Progetto di Programmazione a Oggetti

Sara Righetto, matricola 1174009

Anno accademico 2018/2019

Sommario

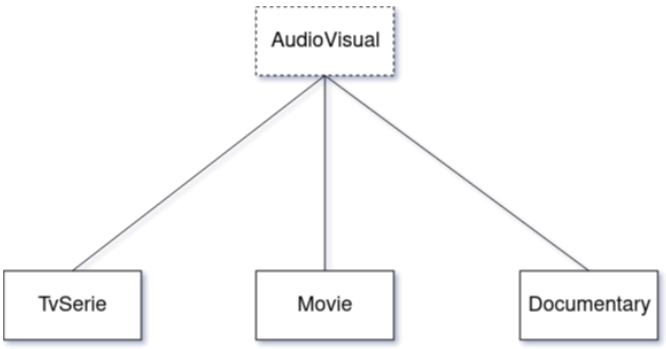
QPlay è un applicazione che consente, tramite l'uso di un contenitore, la gestione e la visualizzazione di file audio-visivi di vario tipo. L'utente può inserire, modificare ed eliminare file, eseguire ricerche in base a diversi campi dati e caricare/salvare su disco la propria lista di file.

Ambiente di sviluppo e direttive di compilazione

Il progetto è stato sviluppato su sistema operativo Manjaro Linux, utlizzando l'IDE *QtCreator* con il relativo framework QT v. 5.12 e compilatore GCC v. 9.1.0.
Poichè il progetto fa ampio uso di alcune *keywords* di C++11 (per esempio `auto` e `nullptr`) viene fornito il file `.pro` da compilare tramite comandi `qmake` e successivamente `make` che creano il relativo file eseguibile.

Gerarchia dei tipi

La gerarchia vuole modellare degli oggetti che comprendono alcuni file audio-visivi ed è così composta: una classe base astratta denominata `AudioVisual` e tre classi derivate concrete `TvSerie`, `Movie`, `Documentary` che implementano i metodi puri della classe base.



AudioVisual La classe base della gerarchia fornisce le informazioni di base comuni a tutti i file di tipo audio-visivo, ossia il *titolo* del file, il percorso dell'*immagine* di copertina, la *descrizione* della trama, l'*anno* di uscita, la *durata* in minuti, il nome del *regista*, l'appartenenza ai *preferiti* dell'utente, se l'audio è *compresso* o meno, la *risoluzione* dell'immagine, i *frame per secondo*.

TvSerie TvSerie è una classe derivata concreta che rappresenta i file riguardanti le serie tv, infatti aggiunge i campi dati riguardanti il numero di *episodi* della serie, il numero di *stagioni*, se la serie è *terminata*, il *rating*, il *genere*, il *cast* formato dai personaggi.

Documentary Documentary è una classe derivata concreta che rappresenta i file di tipo documentario, in cui si ha un *topic* che spazia dall'argomento scientifico, a quello storico o biografico e un campo dati riservato al *narratore*.

Movie Movie è anch'essa una classe derivata concreta che vuole rappresentare i file di tipo film; aggiunge i campi dati riguardanti il *cast*, il *rating* ed il *genere*.

La gerarchia attuale è stata pensata per essere estensibile in futuro, sia in orizzontale, per esempio implementando una classe riguardante i video di YouTube, che in verticale, per esempio derivando da `TvSerie` una classe che rappresenti le telenovelas.
Inoltre non fa uso di dati o classi riguardanti il framework QT, per cui è indipendente da esso.

DeepPtr Il progetto fa uso anche di un template di classe `DeepPtr<T>` di puntatori polimorfi al tipo T che implementano la gestione automatica della memoria cosidetta profonda. È quindi necessario che ogni classe concreta della gerarchia implementi il metodo relativo alla clonazione e anche la distruzione polimorfa.

Polimorfismo All'interno della gerarchia, nella classe base, sono stati dichiarati 5 metodi virtuali puri che vengono poi implementati nelle classi derivate concrete.

1. `virtual AudioVisual* clone() const =0;` utilizzato dallo *SmartPointer* per la copia profonda dell'oggetto.
2. `virtual unsigned int getTotalRunningTime() const =0;` che si occupa di calcolare la durata totale di ogni oggetto.
3. `virtual std::string getType() const =0;` restituisce sottoforma di stringa il tipo dell'oggetto.
4. `virtual bool isHighQuality() const =0;` che determina se un oggetto è di qualità o meno in base alla risoluzione dell'immagine, della compressione dell'audio e dei frame per secondo.
5. `virtual bool matureContent() const =0;` si occupa di determinare se l'oggetto è visionabile solo da adulti.

Inoltre, sebbene non siano metodi puri, sono stati dichiarati virtuali anche i metodi `virtual bool operator==(const AudioVisual&) const;` che fa un confronto tra ogni campo dati presente nella classe e anche il distruttore.

Container<T>

Il contenitore templetizzato è stato sviluppato secondo la linea del contenitore `List` della libreria STL.
Il contenitore fa utilizzo delle classi annidate `Node` che rappresenta un nodo della lista, con campi dati *info*, il puntatore al nodo precedente *prev* e al nodo successivo *next*, la classe `Iterator` e

`Const_Iterator` usate per scorrere gli elementi della lista.
Come da specifica il contenitore offre metodi per l'inserimento di nodi, quali `push_front` e `push_back`, la rimozione di nodi tramite `clear`, `erase`, `pop_front` e `pop_back`, e la ricerca di elementi grazie all'uso del metodo `search` reso disponibile sia costante che non.
Sebbene il contenitore venga utilizzato nel progetto istanziandolo con parametro uguale a `DeepPtr<AudioVisual>`, il parametro T del contenitore può assumere qualsiasi tipo, che sia primitivo o definito dall'utente.

Persistenza delle informazioni

Come da specifica il progetto consente il caricamento e salvataggio delle informazioni relative al contenitore. Si è scelto di adottare l'uso dei file XML come formato di file per la persistenza.
Sono state implementate le funzioni `load` e `save` che fanno uso delle classi `QXmlStreamWriter` e `QXmlStreamReader` per la codifica e decodifica del file XML, e le classi `QFile` e `QSaveFile` per l'apertura e in generale la gestione dei file.
Entrambe le funzioni devono essere chiamate esplicitamente dall'utente e non avvengono in automatico, tranne all'uscita dell'applicazione in cui tramite un popup apposito viene chiesto se si vuole salvare o meno i dati correnti.

GUI

Per lo sviluppo della parte relativa alla GUI è stato adottato il design pattern *Model-View*.
La parte grafica è composta da:

- `MainWindow` derivata pubblicamente da `QMainWindow`, è la finestra principale che viene lanciata all'avvio dell'applicazione.
- `AddDialog` deriva pubblicamente da `QDialog` e si occupa di inserire nuovi oggetti nella collezione.
- `AudioVisualItem` derivato pubblicamente da `QWidget`, ha lo scopo di mostrare i dettagli principali di ciascun oggetto contenuto nella lista.
- `EditWidget` derivato anch'esso pubblicamente da `QWidget`, si occupa di modificare un singolo elemento della lista.
- `DisplayWidget` deriva pubblicamente da `QDialog`, si occupa di visualizzare tutti i dettagli relativi ad ogni oggetto.

Ripartizione ore

La realizzazione del progetto è costata circa 50 ore ripartite nel seguente modo:

- Analisi preliminare del problema: 1h
- Progettazione modello e GUI: 6h
- Apprendimento libreria Qt: 8h
- Codifica modello e GUI: 31h
- Debugging e testing: 6h