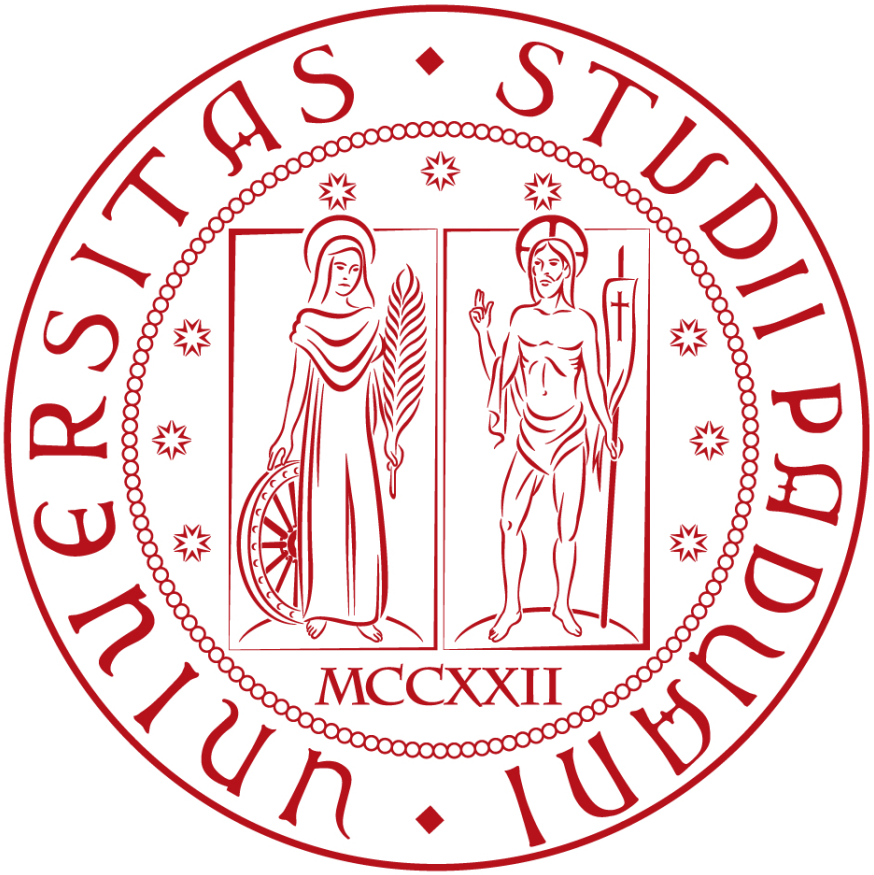
06 febbraio 2016

**Università degli studi di Padova**
**Corso di Laurea in Informatica**
**AA. 2015/16**

# Relazione Progetto di Programmazione ad oggetti




Email: pietro.gabelli@studenti.unipd.it

Pietro Gabelli | 1008028

# Indice



# Elenco delle figure




pietro.gabelli@studenti.unipd.it

# 1 Introduzione

## 1.1 Scopo del documento

Lo scopo di questo documento è di presentare in maniera chiara le scelte architetturali fatte.

## 1.2 Scopo del progetto

Il progetto ha come scopo lo sviluppo di un'applicazione per la gestione di una biblioteca. Gli oggetti che compongono la biblioteca sono inseriti in un database che viene utilizzato attraverso una interfaccia grafica. Lo sviluppo è stato fatto utilizzando C++ e Qt.

## 1.3 Specifiche progettuali

Il progetto è destinato ad immagazzinare i titoli di una biblioteca domestica. Gli oggetti che si possono salvare sono:

- Libri;
- Film, siano questi VHS o DVD.
- CD;

Le funzionalità principali che si intende modellare sono:

- Aggiunta di nuovi elementi;
- Ricerca tra i titoli presenti;
- Cancellazione di titoli, tra quelli presenti;
- Persistenza su disco dei dati inseriti.

I vincoli di progetto sono i seguenti:

1. Definizione ed utilizzo di una gerarchia G di tipi di altezza >= 1 e larghezza >= 1.
2. Definizione di un opportuno contenitore C, con relativi iteratori, che permetta inserimenti, rimozioni, modifiche.
3. Utilizzo del contenitore C per memorizzare oggetti polimorfi della gerarchia G.
4. Il front-end dell'applicazione deve essere una GUI sviluppata nel framework Qt.

Il progetto è stato sviluppato in un sistema XUbuntu; è stato utilizzato l'IDE Qt Creator nella versione 3.0.1, con le librerie alla versione 5.2.1.
Si è provato il progetto sui computer del Laboratorio Informatico Plesso Paolotti (LabP140 - labP036) dove compila ed esegue correttamente.
Il database che compone l'applicazione viene aperto e salvato usando il formato XML: entrambe le operazioni avvengono in maniera automatica grazie ad un path impostato di default.

Nello sviluppo è stata curata la separazione tra la parte logica e la parte grafica, senza utilizzare il design pattern MVC. Si e' preferita invece un'architettura Model/View, che rende possibile la separazione del modo in cui sono immagazzinati i dati e la loro presentazione all'utente.

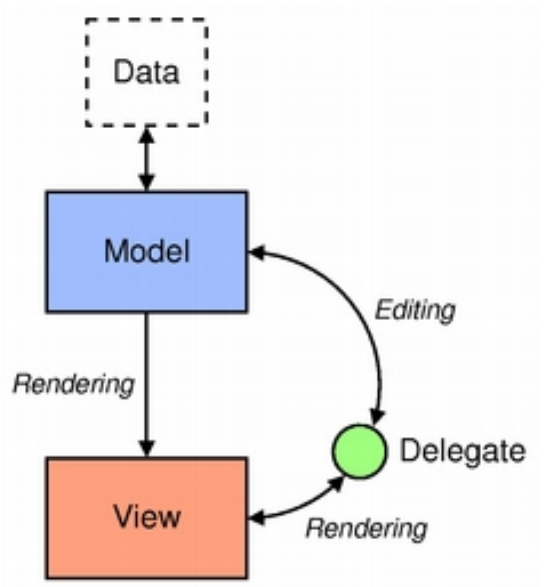



Figura 1: Model/View architecture

# 2 Classi logiche

## 2.1 Gerarchia oggetti memorizzati

La gerarchia sviluppata è composta da 5 classi:

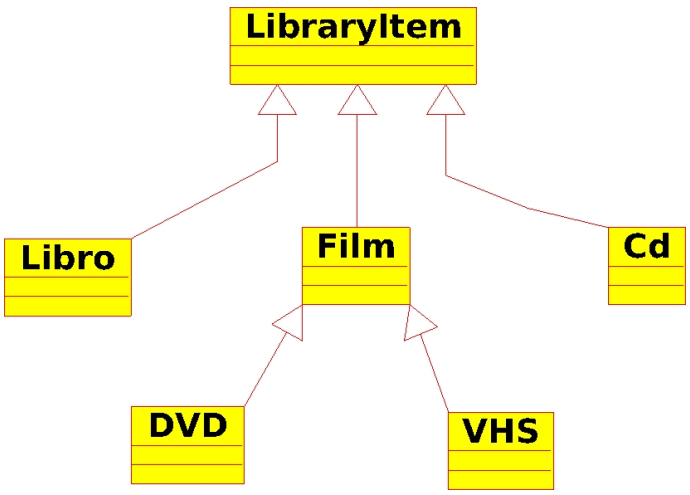



Figura 2: Gerarchia delle classi

La classe **LibraryItem** rappresenta un oggetto generico inserito nella libreria; per questo motivo ho scelto di renderla polimorfa ed astratta: non è possibile dichiarare oggetti di questa classe. All'interno di questa classe sono memorizzati gli attributi comuni a tutti gli oggetti della gerarchia: *titolo,genere*, entrambi di tipo string.
Per questa classe e per le altre classi nella gerarchia verranno resi disponibili:

- Un costruttore di defalut ed un costruttore per i campi dati della classe;
- Un distruttore virtuale;
- Un metodo *clone* che restituisce il puntatore ad un copia all'oggetto su cui viene invocato;


pietro.gabelli@studenti.unipd.it

- Un operatore di uguaglianza ed uno di disuguaglianza;
- Metodi **get** che restituiscono una copia dei campi dati, per ogni campo dati richiesto.

Le classi **Libro, CD, Film** sono classi derivate direttamente dalla classe base **Utente**; sono tutte concrete, a differenza di **Film**, polimorfa ed astratta come **LibraryItem**.
Le classi **DVD, VHS** sono classi concrete derivate da **Film**.
Per quanto riguarda i campi dati:

- La classe **Libro** contiene:
  - Un *autore*, memorizzato tramite uno string;
  - Un *anno di uscita*, memorizzato tramite un intero;
  - Un *editore*, memorizzato tramite uno string;
  - La definizione dei metodi virtuali definiti nella classe **LibraryItem**.
- La classe **Film** contiene:
  - Un *regista*, memorizzato tramite uno string;
  - Una *durata* in minuti, memorizzata tramite un intero;
  - Una *data d'uscita*, memorizzata tramite un QDate;
  - La definizione dei metodi virtuali definiti nella classe **LibraryItem**.
- La classe **CD** contiene:
  - Un *artista*, memorizzato tramite uno string;
  - Un *anno di uscita*, memorizzato tramite un intero;
  - Un *numero di dischi*, memorizzato tramite un intero;
  - La definizione dei metodi virtuali definiti nella classe **LibraryItem**.
- Le classi **CD, DVD** sono implementazioni di **Film**; non contengono ulteriori campi dati. Rendono disponibili costruttori, un distruttore, il metodo clone, operator == e != che richiamano quelli della superclasse **Film**.

## 2.2 Classe contenitore

Per memorizzare gli oggetti della biblioteca è stato creato il template di classe **Contenitore**. La classe è dichiarata amica di *operator«*, delle classi **iterator** e *const_iterator*; all'interno è dichiarata privata la classe **ContainerItem**; sono invece pubbliche le classi **Iterator**, **const_Iterator**.

### 2.2.1 ContainerItem

La classe è caratterizzata da:

- l'oggetto *info*;
- puntatore all'oggetto **ContainerItem** successivo;
- il costruttore di default ed il costruttore a 2 parametri;
- operatore d'uguaglianza e disuguaglianza;

Essendo la classe racchiusa nella parte privata, metodi e campi dati sono stati dichiarati pubblici.


pietro.gabelli@studenti.unipd.it

#### 2.2.2 Iterator

La classe è caratterizzata da:

- la dichiarazione di amicizia con la classe **Container**
- un puntatore a **ContainerItem**, dichiarato privato;
- costruttore di default;
- operatore d'uguaglianza e disuguaglianza;
- operatori d'indirezione ed indicizzazione;
- operator ++ prefisso, postfisso;
- operator* che restituisce copia dell'oggetto puntato.

#### 2.2.3 const_Iterator

La classe è caratterizzata da:

- la dichiarazione di amicizia con la classe **Container**
- un puntatore a **const ContainerItem**, dichiarato privato;
- costruttore di default;
- convertitore di tipo da **Iterator**;
- operatore d'uguaglianza e disuguaglianza;
- operator ++ prefisso, postfisso;
- operatore di dereferenziazione per ottenere un riferimento costante all'oggetto puntato, marcato costante.

### Metodi di Container

Sono dichiarati privati:

- puntatore al primo elemento **ContainerItem**;
- un metodo dichiarato statico *deepcopy* che fa una copia profonda della lista;
- un metodo dichiarato statico *deepremove* che esegue la distruzione profonda della lista;
- un metodo dichiarato statico *equals* che, dati due puntatori a **ContainerItem**, l'uguaglianza delle liste puntate.
- un metodo dichiarato statico *printcontainer* per effetturare la stampa su ostream.

Sono pubblici:

- un costruttore di default;
- il costruttore di copia (profonda);
- l'operatore d'assegnazione (profonda);
- overloading del distruttore di default per fare la distruzione profonda;
- metodo *isEmpty* che dice se il contenitore è vuoto;
- metodo *insert* per inserire in testa un nuovo elemento nel contenitore;

- metodo *remove* per rimuovere un elemento;
- metodo *size*, restituisce il numero d'elementi presenti nel contenitore;
- metodo *replace* per sostituire un elemento con un altro;
- operatori d'uguaglianza, disuguaglianza;
- metodo *search* che restituisce un nuovo contenitore dove sono memorizzati gli elementi che corrispondono alla ricerca;
- metodi *begin*, *end*, che rispettivamente ritornano l'iteratore al primo nodo della lista ed un iteratore che punta ad un nodo vuoto (come quando si sta scorrendo la lista e non si è ancora arrivati al past-the-end); sono disponibili metodi equivalenti per ritornare gli equivalenti iteratori costanti;
- l'operatore di dereferenziazione che restituisce l'oggetto puntato da un iteratore; disponibile anche per l'iteratore costante, viene restituito un riferimento costante;

## 2.3 Classe SmartPtr

È stata definita una classe di puntatori smart, **SmartPtr**, che ha come campo dati privato un puntatore ad un oggetto della superclasse **LibraryItem**.
I metodi che sono stati resi disponibili per questa classe sono i metodi d'utilità per una classe di puntatori: *costruttore di default e ad un argomento, costruttore di copia*, overloading degli operatori di: *assegnazione, uguaglianza, disuguaglianza, indicizzazione, indirezione; distruttore* di default. Si è infine definito un metodo *getItem* che restituisce copia dell'oggetto puntato; ne viene definito anche uno segnato costante.

## 2.4 Classe Biblio

Questa classe serve a concretizzare la classe templetizzata **Container** con le classi della gerarchia radicata in **LibaryItem** utilizzando **SmartPtr**, rendendo disponibili metodi di utilità che serviranno all'applicazione. Contiene:

- Un **Container** di puntatori a **SmartPtr**;
- Un *distruttore*;
- Un metodo *isEmpty* che dice se non sono presenti elementi nella biblioteca;
- Un metodo *addItem*, che aggiunge un nuovo elemento alla biblioteca. Restituisce un booleano che indica la presenza di un'uguale opera all'interno della libreria. Sono permesse più copie d'uno stesso elemento.
- Un metodo *removeItem* per togliere un elemento dalla biblioteca;
- Un metodo *getItem* che data una posizione *i*, restituisce un puntatore all'i-esimo elemento nel **Container**;
- Il metodo *save* che su file il contenuto della biblioteca, in formato **XML**;
- Il metodo *load* che carica dal file XML il contenuto della biblioteca. Entrambi i metodi segnalano con un messaggio d'errore se non e' stato possibile aprire il file.

# 3 Classi Grafiche

La parte View che si è modellata intende permettere all'utente di:

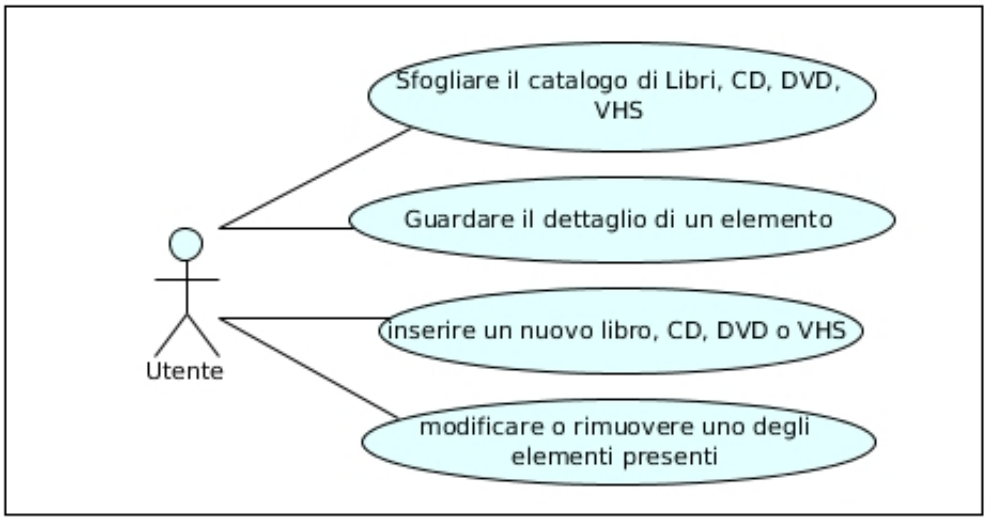



Figura 3: Operazioni previste

Per questo motivo, è stata creata una classe **QWidget**.

## 3.1 MainWindow

Questa è la classe invocata direttamente dal **main**. L'obiettivo di questa classe è mostrare le operazioni che è possibile eseguire, raggruppandole in base agli oggetti della libreria in modo da semplificare l'utilizzo dell'applicazione.
Al suo interno è presente:

- Un oggetto **Contenitore<SmartPtr>**, contenente puntatori agli elementi della libreria, dichiarato privato;
- Il costruttore della **MainWindow**, che crea l'oggetto **Contenitore** nello heap, fa il setup della finestra e istanzia
- Un metodo *setupView()* che s'occupa di
- I seguenti slot:
  - slotAddItem() per l'inserimento d'un nuovo elemento nella libreria;
  - slotMostraCatalogoQDialog() per vedere il contenuto della libreria;
  - slotTrovaQDialog() per effettuare la ricerca d'un elemento all'interno della libreria
  - slotReplaceDVD() per sostituire un film in DVD;
  - slotReplaceVHS() per sostituire un film in videocassetta;
  - slotReplaceCD()
  - slotReplaceLibro()
  - slotAggiornaRisultati()


pietro.gabelli@studenti.unipd.it

## 3.2 QWidget

Un oggetto di **QWidget** viene creato dal main senza parent, costruendo così un'*indipendent window*.
Questa classe funge da layout manager, contiene 5 groupbox al cui interno sono contenute le principali operazioni e funzionalita‘ disponibili, divise nel modo seguente:

- *qgb_menu* rappresenta la groupbox in cui sono racchiusi i pulsanti: *Gestione CD, Gestione DVD, Gestione Libri, Gestione VHS, Trova Elemento, Chiudi.*
- *qgb_CD, qgb_Libri, qgb_DVD, qgb_VHS*, in cui sono racchiuse le tabelle che rappresentano - rispettivamente - i CD, Libri, DVD e VHS presenti nella bibioteca

Ho scelto di usare le groupboxes per la possibilita‘ di inserire un frame ed un titolo attorno alle tabelle.


pietro.gabelli@studenti.unipd.it

# 4 Gestione della memoria

Nel progetto **Qbiblio** gli oggetti sono stati memorizzati nello heap.
Le classi sono state dotate d'un distruttore profondo, che permette la deallocazione degli oggetti nello heap al momento della distruzione del puntatore.
Al momento della chiusura delle finestre il contenitore viene svuotato nel modo seguente: prima vengono deallocati gli oggetti puntati dagli **SmartPtr** e poi rimuove gli elementi del contenitore.


pietro.gabelli@studenti.unipd.it