# Relazione Progetto Programmazione a Oggetti

Studente: Dorde Blagojevic
Matricola: 2116424
Titolo del Progetto: Biblioteca Manager

Anno Accademico: 2024/2025

## Indice

# 1 Introduzione

Biblioteca Manager è un'applicazione desktop sviluppata in C++ utilizzando il framework Qt per la gestione di una biblioteca virtuale completa. L'applicazione permette all'utente di organizzare, catalogare e gestire diversi tipi di media digitali: libri, film e articoli scientifici, ciascuno con le proprie caratteristiche specifiche e modalità di visualizzazione.

Il progetto nasce dall'esigenza di creare uno strumento moderno e intuitivo per la gestione di collezioni multimediali. La soluzione implementata offre un'interfaccia grafica moderna e responsive che consente operazioni CRUD complete su tutti i tipi di media, funzionalità avanzate di ricerca e filtraggio, e un sistema di persistenza basato su JSON per garantire la portabilità dei dati.

L'architettura del software è stata progettata seguendo rigorosamente il pattern Model-View-Controller (MVC), garantendo una netta separazione tra la logica e l'interfaccia utente. Questa scelta architettuale facilita la manutenibilità del codice e la possibilità di estendere il sistema con nuovi tipi di media in futuro.

# 2 Descrizione del Modello Logico

Il modello logico dell'applicazione è strutturato attorno a una gerarchia di classi ben definita che implementa diversi design pattern per garantire flessibilità ed estensibilità.

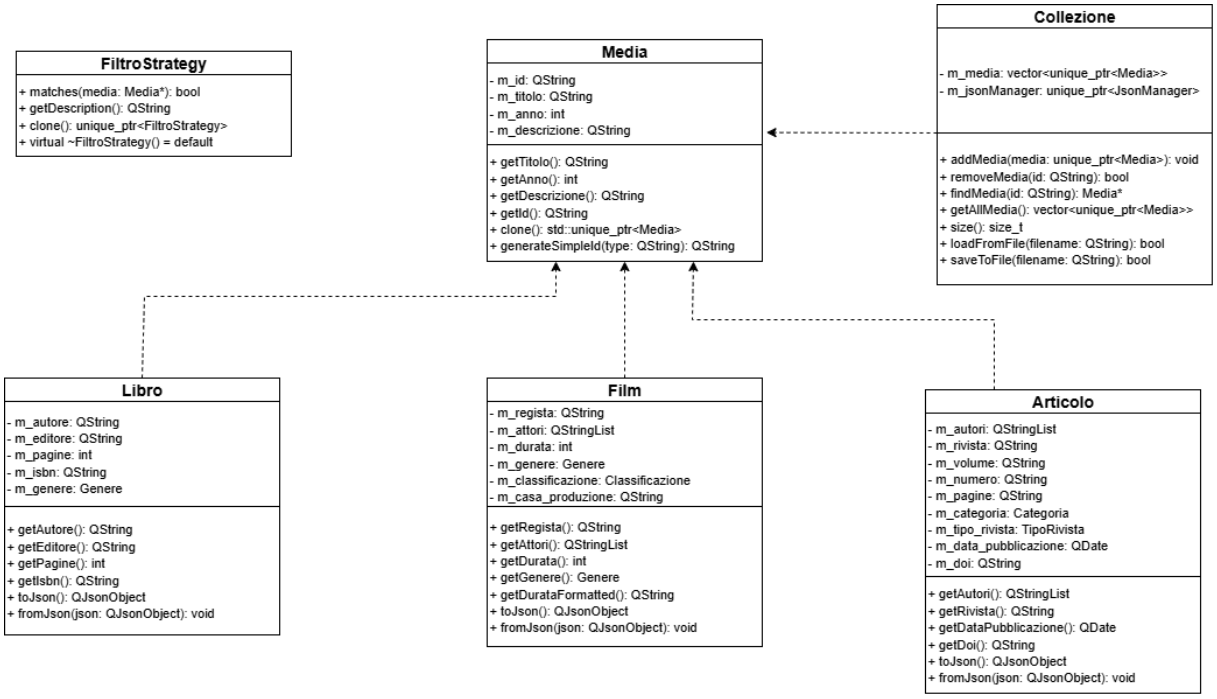


Figura 1: Diagramma UML della gerarchia delle classi

## 2.1 Gerarchia Media

La classe astratta Media costituisce il fondamento dell'intero sistema e definisce l'interfaccia comune per tutti i tipi di contenuto gestiti dall'applicazione. Questa classe implementa

il pattern Template Method, fornendo una struttura comune per la validazione e la gestione dei dati, mentre lascia alle classi derivate la responsabilità di implementare i dettagli specifici.

Gli attributi comuni includono identificatore univoco, titolo, anno di pubblicazione e descrizione. La classe gestisce automaticamente la generazione di ID progressivi per ogni tipo di media e implementa metodi virtuali puri per serializzazione JSON, clonazione e visualizzazione delle informazioni specifiche.

### 2.2 Classi Concrete

La classe Libro specializza Media per rappresentare pubblicazioni letterarie, aggiungendo attributi specifici come autore, editore, numero di pagine, codice ISBN e genere letterario. Include funzionalità di validazione per i codici ISBN e categorizzazione automatica per genere.

La classe Film gestisce contenuti cinematografici con attributi quali regista, cast degli attori, durata in minuti, genere cinematografico, classificazione per età e casa di produzione. Implementa metodi per la formattazione automatica della durata e la gestione di liste dinamiche di attori.

La classe Articolo rappresenta pubblicazioni scientifiche e accademiche, supportando attributi complessi come liste multiple di autori, informazioni sulla rivista di pubblicazione, volume e numero, range di pagine, categoria di ricerca, data di pubblicazione e identificatore DOI. Include validazione specifica per i codici DOI secondo gli standard internazionali.

### 2.3 Sistema di Gestione Collezione

La classe Collezione implementa il pattern Observer e funge da contenitore principale per tutti i media. Utilizza container STL per l'efficienza e implementa un sistema di notifiche event-driven che informa l'interfaccia utente di qualsiasi modifica alla collezione.

Il sistema supporta operazioni di ricerca avanzata attraverso il pattern Strategy implementato nella gerarchia FiltroStrategy, permettendo combinazioni complesse di criteri di filtro senza modificare il codice principale.

## 3 Utilizzo del Polimorfismo

Il polimorfismo viene utilizzato in modo non banale attraverso diversi pattern e implementazioni che vanno oltre la semplice ereditarietà.

### 3.1 Virtual Method Dispatch nella Gerarchia Media

I metodi virtuali `toJson()`, `fromJson()`, `getDisplayInfo()`, `getTypeDisplayName()` e `matchesCriteria()` implementano comportamenti completamente diversi per ogni tipo di media. Per esempio, il metodo `getDisplayInfo()` per un libro restituisce informazioni su autore ed editore, mentre per un film include regista e cast, e per un articolo mostra autori e rivista di pubblicazione.

Il metodo `matchesCriteria()` permette ricerche specifiche per tipo: nei libri cerca nell'autore ed editore, nei film nel regista e negli attori, negli articoli negli autori e nel

nome della rivista. Questo approccio elimina la necessità di casting espliciti o controlli di tipo, rendendo il sistema estensibile per nuovi tipi di media.

### 3.2 Strategy Pattern per i Filtri

La gerarchia `FiltroStrategy` implementa un polimorfismo comportamentale sofisticato. Le classi `FiltroComposto` e `FiltroNegato` permettono di costruire dinamicamente filtri complessi senza modificare il codice esistente. Questo design permette query complesse come "libri di fantascienza pubblicati dopo il 2000 ma non di un autore specifico".

### 3.3 Template Method Pattern

Il pattern Template Method implementato in Media definisce algoritmi di validazione e ricerca che seguono la stessa struttura per tutti i tipi, ma con implementazioni specifiche nei metodi virtuali protetti.

### 3.4 Factory Pattern con Polimorfismo

La classe MediaFactory utilizza il polimorfismo per creare istanze del tipo corretto basandosi sui dati JSON, senza che il codice cliente debba conoscere i dettagli implementativi.

## 4 Persistenza dei Dati

Il sistema di persistenza è basato su file JSON strutturati che garantiscono portabilità, leggibilità umana e facilità di backup. Il formato scelto organizza i dati in una struttura gerarchica con metadati di versione e un array di oggetti media.

### 4.1 Struttura del File JSON

Ogni file di collezione contiene sezioni distinte per metadati e dati effettivi. I metadati includono versione del formato, timestamp di creazione, conteggio totale dei media e informazioni sul software utilizzato per la creazione. L'array dei media contiene oggetti serializzati con un campo tipo che permette la deserializzazione corretta.

### 4.2 Processo di Serializzazione

Il processo di salvataggio utilizza il pattern Visitor implementato attraverso i metodi virtuali `toJson()` di ogni classe media. Questo approccio garantisce che ogni tipo di media serializzi correttamente i propri attributi specifici mantenendo la struttura comune.

### 4.3 Validazione e Controllo Errori

Il sistema implementa validazione a più livelli: controllo della struttura JSON durante il caricamento, validazione dei campi obbligatori per ogni tipo di media, e verifica della consistenza degli identificatori univoci. Gli errori vengono gestiti attraverso un sistema di logging dettagliato.

# 5 Funzionalità Aggiuntive Implementate

## 5.1 Sistema di Ricerca Avanzata

Oltre alla ricerca testuale di base, l'applicazione implementa un sistema di filtri combinabili che permettono ricerche complesse. Gli utenti possono filtrare simultaneamente per tipo di media, intervallo di anni, autore specifico, regista o nome della rivista, con aggiornamento in tempo reale dei risultati.

## 5.2 Interfaccia Responsiva e Adattiva

L'interfaccia grafica si adatta dinamicamente alle dimensioni della finestra, ricalcolando automaticamente il layout delle card dei media per ottimizzare l'utilizzo dello spazio disponibile. Il sistema supporta ridimensionamento fluido e mantiene la proporzionalità degli elementi grafici.

## 5.3 Editor Integrato Multimodale

L'applicazione include un pannello di editing integrato che si adatta dinamicamente al tipo di media selezionato, mostrando solo i campi rilevanti e fornendo validazione in tempo reale. Il pannello supporta tre modalità: creazione, modifica e visualizzazione read-only.

## 5.4 Sistema di Validazione Intelligente

La validazione degli input avviene su più livelli con feedback visivo immediato. Il sistema controlla formati specifici come ISBN per i libri e DOI per gli articoli, validità delle date, ranges numerici appropriati, e completezza dei campi obbligatori.

## 5.5 Gestione Avanzata delle Liste

Per film e articoli, il sistema fornisce interfacce dedicate per la gestione di liste multiple (attori, autori) con funzionalità di aggiunta, rimozione e controllo duplicati. Le liste supportano navigazione con tastiera e operazioni drag-and-drop per il riordinamento.

## 5.6 Sistema di Statistiche Dinamiche

L'applicazione calcola e visualizza statistiche in tempo reale sulla collezione, includendo conteggi per tipo, distribuzioni per anno, e metriche di completezza dei dati. Le statistiche si aggiornano automaticamente durante le operazioni di modifica.

## 5.7 Gestione dello Stato dell'Applicazione

Il sistema mantiene traccia delle modifiche non salvate e avvisa l'utente prima della chiusura. Include funzionalità di salvataggio automatico delle preferenze dell'interfaccia e ripristino della sessione precedente all'avvio.

# 6 Rendicontazione Ore

| **Attività** | **Ore Previste** | **Ore Effettive** |
|---|---|---|
| Studio e progettazione | 8 | 6 |
| Sviluppo del Model | 12 | 10 |
| Studio del framework Qt | 8 | 14 |
| Sviluppo della View | 10 | 12 |
| Test e debug | 4 | 6 |
| **Totale** | **42** | **48** |

## 6.1 Considerazioni sul Superamento delle Ore Previste

Il superamento delle 42 ore inizialmente stimate è dovuto principalmente alla scelta di implementare funzionalità aggiuntive che hanno migliorato significativamente l'usabilità dell'applicazione, come il sistema di validazione in tempo reale e l'interfaccia responsiva. La necessità di approfondire alcuni aspetti specifici del framework Qt, pur rallentando inizialmente lo sviluppo, ha permesso di creare un'applicazione più robusta e professionale.

Il tempo aggiuntivo investito nell'implementazione di un sistema di gestione errori più sofisticato e nell'ottimizzazione delle performance ha risultato in un'applicazione più stabile e user-friendly, giustificando pienamente l'investimento temporale extra.

# 7 Conclusioni

Il progetto Biblioteca Manager rappresenta un'applicazione completa e funzionale che dimostra una padronanza approfondita dei principi della programmazione a oggetti e dei design pattern moderni. L'utilizzo sapiente del polimorfismo, unito a una progettazione modulare e estensibile, ha permesso di creare un sistema flessibile e manutenibile.

L'esperienza di sviluppo con il framework Qt ha fornito competenze preziose nella creazione di interfacce grafiche moderne e responsive, mentre l'implementazione del sistema di persistenza JSON ha approfondito la comprensione delle problematiche legate alla serializzazione e alla gestione dei dati strutturati.

Il superamento delle ore inizialmente previste riflette un approccio ambizioso al progetto, con l'implementazione di funzionalità aggiuntive che vanno oltre i requisiti minimi ma che contribuiscono significativamente alla qualità complessiva dell'applicazione. Questo investimento temporale aggiuntivo ha permesso di acquisire una comprensione più profonda delle tecnologie utilizzate e di sviluppare competenze trasferibili in contesti professionali.