2019

UNIVERSITÀ DEGLI STUDI DI PADOVA

Progetto di Programmazione ad Oggetti

UNIVERSITÀ DEGLI STUDI DI PADOVA

GIUSEPPE VITO BITETTI 1143329

Sommario

1 Introduzione 1
1.1 Abstract 1
1.2 Funzionalità 1
2 Manuale d'uso 1
2.1 Apertura e prima finestra 1
2.2 Barra menù 2
2.2.1 File 2
2.2.2 Strumenti 2
2.3 *Tabs* 2
2.3.1 Tabella di visualizzazione 2
2.3.2 Menù relativo 2
2.4 Finestre di aggiunta/modifica di un file 3
3 Progettazione 3
3.1 Model 4
3.1.1 Gerarchia 4
3.1.2 Classe Container 4
3.1.3 Classe Model 5
3.2 GUI (Graphical User Interface) 5
3.2.1 Classi AddDialog e AddDialogLib 5
3.2.2 Classe SingleTab 5
3.2.3 Classe TabWindow 5
3.2.4 Classe MainWindow 5
3.3 Gestione dei Dati 6
3.4 Polimorfismo 6
3.5 Gestione degli errori 6
3.6 Scelte Implementative 6
4 Conclusioni 7
4.1 Timeline 7
4.2 Ambiente di sviluppo e Compilazione 7
4.3 Note dello sviluppatore 8

1 Introduzione

1.1 Abstract

Si vuole realizzare un programma per la gestione e l'organizzazione di raccolte di file di varia natura. Il programma deve gestire raccolte di file differenti in quanto ogni libreria è caratterizzata non dalla similarità del tipo di file bensì da un'organizzazione decisa dall'utente.

Le tipologie dei file di base messi a disposizione sono:

Film: sono file video caratterizzati da una estensione specifica del *"genere"*.

File Generico: sono file di varia natura con un'estensione variabile non ricollegabile al tipo stesso.

Immagine: sono file immagine caratterizzati da una estensione specifica del *"genere"*.

Ogni raccolta è caratterizzata da un titolo e dalla possibilità di ricercare in ogni libreria un file per i suoi attributi.

1.2 Funzionalità

Il programma permette di gestire più librerie simultaneamente, vengono resi disponibili le seguenti funzionalità:

- Caricamento e salvataggio dell'intera raccolta di librerie in file JSON.
- Aggiunta di una nuova libreria.
- Eliminazione di una libreria.
- Aggiunta di un singolo file ad una specifica libreria.
- Eliminazione singola e multipla dei file in una specifica libreria.
- Visualizzazione di un singolo file da una specifica libreria.
- Ricerca per attributi dei file in una specifica libreria.
- Visualizzazione tramite tabella dei file presenti in una libreria.
- Visualizzazione delle librerie tramite *tabs* apposite.
- Visualizzazione degli avvisi in caso di errori.

2 Manuale d'uso

2.1 Apertura e prima finestra

All'apertura del programma viene visualizzata una finestra con all'interno una barra menù e una *tab "Istruzioni"* contenente le istruzioni di base. In seguito alla creazione della prima libreria questa *tab* scomparirà e riapparirà se non saranno presenti librerie.

Il programma non esegue alcun caricamento da file all'avvio in quanto l'utente può decidere se caricare un salvataggio oppure iniziare una nuova raccolta di librerie.

2.2 Barra menù

La barra menù è molto semplice, è composta da due pulsanti ***File*** e ***Strumenti***.

2.2.1 File

Questo pulsante apre un menu a tendina che permette la gestione delle raccolte tramite file. Presenta tre pulsanti:

- ***Apri***: Questo pulsante aprirà una nuova finestra di navigazione per selezionare il file JSON da cui caricare le raccolte. In caso il file selezionato sia non valido o corrotto il *contesto*, ossia tutti i dati inseriti finora nell'applicazione non verranno eliminati e si potrà continuare a lavorare sui dati stessi; altrimenti i dati presenti verranno eliminati e verranno caricati i dati presenti nel file selezionato.
- ***Salva***: Questo pulsante è utilizzabile solo quando sono stati inseriti dati nel programma (anche una libreria vuota è valida) sia via caricamento tramite file che tramite inserimento manuale. Il pulsante si disattiverà non appena non ci saranno più dati validi nel programma.
- ***Esci***: Permette di uscire dal programma. Cliccando su questo pulsante o sulla "***X***" della finestra il programma uscirà direttamente senza chiedere di salvare e quindi perdendo tutti i dati modificati/aggiunti nel programma.

2.2.2 Strumenti

Contiene un solo pulsante, ***Aggiungi libreria***, il quale permette di aggiungere una libreria alla raccolta. Premendolo verrà creata una nuova finestra nella quale verrà richiesto l'inserimento del titolo della nuova libreria.

2.3 *Tabs*

Ogni *tab* è divisibile in due sezioni: un menù statico sulla destra ed una tabella di visualizzazione sulla sinistra.

2.3.1 Tabella di visualizzazione

Questa tabella è composta da 5 colonne che rappresentano gli attributi principali dei file ed un numero variabile di righe in quanto ogni riga rappresenta un file. È possibile ordinare in ordine alfabetico (crescente o decrescente) per ogni attributo. Questa visualizzazione mostra i dati principali e in maniera ridotta.

L'interazione con la tabella è garantita dalla possibilità di selezionare più oggetti contemporaneamente e scorrere la tabella sull'asse verticale senza modificare le dimensioni finestra.

2.3.2 Menù relativo

Ogni *tab* è accostata da un menù che riguarda la specifica *tab* selezionata. Esso è composto da una prima parte istanziata alla ricerca e da una seconda utilizzata per la gestione degli oggetti presenti nella libreria.

- ***Ricerca***: la barra di ricerca permette all'utente di ricercare tramite stringa, la ricerca è dinamica, ossia i risultati cambieranno man mano che la stringa cambia. Si offre la possibilità di selezionare il campo di ricerca, basato sugli attributi, tramite il menu a tendina posto sopra la barra di ricerca.
- ***Bottoni***: questi bottoni sono relativi alla singola libreria selezionata, questi insieme è composto da sei bottoni distinti:

- ***Aggiungi file***: premendolo si aprirà una finestra dove sarà possibile inserire i dati del nuovo file.
- ***Modifica file***: premendolo si aprirà una finestra dove sarà possibile modificare alcuni dati del file.
- ***Rimuovi file***: premendolo si rimuoveranno tutti i file selezionati.
- ***Visualizza File***: premendolo si aprirà una finestra che mostrerà tutti i dati di un file nella loro interezza.
- ***Modifica Libreria***: premendolo si aprirà una finestra dove sarà possibile modificare i dati della libreria selezionata.
- ***Rimuovi libreria***: premendolo si rimuoveranno tutti i file nella libreria nonché la libreria stessa.

2.4 Finestre di aggiunta/modifica di un file

La finestra è composta da differenti caselle modificabili:

- ***Nome file***: è il titolo del file e potrà essere modificata.
- ***Dimensione file***: è la dimensione del file espressa in KB e potrà essere modificata, questa deve essere maggiore uguale a 0, un valore minore di 0 non verrà accettato.
- ***Descrizione***: è la descrizione del file e potrà essere modificata, è utile per avere una maggiore comprensione del file memorizzato.
- ***Estensione file***: è l'estensione del file, l'inserimento del file potrà andare a buon fine se l'estensione è valida per la ***Categoria*** selezionata.
- ***Categoria***: definirà l'appartenenza del file a categorie predefinite, quest'ultime supporteranno alcune estensioni. Le categorie presenti saranno selezionabili da un menu a tendina e sono:
  - ***File generico***: include tutti i file con una qualsiasi estensione.
  - ***Immagine***: include tutti i file con una estensione del tipo "jpeg" o "jpg" oppure "png".
  - ***Film***: include tutti i file con una estensione del tipo "mp4" o "mkv" oppure "avi".

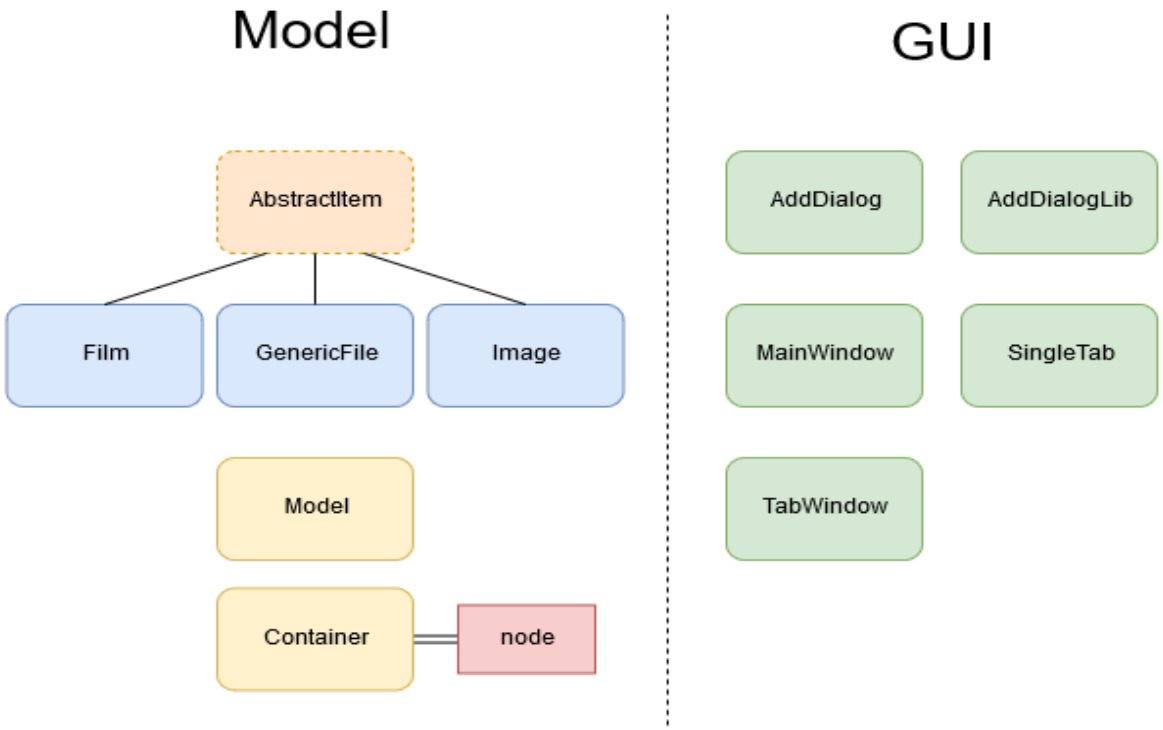
  Un ***File generico*** può avere un'estensione appartenente ad una qualsiasi categoria. Inoltre, una volta creato il file non sarà possibile modificare la sua categoria. Se al momento dell'inserimento l'estensione non è compatibile con la categoria selezionata verrà mostrato un errore.

Tutti i campi compilabili possono essere lasciati vuoti, così facendo i relativi campi verranno "riempiti" con una stringa vuota o 0.

3 Progettazione

Lo sviluppo del progetto è avvenuto seguendo il metodo ***Model-View*** di *Qt* con sviluppo ***TDD*** (*Test driven develpment*) e metodologia ***Bottom-Up***. Si è scelto di utilizzare il ***TDD*** in quanto fornisce del codice "*sicuro*" e relativamente affidabile da poter essere utilizzato in seguito grazie alla metodologia ***Bottom-Up***, inoltre questo approccio porta ad una riduzione delle tempistiche di *debugging*.

Oltre alla gerarchia sono stati realizzati *GUI* e *Model* basandosi su oggetti già presenti nella libreria *Qt*, nonché un oggetto che consentisse la gestione in stile lista come *container*.



3.1 Model

3.1.1 Gerarchia

La gerarchia è composta dalla classe *AbstractItem* che è la classe base virtuale pura, quindi non istanziabile, e fornisce una *firma* per tutte le classi derivate. Tutti i metodi di questa classe sono virtuali ed in particolare i metodi *GetSizeData()*, *GetType()* e *CheckFormat()* sono virtuali puri. Questa classe memorizza le informazioni base di un *"file"* quali *estensione, dimensione, titolo* e *descrizione*.

Derivate dirette dalla classe base sono le classi *Image, Film* e *GenericFile*. Queste sono molto simili tra loro ed ognuna di loro implementa i metodi virtuali puri della base. Si è deciso di non far derivare le classi *Film* e *Image* da *GenericFile* in quanto si vuole mantenere una distinzione concreta delle categorie, infatti l'intento delle classi è fornire *categorie* di file e non *tipi*.

3.1.2 Classe Container

La classe *container* fornisce un *template* simulando una lista doppiamente *linkata*, essa memorizza il primo ed ultimo nodo e il numero totale di nodi. Fornisce metodi di *inserimento* (in coda, in testa oppure ad un certo indice)*, rimozione, ricerca, assegnazione profonda, somma e subscripting.* Questa classe ha al suo interno una classe ausiliaria *nodo* che garantisce il *linkaggio* dei vari nodi, l'eliminazione profonda e la ovvia memorizzazione dell'oggetto *templetizzato*.

Si è scelto di non usare puntatori smart in quanto non ci sarebbero stati benefici tangibili in questo progetto e tutte le operazioni di gestione della memoria sono gestite dall'interazione tra *container* e *nodo*.

3.1.3 Classe Model

La classe *Model* fa uso sia della gerarchia che della classe *Container* e deriva dalla classe *QAbstractListModel* della libreria *Qt*. Si occupa di gestire una singola raccolta di file utilizzando un oggetto di tipo *Container<AbstractItem*>*, essa quindi implementa i metodi virtuali della classe base oltre a nuovi metodi quali *getLib(), dataType()* e *setDataItem()*. Inoltre, il puntatore alla classe base viene dichiarato come metatipo per consentire la conversione tra *AbstractItem** e *QVariant* che si utilizzerà per avere una comunicazione il più agnostica possibile con la *GUI*.

Fornisce tutti i metodi per *inserire, eliminare* e *modificare* gli oggetti presenti nel *Container*.

3.2 GUI (Graphical User Interface)

La *GUI* ha cinque classi distinte tra loro, tra queste ci sono classi che sono "puramente grafiche" (non si interessano di comunicare con il model) e quelle che forniscono adattatori per la ricerca, la comunicazione e gestione di più finestre.

3.2.1 Classi AddDialog e AddDialogLib

Queste due classi derivano da *QDialog* e sono "puramente grafiche", esse infatti comunicano solamente con altre classi della *GUI* ed hanno lo scopo di fornire un'interfaccia di inserimento e modifica dei dati per l'utente.

3.2.2 Classe SingleTab

Questa classe deriva da QWidget e offre una serie metodi e adattatori. Ogni oggetto di questa classe usa un *Model*, un *QSortFilterProxyModel*, una *QTableView* ed una serie di altri piccoli *widget* che permettono l'interazione con l'utente.

Questa classe è la finestra di interazione principale per la singola raccolta di oggetti e permette, grazie al *ProxyModel* di effettuare ricerche senza disturbare l'organizzazione interna del *Model*. Vengono resi disponibili i metodi di caricamento e salvataggio su file JSON come anche metodi per l'inserimento, modifica ed eliminazione degli oggetti del *Model*.

3.2.3 Classe TabWindow

TableWindow è una classe derivata da *QTabWidget* il cui scopo è gestire tutte le *view* e i *model* presenti nel programma, fa questo utilizzando un oggetto di tipo *Container<SingleTab*>*. In questo modo non viene posto un limite numerico alla quantità di librerie creabili dall'utente.

Questa classe fornisce metodi di caricamento e salvataggio di livello più alto in quanto essa gestisce ogni singola libreria. Vengono forniti i metodi per l'aggiunta, la modifica e la rimozione di una libreria. In un oggetto *TabWindow* è sempre presente una *tab* (nascosta o visibile a seconda dello stato dell'applicazione) in cui sono mostrate le istruzioni di base per operare il programma.

3.2.4 Classe MainWindow

È la finestra principale del programma, questa classe deriva dalla classe *QMainWindow* presente in *Qt* ed ospita una barra menù ed un oggetto di tipo *TabWindow. Fornisce all'utente le basilari opzioni di interazione con il programma.*

3.3 Gestione dei Dati

Il programma utilizza il formato ***JSON*** per salvare e caricare i dati. I dati vengono gestiti a in parte dal modello, che gestisce una singola raccolta, ed in parte dalla *GUI* che raccoglie ed organizza i vari model per poi procedere al salvataggio.

Il file *.json* che viene creato consiste in un *array* di *oggetti json*, ognuno di essi rappresentanti una *libreria*. Ogni *oggetto json* è a sua volta composto da un *"Name"*, ossia il titolo della libreria, e da un *array* che contiene oggetti che saranno gli elementi della suddetta libreria.

3.4 Polimorfismo

Il polimorfismo è fortemente presente sia nel *Model*, grazie all'oggetto *Container<AbstractItem*>*, sia nella *GUI*, grazie alle funzioni derivate dalla classe *QWidget. Si utilizzano* ***chiamate polimorfe*** a metodi virtuali ogni qual volta si crea un oggetto della gerarchia, per esempio per il controllo del formato della categoria. Inoltre, la classe base, virtuale pura, rende la gerarchia molto espandibile.

3.5 Gestione degli errori

Per la gestione degli errori si è scelto di gestirli in maniera *run*-time e di usare le classi della libreria standard *stdexcept* ed in seguito visualizzati tramite l'utilizzo della classe *QMessageBox* presente in *Qt*. La scelta di utilizzare le classi già presenti nella libreria è stata fatta basandosi sul *genere* dell'errore, ossia si è trovato che le classi presenti risultavano logicamente combacianti con il *genere* di errore che si voleva *lanciare*. Questo ha portato a non sviluppare una classe personalizzata degli errori.

3.6 Scelte Implementative

- Si è scelto di utilizzare un contenitore di tipo **lista doppiamente linkata** in quanto ritenuta più agevole in un ambiente molto dinamico come quello del programma, infatti si è reso possibile aggiungere elementi in qualsiasi punto della lista.
- È stato scelto di utilizzare la classe *TabWindow* come "*Gestore dei model*" in quanto la creazione di un modello che gestisse il *Model* e la *View* di ogni singola raccolta sarebbe stata troppo onerosa sul tempo a disposizione. Quindi la scelta è stata fatta consapevolmente a discapito di una totale separazione ***Model/View*** che comunque rimane marcata. Infatti, questa classe potrebbe essere definita come un ***Controller***.
- A causa della scelta sopra citata anche la **gestione dei file** è trasversale. Da notare però che ogni *Model* può generare un documento completo per il suo modello e solo il salvataggio finale viene affidato al "*Gestore dei model*".
- Si è scelto di usare sia nel **modello** che nella **gerarchia** il tipo *QString* al posto di *string*. La scelta è dovuta al fatto che la libreria *Qt* sarebbe stata utilizzata nella *GUI*, il che avrebbe reso il tutto più compatibile e leggiero, omettendo le conversioni esplicite.

4 Conclusioni

4.1 Timeline

Il progetto è stato iniziato a luglio 2019 ed è stato concluso ad intorno al 16 agosto 2019. Lo sviluppo è stato interrotto per motivi personali per circa una settimana e mezza verso la fine di luglio.

Per quanto riguarda le ore lavorative il progetto, nella sua completezza, ha richiesto circa **53 ore** sulle 50 previste, quindi sono abbastanza soddisfatto dalle tempistiche di realizzazione. La sezione che ha richiesto più lavoro è di certo lo studio della libreria *Qt* in quanto si è scelto di realizzare sia il *Model* che la *GUI* utilizzando questa libreria. La classe *Model* è stata realizzata due volte in quanto la prima volta si voleva realizzare un modello per i *Model*, come spiegato nelle scelte implementative, per poi scartare l'idea a causa delle tempistiche.

Fasi Progettuali	**Ore utilizzate**
Analisi del problema	3
Progettazione	9
Apprendimento *Qt*	10
Codifica *Model*	9
Codifica *View*	12
Test e Debug	6
Realizzazione relazione	4
Ore totali	**53**

4.2 Ambiente di sviluppo e Compilazione

Il progetto è stato sviluppato con ***Qt creator v4.7.0*** e ***Atom v1.40.1***, come compilatore si è usato ***mingw v4.9.2***. Il sistema operativo utilizzato per lo sviluppo è stato ***Windows 10 Pro N v1809*** per poi muoversi su ***Ubuntu 18.04.1 LTS*** con compilatore ***gcc v7.3.0*** per la fase di debug finale.

Il progetto prevede la compilazione tramite il file *.pro* fornito. Inoltre, viene fornito un file *.json* di esempio di un salvataggio che è possibile usare nel programma.

```
$ qmake P2.pro
$ make
$ ./P2
```

4.3 Note dello sviluppatore

Il progetto ha richiesto un impegno moderato, dal momento che la finestra temporale era molto ristretta e che molte ore sono state impiegate nello studio della libreria *Qt*. Risolti i primi problemi di riprogettazione del *Model* l'implementazione è stata abbastanza "semplice" grazie anche alla documentazione disponibile online.

La scelta metodologica di usare il ***Bottom-up*** in accoppiata al ***TDD*** è stata vincente facendo risparmiare tempo nella fase di debug e portando alla produzione di codice più pulito. Inoltre, è stato usato un sistema di ***versioning*** (***GitHub***) per rendere il processo produttivo più fluido.

Sempre per una restrizione temporale la separazione tre *GUI* e *Model* non è delle migliori ma rimane comunque concreta. Con più tempo a disposizione si sarebbe potuto optare per una gestione dei file interna ad un *Controller dei Model* vero e proprio. In modo analogo la *GUI*, nonostante sia funzionale, manca di "*carisma*". Questo problema si potrebbe risolvere aggiungendo un file *.css* che permetterebbe una cura estetica maggiore di quella presente tuttora.

In linea generale sono soddisfatto del progetto, date le tempistiche, ma la gerarchia è un po' scarna. Avrei voluto migliorarne la qualità ma le scadenze erano serrate.