Media Qllection

Progetto di
Programmazione ad
Oggetti
2019/2020

Luca Marcon
1070602

Descrizione

Media Qllection è un'applicazione che nasce con lo scopo di gestire un contenitore di media - musica, audiolibri, film e libri digitali - e che ha la principale utilità di archiviare i media di cui si è in possesso, di cui si ha usufruito o di cui si vorrebbe usufruire (con utilizzo simile alla "lista della spesa").

L'applicazione, oltre al salvataggio e al caricamento automatico della lista, permette di inserire, modificare, rimuovere e cercare media secondo determinati parametri e filtri (tra i quali la ricerca testuale e il filtro per "preferiti" che mostra solo i media che hanno una valutazione di 5 su 5).

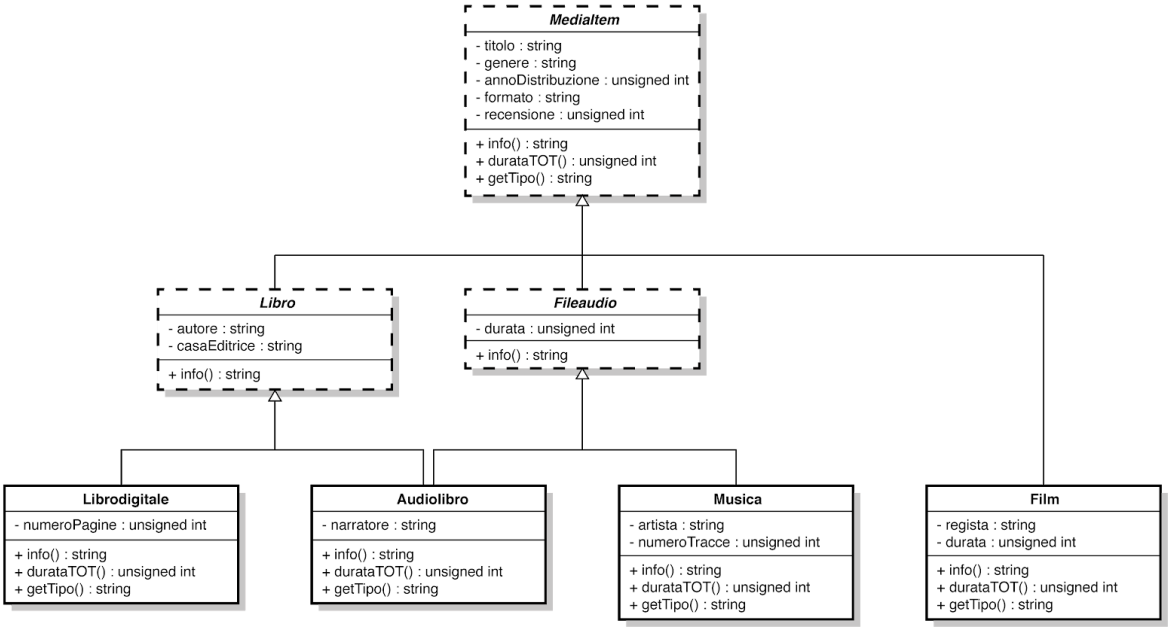
Inoltre, l'utente può caricare e salvare la lista da/su file esterni aggiuntivi, senza dover per forza utilizzare il database già integrato che viene caricato all'apertura dell'applicazione, riuscendo così potenzialmente ad utilizzare numerose liste in base allo scopo o all'utilizzo che si intende fare di Media Qllection: grazie alle molteplici funzionalità, l'utente è quindi libero di utilizzare il contenitore a proprio piacimento.

Sono presenti infine delle funzioni di ordinamento alfabetico per titolo e di calcolo dell'intrattenimento totale, secondo una formula che somma la durata di tutti i film, di tutta la musica, di tutti gli audiolibri e il tempo di lettura totale delle pagine dei libri digitali (considerando tre minuti per la lettura di una pagina).

Progettazione

Il design pattern a cui si è deciso di aderire è il Model-View. Gli strumenti forniti da Qt, in relazione alla flessibilità di questo pattern, hanno permesso facilità nel mantenere modello e vista separati e uno sviluppo del progetto con tempistiche adeguate a quelle richieste. Pur consapevoli della possibilità di utilizzare il pattern Model-View-Controller, è stata fatta a priori la scelta del Model-View per poter completare il progetto soddisfando le richieste della consegna nel modo più aderente possibile.

Gerarchia di tipi



La classe base astratta polimorfa MediaItem è la classe base della gerarchia e rappresenta un media generico che non ha una tipologia, ma è dotato di attributi generici comuni a tutti i media: le *stringhe* titolo, genere e formato, gli *unsigned int* annoDistribuzione e recensione - rilasciata dall'utente (espressa in numero di cuori da 1 a 5, dove 1 indica il minimo e 5 indica il massimo). Si tratta di una classe base astratta perché contiene i metodi virtuali puri **`durataTOT()`** e **`getTipo()`** i quali verranno implementati dalle classi discendenti: come già accennato nella descrizione, il primo metodo virtuale puro permetterà, tramite chiamate polimorfe, di calcolare la durata totale dei media; il secondo, sempre tramite chiamate polimorfe, restituirà una stringa contenente il tipo dinamico dell'oggetto (questa funzionalità si rivela utile in particolare nella GUI per mostrare all'utente il tipo dell'oggetto selezionato). Oltre a ciò, contiene dei semplici metodi di accesso e inizializzazione dei campi dati ("getters" e "setters"), il distruttore virtuale e un metodo virtuale **`info()`** che stampa nello stream di output una stringa con i campi dati. Ovviamente di quest'ultimo metodo, essendo virtuale, verrà fatto l'override nelle classi figlie, dove manterrà l'output dei campi dati di MediaItem, aggiungendoci inoltre i campi dati della classe che lo ha implementato. Infine sono presenti anche dei metodi per il comportamento degli operatori di uguaglianza, disuguaglianza e confronto. Il metodo **`info()`** e gli operatori di uguaglianza implementati nella gerarchia, pur non trovando un utilizzo effettivo nella sezione della vista che gestisce la GUI, sono stati introdotti per semplificare le possibili future implementazioni e rendere quindi estensibile il codice: essi rispecchiano la separazione tra parte logica e grafica e l'evolvibilità dell'applicazione.

Da MediaItem derivano altre due classi astratte: **Libro** e **Fileaudio**

- La classe **Libro** è caratterizzata dai campi dati autore (di tipo *string*), in cui si riportano nome e cognome dell'autore del libro e casaEditrice (di tipo *string*) che identifica con un'altra stringa il nome della casa editrice. Libro contiene inoltre l'override dei metodi virtuali **`info()`** e operatori di uguaglianza e disuguaglianza. Ha dei metodi per accedere e modificare i campi dati. Si tratta di una classe astratta in quanto non implementa i metodi virtuali puri ereditati da MediaItem **`durataTOT()`** e **`getTipo()`**.
- **Fileaudio**: in modo similare a Libro, Fileaudio fa l'overriding di **`info()`**, == e !=, ma non implementa in modo personalizzato **`durataTOT()`** e **`getTipo()`,** confermandosi così classe astratta. Il campo dati specifico di Fileaudio è durata (di tipo *unsigned int*).

Le classi concrete della gerarchia sono:

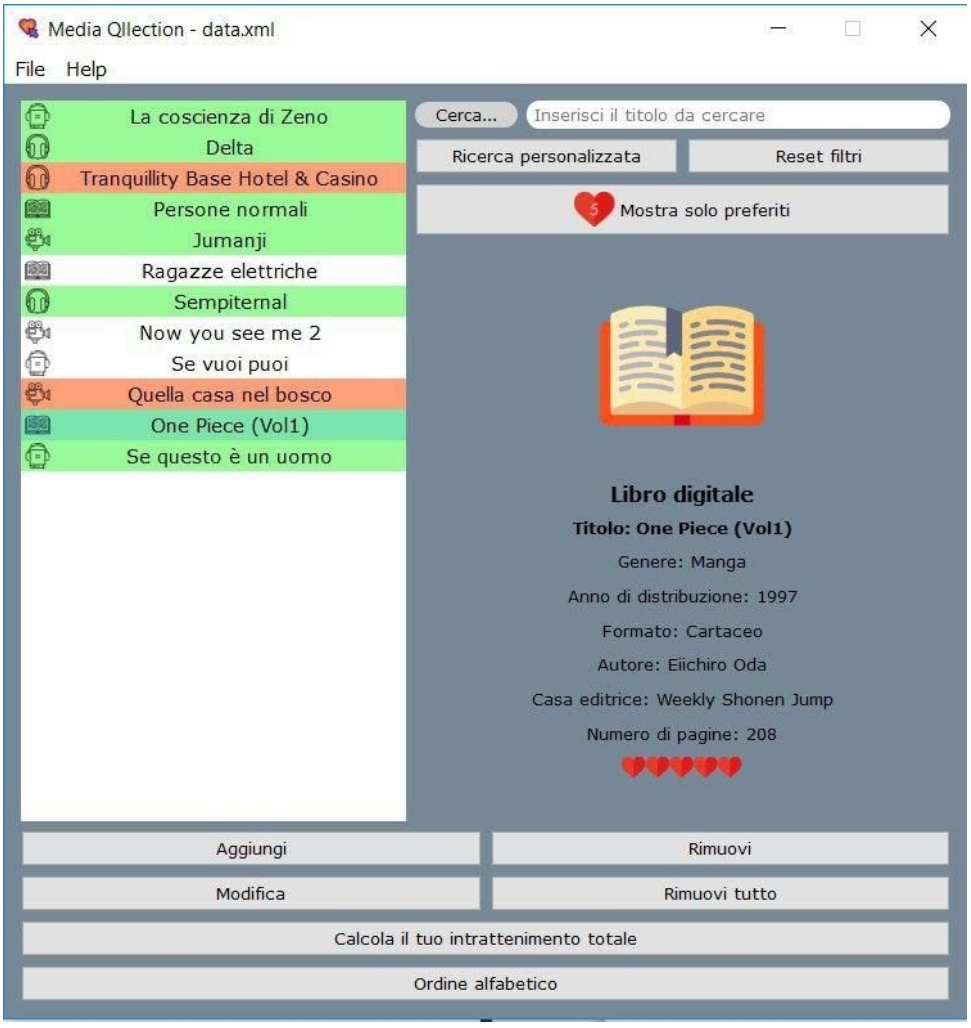
- **Librodigitale:** La classe Librodigitale deriva da Libro ed è caratterizzata dal campo dati numeroPagine (di tipo *unsigned int*) che memorizza il numero di pagine del libro. Librodigitale contiene inoltre l'override dei metodi virtuali **`info()`** e operatori di uguaglianza e disuguaglianza, oltre ai metodi per accedere e modificare i campi dati. Infine implementa i metodi virtuali **`durataTOT()`**, restituendo il numero delle pagine moltiplicato per 3 (ovvero 3 minuti per la lettura di una pagina), e **`getTipo()`**, che restituisce la stringa "Libro digitale".
- **Film:** deriva direttamente da MediaItem e contiene due campi dati oltre a quelli ereditati da MediaItem: il nome del regista di tipo *string* e la durata del film di tipo *unsigned int*. Oltre ai classici metodi per accedere ("*get*") e per modificare ("*set*") i campi dati, anche Film ridefinisce **`info()`** e gli operatori == e !=. Qui l'implementazione di **`durataTOT()`** semplicemente restituisce la durata del film ottenuta tramite **`getDurata()`**, mentre **`getTipo()`** restituisce la stringa "Film".
- **Musica**: deriva da Fileaudio. In modo similare alle altre classi dello stesso livello di gerarchia, Musica fa l'overriding di **`info()`**, == e != ed implementa in modo personalizzato **`durataTOT()`** e **`getTipo()`**. Poichè i campi dati di musica sono artista (di tipo *string*) e numeroTracce (che tiene nota del numero di canzoni dell'album), allora **`durataTOT()`** sfrutta il metodo **`getDurata()`** ereditato da Fileaudio per ottenere la durata totale dell'album. Questa classe lascia libera interpretazione all'utilizzatore dell'applicazione, in quanto può tenere nota sia di album musicali, che di raccolte, che di singole canzoni. In quest'ultimo caso (pur poco verosimile visto lo scopo primario dell'applicazione di tenere nota di grandi quantità di dati) sarà sufficiente impostare il numero di tracce a 1.
- **Audiolibro**: grazie all'ereditarietà multipla, permessa in modo particolare dal linguaggio di programmazione C++, Audiolibro deriva sia da Libro che da

Fileaudio. In questo caso specifico, Audiolibro contribuisce alla formazione del cosiddetto “diamante”, insieme alle classi astratte MediaItem, Fileaudio e Libro. Audiolibro ha un suo campo dati (la stringa narratore) e implementa **`durataTOT()`** e **`getTipo()`**.

Metodi virtuali e chiamate polimorfe

Risulterà quindi chiaro che i metodi virtuali utilizzati sono gli operatori di uguaglianza, di disuguaglianza, il distruttore di MediaItem, della stampa di informazioni (metodo **`info()`**), di calcolo della durata totale (metodo **`durataTOT()`**) e di stampa del tipo (metodo **`getTipo()`**), dei quali si sono ampiamente spiegate le funzionalità in precedenza. In particolare, vengono spesso utilizzate, nella parte della vista, chiamate polimorfe su **`durataTOT()`** e **`getTipo()`**: esse permettono di avere quindi una funzionalità personalizzata in base al tipo dinamico dell’oggetto di invocazione.

GUI ed Utilizzo del software



La GUI presenta tre layout principali:

- il primo, a sinistra, contiene una lista di tutti gli oggetti finora inseriti denominati per titolo e contraddistinti da un simbolo personale: Libro digitale, Film, Audiolibro e Musica sono ben distinguibili grazie a quattro diverse icone. Inoltre, per una più semplice visualizzazione "a colpo d'occhio", si è deciso di impostare un colore di sfondo per ogni elemento in base alla recensione ottenuta al momento dell'inserimento: i media che hanno ricevuto una recensione molto bassa (1 cuore) hanno lo sfondo rosso, quelli che invece hanno ottenuto una valutazione ottima o eccellente (4 e 5 cuori) hanno lo sfondo verde.
- il secondo layout, a destra, contiene le principali funzionalità di ricerca e di filtro. Oltre alla ricerca testuale per titolo è possibile impostare un filtro personalizzato che mostra solo i media di tipo Libro digitale, o quelli di tipo Film, o quelli di tipo Audiolibro, o quelli di tipo Musica. Il filtro di ricerca testuale è dinamico: quando l'utente inserisce una chiave di ricerca attraverso la tastiera, l'applicazione reattivamente imposta il filtro sulla lista. Dunque, man mano che l'utente inserisce o elimina le varie lettere dalla barra di ricerca per comporre le parole, il filtro si raffina, mostrando quindi in tempo reale gli elementi corrispondenti alla chiave attualmente inserita.

  Altra funzionalità è il filtro per "preferiti": permette di visualizzare solo i media a cui è stata conferita una valutazione di 5 cuori su 5. Infine, il tasto "Reset filtri" permette di resettare gli eventuali filtri inseriti, reimpostando la lista completa.

  Sotto a questi pulsanti, una volta selezionato un media, vengono visualizzate tutte le informazioni relative ad esso. Ogni media mostra quindi tutti i suoi campi dati con i valori ad essi assegnati nel momento dell'inserimento nella lista (o dopo la modifica)
- il terzo layout è collocato sotto ai due precedenti e presenta le funzionalità di inserimento, modifica e rimozione. Sono inoltre presenti i pulsanti per ordinare la lista secondo l'ordine alfabetico e per mostrare "l'intrattenimento totale" che fa affidamento sul metodo virtuale **`durataTOT()`** e attraverso le chiamate polimorfe somma la durata di tutti i film, la durata di tutta la musica e il tempo necessario per la lettura di tutti i libri (3 minuti per pagina, quantità considerata generalmente ragionevole).

Container

Il contenitore di oggetti che viene utilizzato dal modello è stato implementato tramite una doubly linked list (lista doppiamente linkata). Esso permette le operazioni di inserimento e rimozione di elementi in qualsiasi posizione in maniera efficiente, ovvero consentendo tali operazioni in tempo O(1) costante.

Estensibilità

Per la migliore chiarezza e manutenibilità del codice, si è deciso di separare il più possibile tutte le varie sezioni, sia nella parte del modello che nella parte della vista. Il progetto, visto il monte ore limitato, presenta una gerarchia alquanto basilare e per ciò questa gerarchia risulta facilmente sviluppabile in "verticale": Librodigitale, Film, Audiolibro e Musica potrebbero a loro volta avere delle classi derivate. L'implementazione risulterebbe semplice: sarebbe sufficiente inserire i nuovi file relativi alle nuove classi, modificare parzialmente il file xml per lettura e scrittura, e implementare le nuove funzionalità. Le modifiche al codice preesistente sarebbero dunque molto limitate. Allo stesso modo basterebbe implementare le modifiche nella vista potendo sfruttare le strutture già esistenti ed inserendo solo i nuovi campi.

Caricamento e salvataggio del contenitore

Il caricamento e il salvataggio del contenitore avvengono attraverso l'utilizzo di un file XML e l'implementazione di alcuni metodi di lettura e scrittura nei file xml.h e xml.cpp.
Il caricamento del contenitore avviene in automatico quando l'applicazione viene lanciata, mentre per accedere alla funzione di salvataggio sarà sufficiente fare click su File nel menù principale e conseguentemente salvare (verrà mostrata una finestra per informare dell'avvenuto salvataggio). L'app carica automaticamente il file **`data.xml`**.
In alternativa è possibile utilizzare le funzioni di caricamento e salvataggio da/su file esterno scelto dall'utente, tramite i percorsi File -> Apri e File -> Salva con nome. Per una prova pratica del funzionamento forniamo nella repository il file **`data2.xml`**.
E' stato scelto il formato XML in quanto facilmente fruibile da tutti gli utenti, anche quelli meno esperti: i tag sono di semplice interpretazione e rappresentano direttamente i campi dati della gerarchia. Inoltre la libreria Qt permette di integrare in maniera efficace XML grazie a dei metodi specifici.
In scrittura, la funzione **`void xml::write(Container<MediaItem *> & lista) const`** verifica il risultato del downcasting sull'oggetto MediaItem e scrive i campi dati relativi alla sottoclasse in questione (Audiolibro, Librodigitale, Film, Musica).
In lettura invece, **`Container<MediaItem *> xml::read() const`** tramite un ciclo while legge tutti i tag e li inserisce nei relativi campi.

Suddivisione del lavoro

I responsabili di questo progetto sono:

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

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.

Tempo impiegato

Circa 50 ore, di cui:

- Analisi preliminare del problema: 3 ore
- Progettazione modello e GUI: 6 ore
- Apprendimento libreria Qt e partecipazione al tutorato: 15 ore
- Codifica modello: 7 ore
- Codifica GUI: 14 ore
- Debugging: 3 ore
- Testing: 2 ore

Note

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, come ad esempio l'istruzione **`QMAKE_CXXFLAGS += -std=c++11`**. In particolare, nel progetto sono stati utilizzati alcune keywords come *default* e *override*, specifiche di c++11. Il progetto compila ed esegue correttamente nella macchina virtuale Linux fornita.
Utilizzare i comandi **`qmake; make;`** per la compilazione.
Per avviare l'applicazione al termine delle operazioni di build, utilizzare il comando **`./ProgettoOOP`**

E' richiesto inoltre che la cartella in cui risiedono tutti i file e tutto il codice si chiami **ProgettoOOP** (nomenclatura che aveva durante lo sviluppo, abbreviazione di Progetto Object Oriented Programming) in quanto sia lettura e scrittura attraverso file xml, che fogli di stile e risorse (con immagini annesse), fanno riferimento a una cartella denominata in quel modo.
L'intero 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 4.9.2