oggetto	Relazione progetto Programmazione a Oggetti
gruppo	Bertin Samuele, mat. 2042387
titolo	Libreria

Introduzione

L’applicazione “Libreria” è un semplice software desktop che permette all’utente di creare, cercare, visualizzare, modificare, cancellare e salvare elementi di diversi tipi (in questo progetto Book, Film e Article) tramite un’interfaccia grafica pulita e semplice.

L’argomento del progetto è stato assegnato dal corso, nonostante ciò ho scelto di personalizzare e approfondire aspetti che ritengo più didattici e significativi per il mio percorso: in particolare mi sono concentrato sul design polimorfico della gerarchia dei media, sulla separazione netta tra modello e interfaccia, e sulla robustezza della persistenza.

Descrizione del modello

Il modello logico è composto da una gerarchia di classi che rappresentano i vari tipi di media gestiti dall’applicazione, dal MediaManager e da alcune classi di supporto per implementare i design pattern Visitor e il comportamento osservabile (observer) necessario per sincronizzare modello e vista. Di seguito è riportato il diagramma UML.

La gerarchia delle entità parte da una classe astratta Media che incapsula l’informazione comune a tutti i media: il title e l’interfaccia virtuale per le operazioni fondamentali. Per ogni attributo principale sono previsti metodi getter/setter nei tipi concreti dove necessario; in particolare la classe base fornisce metodi virtuali puri per la serializzazione (toJson() / fromJson()), per la clonazione polimorfica (clone()) per la costruzione dinamica di un widget di dettaglio (createDetailWidget(QWidget*))e per l’esecuzione di un’azione specifica del media (performAction()).

La sotto-gerarchia concreta comprende Book, Film e Article, ciascuna con attributi specifici e metodi corrispondenti:

- Book definisce author e year ed implementa toJson()/fromJson() per salvare/leggere questi campi, getSummary() per una breve descrizione e createDetailWidget() per fornire la vista di dettaglio adatta ad un libro;

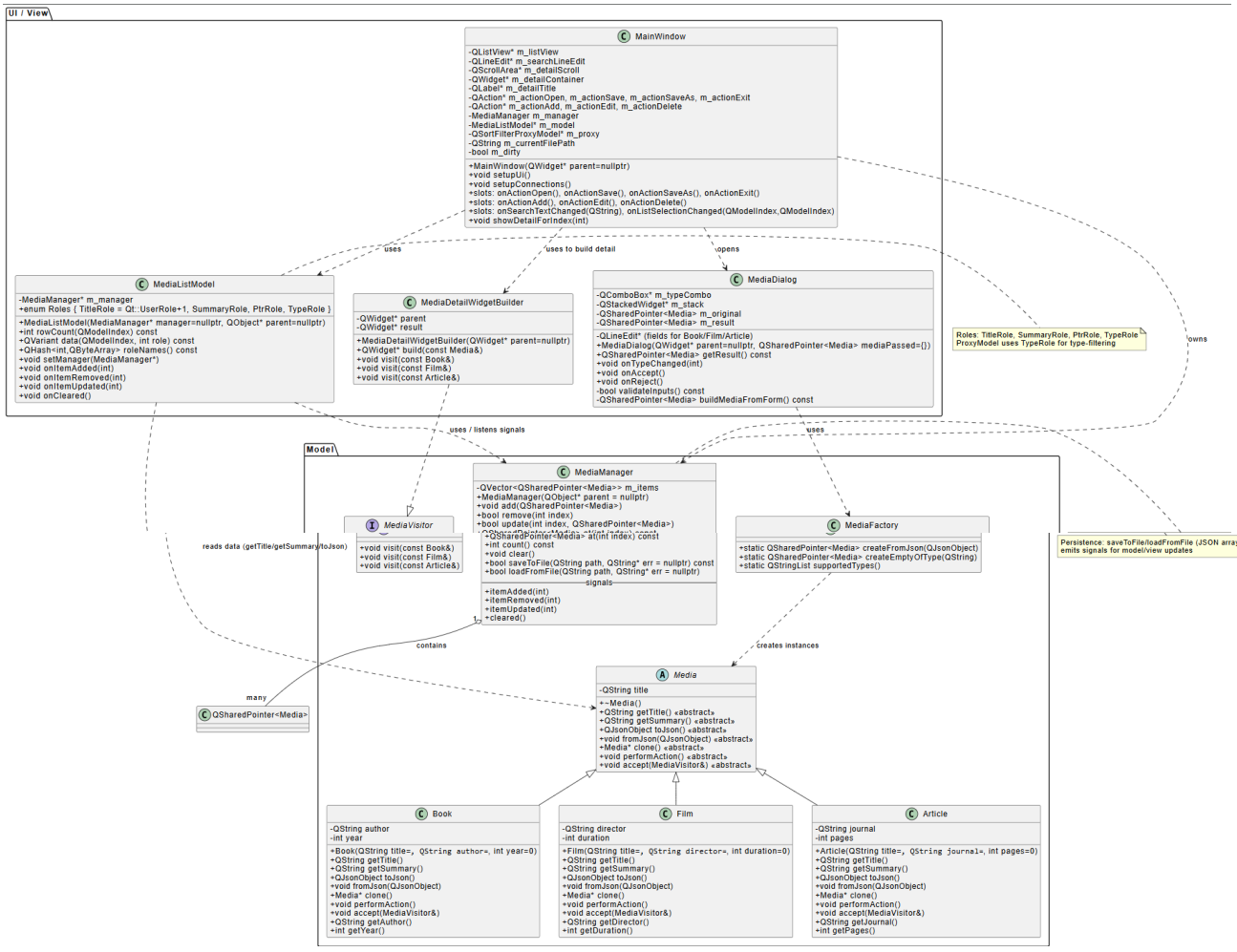
- Film definisce director e duration e fornisce implementazioni analoghe adattate per Film;

- Article definisce journal e pages e fornisce implementazioni analoghe adattate per Article.

Queste implementazioni concrete non si limitano a semplici getter: ciascuna sovrascrive metodi che si comportano in modo profondamente diverso a seconda del tipo dinamico dell’oggetto.

La classe MediaManager agisce come contenitore della collezione: offre operazioni CRUD ben definite (add(), remove(index), update(index, media), at(index), count(), clear())oltre ai metodi di persistenza saveToFile(path) e loadFromFile(path) che leggono/scrivono file JSON strutturati. MediaManager emette segnali Qt ogni volta che lo stato della collezione cambia; questi segnali sono l’equivalente del pattern Observer e permettono al modello di interfaccia (MediaListModel)e

ad altre componenti di aggiornarsi in modo reattivo senza dipendere direttamente dall’interfaccia grafica.



l layer di integrazione con Qt è rappresentato, come accennato in precedenza, da MediaListModel. Il modello espone ruoli personalizzati (TitleRole, SummaryRole, TypeRole, PtrRole) che forniscono tutte le informazioni necessarie per visualizzare e filtrare gli elementi. MediaListModel implementa i metodi necessari per emettere chiamate di modifica del modello (beginInsertRows, ecc) quando opportuno, garantendo così l’aggiornamento immediato e corretto delle viste Qt. In questo modo la separazione netta tra modello logico e interfaccia è mantenuta: il modello può essere riutilizzato in contesti non-GUI.

Per la deserializzazione polimorfica è presente MediaFactory, che legge il campo discriminante type (Book, Film o Article) dal QjsonObject e restituisce un QsharedPointer<Media> dell’appropriato sottotipo, chiamando fromJson() per riempire gli attributi.

Per costruire l’interfaccia di dettaglio in modo polimorfico è stato adottato il pattern Visitor/Builder: MediaDetailWidgetBuilder visita l’istanza concreta e costruisce un widget Qwidget * contenente i campi specifici.

Per l’editing/creazione degli oggetti è stato realizzato MediaDialog, un dialog dinamico che usa un QstackedWidget per mostrare la form appropriata in base al tipo selezionato.

La ricerca e i filtri sono implementati tramite QSortFilterProxyModel: vengono usati TitleRole per la ricerca testuale e TypeRole per i filtri per tipo.

Il sistema usa i segnali e gli slot di Qt come meccanismo observer per mettere in comunicazione le parti dell'applicazione. Quando il MediaManager aggiunge, rimuove o modifica un elemento, invia un segnale: il MediaListModel (che funge da osservatore) lo riceve e a sua volta notifica l'aggiornamento.

Polimorfismo

L'applicazione soddisfa il requisito non banale del polimorfismo in diversi punti:

- la gerarchia Media espone metodi virtuali usati in modo differenziato come toJson()/fromJson(), createDetailWidget(), performAction();

- il pattern Visitor viene utilizzato per costruire dinamicamente il widget di dettaglio in base al tipo concreto dell'oggetto passato (così si evita l'utilizzo di getType());

- il pattern clone() è usato per creare copie polimorfe degli oggetti.

Persistenza dei dati

La persistenza dei dati è stata implementata in formato JSON e supporta:

- formato array-foot: il file è un array di oggetti media;

- formato object-root con campo items: ({format: library_v1, items: [...]})

Le funzioni di caricamenti (loadFromFile) e di salvataggio (saveToFile) effettuano controlli robusti: apertura del file, parsing, validazione della struttura e reporting degli errori. Quando si carica un file, createFromJson viene chiamata per tradurre ogni QJsonObject in un QsharedPointer<Media> del tipo corretto. Elementi non riconosciuti e malformati sono saltati con warning.

Funzionalità implementate

Funzionalità logiche:

- CRUD completo per Book/Film/Article tramite MediaManager;

- creazione polimorfica da JSON e serializzazione su file;

- notifiche tramite segnali Qt per aggiornare il modello e la view.

Funzionalità grafiche:

- MainWindow costruita interamente via codice con:

    - barra dei menù (apri, salva, salva come, esci);

    - top bar con campo di ricerca e pulsanti filtro (All, Book, Film e Article);

    - area principale con lista a sinistra e descrizione con titolo del media a destra;

    - bottom bar con pulsanti Aggiungi, Modifica ed Elimina;

- MediaDialog per la creazione/modifica dei media;

- inserimento di media di esmpio all'avvio dell'applicazione tramite sample_library.json

- conferme per azioni distruttive (elimina), disabilitazioni di azioni non valide (provare ad eliminare un media senza aver selezionato qualcosa);

- generazione dinamica di widget di dettaglio tramite MediaDetailWidgetBuilder.

Le funzionalità elencate sono intese in aggiunta a quanto richiesto dalle specifiche del progetto.

Rendicontazione ore

Attività	Ore Previste	Ore Effettive
Studio e progettazione	10	8
Sviluppo del codice del modello	10	14
Studio del framework Qt	10	9
Sviluppo del codice della GUI	10	12
Test e debug	5	8
Stesura della relazione	5	4
totale	50	**55**

Il monte ore è stato leggermente superato perché l’implementazione ha richiesto più tempo del previsto: in particolare ho dedicato molte ore a test e debug per individuare e correggere errori di runtime non evidenti inizialmente.
Questa attività extra è servita a rendere il software più robusto.
Nel complesso il lavoro ha richiesto più iterazioni di verifica rispetto alla stima iniziale, ma ha migliorato la qualità finale del progetto.