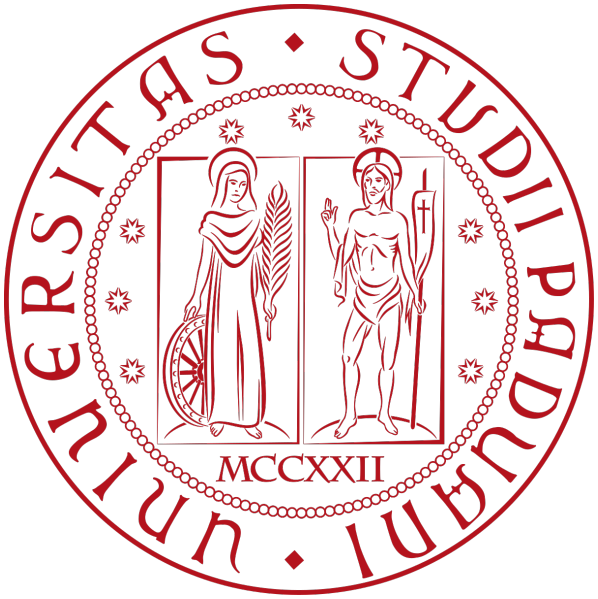
Università degli Studi di Padova



Relazione per:

Programmazione ad Oggetti

Per la creazione di una:

Biblioteca Digitale

Realizzata da:

Riccardo Baldin, matricola 2075548

GitHub Repository:

https://github.com/RiccardoBaldin/UniPD_OOP_Project

1 Introduzione

L'**applicazione** sviluppata consente di gestire una biblioteca virtuale, tenendo traccia di **libri**, **video** e **serie TV**.

Tramite un'interfaccia grafica realizzata con il framework **Qt**, il programma permette di creare, inserire, modificare, cercare, eliminare e salvare i file. L'**applicazione** consente non solo di salvare la propria libreria di contenuti, ma anche di conservare tutte le informazioni relative a ciascun elemento, rendendo più semplice per l'utente la consultazione dei dati specifici.

Il programma è inoltre dotato di un sistema di preferiti, che consente all'utente di accedere rapidamente agli elementi più importanti. Il salvataggio e il caricamento delle librerie, sia nella loro interezza sia per singoli elementi, avvengono tramite file `.JSON` creati automaticamente dal programma al momento del salvataggio.

2 Descrizione del modello

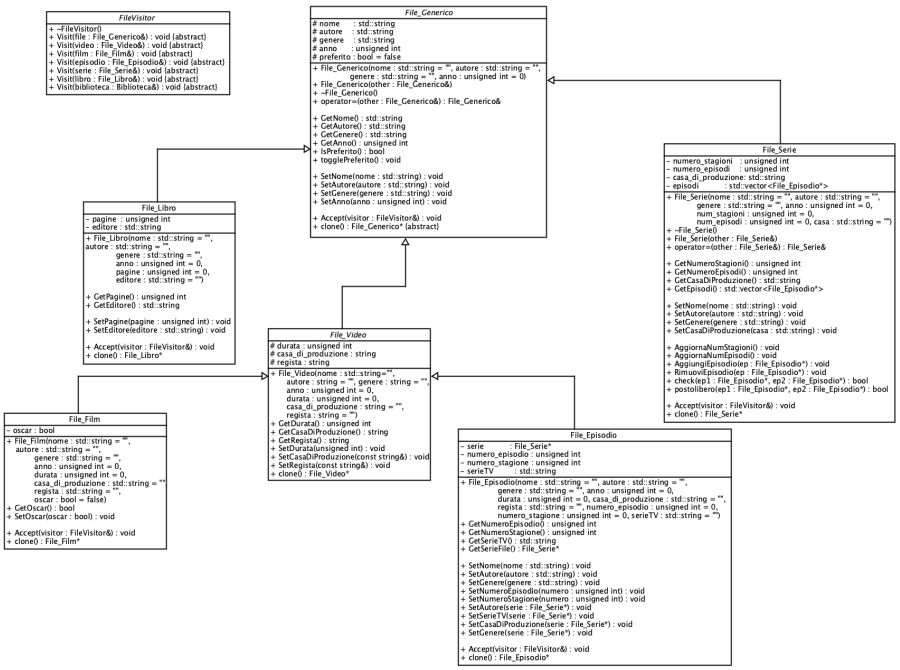


Figura 1 : Diagramma delle classi

La gerarchia delle classi, come mostrato in *Figura 1*, deriva dalla classe astratta ***File_Generico*** che contiene tutte le informazioni comuni di tutti i file e quindi le sottoclassi. In questo caso tutti i file che derivano da questa classe astratta avranno: **nome**, **autore**, **genere**, **anno** e l'indicatore booleano di **preferito**. Per ciascuna classe sono stati definiti opportuni costruttori e distruttori, e opportuni metodi *getter* e *setter* che permettono di interagire con i campi privati delle classi, riducendo l'impatto che una modifica alla progettazione della classe base causerebbe alle classi derivate.

La classe genitore ***File_ Generico*** ha come eredi due classi concrete (***File_ Libro*** e ***File_ Serie***) e una classe astratta (***File_ Video***) che a sua volta ha due classi concrete come eredi (***File_ Film*** e ***File_ Episodio***)

La funzione di ogni classe e i propri attributi sono i seguenti:

- **Libro:** classe concreta che rappresenta un libro tramite gli attributi specifici relativi al numero di pagine e l'editore.
- **Serie:** classe concreta che rappresenta una serie televisiva tramite gli attributi specifici relativi al numero di stagioni, il numero di episodi e la casa di produzione, oltre che un vettore contenente puntatori a ***File_ Episodio***.
- **Video:** classe astratta creata per gestire correttamente gli attributi comuni dei file figli (*regista, durata e casa di produzione*). Da questa ereditano:
  - **Film:** classe concreta che rappresenta un film tramite l'attributo specifico oscar, oltre a quelli ereditati da Video.
  - **Episodio:** classe concreta che rappresenta un episodio di una serie televisiva tramite gli attributi specifici relativi al nome della serie, la stagione di appartenenza, la posizione all'interno della stagione ed un puntatore alla stessa, oltre a quelli ereditati da Video.

3 Polimorfismo

Oltre al polimorfismo presente nelle classi dei File, ad esempio nel metodo di clonazione, è stato implementato anche il design pattern *Visitor*. Tramite il metodo `Accept`, un oggetto File può essere visitato da diverse istanze di `FileVisitor`, permettendo comportamenti diversi a seconda sia del tipo concreto dell'oggetto sia del tipo del visitor, senza ricorrere a controlli espliciti sul tipo dell'oggetto.

La classe astratta `FileVisitor` contiene metodi virtuali puri `visit()` ed è utilizzata per trattare gli oggetti da manipolare durante l'utilizzo dell'**applicazione** in maniera efficiente. In questo modo, le operazioni sono separate dai dati, migliorando la manutenibilità del codice.

I *visitor* concreti implementano comportamenti specifici:

- `GrigliaVisitor` gestisce la visualizzazione dei file in una griglia;
- `ListaVisitor` gestisce la visualizzazione dei file come elementi di una lista;
- `JSonVisitor` si occupa della serializzazione in formato `.JSON`;
- `ModificaVisitor` permette di aggiornare i dati dei file;
- `MostraVisitor` visualizza le informazioni dettagliate degli oggetti.

Grazie a questi *visitor*, l'**applicazione** può estendere facilmente *nuove funzionalità* senza dover modificare le classi dei file.

4 Persistenza dei dati

La *persistenza dati* è implementata attraverso l'uso di un formato strutturato `.JSON`, in particolare affidandosi alla classe ***QJsonObject*** presente all'interno della *libreria* ***Qt***.

L'**applicazione** offre all'utente la possibilità di salvare con nome e di importare dal sistema i file `.JSON` relativi ai tipi di oggetti supportati dal programma attraverso una finestra di dialogo.

Il *salvataggio dell'intera biblioteca* può essere eseguito nei seguenti modi:

- Pulsante `"Salva la biblioteca"`
- Shortcut `"Ctrl+S"` nella finestra principale

Il *salvataggio dei singoli file* può essere eseguito nei seguenti modi:

- Pulsante `"Salva su disco"` nella finestra di visualizzazione relativa al file
- Voce `"Salva"` del menù contestuale accessibile tramite tasto destro del mouse
- Shortcut `"Ctrl+S"` mentre si visualizza il file

Nella *finestra principale* dell'applicazione sono disponibili due pulsanti: `"+ Biblioteca +"`, per *importare un'intera biblioteca* di file `.JSON`, e `"+ File +"`, per *importare un singolo file* `.JSON` dal sistema locale. Se si carica una nuova biblioteca, l'**applicazione** chiede prima se si desidera salvare quella corrente. L'*importazione* di una biblioteca è inoltre accessibile tramite la scorciatoia da tastiera `Ctrl+O`.

Qualora ci si trovasse all'interno della finestra di visualizzazione di una Serie, la scorciatoia `"Ctrl+O"` gestirebbe l'upload di un episodio della serie.

In allegato al progetto, come alla pagina di ***Github***, è possibile trovare una cartella di prova con la quale sperimentare tutte le funzionalità descritte in questo paragrafo.

5 Funzionalità implementate

L'**applicazione** implementa le seguenti funzionalità:

- Creazione di nuovi elementi della libreria tramite GUI (anche tramite scorciatoia `Ctrl+N`)
- Importazione ed esportazione di intere librerie in formato strutturato `.JSON`
- Importazione ed esportazione di singoli elementi in formato strutturato `.JSON`
- Modifica delle caratteristiche di un elemento tramite GUI
- Eliminazione di un singolo elemento
- Marcatura di un elemento come "Preferito" per personalizzare l'esperienza
- Gestione dei file tramite menù contestuale (*accessibile tramite tasto destro del mouse*) senza doverne visualizzare il contenuto
- Visualizzazione dei file in modalità Griglia o Lista
- Filtraggio dei file in ordine alfabetico o cronologico
- Filtraggio dei file per tipo (*Libri, Film, Serie*) cliccanto il tipo di interesse nell'albero di navigazione laterale
- Filtraggio per preferenza tramite pulsante intuitivo a forma di stellina
- Ricerca di elementi per nome, anche parziale
- Utilizzo di scorciatoie da tastiera per velocizzare le operazioni

6 Rendicontazione ore

Attività	**Ore Previste**	**Ore Effettive**
Studio e progettazione	10	11
Sviluppo Del Codice Del Modello	10	11
Studio Del Framework Qt	7	8
Sviluppo Del Codice Della GUI	5	10
Test e debug	5	8
Stesura Della Relazione	3	3
TOTALE	40	51

Tabella 1: Ore di sviluppo

Come evidenziato in *Tabella 1*, lo sforamento delle ore previste ha riguardato principalmente lo sviluppo della GUI e la fase di *Testing* e *Debugging*.

La causa è legata soprattutto alla gestione dei ***File_ Serie*** e dei ***File_ Episodio***. Questa scelta ha reso necessario creare codice *ad hoc*, come la classe `MostraVisitorHelper`, che consente di utilizzare la stessa funzione per visualizzare ogni tipo di file.

Durante la fase di testing si sono verificati anche errori di tipo segmentation fault. Per risolverne uno particolarmente problematico, presente solo su **Ubuntu 22.04** e non su **macOS**, è stato fondamentale l'utilizzo del tool `Valgrind`, un *debugger* che consente di rilevare errori di accesso e perdite di memoria.

7 Ambiente Di Sviluppo

Per testare l'**applicazione** e le varie funzioni è stata utilizzata la *macchina virtuale* messa a disposizione dal corso. Per facilitare la build del progetto è stato creato uno script aggiuntivo, `build_project.sh`, che compila automaticamente tutti i file necessari, mostra una *progress bar* all'utente e prepara l'applicazione per l'esecuzione.

SO:	macOS 15.6.1 24G90 arm64
Editor:	Visual Studio Code
Compilatore:	Apple clang version 17.0.0 (clang-1700.0.13.5)
QT:	Qt version 6.9.2

Tabella 2: Ambiente di sviluppo