UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Media Qllection

Progetto di Programmazione ad Oggetti 2019/2020

Serban Eduard George

Matricola 1052741

eduardgeorge.serban@studenti.unipd.it

Maggio 2020

Descrizione

Relazione sul progetto Media Qllection di Programmazione ad Oggetti 2019/2020.

Indice

1 Descrizione del progetto

Lo scopo del progetto è realizzare un'applicazione chiamata **Media Qllection** che l'utente potrà utilizzare per archiviare *Musica*, *Audiolibri*, *Libri digitali* e *Film*. L'archivio viene gestito come una semplice lista sia all'interno della GUI che sul file esterno in cui la lista viene salvata. L'utente ha la possibilità di creare numerose liste e salvarle su file separati, in base alle sue esigenze. Le funzionalità offerte per gestire la lista sono molteplici e verranno analizzate nella prossima sezione. L'applicazione non è estremamente elaborata dal punto di vista grafico nonostante ciò la *user experience* risultante è piacevole e godibile poiché i layout e i colori scelti rendono le sue funzionalità facilmente accessibili.

2 GUI

La GUI è formata da una sola finestra suddivisa in 3 layout per rendere più chiara l'usabilità e dividere logicamente le funzionalità. La suddivisione della GUI in più finestre principali è stata valutata negativamente in quanto avrebbe reso l'esperienza utente più problematica.

Figura 1: GUI Media Qllection

I tre layout sono così suddivisi:

- a sinistra viene visualizzata la lista degli elementi aggiunti dall'utente. Tale lista è composta dal titolo di ogni elemento con all'inizio il simbolo del tipo dell'elemento (musica, audiolibro, libro digitale o film). Inoltre siccome l'utente ha la possibilità di inserire un voto, come recensione, da 1 a 5, gli elementi con voto 1 vengono colorati di rosso e gli elementi con voto 4 o 5 vengono colorati di verde;
- a destra sono presenti le varie funzionalità legate alla ricerca e nel caso l'utente selezioni un elemento della lista vengono visualizzate tutte le informazioni di quell'elemento;
- in basso sono presenti i tasti per le altre funzionalità offerte dall'applicazione.

3 Funzionalità

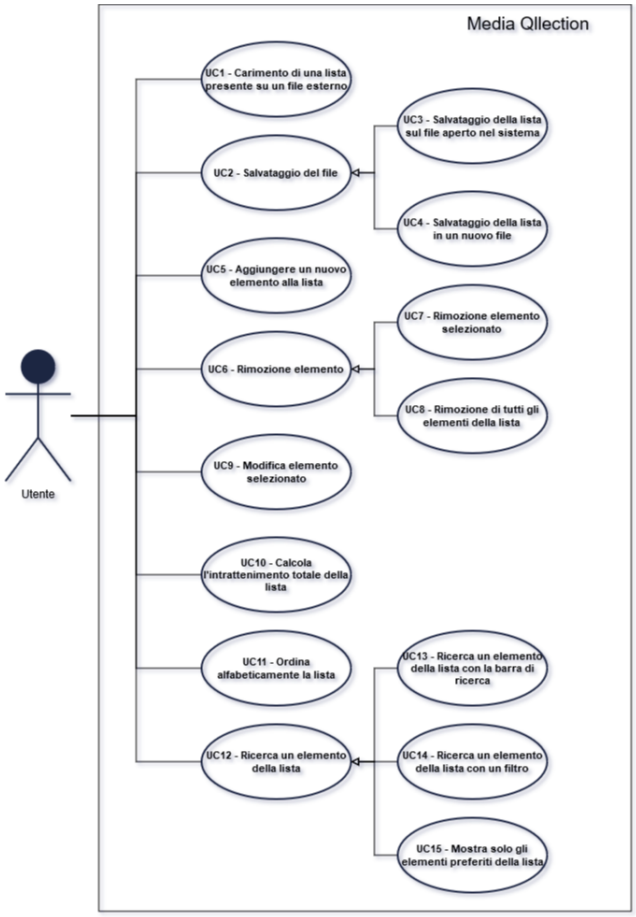

Figura 2: Casi d'uso Media Qllection

3.1 Gestione dei file esterni

L'applicazione permette la scrittura e la lettura su file esterni in formato *XML* (UC1, UC3, UC4). Quando l'applicazione viene avviata la prima volta essa carica automaticamente nel sistema il file data.xml con all'interno almeno un elemento per ogni tipo disponibile cosicché l'utente possa capire facilmente le potenzialità dell'applicazione. E' stato scelto l'XML sia perché è molto comune e facilmente comprensibile sia perché la libreria QT offre numerose facilitazioni per questo formato.

3.2 Gestione degli elementi della lista

L'applicazione permette di aggiungere un nuovo elemento alla lista (UC5), rimuovere uno (UC7) o tutti gli elementi della lista (UC8) e la modifica di un elemento della lista (UC9). L'utente inoltre può decidere di ordinare alfabeticamente gli elementi della lista (UC11). Attualmente questa funzionalità è irreversibile. Infine l'utente ha la possibilità di calcolare l'intrattenimento totale della lista.

3.3 Funzionalità di ricerca

L'utente ha a disposizione diverse funzionalità per ricercare o filtrare degli elementi presenti nella lista. Il modo più comune è quello di scrivere il titolo dell'elemento che cerca nella barra di ricerca (UC13). La lista degli elementi si aggiorna dinamicamente non appena l'utente comincerà a digitare le lettere del titolo che sta cercando. Un altro modo è fare una ricerca personalizzata (UC14) che permetterà all'utente di visualizzare solamente gli elementi di un tipo (musica, libro, etc). Infine "Mostra solo preferiti" (UC15) permette all'utente di visualizzare solamente gli elementi con una recensione massima (5).

4 Progettazione

Il progetto è stato sviluppato con sistema operativo Windows 10, Libreria Qt in versione 5.9.5, compilatore MinGW 5.3.0 32 bit ed editor Qt Creator in versione 4.10.2 (Community). Per la progettazione si è aderito al design pattern Model-View sia perché Qt include un insieme di classi di "view" che usano una architettura "model/view" per gestire la relazione tra i dati logici della GUI ed il modo in cui essi sono presentati all'utente sia perché la divisione tra modello e vista permette di fare modiche all'una o all'altra senza stravolgere completamente l'intero progetto.

4.1 Gerarchia dei tipi

La classe base astratta polimorfa *MediaItem* è la classe base della gerarchia che rappresenta un elemento generico. Esso contiene degli attributi generici comuni a tutti gli elementi come titolo, genere, anno di distribuzione, formato e recensione. Si tratta di una classe base astratta polimorfa poiché contiene i metodi virtuali

puri **durataTOT()** e **getTipo()**. Oltre a questi due metodi è presente il metodo virtuale **info()** che ritorna una stringa con tutti i campi dati a cominciare da quelli della classe base *MediaItem* fino all'oggetto di invocazione. Sono infine presenti i metodi **set** e i metodi **get** in ogni classe della gerarchia oltre ai metodi di uguaglianza e disuguaglianza. Qui di seguito viene mostrato il diagramma delle classi in cui, per aumentare la leggibilità, vengono mostrati solamente i metodi più "interessanti".

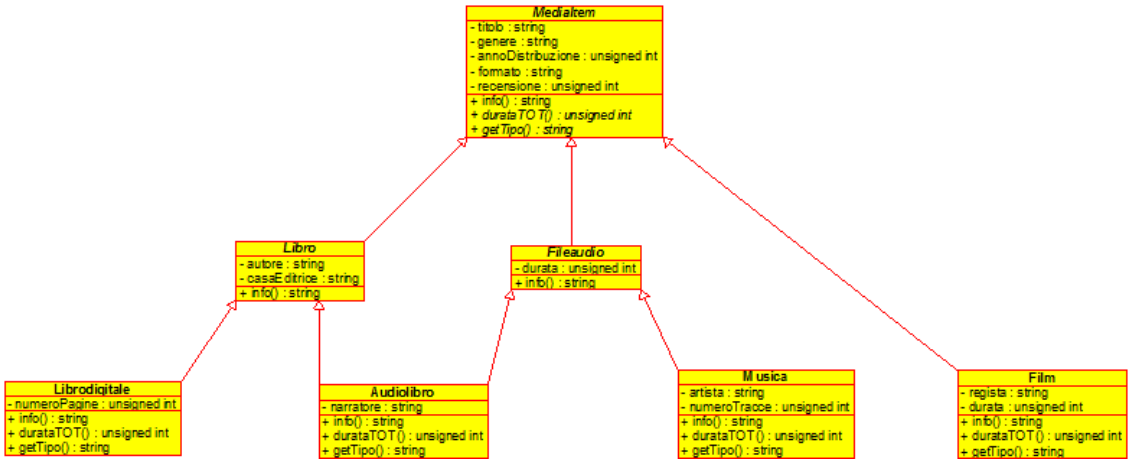

Figura 3: Diagrammi delle classi

Le classi *Libro* e *Fileaudio* derivano pubblicamente da *MediaItem* tuttavia rimangono classi astratte in quanto non implementano i metodi virtuali puri, **durataTOT()** e **getTipo()**, definiti nella classe base della gerarchia. Eseguono invece l'*override* del metodo **info()** invocando da prima il metodo per la classe base *MediaItem* e aggiungendo infine a questa stringa le informazioni dei propri campi dati: autore e casa editrice per *Libro* e la durata per *Fileaudio*.

Le classi concrete della gerarchia sono **Librodigitale**, **Audiolibro**, **Musica** e **Film**. Queste classi sono concrete in quanto implementano i metodi virtuali puri, citati precedentemente, della classe base della gerarchia.

Film è un sottotipo diretto della classe *MediaItem*, oltre all'implementazione dei metodi virtuali puri, esso esegue l'overriding del metodo **info()**, aggiungendo alla lista dei campi dati della classe *MediaItem* i propri attributi: regista e durata.

Librodigitale e **Musica** derivano rispettivamente da *Libro* e *Fileaudio*. Come per **Film**, oltre ad implementare i metodi virtuali puri della classe base, ridefiniscono il metodo **info()** aggiungendo alla lista, che il metodo ritorna, i propri campi dati: numero di pagine per **Librodigitale** e l'artista e il numero di tracce per **Musica**.

Audiolibro è una classe concreta in quanto implementa i metodi virtuali puri della classe base. La particolarità di questa classe sta nel fatto che derivi pubblicamente sia da *Libro* che da *Fileaudio*, sfruttando la cosidetta ereditarietà multipla, permessa dal linguaggio di programmazione C++. In questo caso si tratta dell'ereditarietà multipla a diamante poiché si può osservare che graficamente, considerando anche *MediaItem*, si crea un diamante nel diagramma delle classi. Questa situazione è stata opportunamente gestita, per non avere inconvenienti e problemi di compilazione, dichiarando virtuale la derivazione di *Libro* e *Fileaudio* da *MediaItem*. Infine **Audiolibro** fa l'overrinding del metodo **info()** aggiungendo alla stringa che viene ritornata il proprio campo dati narratore.

4.2 Uso di chiamate polimorfe

MediaItem ha i seguenti metodi virtuali (in alcuni casi puri) che permettono al programma di invocare il metodo corretto delle sottoclassi in base al tipo dinamico dell'oggetto di invocazione:

- virtual string ***info()*** const; //per ottenere le informazioni dell'oggetto
- virtual unsigned int ***durataTOT()*** const=0; //per calcolare la durata
- virtual string ***getTipo()*** const=0; //per ottenere il tipo dell'oggetto

Il comportamento di **info()** è stato abbondantemente spiegato in precedenza. Il metodo **durataTOT()** calcola la durata dell'oggetto dinamico che ha invocato il metodo. **Librodigitale** implementa questo metodo sfruttando il numero di pagine del libro moltiplicato per 3 minuti (lettura in media di una pagina), le altre tre classi concrete sfruttano semplicemente il campo dati durata (presente in **Film** o in *Fileaudio* per le classi **Audiolibro** e **Musica**). Il metodo **getTipo()** ritorna il tipo (nome della classe) dell'oggetto di invocazione. Questo metodo viene utilizzato per mostrare all'utente il tipo dell'elemento selezionato.
Vengono omessi nella lista, essendo poco interessanti, il distruttore virtuale e gli operatori di uguaglianza e disuguaglianza.

4.3 Container

Per la gestione della lista viene utilizzato un container implementato tramite una double linked list (lista doppiamente linkata). Il contenitore permette le tipiche operazione basilari come l'inserimento, la rimozione etc.

5 Informazioni utili sul progetto

5.1 Istruzioni per la compilazione

Il progetto prevede l'utilizzo di un file .pro diverso da quello ottenibile con l'esecuzione del comando qmake-project, per la presenza di alcuni flag relativi alla versione *c++11*. Utilizzare dunque i comandi **qmake** e poi **make**;

E' richiesto inoltre che la cartella in cui risiedono tutti i file e tutto il codice si chiami **ProgettoOOP** in quanto sia lettura che scrittura attraverso file xml, che fogli di stile e risorse (con immagini annesse), fanno riferimento a una cartella denominata in quel modo. Per avviare l'applicazione alla fine della compilazione utilizzare il comando **./ProgettoOOP**

5.2 Tempo impiegato

Sono state impiegate circa 50 ore così suddivise:

- Analisi preliminare del problema: 3 ore;
- Progettazione modello: 7 ore;
- Progettazione GUI: 3 ore;
- Apprendimento libreria *Qt* e tutorato: 16 ore;
- Codifica modello: 11 ore;
- Codifica GUI: 4 ore;
- Debugging: 3 ore;
- Testing: 3 ore;

5.3 Suddivisione del lavoro progettuale

Gruppo responsabile del progetto

I responsabili di questo progetto sono:

- Eduard George Serban - Matricola: 1052741;
- Luca Marcon - Matricola: 1070602.

I membri del gruppo hanno progettato in collaborazione sia il modello che la GUI. Tuttavia per sfruttare le potenzialità della suddivisione del lavoro il gruppo ha deciso di assegnare ad Eduard la parte riguardante il modello ed a Luca la parte riguardante la GUI. Concretamente questo si è tradotto in un paio di ore in più per Eduard per quanto riguarda la progettazione del modello, per poter raffinare quanto pensato in collaborazione e molte ore in più per la sua codifica, rispetto a quelle dedicate alla parte grafica. Lo stesso discorso ha riguardato Luca, in modo speculare, per la parte della GUI. Nonostante tale suddivisione, entrambi i componenti si sono attivati per conoscere e per scrivere codice nella parte non assegnata. Ad esempio Eduard ha scritto il codice della GUI per l'apertura o il salvataggio su un file esterno. Luca invece ad esempio ha scritto nel modello il metodo durataTOT() che abbiamo visto in precedenza. La classe Container è stata scritta in collaborazione.