Relazione progetto Programmazione ad Oggetti

Alessandro Bertolazzi

1227274

Sommario

Relazione del progetto "Biblioteca Virtuale" per il corso "Programmazione ad Oggetti". Un'applicazione scritta in C++ con framework Qt che permette la gestione di una biblioteca.

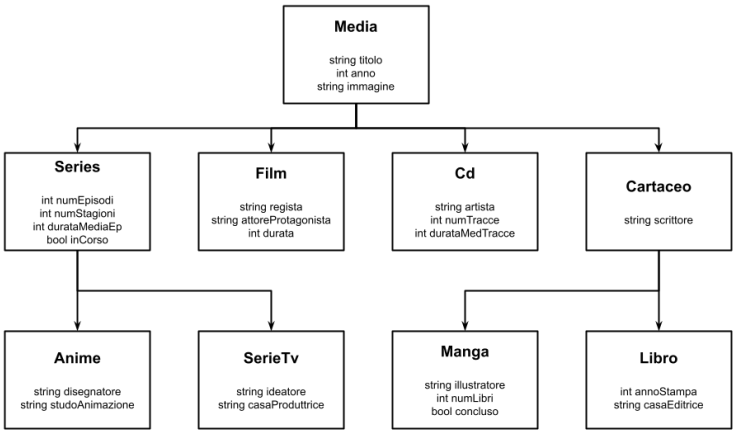
1 Introduzione

Bibliotheca procurator dal latino "direttore della biblioteca", il nome che ho deciso di dare all applicazione. Il progetto proponeva di creare un'applicazione che permettesse la gestione di una biblioteca. L'applicazione che ho deciso di creare non riguarda l'organizzazione di una biblioteca pubblica, bensì l'organizzazione di una biblioteca privata in modo da poter strutturare e catalogare libri, film, CD e simili per un privato con collezioni ampie. Mi sono dedicato questo obbiettivo poiché, oltre a poter avere un'utilità per me, ho notato che non si trovano prodotti simili online e, se si trovano, si tratta di applicazioni datate, a pagamento o non più mantenute.

2 Struttura del Progetto

2.1 Logic

Nella cartella `Logic` si trova tutta la parte logica dell'applicazione. Le specifiche del progetto richiedevano almeno 3 classi concrete per i progetti svolti singolarmente, ma per completezza ho deciso di farne 6: `Anime`, `SerieTv`, `Film`, `Cd`, `Manga` e `Libro`. Come classi astratte invece ho utilizzato `Media` come superclasse e ulteriori due classi `Series` e `Cartaceo`.



Struttura logica dell'applicazione

2.2 UI

Per quanto riguarda l'interfaccia grafica, `mainWindow` gestisce la finestra principale. Ho deciso poi di alleggerire relativamente il file creando file `Widget` che si occupano di generare parti singole dell'interfaccia. `topMenuWidget` va a generare la parte di pulsanti che si occupa della creazione, caricamento e chiusura della biblioteca e un bottone per la creazione di nuovi `Media`. `rightLayoutWidget` si occupa di gestire la parte destra del layout in cui vengono proposti tutti gli elementi media contenuti nella libreria. `createItemWidget` invece è la finestra che viene utilizzata a seguito del click del tasto "crea media" per creare un nuovo media. Questa finestra va a sostituire quello che viene inizialmente generato da `rightLayoutWidget` per poi, una volta confermata la creazione, nascondersi.

2.3 Service

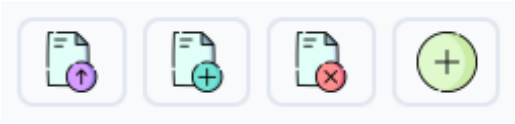
Nella parte `Service` ho cercato di inserire tutta la parte di codice non legata alla generazione della UI e alla logica dell'applicazione. Il file `jsonService` si occupa della gestione del file JSON per la persistenza dei dati, gestendo il caricamento e il salvataggio del file, conversione e aggiunta di un nuovo media e la rimozione. `styleUtils` può essere visto come un file CSS nella programmazione web: qui ho raggruppato tutta la parte che si occupa dello stile dell'applicazione (a volte ho lasciato la gestione dello stile direttamente nei file Widget). `uiService` si occupa invece del "popolamento" dell'interfaccia grafica. `mediaType` è un file che ho utilizzato per raggruppare i `dynamic_cast` presenti nel codice, permettendo una gestione più semplice e pulita dei vari tipi di media. `mediaService` è la parte che si occupa della gestione generale di tutti i servizi, visto un po' come file "mainService".

3 Persistenza dei Dati

Come metodologia di persistenza dei dati ho scelto di utilizzare un file JSON, scelta dettata innanzitutto dalla semplicità di utilizzo e dalla facile manutenibilità. Le modifiche che vengono fatte nell'interfaccia grafica, a seguito della conferma tramite pop-up, vengono scritte subito nel file JSON. La maggior parte del codice di gestione del file JSON è contenuta nel file `jsonService`.
Il codice JSON è così strutturato:

```
"media": [
    {
        "anno": 1986,
        "disegnatore": "Akira Toriyama",
        "durataMediaEp": 24,
        "immagine": "../resources/img/dragonball.jpg",
        "inCorso": false,
        "numEpisodi": 153,
        "numStagioni": 5,
        "studioAnimazione": "Toei Animation",
        "titolo": "Dragon Ball",
        "type": "Anime"
    }
]
```

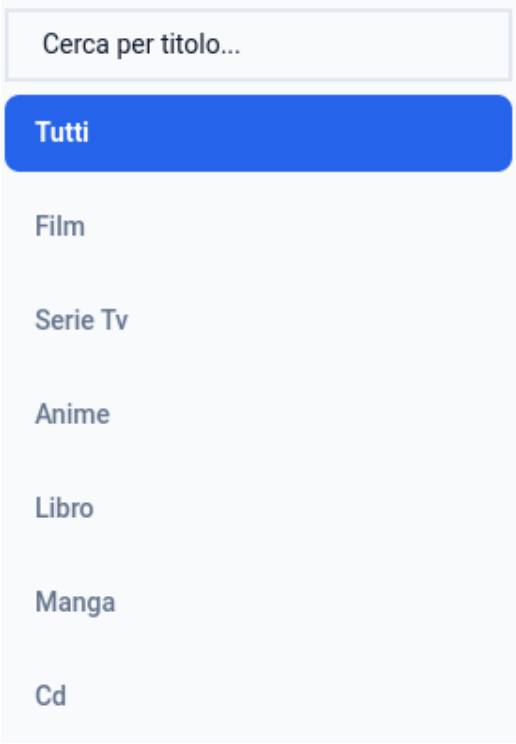
4 Interfaccia Grafica



Bottoni del `topMenuWidget`.
In ordine:

- Carica una biblioteca esistente
- Crea e apre una nuova biblioteca
- Chiudi la biblioteca corrente
- Crea un nuovo elemento media

Il secondo bottone, nel caso non sia stata caricata una biblioteca e siano già stati creati dei media, salva automaticamente i media in cache nella nuova biblioteca.



Nel `leftMenu` viene creata la barra di ricerca che, ad ogni cambiamento (ogni volta che viene digitato un carattere), cerca l'elemento nella biblioteca.
La ricerca è "case insensitive" e funziona anche con ricerca parziale. Sotto sono presenti i media categorizzati per tipologia.

Un esempio di elemento media: a sinistra viene caricata l'immagine, mentre a destra vengono visualizzati tutti i dettagli. All'estrema destra ci sono due bottoni: il primo permette di modificare il media, il secondo di cancellarlo dalla libreria. Entrambe le funzioni di modifica ed eliminazione, appena confermate, vanno a modificare direttamente il file JSON.
Alcuni miglioramenti che non ho implementato: una gestione migliore dello scaling dell'immagine ed un'eventuale ricerca automatica online, per alleggerire l'utente dal dover scaricare e gestire manualmente le immagini.

5 Polimorfismo

5.1 getSpecificDetails()

```
\\ Dichiarazione
    virtual std::vector<std::pair<string, string>> getSpecificDetails() const =
       0;

\\ Implementazione
    std::vector<std::pair<string, string>> Anime::getSpecificDetails() const {
    auto details = getSeriesBaseDetails(); // Eredita da Series
    details.insert(details.end(), {
        {"Disegnatore", disegnatore},
        {"Studio", studioAnimazione}
    });
    return details;
    }
```

La funzione `getSpecificDetails()` è una funzione virtuale pura definita nella classe astratta `Media` e ridefinita in ogni classe concreta. Questa funzione viene utilizzata per ottenere i dettagli specifici di ogni tipo di media, restituendo un vettore di coppie chiave-valore che rappresentano gli attributi specifici dell'istanza. Ad esempio, la classe `Anime` eredita da `Series` e aggiunge dettagli specifici come "Disegnatore" e "Studio". Ho scelto di utilizzare questo approccio contro all utilizzo di dinamyc_cast per ottenere i valori specifici di ogni media visto che mantiene il codice piú pulito e leggibile.

5.2 toJsonSpecific() / fromJsonSpecific()

```
\\ Dichiarazioni

    virtual QJsonObject toJsonSpecific() const = 0;
    virtual void fromJsonSpecific(const QJsonObject& json) = 0;

\\ implementazione

    QJsonObject SerieTv::toJsonSpecific() const {
    auto json = getSeriesBaseJson();
    json["ideatore"] = QString::fromStdString(ideatore);
    json["casaProduttrice"] = QString::fromStdString(casaProduttrice);
    return json;
    }
```

Ho deciso di implementare questo approccio dopo aver provato a gestire i vari tipi di media con dei dynamic_cast. La gestione dei vari tipi di media con i cast risultava comunque semplice ma rendeva il codice meno leggibile. Con questo approccio poi la serializzazione é "automatica" e non richiede di scrivere i campi specifici per ogni tipo di media. Eventuali modifiche o aggiunte possono essere gestite direttamente nella classe specifica.

5.3 Clone

```
\\ Dichiarazione

    virtual Media* clone() const = 0;
```

Implementazione del metodo clone senza essere a conoscenza del tipo concreto. Ogni classe specifica gestisce l' allocazione della memoria per il proprio oggetto e garantisce che tutti i dati vengano copiati correttamente.

5.4 getMediaType()

```
\\ Dichiarazione

    virtual string getMediaType() const = 0;
```

Implementando questa funzione ho potuto rimuovere tutti i dynamic_cast e typeid. Ho una chiamata diretta permettendomi di non utilizzare cast che in termini di computazione sono costosi e a rischio di errori. Anche qui ogni classe specifica gestisce il proprio tipo.

6 Rendiconto delle tempistiche

Quantificare le ore di lavoro richieste dal progetto risulta complicato, in quanto è stato sviluppato trovando il tempo tra lo studio e il lavoro. Escludendo lo studio personale necessario per comprendere il funzionamento di Qt e delle librerie utilizzate, e lo studio più approfondito del linguaggio C++, posso stimare circa 65 ore di sviluppo.

Jobs	**Scheduled hours**	**Actual hours**	**Notes**
Modellazione diagramma UML	1	2	Il tempo é stato sforato perché nel corso dello sviluppo del progetto sono state necessarie delle modifiche.
Bozza interfaccia Grafica	1	1	
Implementazione modello Logico	5	5	Implementazione veloce grazie al diagramma UML già deciso.
Implementazione Interfaccia grafica	15	20	
Implementazione parte Service	15	15	
Affinamento Progetto	5	10	Questo Job sarebbe da includere nella implementazione della parte di Service, ma visto che é stato lavorato anche su altro ho deciso di considerarlo a parte.
Debug	5	7	Quantificare il tempo di debug risulta difficile visto che il progetto é stato sviluppato per parti e dopo ogni modifica spesso risultava necessario sistemare qualcosa. Alcuni bug invece sono stati trovati con un uso piú attento dell applicazione.
Modifiche Finali	5	5	

7 Possibili Miglioramenti

- Implementare una ricerca online della copertina dei media.
- Attualmente Serie Tv, Anime e manga sono stati implementati pensando che l' utente sia in possesso di tutti i volumi o episodi di questi. Sarebbe da rendere possibile all'utente di inserire quali volumi o episodi é effettivamente in possesso.
- Implementare uno scaling e ridimensionamento delle immagini di copertina migliore.
- Implementare tema scuro.

8 Used Words

- Factory Method Pattern: Definizione da Wikipedia In programmazione, la "factory" o Factory Method Pattern è un design pattern creazionale che permette di creare oggetti senza specificare la classe esatta dell'oggetto da istanziare. Funziona definendo un'interfaccia per la creazione di oggetti, ma lasciando che le sottoclassi decidano quale oggetto specifico creare, delegando così la responsabilità della creazione. Ho utilizzato la parola factory dove ho utilizzato questo desing pattern, essendo una cosa che ho scoperto durante la creazione di questo progetto mi sono fatto aiutare molto da esempi o altri riferimenti online, non sono sicuro dell uso corretto di questo metodo.
- Media: viene inteso come qualsiasi elemento singolo che puó essere Cd, Manga, Serie Tv, Libro e Film..
- Cast: usata indicando gli static_cast ma principalmente in questo caso per i dynamic_cast.