Relazione per il progetto di Programmazione a Oggetti

Biblioteca Virtuale Multimediale

Gruppo
Marchioro Elisa – 2111941
De Guio Filippo – 2113177

Indice

1 Introduzione

Il progetto presentato consiste nello sviluppo di un'applicazione che funge da biblioteca virtuale, in grado di gestire e rendere disponibili contenuti multimediali di diversa natura: libri, film, videogiochi, fotografie e musica. L'obiettivo principale è quello di offrire una piattaforma unificata e facilmente accessibile, che permetta all'utente di esplorare e organizzare le proprie risorse culturali e di intrattenimento in un unico ambiente digitale.
L'applicazione consente di creare, modificare, eliminare e visualizzare i contenuti, ciascuno caratterizzato da proprietà specifiche. Ad esempio, i libri sono corredati da autore e anno di pubblicazione, i film da regista e durata, i videogiochi da piattaforma di riferimento, la musica da artista e genere, mentre le fotografie possono includere informazioni su luogo e data di scatto. Questa varietà di tipologie di contenuto rappresenta un'ottima occasione per sfruttare a fondo i principi della programmazione a oggetti, in particolare l'ereditarietà e il polimorfismo, per modellare entità con attributi comuni ma anche con caratteristiche distintive.
Il valore aggiunto del progetto risiede nella capacità di integrare media diversi in una singola applicazione, garantendo una gestione coerente e modulare. Inoltre, l'architettura progettata è stata pensata per essere estensibile, permettendo in futuro di arricchire la piattaforma con nuove tipologie di contenuto o funzionalità avanzate, come sistemi di ricerca personalizzati o suggerimenti basati sulle preferenze dell'utente.

2 Descrizione del modello

Il modello logico della biblioteca virtuale si basa su una gerarchia di classi progettata per rappresentare contenuti multimediali eterogenei, come libri, film, videogiochi, fotografie e musica. In cima alla gerarchia c'è la classe astratta `Product`, che raccoglie le informazioni comuni a tutti gli oggetti gestiti: nome, descrizione, genere, nazione, costo, stelle (recensione) con relativi metodi getter e setter. Questa struttura consente di definire comportamenti comuni pur lasciando spazio a specializzazioni concrete.
La gerarchia dei contenuti concreti include diverse sottoclassi: per esempio, la musica, i film e i videogiochi sono rappresentati da classi derivate di `DigitalProduct` che aggiungono attributi specifici come autore, durata o piattaforma. Le fotografie e i brani musicali seguono uno schema analogo e sono sottoclassi di `PhysicalProduct`. Questa suddivisione permette di sfruttare l'ereditarietà per modellare entità con proprietà comuni e caratteristiche distinte, garantendo al contempo un'architettura modulare e facilmente estendibile.
L'arricchimento dinamico delle classi è garantito dal pattern *Visitor*, tramite le classi astratte `Visitor` e `InfoVisitor`.
Un aspetto fondamentale del progetto è la gestione dei dati persistenti tramite file JSON e XML. Per ciascun formato sono state realizzate librerie separate, ciascuna con visitor dedicati per la lettura e la scrittura, garantendo che il caricamento e il salvataggio dei dati siano autonomi e indipendenti tra loro.
Questa architettura consente una gestione coerente e scalabile dei contenuti multimediali, permettendo di aggiungere nuove tipologie di media o funzionalità avanzate senza modificare le classi esistenti, grazie all'uso combinato dei principi di programmazione a oggetti e dei pattern *Visitor*.
All'interno dell'applicazione, un ruolo fondamentale è svolto dal `LibraryModel` e dal `LibraryFilterProxyModel`.
Il `LibraryModel`, estendendo `QAbstractTableModel`, rappresenta la struttura dati principale che gestisce l'elenco dei prodotti presenti nella libreria virtuale. Si occupa non solo di fornire le informazioni testuali (ad esempio il nome dell'oggetto), ma anche di caricare e visualizzare le immagini associate ad ogni prodotto. Grazie a questa classe, l'interfaccia grafica può accedere in modo uniforme ai dati,

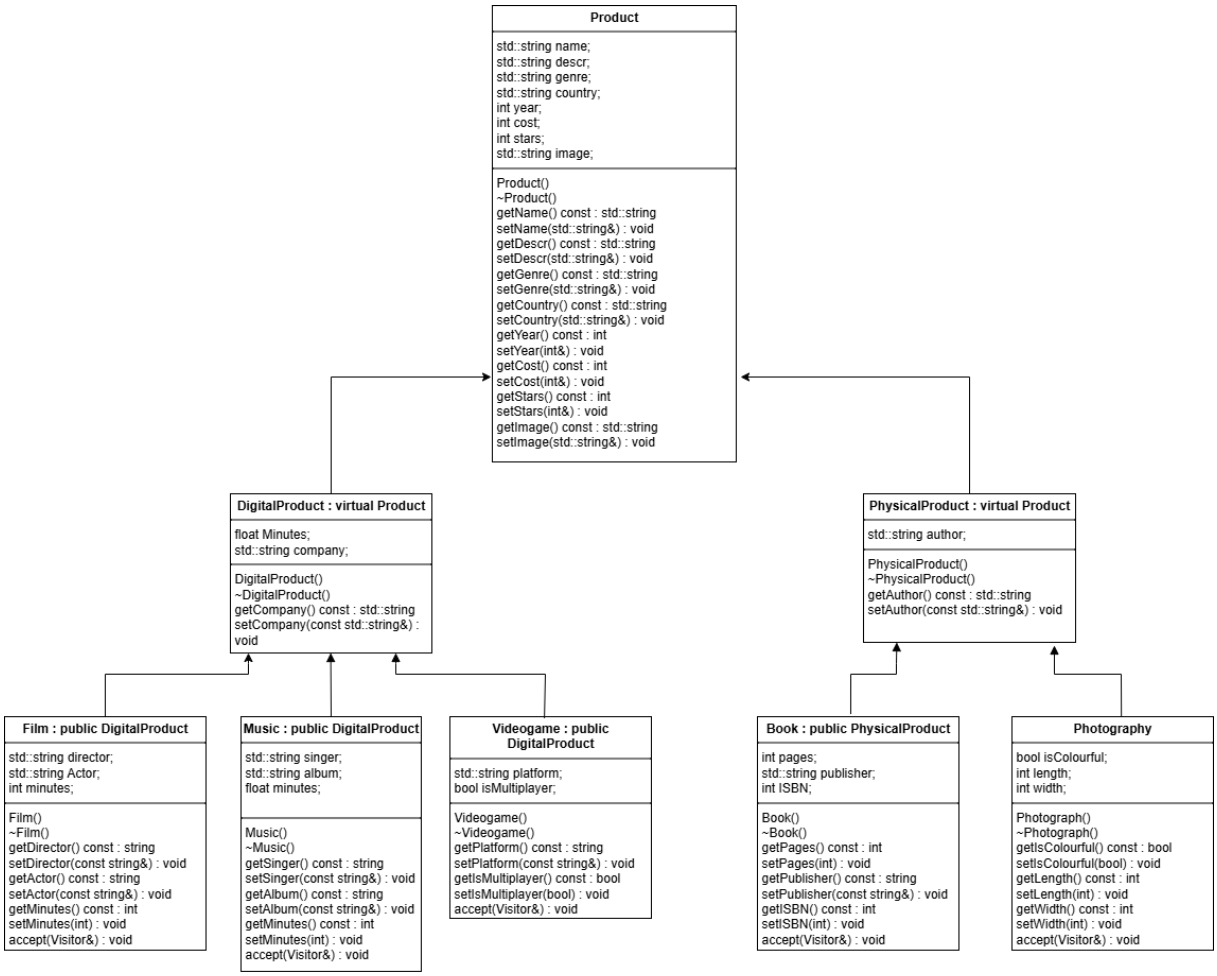

Figura 1: Diagramma delle classi del modello.

permettendo l'aggiornamento dinamico della vista ogni volta che vengono aggiunti, rimossi o modificati elementi.

Il `LibraryFilterProxyModel`, invece, funge da strato intermedio tra il modello e la vista. La sua funzione principale è permettere il filtraggio e la ricerca dei prodotti in base a criteri specifici, come la tipologia (Film, Libro, Musica, ecc.) o il nome inserito dall'utente nella barra di ricerca. Questo approccio separa in maniera chiara la logica di gestione dei dati dalla loro visualizzazione e filtraggio, rendendo l'architettura dell'applicazione più modulare, estensibile e semplice da mantenere.

3 Polimorfismo

Questa sezione descrive l'utilizzo non banale del polimorfismo nella biblioteca virtuale multimediale. Per "banale" si intende un uso del polimorfismo scontato, come rendere virtuali i distruttori in una superclasse o definire metodi virtuali per piccole variazioni di comportamento: questi casi non richiedono documentazione.

Nel progetto, il polimorfismo è sfruttato in maniera consapevole tramite il design pattern *Visitor*, implementato nella gerarchia `Product` e nelle sottoclassi concrete (`Film`, `Book`, `Music`, `Videogame`, `Photograph`). In particolare, l'`InfoVisitor` costruisce dinamicamente l'interfaccia grafica per ciascun tipo di oggetto, rispettando le caratteristiche specifiche dei contenuti. Questo approccio permette di:

- mostrare informazioni comuni a tutti i prodotti (*title*, *description*, *genre*...);

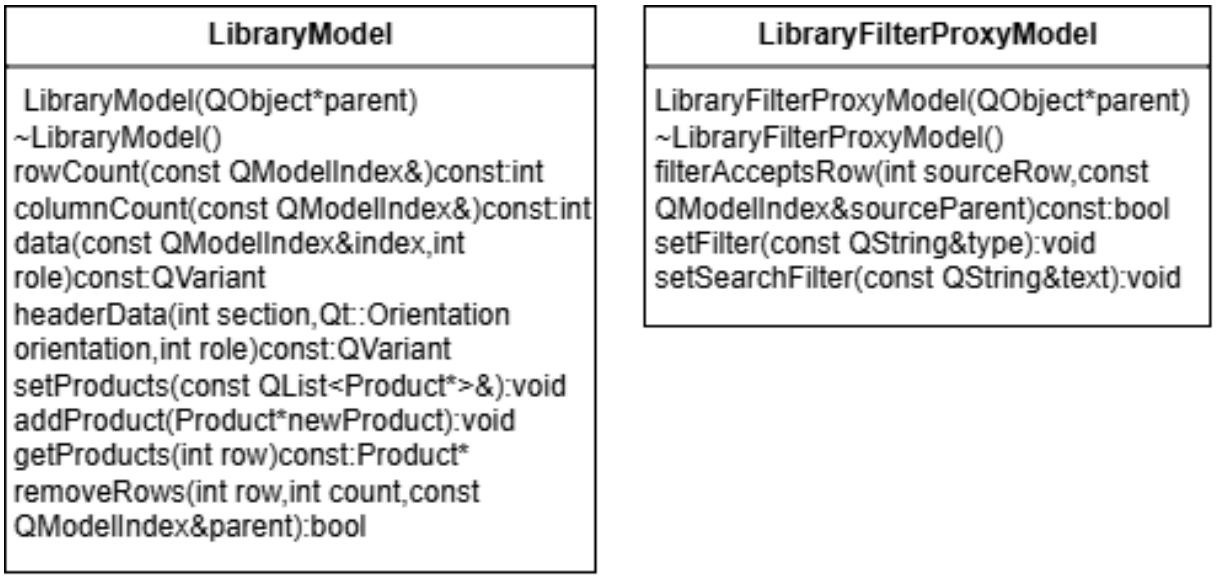

Figura 2: Diagramma di LibraryModel e LibraryFilterProxyModel.

- aggiungere campi specifici a seconda del tipo concreto, come regista e attori per i film, autore e pagine per i libri, cantante e album per la musica, piattaforma e multiplayer per i videogiochi, autore e dimensioni per le fotografie;
- gestire i widget in maniera uniforme, mantenendo il codice modulare e facilmente estendibile.

Il polimorfismo consente quindi all'applicazione di invocare il metodo `accept` sugli oggetti `AbstractItem` e delegare all'`InfoVisitor` la costruzione del widget corretto, senza controlli condizionali sul tipo dell'oggetto. In altre parole, ogni oggetto "si comporta secondo il suo tipo concreto", e l'interfaccia si adatta automaticamente alle caratteristiche specifiche.
Inoltre, l'uso combinato di polimorfismo e *Visitor* consente di:

- aggiornare dinamicamente i dati mostrati nei widget;
- abilitare la modifica di campi specifici tramite `enableEdit` e applicare le modifiche con `applyEdits`, mantenendo le operazioni separate per ciascun tipo di media;
- integrare facilmente nuove tipologie di media future senza modificare il visitor esistente.

Infine, il polimorfismo è impiegato anche nella gestione della visualizzazione dei contenuti multimediali tramite strategie diverse di layout (elenco, griglia, teaser, full view), rendendo il sistema flessibile, modulare e pronto per estensioni future.

4 Persistenza dei dati

La persistenza dei dati nella biblioteca virtuale multimediale è realizzata tramite due formati separati e autonomi: JSON e XML. Entrambi i formati contengono un catalogo di prodotti, ma vengono gestiti da classi e visitor distinti, per garantire modularità e facilità di estensione.

JSON

Per i file JSON, ogni catalogo è rappresentato come un array di oggetti, ciascuno dei quali contiene gli attributi comuni a tutti i prodotti (nome, descrizione, genere, paese, anno di pubblicazione, prezzo, stelle,

percorso immagine) e quelli specifici della tipologia (ad esempio regista e attori per i film, autore e pagine per i libri). Il campo `"type"` permette di distinguere tra Film, Book, Music, Videogame e Photograph.
La classe `JsonReader` gestisce la lettura dei file JSON, creando istanze delle classi concrete corrispondenti in base al tipo specificato. La funzione `readAll` apre il file, verifica che contenga un array e itera su ciascun elemento, delegando la costruzione dell'oggetto al metodo dedicato (`readFilm`, `readBook`, ecc.). Questa separazione permette di aggiungere nuovi tipi di media senza modificare le classi già esistenti.

XML

I file XML sono trattati in modo analogo ma utilizzando la classe `XmlReader`, che si occupa di leggere i prodotti tramite un parser a streaming (`QXmlStreamReader`). Ogni elemento `<Product>` contiene un attributo `type` che identifica la sottoclasse concreta, mentre i sotto-elementi rappresentano gli attributi comuni e specifici del prodotto. La lettura è realizzata attraverso metodi dedicati per ciascun tipo di media, come `readFilm`, `readBook`, `readMusic`, `readVideogame` e `readPhotograph`.
Questa gestione separata consente di mantenere una struttura modulare: le librerie JSON e XML non si influenzano a vicenda, e ciascun formato ha i propri visitor e strumenti di parsing. In questo modo, la libreria virtuale può leggere e scrivere cataloghi in entrambi i formati senza conflitti, facilitando la manutenzione e l'eventuale estensione a nuovi tipi di media o a formati di persistenza differenti.

5 Funzionalità implementate

La GUI dell'applicazione è stata progettata per rendere semplice e intuitivo l'uso delle funzionalità della biblioteca digitale, pur offrendo alcune caratteristiche avanzate che potrebbero non essere immediatamente visibili. La finestra principale consente di navigare tra l'elenco dei prodotti e i dettagli di ciascun articolo tramite uno `QStackedWidget`, e integra meccanismi di ricerca, filtro e gestione della persistenza dei dati.
Le funzionalità implementate possono essere suddivise in due categorie principali: funzionali ed estetiche.

Funzionalità

- Gestione di cinque tipologie di prodotti: `Film`, `Music`, `Videogame`, `Book` e `Photograph`.
- Caricamento dei dati da file `JSON` e `XML` tramite i reader dedicati (`JsonReader` e `XmlReader`).
- Salvataggio dei dati modificati in formato `JSON` o `XML` tramite i visitor `JsonWriterVisitor` e `XmlWriterVisitor`.
- Ricerca dei prodotti tramite una barra di ricerca (`QLineEdit`) con aggiornamento in tempo reale dei risultati.
- Filtri per tipologia di prodotto attraverso pulsanti dedicati (All, Book, Film, Music, Videogame, Photograph), integrati con un `QSortFilterProxyModel` per gestire la pertinenza dei risultati.
- Visualizzazione dettagliata di ciascun prodotto nella schermata dedicata con possibilità di tornare alla lista principale.
- Aggiunta e rimozione di prodotti tramite dialog dedicati, con aggiornamento automatico del file di persistenza.

- Gestione coerente della persistenza dei dati: il formato di salvataggio viene rilevato automaticamente dal path del file caricato, evitando errori di compatibilità.

Funzionalità estetiche

- Barra dei menù personalizzata in alto con collegamento diretto alle azioni di caricamento e aggiunta prodotti.
- Layout principale diviso in due sezioni: pulsantiera verticale a sinistra per i filtri e area principale per la lista e la visualizzazione dettagliata.
- Utilizzo di icone, immagini di anteprima e stili grafici coerenti per ogni tipologia di prodotto.
- Effetti grafici per i pulsanti e la lista, inclusi gradiente, bordi arrotondati, ombre e cambio di colore al passaggio del mouse.
- Ombreggiature e stili grafici applicati a `QListView`, `QLineEdit` e pulsanti per un effetto moderno e tridimensionale.
- Possibilità di navigare tra schermata principale e dettagli di un prodotto tramite interfaccia con animazioni e gestione degli oggetti dinamici.
- Ridimensionamento dinamico della finestra e dei widget, mantenendo proporzioni e spaziatura coerenti.
- Ogni prodotto ha una propria visualizzazione personalizzata nella schermata dei dettagli, con layout adattivo per immagini, testi e pulsanti di azione.

Le funzionalità elencate integrano e ampliano quanto richiesto dalle specifiche del progetto, offrendo un'interfaccia completa e moderna che combina gestione dei dati, ricerca avanzata e un'esperienza utente gradevole e intuitiva.

6 Rendicontazione ore

Questa sezione riporta le attività svolte durante lo sviluppo del progetto, indicando per ciascuna le ore previste e quelle effettivamente impiegate.

Attività	**Ore Previste**	**Ore Effettive**
Studio e progettazione	8	8
Sviluppo del codice del modello	8	11
Studio del framework Qt	8	10
Sviluppo del codice della GUI	8	19
Test e debug	4	3
Stesura della relazione	4	2
Totale	40	53

Tabella 1: Rendicontazione delle ore previste ed effettive per ciascuna attività.

Il monte ore è stato superato principalmente a causa della risoluzione di bug complessi legati alla gestione della memoria (‘delete‘), alla scrittura su file XML (‘XmlWriter‘) e al salvataggio dei dati nella GUI. Queste problematiche hanno richiesto numerosi test e debug per garantire la correttezza della persistenza dei dati e la stabilità complessiva dell'applicazione. Inoltre, lo sviluppo della GUI ha richiesto più tempo del previsto per integrare correttamente le funzionalità di filtro, ricerca e navigazione tra diverse

schermate e i relativi widget. Anche lo studio si è rivelato più complesso del previsto, in particolare la ricerca di elementi specifici nelle documentazioni di Qt.

7 Suddivisione lavoro di gruppo

Attività	**Filippo**	**Elisa**
Studio di Qt	X	X
Progettazione	X	X
Sviluppo modello logico	X	
Model e FilterModel		X
JSON (e classi correlate)		X
XML (e classi correlate)	X	
Funzionalità Add Product	X	
Funzionalità Delete	X	
Funzionalità Modify		X
Funzionalità Save		X
MainPage e InfoPage	X	X
Test e Debugging	X	X