Relazione del progetto di Programmazione ad Oggetti: Qontainer Elisabetta Piombin 1142189

Indice

1	Abstract	2
2	Manuale utente2.1 Finestra principale2.2 Aggiunta di contenuto2.3 Ricerca del Contenuto	4
3	Compilazione ed esecuzione	g
4	Descrizione della gerarchia di classi	9
5	Descrizione delle chiamate polimorfe	10
6	Descrizione del formato del file di salvataggio e caricamento del contenitore	10
7	Tempistiche	10

1 Abstract

Qontainer è un progetto realizzzato al fine di fornire un contenitore che gestisca una libreria di contenuti multimediali: file audio e file video, che si dividono a loro volta in canzoni, podcast, serie tv e film. Per farlo, la classe templatizzata container a sua volta fa uso di altre classi annidate:

- nodo: inserita nella parte privata di container, viene usata per memorizzare i vari contenuti multimediali, visti come se fossero una lista concatenata di elementi, con ogni nodo diviso nel suo campo info (di tipo parametrico T, next e prev(di tipo nodo*). In container è presente un puntatore al primo elemento della lista.
- 2. const_iterator: inserita nella parte pubblica di container, è la classe che permette l'implementazione di iteratori costanti.
- 3. iterator: inserita nella parte pubblica di container, è la classe che permette l'implementazione di iteratori non costanti.

Vista l'assenza di puntatori ad altre classi nella gerarchia, è stata ritenuta superflua l'implementazione di una classe per un eventuale *smart pointer*.

Ho scelto di implementare il contenitore come una lista linkata composta da nodi perché offro la possibilità di rimuovere contenuti all'interno della libreria in posizioni arbitrarie, e questo mi porta alla necessità di scegliere l'implementazione sotto forma di lista, che rende tale operazione di rimozione più efficiente (viene operata in tempo costante) rispetto alla scelta di implementare tale contenitore come un vettore di elementi.

2 Manuale utente

In questa sezione vengono descritte le varie schermate della GUI.

2.1 Finestra principale



Figura 1: Finestra principale del progetto

Da qui si ha un'*overview* dell'intera applicazione, con tutti i contenuti in essa inseriti dall'utente, suddivisi per **Canzoni**, **Podcast**, **Film** e **Serie**. Tramite la finestra principale si da' all'utente la possibilità di cercare o di aggiungere contenuto.

2.2 Aggiunta di contenuto

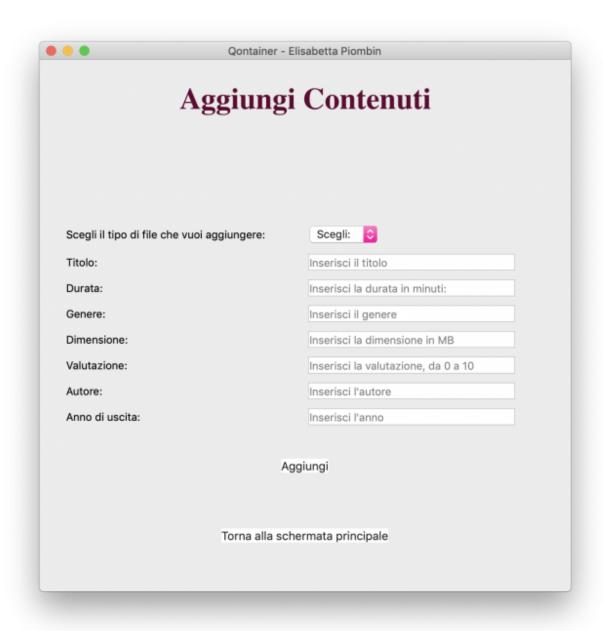


Figura 2: Finestra base di aggiunta contenuti

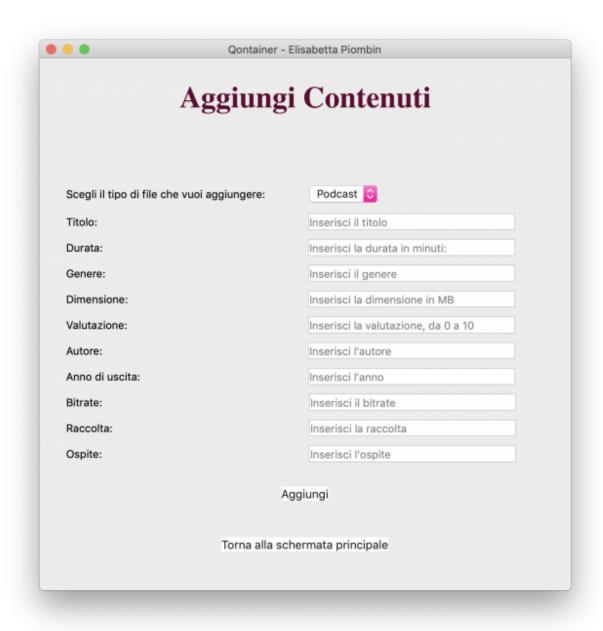


Figura 3: Finestra specifica di aggiunta contenuti

Nella schermata di aggiunta dei contenuti l'utente deve scegliere il tipo di contenuto che vuolei inserire, e a seconda della scelta compariranno i campi dati specifici di quella classe.

2.3 Ricerca del Contenuto

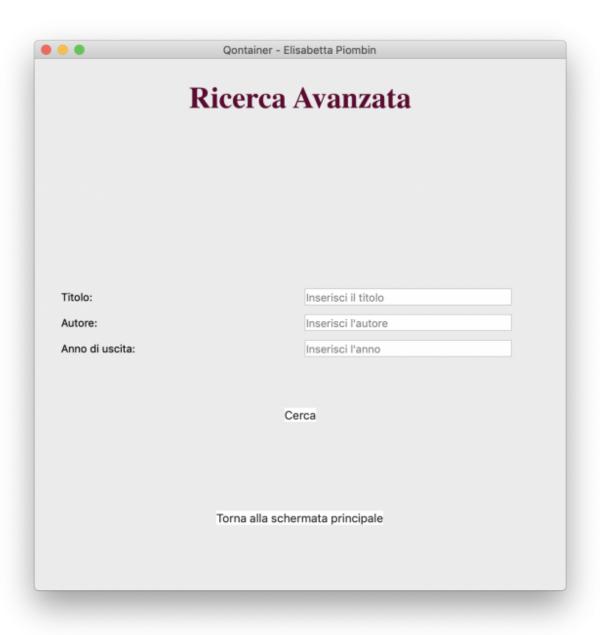


Figura 4: Prima finestra di ricerca

All'utente viene data la possibilità di modificare i contenuti esistenti, e la ricerca per la modifica è imlementata in modo che accetti sia stringhe incomplete che *upper* o *lower case*. All'utente viene data solamente la possibilità di inserire i campi dati che è più probabile vengano ricordati a memoria, per avere una ricerca realistica: sarebbe impossibile che l'utente si ricordi il valore di ogni singolo campo dato di ogni oggetto inserito nell'applicazione.

Una volta inserito anche solo un campo dati parziali (nell'esempio ho inserito solamente "iron" nel campo del titolo), l'utente si ritrova con tutti i match trovati all'interno dell'applicazione:

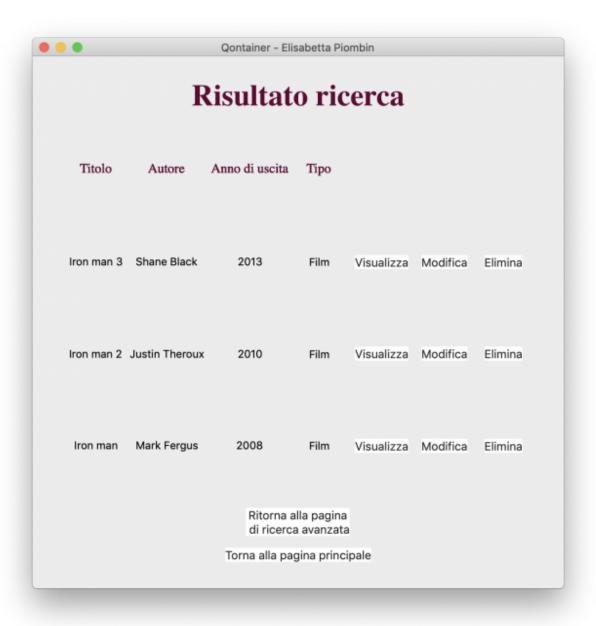


Figura 5: Risultato della ricerca

Una volta aperta la finestra dei risultati, all'utente viene data la possibilità di **visualizzare** nel dettaglio, **modificare** oppure **eliminare** del contenuto.

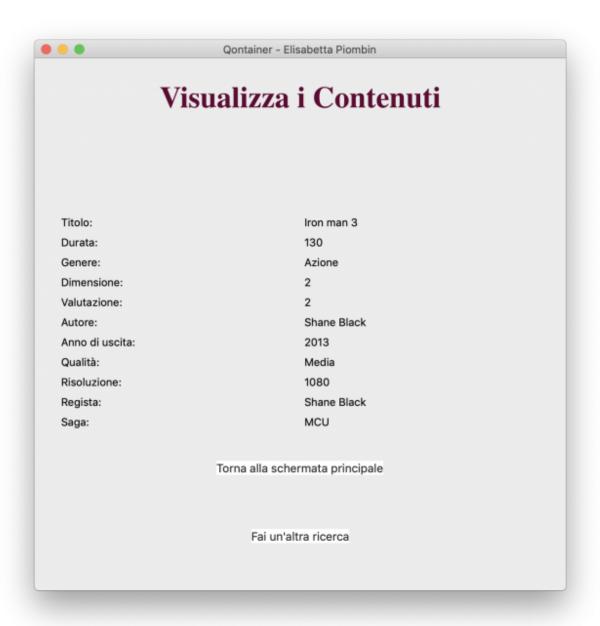


Figura 6: Visualizzazione dettaglio contenuti



Figura 7: Modifica dei contenuti

3 Compilazione ed esecuzione

La compilazione si esegue tramite i comandi quake e in seguito make.

4 Descrizione della gerarchia di classi

La classe base astratta da cui deriva tutta la gerarchia è ContenutoMultimediale, che verrà poi concretizzata tramite sue classi derivate. Da ContenutoMultimediale derivano immediatamente altre due classi: audio e video, che sono le due macrocategorie di appartenenza dei file che vengono memorizzati nella libreria.

Da video derivano due classi: film e episodio, mentre da audio derivano altre due classi, podcast e canzone.

Non si verifica la situazione di ereditarità multipla.

5 Descrizione delle chiamate polimorfe

Sono presenti tre metodi virtuali nella gerarchia, i metodi di serializza(char), l'overloading dell'operatore di confronto operator==(const T&), e il metodo qualita(), che è puro.

- string serializza(char): definito virtuale all'interno di ContenutoMultimediale, viene implementato diversamente per i campi dati specifici di ogni classe derivata. Viene richiamata all'interno della classe database, all'interno della funzione void save(container<T>&), in quanto questa è la funzione che si occupa del salvataggio su file degli oggetti inseriti dentro al container.
- bool operator== (const T&) è usato nella funzione void remove(T) all'interno di ContenutoMultimediale, e viene usata in risultatoricerca (nella GUI) per rimuovere un elemento dal contenitore.
- string qualita() const, metodo virtuale puro in ContenutoMultimediale, utilizzato in visualizzaelemento per dare la possibilità all'utente per vedere se il file che si sta visualizzando è di qualità media, alta o bassa. Ha diverse implementazioni in audio e in video.

Nella classe base ContenutoMultimediale è presente il distruttore virtuale di default.

6 Descrizione del formato del file di salvataggio e caricamento del contenitore

Per il caricamento dei file che compongono la libreria, e per il loro salvataggio, mi appoggio ad una classe esterna alla gerarchia, che ho chiamato database. Questa classe contiene i metodi load() e save(). La classe database non è un template perché dipende strettamente da ContenutoMultimediale.

- void load(container<ContenutoMultimediale*>&), tramite un'operazione di deserializzazione implementata ad hoc in ogni classe derivata concreta della gerarchia, permette la lettura del contenuto in ogni file .txt. Esiste un unico file .txt, in cui sono scritti tutti gli oggetti che verranno caricati nel container. Database si occupa di caricare questi oggetti nella libreria tramite static film* film::deserializza(const vector<string>&), static episodio* episodio::deserializza(const vector<string>&) e static canzone* canzone::deserializza(const vector<string>&), in modo che possano essere gestiti dal contenitore templetizzato.
- void save(container<ContenutoMultimediale*>&) si occupa di memorizzare sul file .txt i caratteri
 corrispondenti agli oggetti contenuti in container, invocando il metodo serializza(char) corretto,
 che è definito come virtuale.

7 Tempistiche

Ore	Fase
2	Analisi preliminare del problema
1	Progettazione del modello
1	Progettazione della GUI
10	Apprendimento libreria Qt
30	Codifica modello e GUI
10	Debugging
5	Testing