singolo ed una *Collezione* un insieme di entità *Video.* Entrambe avrebbero dovuto estendere una entità *Media* in quanto sia un *Video* che una *Collezione* sono *Media*. Per rendere più semplice il tutto (anche per non sforare il tempo) è stato deciso di implementare il concetto di collezione come campo dell'entità *Video (*la quale nella nostra applicazione si chiamerà *Media)* a discapito anche delle prestazioni.

Per migliorare le prestazioni di alcune operazioni non sequenziali è stato introdotto in ultimo momento la suddivisione di alcuni compiti non sequenziali in un numero di thread basati sui processori disponibili.