Università degli studi di Padova
Corso di laurea in Scienze Informatiche
Scuola di Scienze - Dipartimento di Matematica
Professore: Francesco Ranzato
Anno Accademico 2021/2022

Progetto di Programmazione ad Oggetti

UNIPD LIBRARY

—

Submitted by: Marco Brugin

Matriculation number: 2010012

Submission date August 27, 2022

1 Introduzione

Sin dai tempi più antichi, è nota l'importanza delle biblioteche e della conoscenza contenuta nei libri.
Con l'avvento delle nuove tecnologie tali ambienti si sono trasformati adottando nuovi strumenti, che ne consentono, attraverso l'utilizzo di strumenti digitali, non soltanto la fruizione in formato digitale, ma pure l'opportunità realizzare delle statistiche su di essi.
Per la catalogazione e la realizzazione di tali statistiche, è necessario conoscere per ogni libro : il titolo e l'autore, che vanno ad identificare univocamente un libro, in quanto non possono esistere due libri dello stesso autore con stesso titolo;la data di pubblicazione, il genere, e il numero delle pagine che lo vanno a costituire (si assume che ogni libro abbia almeno una pagina per essere considerato tale); il numero di lettori che hanno avuto piacere di leggerlo (nel caso di nuove pubblicazioni tale valore può essere anche uguale a zero) e dalla presenza oppure no di formati digitali.
Si sottolinea che l'autore e il titolo sono due caratterristiche essenziali perchè consentono l'identificazione di copie di uno stesso libro.

2 Manuale d'uso GUI

Unipd Library è un applicativo che elabora tali informazioni e fornisce delle rappresentazioni grafiche utili a tali scopi. Una volta aperta l'applicazione, si aprirà il messaggio di benvenuto, il quale dopo aver premuto il pulsante "OK" lascerà il posto alla finestra Home. Quest'ultima da l'opportunità di caricare una Biblioteca già esistente o di crearne una nuova. Come esempio viene fornita una biblioteca di prova, salvata nel file "prova-biblioteca.json".
Una volta premuto il pulsante New verrà richiesto di inserire il nome della Biblioteca che si vorrà andare a creare. Dopo di che, si aprirà la schermata Main che da l'opportunità di modificare la lista dei libri presenti nella biblioteca (prima tabella sulla sinistra), modficare l'elenco dei generi e degli autori presenti nella biblioteca e visualizzare le rappresentazioni grafiche fornite. Le operazioni di "add" e "remove" saranno possibili attraverso i pulsanti "+" dopo aver compilato la barra di aggiunta o attraverso i pulsanti "-" presenti accanto ad ogni entry delle tabelle. Le stesse operazioni saranno possibili anche premendo il pulsante "open".
Inoltre sono presenti altri due pulsanti che permettono il salvataggio della biblioteca corrente su file o il ritorno alla schermata Home attarverso il pulsante con la medesima etichetta.

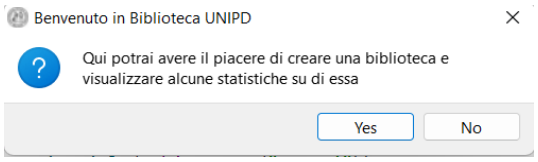

Figure 1: Schermata di Benevenuto

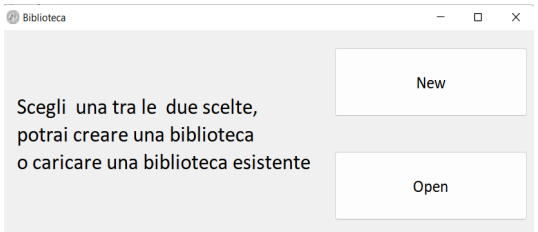

Figure 2: Schermata Home

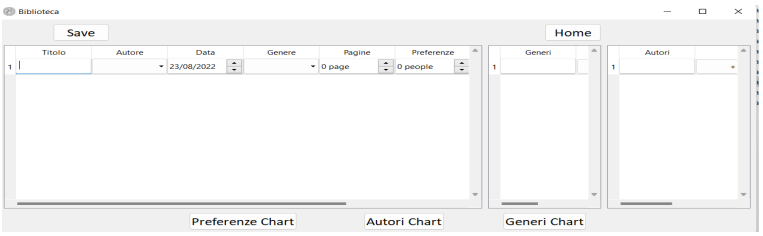

Figure 3: Schermata Main

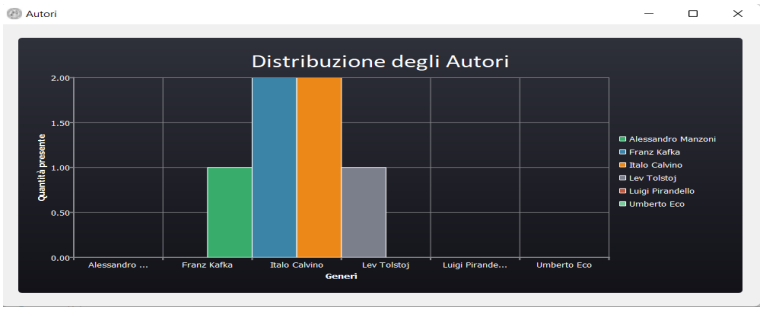

Figure 4: Schermata Charts

3 Funzionalità offerte

- Mantenimento dell'integrità e della correttezza dei dati inseriti, ovvero per ogni informazione inserita viene verificato se quest'ultima rispetta il parsing di quel determinato tipo di informazione. Tutto ciò viene ottenuto minimizzando l'inserimento diretto dell'utente da tastiera.

- L'applicazione è stata progettata per mentenere al suo interno solo dati univocamente identificati: i doppioni non vengono presi in considerazione. A tal proposito è presente un controllo di unicità sia al momento dell'inserimento da GUI sia nella lettura da file per tutte le informazioni rillevanti: Libri, Generi e Autori. Per quanto riguarda i Libri si è scelto di considerare due libri uguali se e soltanto se hanno stesso autore e titolo (tutti i controlli sono Insensitive Case).

- Inoltre è presente un ulteriore controllo di integrità referenziale su generi e autori: infatti non sarà possibile eliminare alcun autore o genere che risulta referenziato in un libro presente nella biblioteca.

- Salvataggio e caricamento da file JSON. L'applicazione utilizza il nome della biblioteca come proposta del nome del file da salvare. L'utente comunque può sempre ovviare a tale proposta. Inoltre al momento del caricamento della biblioteca da file viene dato all'utente la possibilità all'utente di scegliere solo file di estensione json, nascondendo gli altri. In più, oltre a tale funzionalità, una volta scelto il file di interessa viene verificato che quest'ultimo presenti la seguente struttura: {**"autori"**:[ ],**"data"**:[ ],**"generi"**:[ ]}.

- Viene fornita una rappresentazione grafica dei dati inseriti nei seguenti formati.

  - Rappresentazione con un Pie_Chart delle diversi preferenze del pubblico nei confronti degli autori.
  - Rappresentazione con Bar_Chart del numero di libri presenti nella biblioteca per autore.
  - Rappresentazione con Area_Chart del numero di libri presenti nella biblioteca per genere (il pop-up guiderà la lettura del grafico).

4 Ereditarietà e polimorfismo

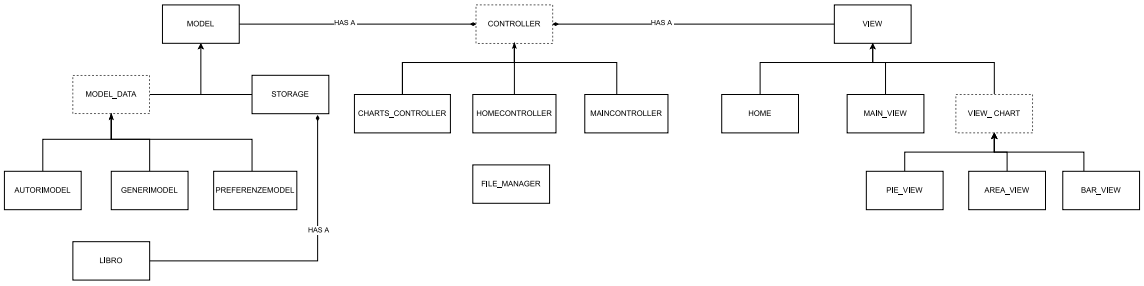

Figure 5: Gerarchia dei Tipi

4.1 Gerarchia dei tipi

Dai criteri di progettazione si è deciso di adottare il modello MVC per dividere la parte logica dalla parte grafica.
La classe base polimorfa **Model** rappresenta una collezione di oggetti interessanti per il modello che si sta andando a modellare. Nelle sue diverse implementazioni assume varie forme e implementa vari contenitori in base alle funzionalità che è tenuto a fornire. Si noti che non è possibile instanziare la classe Model direttamente, in quanto non sono presenti costruttori pubblici; è essenziale instanziare un suo derivato che non sia astratto.
Lo stesso ragionamento lo si può fare con la classe base polimorfa **View** che rappresenta una schermata visualizzata all'utente. Quest'ultima presenta una diversa implementazione per ogni schermata: con adeguati segnali, slot e metodi necessari al suo funzionamento.
Inoltre si precisa che la derivazione delle schermate prosegue secondo due livelli: il primo non fa altro che specializzare e implementare le schermate di presentazione e inserimento dei dati; il secondo invece specializza e implementa la classe astratta **View_Chart: public View**: modello per qualsiasi altra schermata che contiene un Chart.
Un esempio di metodo virtuale puro presente in questa classe è il metodo **void** $Insert(constQMap < QString, uint > \&)$ che in base alla sua implementazione permette di inserire i dati provenienti dal Model all'interno della View. D'altra parte si precisa che anche per View non è permessa l'istanziazione diretta, per l'assenza di costruttori pubblici.
Inoltre è presente anche la classe base astratta polimorfa **Controller:public QObject**, che rappresenta un modello per un controller di una schermata. Ogni Controller al suo interno possiede un pointer al rispettivo Model e View di competenza (in base al ramo di derivazione), attraverso i quali, una volta ricevuto i segnali o le modifiche dei dati, opererà di conseguenza.
Si precisa che nella classe Controller è anche presente uno slot virtuale puro: **void onViewClosed() const=0** che permette di specificare il comportamento da adottare al momento dell'emissione del segnale **void viewClosed() const**.
Infine è presente anche una classe Libro che rappresenta il componente principale di una libreria e una classe statica File_Manager che contiene metodi di utilità per la gestione dei file.

4.2 Polimorfismo

- **Metodo** $constQMap < QString, uint > \&ModelData :: getData()$

  E' un metodo presente nella classe alla base della gerarchia dei modelli dei dati per Chart. Si noti che in ogni classe derivata tale metodo deve essere ridefinito per consentire di instanziare la classe discendente e, anche se tale metodo è un semplice getter, il risultato restituito cambia radicamenete in base al comportamento del costruttore della classe che lo implementa. Si noti che tale definizione può essere utilizzata anche per grafici multipli, in quanto in Qt la QMultiMap non è altro che un sottotipo diretto della QMap.

- **Metodo** $voidView\_Chart :: Insert(constQMap < QString, uint > \&)$

  E' un metodo presente nella classe View_Chart ed in particolare si occupa di tradurre i dati provenienti dal model in dati rappresentabili in un View_Chart. Naturalmente il suo comportamento cambia radicalmente dal tipo di grafico che andrà ad implementare, quindi deve essere ridefinito in ogni classe derivata. Essendo anche alla base del progetto richiesto è un ottimo esempio di polimorfismo.

- **Metodo void Model_Data::Set_Up(const QString&)**

  E' un metodo presente nella classe base astratta View_Chart che si occupa del setting di ogni grafico. Essendo ogni classe derivata un grafico differente il suo comportamento varierà da implementazione a implementazione. Essendo anche alla base del progetto richiesto è un ottimo esempio di polimorfismo.

- **Distruttori Virtuali**

  for (auto i:children())
  delete i;
  delete model; delete view;

  In ogni classe sono stati implementati adeguati dstruttori virtuali. Indipendentemente dal tipo dinamico del puntatore verrà richiamato il distruttore corretto come si può notare dal codice sorgente.

- **Slot void onViewClosed()**

  Alla chiusura di ogni View viene emesso il segnale void viewClosed() a cui è connesso lo slot onViewClosed() che in base alla sua implementazione ne determina un diverso comportamento alla chiusura di una View.

- **Metodo void closeEvent(QCloseEvent *)**

  Metodo appartenente alla classe QWidget invocato quando viene premuto il pulsante "X" di un finestra. Esso viene reimplementato in alcune delle finestre e ne determina un diverso comportamento alla chiusura di esse in base al loro tipo dinamico.

- **Metodi Contains classe Storage**

  Vengono resi virtuali i metodi contains della classe Storage per ulteriori ridefinizioni del significato di tali metodi.

5 Formato dei file per l'Input e l'Output

Come richiesto dalle specifiche, l'applicazione è in grado di leggere e scrivere su file i dati che utilizza per generare le rappresentazioni grafiche. I dati, una volta inseriti dall'utente, per volontà dell'utente possono essere salvati in un file formato JSON. Il file che verrà generato avrà una struttura prefissata: {**"autori"**:[ ],**"data"**:[ ],**"generi"**:[ ]}. Al momento della lettura da file vengono rigettati tutti i file che non sono in formato JSON e non hanno la struttura citata pocanzi. In più oltre ai controlli appena detti, l'applicazione possiede dei controlli di integrità riguardo ai possibili doppioni di informazioni inserite e un controllo sull'integrità dei libri che si andranno a caricare da file. Infatti non sarà possibile memorizzare nell'applicazione libri il cui genere e autore non sia presente nel file (i controlli sono case insensitive).
Nonostante tali protezioni, non è possibile affermare che l'applicazione possieda un vero e proprio controllo di integrità.
Alcune prove si possono effettuare con il file dato in dotazione "prova-biblioteca.json".

6 Istruzione di compilazione

Le seguenti istruzioni devono essere eseguite da terminale. Insieme al progetto viene fornito il file "Biblioteca.pro" necessario alla compilazione. Attenzione non si deve generare alcun file pro con il comando qmake -project, perchè la compilazione non andrebbe a buon fine. Per il progetto sono necessari i seguenti pacchetti:

- qt5-default (installibile su sistema operativo Ubuntu con il seguente comando: sudo apt-get install qt5-default)
- libqt5charts5-dev (installibile su sistema operativo Ubuntu con il seguente comando: sudo apt install libqt5charts5-dev)

Quindi i comandi per eseguire la compilazione sono i seguenti: qmake→make
Infine per avviare l'applicazione sarà necessario digitare il comando ./Biblioteca

7 Ore richieste per il progetto

Attività	**Durata**
Analisi della realtà di interesse e dei Requisiti	2 ore
Documentazione Qt	6 ore
Studio Modello MVC	1 ora
Progettazione nella sua integrezza	5 ore
Creazione modello	7 ore
Creazione Home_View	2 ore
Creazione Main_View	6 ore
Creazione View per Chart	5 ore
Creazione dei vari Controller	10 ore
File Management	2 ore
Debugging	10 ore
TOTALE	56 ore

Come si vede dalla tabella sono state superate le 50 ore previste dalle specifiche, questo perchè in fase di sviluppo si sono verificati dei problemi di realizzazione, in particolare nella gestione della MainView.

8 Ambiente di sviluppo

L'intero progetto è stato creato con l'IDE QtCreator.
Lo sviluppo è stato effettuato con il seguente sistema:

- Sistema Operativo: Windows 11 Home version 21H2
- Compilatore: GNU g++ version 7.10.1
- Libreria di supporto: Qt 5.9.5

Inoltre il programma è stato anche testato sulla macchina virtuale fornita:

- Sistema Operativo: Ubuntu 18.04.3 LTS
- Compilatore:GNU g++ version 7.3
- Libreria di supporto: Qt 5.9.5

9 Warning consapevoli e possibili problemi di visualizzazione

Si segnalano i seguenti warning che possono verificarsi durante l'esecuzione:

- **GtkDialog mapped without a transient parent. This is discouraged.**

Inoltre si segnala che per quanto riguarda la rappresentazione PieChart, con certe porzioni di dati, le label dei dati risultano illegibili. Per compensare tale problema è stata aggiunta una leggenda illustrativa.