QShop

Relazione progetto Programmazione a Oggetti

Banzato Alessio, mat. 2042381

1 Introduzione

QShop è un'applicazione per la gestione dell'inventario dell'omonimo negozio. Il negozio offre tre tipi di prodotti: album musicali, libri, film. I prodotti godono di caratteristiche comuni ma anche specifiche per ogni tipologia. Ogni tipologia di prodotto ha inoltre una propria visualizzazione grafica.
Per poter accedere all'applicazione è necessario utilizzare un account dotato di un nome utente e una password. Dopo aver effettuato l'accesso è possibile:

- Cercare prodotti tramite l'apposita barra di ricerca
- Modificare prodotti tramite l'apposito tasto presente su ogni prodotto
- Eliminare prodotti tramite l'apposito tasto presente su ogni prodotto
- Aggiungere prodotti all'inventario utilizzando il menu a tendina posto in alto a sinistra

L'account messo a disposizione è il seguente:

Nome utente	**Password**
Proprietario	qshop12

2 Descrizione del modello

Il modello logico si basa su una gerarchia di prodotti che il negozio vende, le cui caratteristiche comuni sono raggruppate in una classe astratta `Item`, che rappresenta quindi il generico oggetto venduto. Tutti i prodotti del negozio sono contenuti in un inventario, a cui è dedicata un'apposita classe.

La classe `Inventario` è stata implementata seguendo il design pattern Singleton, ed è caratterizzata da un `Vettore<Item*>`, e da un valore booleano `unsavedChanges` che segnala se l'inventario è stato modificato. Il costruttore di questa classe non ha parametri e imposta il valore booleano a `false`. Il puntatore `instance` punta all'unica istanza della classe, resa unica dal fatto che

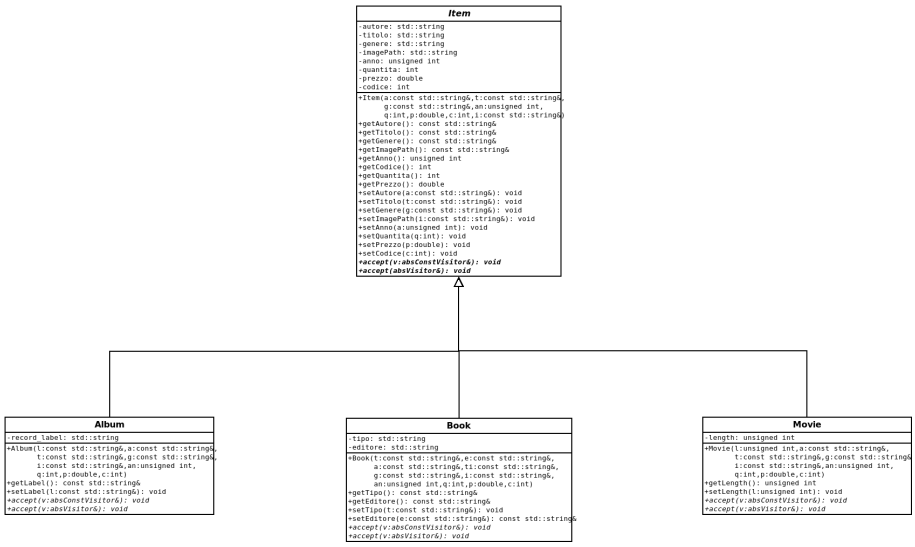


Figure 1: Gerarchia modello logico

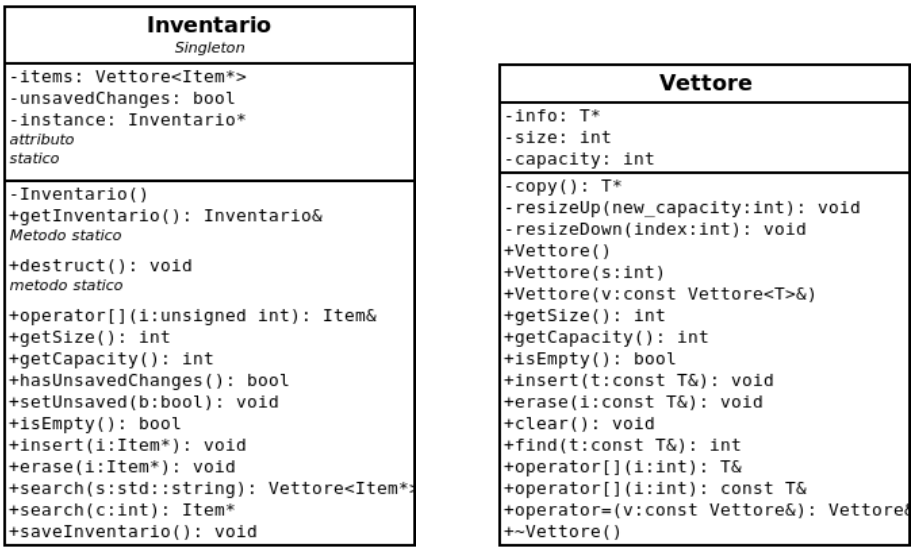
il costruttore è privato ed è chiamato soltanto in seguito a un controllo effettuato dal metodo statico `getInventario`. Il costruttore di copia e l'operatore di assegnazione sono stati eliminati, in conformità con la definizione del design pattern. È presente l'overloading del metodo `search`:

- `Vettore<Item*> search(std::string)`: è utilizzato quando si vuole cercare un prodotto dalla barra di ricerca. Il vettore ritornato rappresenta l'insieme di prodotti di cui è stata trovata una corrispondenza con la stringa data in input

- `Item* search(int)`: è utilizzato quando si vuole eliminare o modificare un prodotto. Cerca il prodotto nell'inventario tramite il codice ad esso associato e lo ritorna per poter effettuare le operazioni desiderate

Il metodo `saveInventario()` è invece utilizzato per salvare l'inventario in un file. È stato deciso di utilizzare il formato JSON per quanto riguarda la persistenza dei dati, quindi questo metodo, e anche il costruttore dell'inventario, utilizzano strumenti di Qt che riguardano il formato JSON. Infine, i metodi `getSize()`, `getCapacity()`, `isEmpty()`, `insert(Item*)`, `erase(Item*)` richiamano i metodi omonimi della classe `Vettore`, che rappresenta il contenitore implementato nativamente per il progetto. Il contenitore è stato implementato sotto forma di template, in quanto all'interno del progetto è utilizzato sia come contenitore di puntatori ad `Item` (come visto in precedenza, per l'inventario e per contenere i risultati di ricerca), sia come contenitore di puntatori a `QWidget` (contiene i widget che permettono di visualizzare i risultati di ricerca).
Ultima classe facente parte del modello logico è quella dedicata all'utente. È una

classe dedicata a rappresentare l'utente connesso all'inventario, il cui username appare in alto a sinistra all'interno dell'interfaccia grafica.



(a) Classe inventario

(b) Classe template vettore

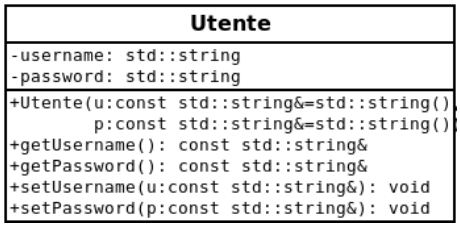


Figure 3: Classe utente

3 Polimorfismo

Per quanto riguarda il polimorfismo è stato utilizzato il design pattern Visitor. In particolare sono presenti due classi astratte pure: `absConstVisitor` e `absVisitor`.

3.1 absConstVisitor

Questa classe astratta pura è alla radice di una gerarchia composta da tre classi, illustrata dal diagramma UML. Due di queste classi rappresentano dei visitor del modello logico, mentre l'altra è un visitor per l'interfaccia grafica.

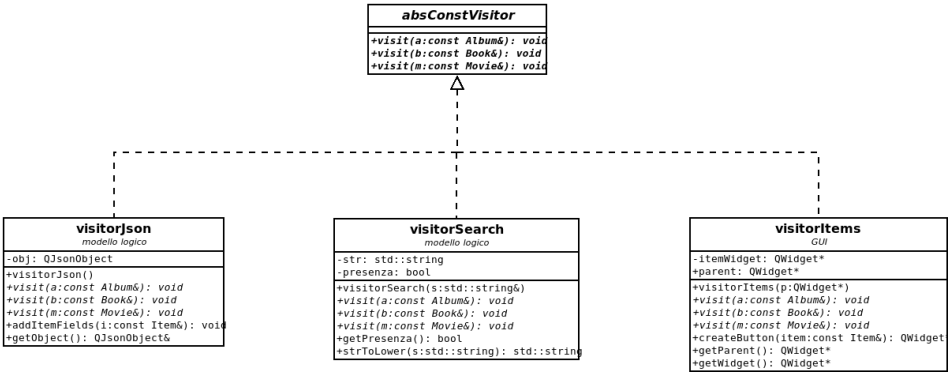
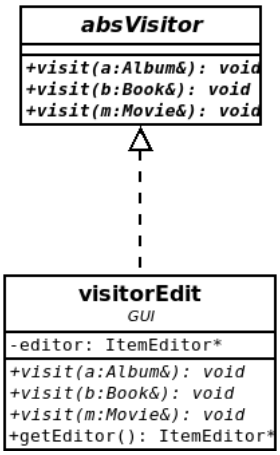


Figure 4: Gerarchia del visitor costante

La classe `visitorJson` è utilizzata all'interno dell'inventario, in particolare nel metodo utilizzato per il salvataggio dati su JSON. Questo visitor ha il compito di creare un `QJsonObject` adatto al tipo di prodotto da salvare: nei tre metodi ereditati dalla classe astratta pura viene aggiunto al `QJsonObject obj` l'attributo caratteristico del prodotto da salvare, ad esempio l'etichetta discografica (`std::string record_label`) per quanto riguarda gli oggetti che rappresentano un album, e viene poi chiamato il metodo `addItemFields` che aggiunge all'oggetto gli attributi comuni a tutti i prodotti del negozio.
`visitorSearch` è sempre utilizzato nell'inventario, ma nella funzione di ricerca che prende una stringa come parametro. In questo caso il visitor prende tutti gli attributi dell'oggetto su cui è chiamato, li inserisce in una stringa unica che trasforma tutta in caratteri minuscoli tramite il metodo `strToLower(std::string s)`, e successivamente modifica il valore booleano `presenza` controllando che nella stringa degli attributi sia presente almeno un'occorrenza della stringa `str`, ovvero della stringa data come input nella ricerca. Questo controllo avviene tramite il confronto tra il risultato del metodo `std::string::find()` e il valore `std::string::npos`.
Infine `visitorItems` è il visitor addetto alla costruzione dei widget grafici utilizzati per visualizzare i risultati di ricerca. Dal punto di vista grafico i widget sono di due tipi: quello per la visualizzazione degli oggetti `Album` è costituito da un layout verticale con immagine quadrata e informazioni riguardo l'album, mentre i widget per `Book` e `Movie` presentano l'immagine rettangolare posta di fianco alle informazioni tramite un layout orizzontale. Per quanto riguarda le informazioni mostrate, invece, c'è un caratterizzazione specifica per ogni tipo della gerarchia: ogni widget rappresenta tre informazioni comuni (titolo, autore, genere) e l'informazione che caratterizza il tipo di oggetto. Nel caso degli oggetti `Movie`, inoltre, l'autore è indicato come "regista".

3.2 absVisitor

Per quanto riguarda questa classe, la gerarchia è formata da una sola classe concreta: `visitorEdit`.
Questa classe è responsabile della costruzione dei widget usati per la creazione e la modifica di oggetti. In base al tipo di oggetto che si vuole modificare o creare viene visualizzato un widget composto da delle coppie di `QLabel` e `QLineEdit`, ognuna rappresentante un attributo dell'oggetto. Ci saranno dunque delle coppie comuni, rappresentanti gli attributi comuni, e delle coppie caratteristiche del tipo specifico.



4 Persistenza dei dati

Come detto più volte in precedenza,
il salvataggio dei dati avviene su file di tipo JSON. Ci sono due file addetti alla persistenza dei dati, uno riguardo gli utenti chiamato `utenti.json` e uno riguardo l'inventario chiamato `inventario.json`.
Il file riguardante gli utenti è gestito nella classe `utenteEditor`, all'interno della quale è presente uno slot privato che permette di aggiungere nuovi utenti al file.
Il file dell'inventario è gestito dall'omonima classe.
Ogni file contiene un array, e nel caso dell'inventario è presente un campo dati `objtype` per discriminare i vari tipi di oggetti durante la creazione dell'inventario nel costruttore della classe.

5 Funzionalità implementate

5.1 Finestra di login

1. Login tramite username e password
2. Login effettuabile tramite tasto `Invio`
3. Possibilità di aggiungere utenti nella schermata di login
4. Visualizzazione messaggio di errore quando si sbaglia a inserire username o password

5.2 Finestra principale

1. Barra superiore
   (a) Visualizzazione nome utente
      i. Possibilità di chiudere l'applicazione dal menu a tendina associato
   (b) Menu a tendina per aggiungere prodotti all'inventario
      i. Ogni opzione crea un editor specifico
      ii. Il pulsante dell'editor è attivabile tramite tasto `Invio`
2. Ricerca di prodotti tramite barra apposita
3. Ricerca effettuabile tramite tasto `Invio`
4. Visualizzazione dei risultati di ricerca tramite layout a griglia usando widget diversi in base al tipo di prodotto
5. Possibilità di modificare un prodotto tramite editor apposito creato dal tasto con l'icona a penna
6. Possibilità di eliminare un prodotto dall'inventario tramite apposito tasto con l'icona del cestino
   (a) Richiesta di conferma per l'eliminazione
   (b) Aggiornamento automatico dell'interfaccia grafica dopo la rimozione
7. Richiesta salvataggio modifiche (se effettuate) alla chiusura dell'applicazione

6 Rendicontazione ore

Attività	**Ore previste**	**Ore effettive**
Studio e progettazione	10	9
Sviluppo del codice del modello	10	12
Studio del framework Qt	8	8
Sviluppo del codice della GUI	13	15
Test e debug	5	5
Stesura della relazione	4	4
Totale	50	53