Relazione progetto Programmazione a Oggetti

De Pasquale Mario

MAT.2076433

Media Library

1 INTRODUZIONE {

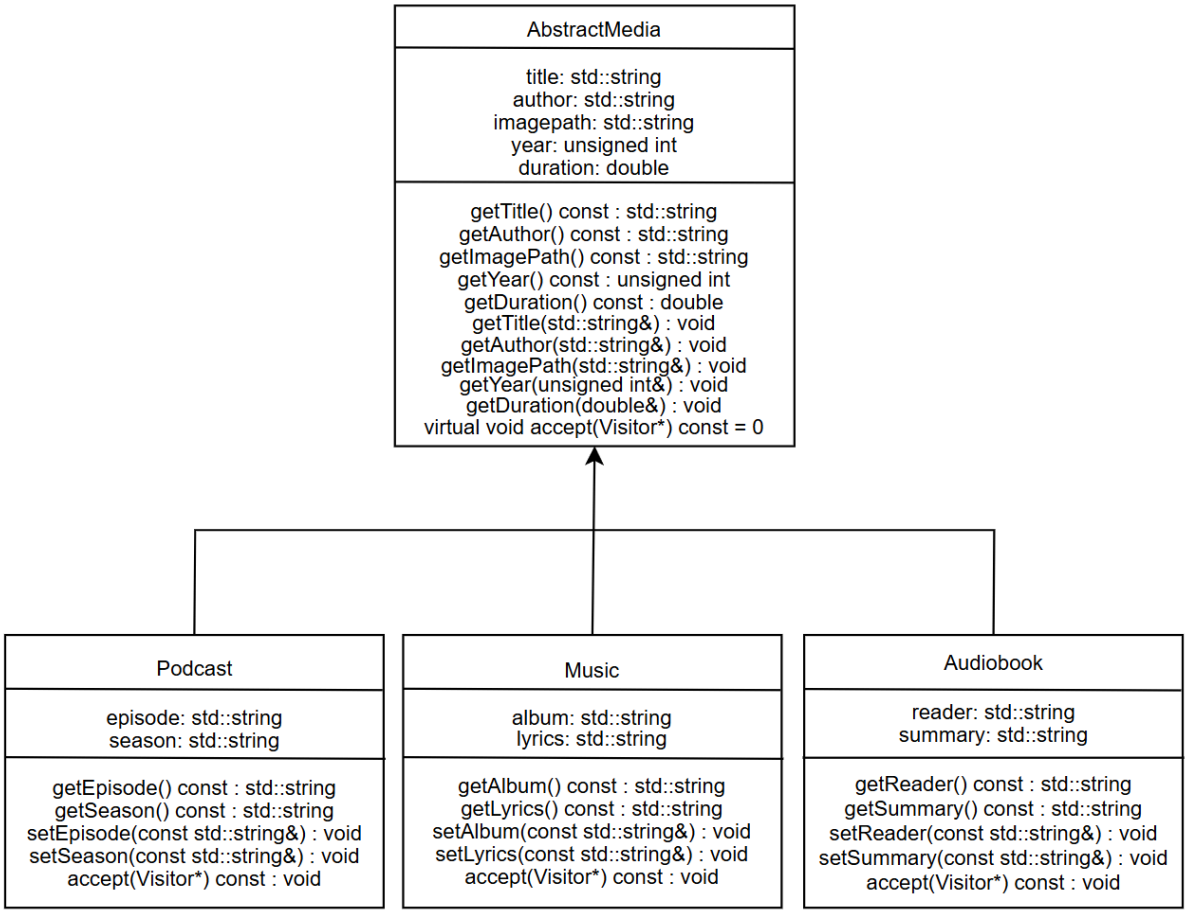
Media Library è una libreria multimediale progettata per salvare, modificare, cancellare, visualizzare i dettagli dei singoli media, oltre ad esportare o importare dati di salvataggio tramite file JSON.
I tipi di media attualmente gestibili sono tre: Audiolibri, musica e podcast; ognuno dei quali presenta delle caratteristiche distintive dagli altri: come per esempio il nome del lettore dell'audiolibro, l'album e testo delle canzoni e stagione ed episodio dei podcast.

L'interfaccia principale è composta da una grossa griglia per contenere la libreria e una barra di ricerca per cercare più velocemente i media.
In alto si ha un menubar, da cui è possibile eseguire azioni chiave come creare un nuovo media, caricare dati esistenti sul computer, oppure consultare le scorciatoie da tastiera.
Infine, in basso, è presente una statusbar dove il programma riuscirà a comunicare all'utente le operazioni eseguite.

}

2 DESCRIZIONE DEL MODELLO {

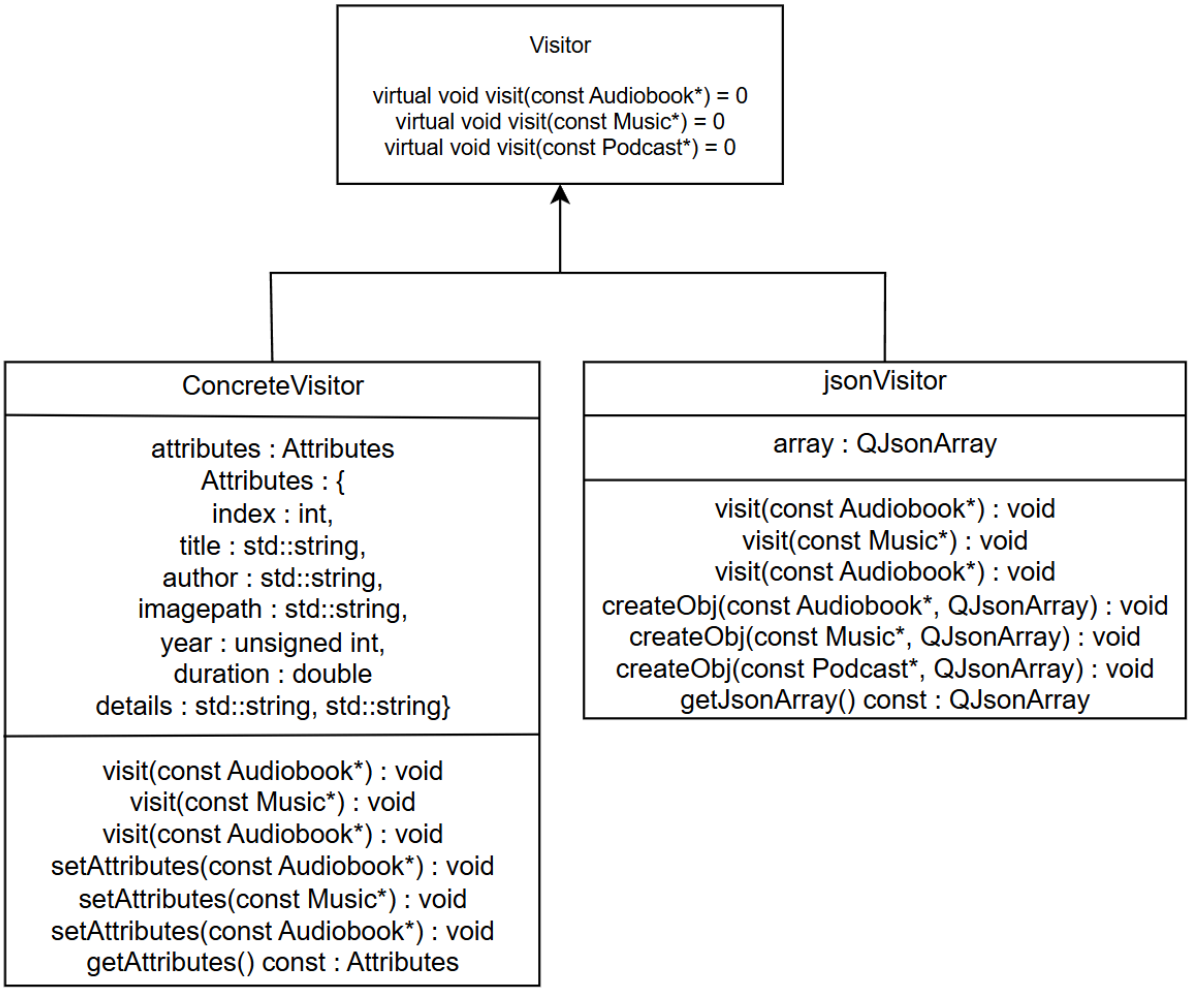
Il modello logico si articola in 4 parti fondamentali facilmente riconoscibili dalle cartelle in src del progetto: media(fig.1), visitor(fig.2), container e manager(fig.3).
Media e Visitor sono classi che sfruttano il polimorfismo, container e manager sono, invece, classi convenzionali. Media si occupa della definizione dei tipi di media, è presente la classe astratta `AbstractMedia` la quale si occupa di definire i componenti comuni dei media: titolo (che a tutti gli effetti e' un identificativo del media), autore, anno di uscita, durata e l'indirizzo dell'immagine.
Successivamente abbiamo 3 classi derivate: `Audiobook`, `Music`, `Podcast`; che definiscono i 3 tipi distinti di media, ognuno ha dei dati contraddistintivi. Audiobook ha il lettore e il riassunto del libro, entrambi definiti dal tipo `std::string`. Music ha l'album e il testo definiti dal tipo `std::string`. Podcast ha l'episodio e la stagione, anch'essi definiti dal tipo `std::string`.
In tutte le classi sono definiti i metodi get e set per ogni dato, inoltre è presente il metodo accept, dichiarato come virtuale puro nella classe base e definito nelle classi derivate, il quale permette di utilizzare il pattern visitor con il tipo di media specifico.



(fig.1)

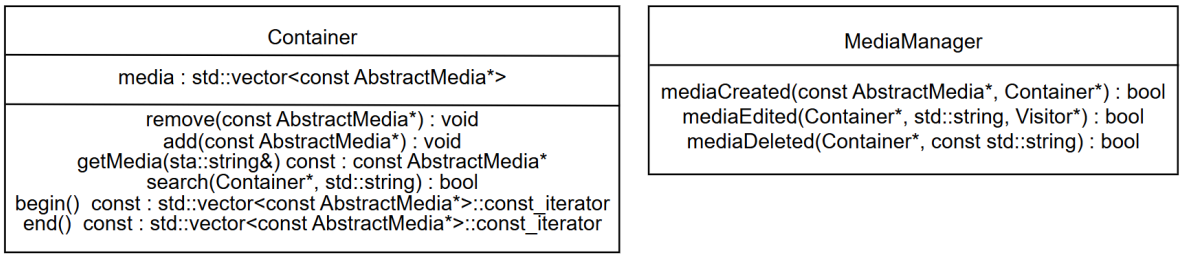
La classe `Visitor` presenta tre metodi virtuali puri, una per tipo di media. Le classi derivate sono `ConcreteVisitor` e `jsonVisitor`.

I metodi `visit(Media*)`, comuni ad entrambe le classi derivate, sono utilizzati per eseguire le azioni sui singoli media.
`ConcreteVisitor` si occupa di assegnare a variabili temporanee il valore dei dati del media, viene usata soprattutto nei metodi dove è necessario riempire i campi dati di un media senza sapere a priori il suo tipo.
I metodi `setAttributes` memorizzano i dati di un media nella struct -Attributes- appositamente definita.
`jsonVisitor` invece si occupa della memorizzazione dei dati in un array di tipo `QJsonArray` il quale viene utilizzato per poter gestire l'export dei dati della libreria. Oltre i metodi visit sono presenti i metodi `createObj(const Media*, QJsonArray)` che effettivamente aggiungono i media all'array.



(fig.2)

La classe manager si interpone tra l'interfaccia di Qute ed il container cosicché si crei a tutti gli effetti una distinzione netta tra le interfacce.
I metodi di mediaManager hanno come tipo di ritorno bool così quando viene invocato si può avere un comportamento diverso a seconda dell'esito del metodo.
Container definisce il contenitore di memorizzazione dei dati standard adattato ai media.



(fig.3)

}

3 POLIMORFISMO {

In questo progetto ho utilizzato il design pattern Visitor, questo tipo di implementazione mi è stato fondamentale per la distinzione di responsabilità tra le classi che definiscono i media e le classi che operano su di esse. Così facendo il progetto potrebbe essere facilmente espanso in futuro sfruttando il visitor pre-esistente nel codice.

Il mio design pattern si suddivide in 3 classi:

`Visitor:` la classe virtuale pura dove vengono semplicemente definiti i metodi visit, i quali specificano su quali elementi il visitor potrà operare.

`ConcreteVisitor:` Questa classe la ho utilizzata principalmente per gestire l'apertura di nuove finestre, per esempio nel caricare i dati del media in mediaEditor quando gli si vogliono modificare gli attributi. Un altro esempio di utilizzo lo troviamo in manager, dove il tipo del media è sconosciuto e quindi l'utilizzo del pattern visitor è estremamente utile. In questa classe vengono definiti i seguenti metodi:

`setAttributes:` memorizza in un tipo Visitor i dati di un media.

`getAttributes:` restituisce i dati memorizzati.

`jsonVisitor:` questa classe la ho utilizzata per gestire i metodi per il salvataggio del media nel file json. In questa classe vengono definiti i seguenti metodi:

`createObj:` questo metodo crea un oggetto da inserire nell'array di tipo QJsonArray, questo metodo è utilizzato quando si vogliono esportare i dati della libreria e viene eseguito in un ciclo per poter memorizzare tutti gli elementi del container.

`getJsonArray:` metodo che viene invocato nelle operazioni preliminare del salvataggio per poter effettivamente trascrivere i dati di salvataggio in un file `.json`.

}

4 PERSISTENZA DEI DATI {

Per la persistenza dei dati ho utilizzato il formato .json, ho incluso nel progetto un file .json dove ho un media per categoria.
Nel progetto è possibile sia esportare sia importare i dati.
Tramite il design pattern visitor gli oggetti serializzati contengono il tipo del media cosicché possano essere inseriti correttamente nella galleria.

}

5 FUNZIONALITÀ IMPLEMENTATIVE {

La GUI e' semplice ed intuitiva, è stata pensata per un utilizzo immediato ed efficace, per questo ci sono poche schermate che garantiscono una navigazione fluida.

La MainWindow presenta: una menubar con due voci: Media, Help; in Media si hanno le funzionalità base del programma, quindi, creare un nuovo media, importare o esportare i dati. Nel menu help è disponibile la descrizione delle scorciatoie da tastiera.
Nel centro della finestra sono presenti una griglia dove verranno mostrati i media, una barra di ricerca ed il nome dell'applicazione.
In basso è presente una statusbar la quale accompagna l'utente nella navigazione e fornisce conferma visiva all'utente delle operazioni eseguite.

Quando si vuole aggiungere un nuovo media il programma chiede quale tipo di media. Dopo la selezione viene istantaneamente mostrato l'editor nel quale è possibile inserire tutti i dati e caricare una foto identificativa. Nel caso in cui l'utente non voglia aggiungere una foto ne viene assegnata una di default per garantire coerenza visiva nella griglia.

Infine, quando si clicca su un media nella griglia vengono mostrate le informazioni con la foto del media grande a sinistra. E' da questa schermata che si puo' interagire con il media dove si puo' decidere di modificarlo o eliminarlo.

}

6 RENDICONTAZIONE ORE {

Attività	Ore Previste	Ore Effettive
Studio e progettazione	5	4
Sviluppo del codice del modello	5	5
Studio del framework QT	10	15
Sviluppo del codice GUI	10	10
Test e debug	5	10
Stesura della relazione	5	4
totale	40	48

La progettazione e lo sviluppo del modello logico l'ho trovato abbastanza immediato, ma ho trovato particolare difficoltà nell'immparare ad utilizzare l'interfaccia di QT anche se, una

volta presa confidenza lo sviluppo è iniziato ad essere più piacevole. La parte dove ho incontrato davvero problemi è stato il test e debug, dove, dopo aver preso confidenza con il debugger di qute, ho trovato che cercare e risolvere bug è un'attività abbastanza onerosa in termini di tempo .

}

7 CONCLUSIONI {

In conclusione, questa esperienza mi è servita per prendere familiarità con la creazione di un'applicazione con un'interfaccia grafica, in particolare, nonostante le iniziali difficoltà per entrare in confidenza con il framework qute, alla fine del progetto ritengo di aver appreso le tecniche per sviluppare efficientemente un progetto con questo framework.
La parte che reputo mi sia servita di più è stata la parte di debug finale dove mi sono dovuto applicare. Ritengo che, dopo aver imparato ad usare il debugger, ed aver speso tante ore per quest'attività mi abbia aiutato a navigare meglio il codice ed individuare problemi più velocemente, soprattutto quelli banali.
L'apprendimento di un design pattern è stato molto interessante e mi ha mostrato che la parte di progettazione non è solo pensare di realizzare il programma ma anche di avere un riguardo per il futuro di esso, che sia estenderlo o manutenzionarlo.

}