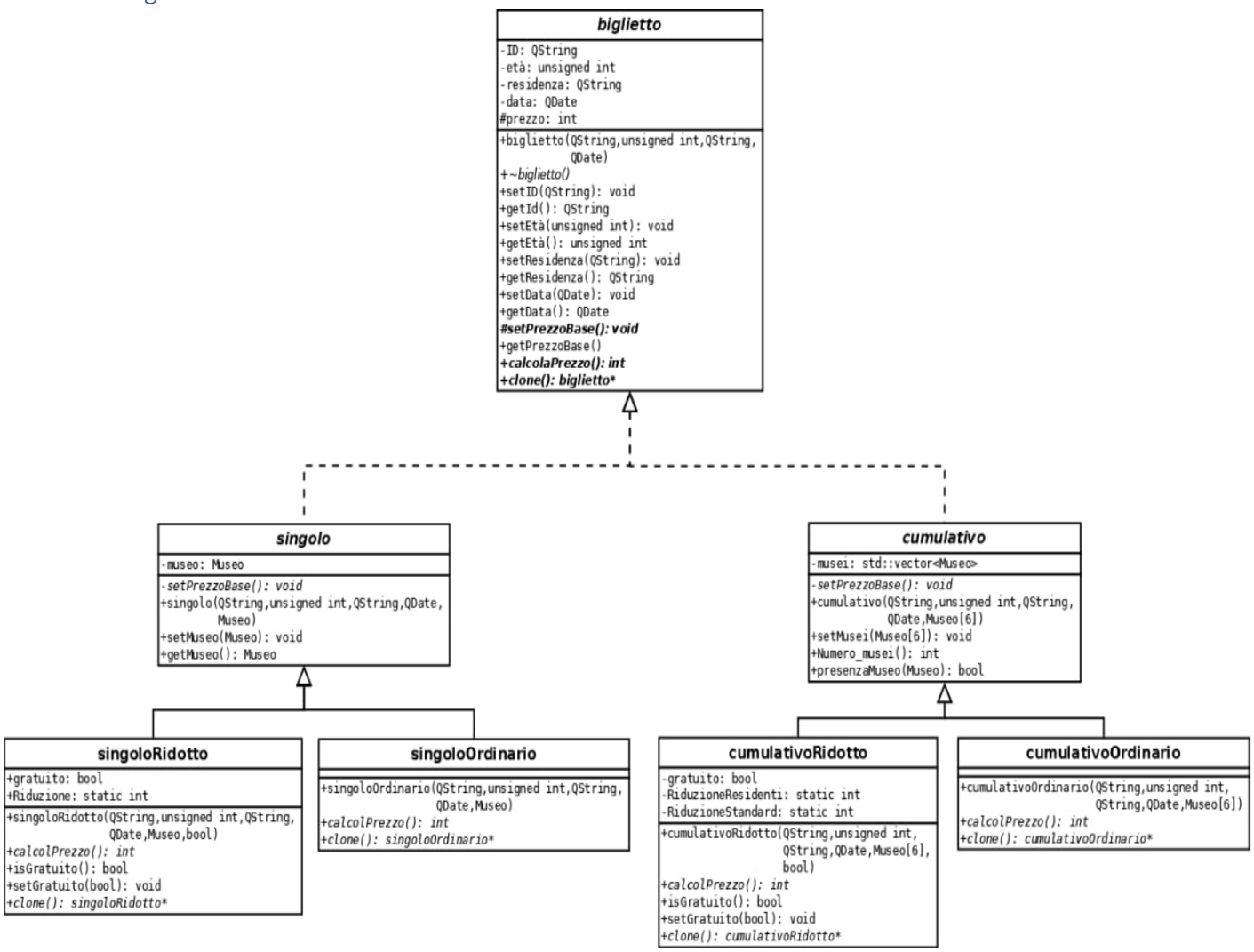
oggetto progetto Programmazione ad Oggetti
autore Furno Francesco, mat.2042327
titolo Gestore tickets Musei Civici di Vicenza

Introduzione

Il progetto che ho sviluppato fornisce un'applicazione, corredata da interfaccia grafica, che permette la vendita e gestione di biglietti-ingresso per i Musei Civici di Vicenza. Per la sua ideazione mi sono ispirato alla mia esperienza "sul campo" come addetto agli ingressi presso il sistema museale di Vicenza.
In particolare, l'applicazione permette l'inserimento, la modifica e la cancellazione di biglietti all'interno di una lista e fornisce delle statistiche riassuntive relative alle visite, tra cui il guadagno totale, il numero di biglietti emessi ed il museo più visitato.
Il progetto è stato sviluppato utilizzando Qt Creator con versione delle librerie Qt 6.2.4 in ambiente OS Ventura 13.3 con compilatore clang.

Il modello logico



Il modello parte da una classe astratta *biglietto* che contiene le informazioni comuni a tutti i biglietti selezionabili, ovvero identificatore, età e residenza del turista e data di acquisto, per ciascuno dei quali sono implementati metodi getter e setter. La sotto-gerarchia *singolo* si riferisce al biglietto associato ad un solo un museo. Si articola in *singoloOrdinario* e *singoloRidotto*. Il prezzo dei biglietti ordinari viene calcolato a seconda dei musei inseriti. Per quelli ridotti inoltre è prevista una riduzione fissa. È possibile creare un biglietto ridotto omaggio settando il boolean *gratuito* a true. La scelta di dividere Ordinario e Ridotto è dovuta al fatto che spesso i musei non sono visitabili integralmente a causa di allestimenti di mostre temporanee: per non modificare il prezzo di un museo da sistema è possibile generare un biglietto ridotto relativo al museo stesso.
La sotto-gerarchia *cumulativo* si riferisce ai biglietti associati ad almeno due musei e si articola in *cumulativoOrdinario* e *cumulativoRidotto*. È analogo alla gerarchia precedente, tuttavia i residenti in provincia di Vicenza, quindi con *residenza* == "VI", ottengono un ulteriore sconto sull'acquisto del biglietto.

La classe *listabiglietti* è il contenitore da me definito: è una lista che memorizza puntatori a biglietto polimorfi. Permette l'inserimento in testa alla lista e rende disponibili alcuni metodi che forniscono delle statistiche: *totaleBigliettiEmessi(), countSingolo(), countCumulativo(), guadagnoTotale(), museoPiuVisitato().*
La rimozione può avvenire cercando l'id di un biglietto specifico, svuotando la lista, oppure facendo il pop da testa o coda.
Il contenitore è dotato di una classe *iteratore*, che racchiude al suo interno un puntatore polimorfo a *biglietto*, che permette di scorrere l'intera lista tramite l'operatore ++ e di accedere ad un dato elemento tramite l'operatore [], grazie alla definizione degli usuali overloading.
La memoria è gestita tramite smart pointers per garantire estendibilità all'applicazione (in caso si volesse, per esempio, tenere più liste in contemporanea e si volesse fare condivisone di memoria su parte di esse).
Gli oggetti di tipo *biglietto* sono allocati esclusivamente sullo heap: a seconda del tipo dinamico del puntatore a biglietto viene invocato il distruttore corretto.

Polimorfismo

La classe *biglietto* è una classe base polimorfa perché possiede quattro metodi virtuali, di cui tre puri. Il distruttore è marcato virtuale, in questo modo quando viene invocata una delete, sia implicitamente che esplicitamente, su di un puntatore alla classe base *biglietto*, viene richiamato il distruttore corretto.
All'interno della contenitore *listaBiglietti* che ho creato, è presente il metodo *guadagnoTotale()* che scorre la lista e, per ogni biglietto, invoca il metodo virtuale *calcolaPrezzo()*: a seconda del tipo dell'oggetto puntato dal puntatore polimorfo verrà calcolato il prezzo diversamente.
Nei metodi *RimuoviDaTesta(), rimuoviDaCoda(), rimuoviBiglietto()*, prima dell'eliminazione del biglietto all'interno della lista, ne viene fatta una copia profonda utilizzando il metodo *clone()*, virtuale puro nella base ed implementato nelle classi derivate concrete, e restituito il puntatore. Le implementazioni del metodo virtuale puro sono davvero implementazioni anche se il tipo di ritorno cambia, in quanto il nuovo tipo è covariante con il tipo specificato nel metodo virtuale puro. Quindi, viene invocato il metodo clone corretto a seconda del tipo dinamico del puntatore a biglietto.

Non sempre è stato possibile implementare le varie funzionalità tramite metodi virtuali, per questo è stato utilizzato il dynamic cast come opreatore di RTTI. Per esempio:
MainWindow::slotSalvaFile()
A seconda del tipo dinamico del biglietto il salvataggio all'interno dell'albero XML richiede elementi diversi.
modificaBW::slotAggiornaDati()
A seconda del tipo dinamico del biglietto che si vuole modificare viene visualizzato un determinato widget: nel caso di un biglietto singolo viene mostrato un *QComboBox*, per un cumulativo vengono mostrati sei *QCheckBox*.
eliminaBW::slotAggiornaUltimo()
A seconda del tipo dinamico del biglietto appena eliminato viene visualizzato un determinato widget: nel caso di un biglietto singolo viene mostrata un *QLineEdit*, per un cumulativo viene mostrato un *QPlainTextEdit* con i relativi musei.

Persistenza dei dati

Per la persistenza dei dati viene utilizzato il formato XML. Un esempio della struttura dei file è dato dal file dati.xml fornito insieme al codice.

Funzionalità implementate

Le funzionalità implementate prevedono, come indicato nell'introduzione, l'inserimento, la modifica, la cancellazione di biglietti all'interno di una lista che è possibile consultare.
Dopo ogni operazione andata a buon fine, le statistiche della schermata principale vengono aggiornate grazie all'emissione di un *signal*.
Se non è la prima volta che si apre il programma e si vuole caricare un lavoro salvato in precedenza, fare click in alto a sinistra su *file->Apri*: i biglietti verranno creati e inseriti all'interno della lista, mentre le statistiche verranno aggiornate.
Inserimento di un nuovo biglietto
All'interno della schermata è possibile inserire i dati relativi al biglietto che si vuole creare. Vengono effettuati alcuni controlli affinché l'utente non inserisca un biglietto privo di ID o di museo. Il campo residenza può essere vuoto, il campo età è impostato di default a 18 ed il campo data al giorno corrente.
Quando il tasto "inserisci biglietto" viene premuto, se tutti i dati sono stati inseriti correttamente, il biglietto viene inserito nella lista e viene mostrato il costo nella parte inferiore della schermata.
Il tipo del biglietto che viene generato è affidato al sistema in base ai dati inseriti (in particolare età e residenza), tuttavia è possibile forzare la creazione di un biglietto ridotto o gratuito selezionando l'apposito bottone.

È possibile tornare alla schermata principale con il tasto indietro.

Modifica di un biglietto

Spostandosi in un'altra casella dopo aver digitato una stringa nel campo ID, la schermata viene aggiornata con i dati relativi al biglietto con quell'id. Nel caso fossero presente più biglietti con lo stesso identificativo verrà selezionato il primo. Se il biglietto viene trovato all'interno della lista, i campi all'interno della schermata vengono aggiornati ed è possibile procedere alla modifica.

È possibile confermare le modifiche con l'apposito tasto in basso alla schermata.

Eliminazione di uno o più biglietti

La schermata mette a disposizione diversi modi per eliminare i biglietti. È possibile eliminare un biglietto inserendo il relativo ID (anche qui nel caso ce ne fossero molteplici con lo stesso identificativo verrà eliminato il primo), oppure eliminando il biglietto in testa o in coda alla lista.

Se l'eliminazione va a buon fine, nella parte inferiore della schermata vengono riportati i dati relativi al biglietto eliminato.

È possibile, inoltre, svuotare la lista con l'apposito tasto.

Visualizza lista

Permette di consultare i biglietti all'interno della lista

Rendicontazione ore

Attività	Ore Previste	Ore effettive
Studio e progettazione	8	8
Sviluppo del codice del modello	10	10
Studio del framework Qt	10	12
Sviluppo del codice della GUI	12	15
Test e debug	5	5
Stesura della relazione	5	5
totale	50	55

Il monte ore è stato leggermente superato in quanto lo sviluppo del codice della GUI ha richiesto più tempo di quanto previsto: in particolare inizialmente è stato utilizzato QT Designer, ma per scelta personale si è preferito ricominciare tramite codice puro.