

ARSENAL

Relazione progetto di programmazione ad oggetti

Federico Perin

1170747

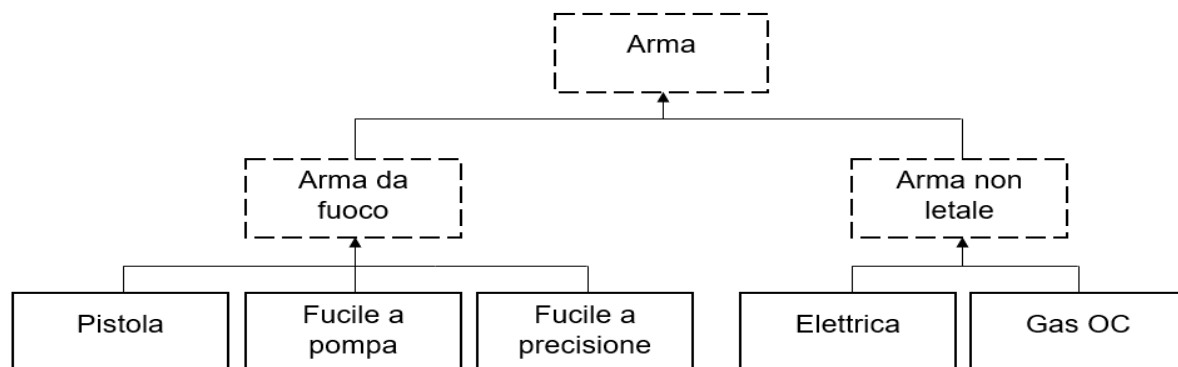
a.a. 2018-2019

Sommario

Introduzione	3
Gerarchia classi.....	3
Polimorfismo e metodi virtuali.....	4
Qontainer <T>.....	4
GUI	5
Esempi d'uso	5
Gestione del formato del file dati.....	7
Estensibilità e libreria Qt.....	7
Conteggio ore:	8
Note:	8

Introduzione

Arsenal è un'applicazione che permette la gestione di un archivio d'ordini di una armeria, consente quindi di gestire per ogni cliente le varie armi ordinate salvando per ogni cliente i propri ordini in file XML. Arsenal offre quindi la possibilità di aprire nuovi file oppure file già esistenti e la possibilità di salvare i dati in formato XML nei file. Come già scritto ogni file contiene tutte le armi ordinate da un cliente quindi può essere visto come un contenitore su cui sono state implementate le funzioni di inserimento, visualizzazione armi ordinate (con la possibilità di mostrare i dettagli e mostrarli in modo ordinato secondo due modalità), eliminazione, ricerca, modifica, e calcolo del costo totale delle armi ordinate.



Gerarchia classi

La gerarchia è composta dalle seguenti classi:

- **Arma:** La classe Arma è la classe base e risulta essere una classe astratta polimorfa la quale rappresenta una generica arma che può essere venduta da una armeria. La classe ha come attributi (privati), marca, nome, foto (contiene in formato stringa l'immagine dell'arma) costo, disponibilità.
- **Arma da fuoco:** È un sottotipo diretto della classe base Arma. Risulta essere una classe astratta polimorfa, rappresenta la categoria delle armi da fuoco perciò come attributi privati si avranno attributi che rappresentano aspetti comuni di un'arma da fuoco, sono quindi presenti gli attributi calibro e cadenza di fuoco.
- **Pistola:** È un sottotipo della classe Arma da fuoco. È una classe concreta polimorfa che rappresenta le pistole che possono essere vendute in una armeria. Sono presenti come attributi torcia (indica se la pistola è provvista di torcia o no), silenziatore (indica se la pistola è provvista di silenziatore o no) e automatica (indica se la pistola è automatica o no).
- **Fucile a pompa:** È un sottotipo della classe Arma da fuoco. È una classe concreta polimorfa che rappresenta i fucili a pompa che possono essere vendute in una armeria. Ha come attributo doppia canna (indica se l'arma è a singola canna o doppia).
- **Fucile a precisione:** È un sottotipo della classe Arma da fuoco. È una classe concreta polimorfa che rappresenta i fucili a precisione che possono essere vendute in una armeria. Sono presenti come attributi silenziatore (come pistola) e mirino (rappresenta il tipo di mirino con il quale è equipaggiato il fucile).
- **Arma non letale:** È un sottotipo diretto della classe base Arma. Risulta essere una classe astratta polimorfa, rappresenta la categoria delle armi non letali perciò come attributi privati si avranno attributi che rappresentano aspetti comuni di un'arma da letale, è presente quindi l'attributo ricaricabile.
- **Elettrica:** È un sottotipo della classe Arma non letale. È una classe concreta polimorfa che rappresenta le armi che fanno uso di scariche elettriche, come attributi privati si hanno, voltaggio, amperaggio, durata scarica, distanza massima.
- **Gas Oc:** È un sottotipo della classe Arma non letale. È una classe concreta polimorfa che rappresenta le armi che fanno uso di sostanze urticanti, come attributo ha capacità.

Sono ovviamente presenti metodi virtuali puri e per ogni attributo è stato implementato una funzione di get e di set.

Polimorfismo e metodi virtuali

Come indicato nelle specifiche, viene utilizzato per gran parte del progetto il polimorfismo infatti sono stati implementati i seguenti metodi virtuali:

- Virtual Arma* clone () const =0: metodo polimorfo puro, restituisce un puntatore ad una copia dell'oggetto d'invocazione.
- Virtual bool Armadafuoco () const =0: metodo polimorfo puro, restituisce true se l'oggetto d'invocazione è sottotipo di Arma_da_fuoco altrimenti se non lo è restituisce false. Viene dichiarato final override perché non avrebbe senso ulteriori override dato che un oggetto se è di sottotipo di Arma_da_fuoco allora sicuramente è un Arma_da_fuoco.
- Virtual string classe () const =0: metodo polimorfo puro, restituisce una stringa in cui viene indicato il tipo delle classi concrete derivate, questo metodo viene implementato per avere una ottimizzazione dell'operazioni, infatti tale metodo viene utilizzato al posto del dynamic_cast che risulta meno efficiente perché utilizzano le vtables mentre classe () sfrutta il dynamic binding.
- Virtual float CalcolaCosto () const = 0: metodo polimorfo puro, restituisce il costo finale il quale varia a seconda del tipo dell'oggetto d'invocazione.
- Virtual ~Arma () =default: distruttore virtuale standard.

Qontainer <T>

L'applicazione fornisce un template di classe Qontainer<T> i cui oggetti rappresentano un contenitore di oggetti di tipo T. Il template Qontainer<T> contiene le funzionalità principali di inserimento, rimozione, ricerca, ma anche metodi per la gestione dei dati, esso contiene le classi interne iterator e const_iterator che simulano le due tipologie di iteratori disponibili in un container standard, vengono poi implementati vari overloading per vari operatori. È stato scelto di implementare come container un array dinamico secondo diverse motivazioni. Come noto un array dinamico consente l'inserimento e la rimozione in coda in tempo ammortizzato costante ma cosa di fondamentale importanza, soprattutto nel contesto dell'applicazione sviluppata è la possibilità di sfruttare l'accesso in posizione arbitraria in tempo costante $O(1)$. È fondamentale ciò nella manipolazione della tabella quando si vuole visualizzare, modificare un determinato oggetto o per ricercare per un determinato campo, si accede ad un elemento in posizione arbitraria in un tempo $O(1)$ senza dover scorrere tutto l'array diversamente dalla lista. Ad un primo approccio si potrebbe pensare che l'eliminazione in posizione arbitraria esclusa l'eliminazione in coda, sia meno efficiente rispetto ad un container lista, ma ciò non è vero grazie all'implementazione dell'applicazione, infatti quando si elimina un oggetto attraverso la manipolazione non si ha un iteratore ma la posizione intera che grazie l'accesso in posizione arbitraria in tempo costante $O(1)$ si punta subito l'oggetto da eliminare mentre con le liste no bisogna scorrere tutta la lista prima di trovare l'oggetto da eliminare, certamente però resta il fatto che una volta eliminato l'oggetto occorre fare lo shift dei oggetti successivi e questo rimane più efficiente nelle liste, ma sul piano dei benefici è un prezzo che si può accettare. Infine, è più probabile che avvengano più operazioni di visualizzazione, modifica e ricerca in insieme rispetto a operazioni di eliminazione, quindi si preferisce avere un array dinamico rispetto ad una lista.

Nell'applicazione è stato implementato un template di classe DPtr<T> che opera in modalità profonda con opportuni metodi. Il contenitore, perciò conterrà oggetti di tipo DPtr<T>, che contengono un campo dati privato di tipo Arma* cioè un puntatore ad un oggetto Arma.

GUI

Come richiesto dalle specifiche del progetto, la GUI è stata separata dal modello logico aderendo al design pattern Model-View, sfruttando le classi messe a disposizione da Qt, perciò la parte Controller è all'interno di View.


Si è scelto di dividere le varie funzionalità della View in varie classi dove ognuna di esse ha un preciso compito (per esempio una classe gestisce la parte grafica della modifica mentre un'altra gestisce la parte grafica dell'inserimento). L'applicazione è quindi formata da le seguenti interfacce che ereditano da QWidget:

- Inserisci, interfaccia per l'inserimento, in base alle opzioni scelte vengono visualizzati i corretti campi dove inserire i dati.
- Menu, interfaccia per gestire la barra menu non eredita da QWidget ma da QMenuBar.
- Modifica, interfaccia modifica una arma selezionata.
- Ricerca, interfaccia per la ricerca secondo i filtri impostati dall'utente, fruttando la ricerca viene implementata anche la funzionalità di ordinamento grazie all'utilizzo di metodi della libreria Qt.
- VisualizzaOggetto, interfaccia per visualizzare i dettagli di una arma selezionata.
- Visualizza, interfaccia che istanzia la QtableView (la tabella per visualizzare i dati principali) e collega la tabella con il modello per gestirla per poter visualizzare i risultati delle varie operazioni implementate (ricerca, modifica, ordinamento e eliminazione)
- MainWindow, interfaccia principale dell'applicazione, si occupa di istanziare le varie interfacce e raccoglie tutti i segnali lanciati da quest'ultime, gestendoli con dei opportuni SLOT propri.

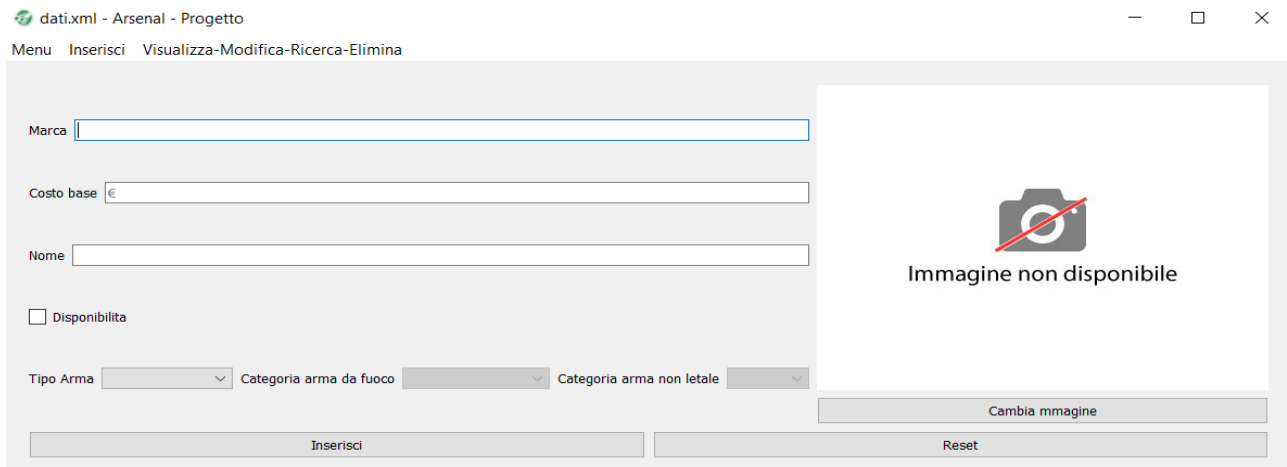
In modo simile a come era stato fatto per la classe Menu si è deciso, per rendere il codice più riutilizzabile anche per sviluppi futuri, per ogni combobox di creare una classe a sé che deriva da QComboBox creando così per ogni combobox una sottoclasse di QComboBox, un esempio di riutilizzo è la classe comboboxMirino che viene utilizzata nell'interfaccia di inserimento e nell'interfaccia di ricerca. Infine, si è scelto di rendere disponibile l'inserimento, visualizzazione, e cambio dell'immagine che avviene tramite i metodi disponibile nella classe Gestionelmg.

Esempi d'uso

L'interfaccia grafica prevede la presenza di una barra menu in cui ci si può spostare nelle schermate implementate, è presente anche un sottomenu che permette le opzioni di salvataggio dati, caricamento dati e creazione di nuovi file XML.

 dati.xml - Arsenal - Progetto

Menu Inserisci Visualizza-Modifica-Ricerca-Elimina



La schermata d'inserimento prevede la possibilità di inserire nei vari campi, i dati corretti e con la possibilità di inserire un'immagine dell'arma. A seconda di cosa si sceglie nelle combobox verranno visualizzati i giusti campi per ogni classe derivata di Arma.

Ordina ↓	Ordina ↑	Ricerca	Marca	Reset
Marca	Costo Finale	Nome	Disponibilit�	Classe
Beretta	266,5	Beretta 93R	false	Pistola
Mossberg	581,5	Mossberg ...	true	Fucile a po...
Barrett Fir...	961,5	Barrett M82	true	Fucile a pr...
Taser	220	Taser X26	true	Elettrica
Sabre USA	25	Sabre US...	true	Gas OC
Ac Milan	4e+07	Piatek	false	Pistola

Visualizza dettagli Modifica Elimina Calcola Costo totale

La schermata di Visualizzazione permette di mostrare attraverso la tabella tutti gli oggetti Arma presenti, in pi  permette di selezionare una per volta, una riga della tabella sulla quale posso essere applicate l'operazione di visualizza dettagli, modifica e eliminazione cliccando i rispettivi tasti. Cliccando sulla calcola costo totale si sommano tutti i costi finali   il risultato viene poi mostrato all'utente.

Arsenal

Tipo Elettrica

Marca: Taser

Costo base: 50.000000

Nome: Taser X26

Disponibilit : Si


Ricaricabile: Si

Voltaggio: 50.000000

Amperaggio: 10.000000

Durata scarica: 60.000000

Distanza massima: 6



Arsenal


Classe Pistola

Marca Beretta

Costo Base 100

Nome Beretta 93R

Disponibilit  false



Cambia Foto

Calibro 9.000000 Cadenza di fuoco 1100

Torcia false

Silenziatore true

Automatica true

Salva

Viene implementata l'operazione di ricerca che in base al campo selezionato nella combobox viene scelto per quale attributo fare la ricerca aggiungendo una colonna in pi  se il filtro selezionato non   presente nella tabella cio  il filtro non   la colonna Marca o Costo o Nome o Disponibilit  o Classe ma   per esempio Calibro che viene messa temporaneamente fino ad una nuova ricerca la quale sostituir  la colonna con una nuova colonna. Se si ricerca per un attributo che   una stringa verr  visualizzata una QLineEdit dove poter inserire la parola da ricercare che deve essere contenuta nell'attributo scelto (di default viene ricercata una stringa vuota che secondo la propriet  del metodo utilizzato per la ricerca, se c'  una stringa vuota da ricercare lascia visualizzate tutti gli oggetti), se si ricerca un booleano vengo visualizzati due bottoni cliccabili che non possono essere selezionati contemporaneamente, per ricercare gli attributi che hanno valore True oppure per False (di default ricerca per True), se invece si ricerca un valore numerico viene visualizzata sempre una QLineEdit dove scrivere il numero e in pi  due bottoni cliccabili per indicare se deve essere maggiore uguale il valore del campo selezionato nella combobox rispetto al valore inserito nella LineEdit oppure minore o uguale (di default ricerca i valori maggiori o uguali a 0). Un'altra funzionalit  implementata   l'ordinamento che in base sempre alla scelta fatta sull'elenco della combobox permette di ordinare in modo crescente o decrescente asseconda di quale bottone ordina   stato premuto. In pratica l'ordinamento sfrutta le funzioni implementate in ricerca quindi se si seleziona una colonna di un attributo presente solo in certe sottoclassi di Arma verr  sfruttata la ricerca per mostrare solo gli oggetti aventi tale attributo per cui si   deciso di ordinare, chiaramente se si effettua una ricerca in cui si ricerca per esempio una stringa in un campo dati saranno ordinati solo i risultati prodotti dalla ricerca.

Gestione del formato dei file dati

Come detto all'inizio, l'applicazione Arsenal permette il caricamento e il salvataggio di dati attraverso file XML. Si è scelto di utilizzare il linguaggio XML per salvare i dati dell'applicazione, perché di per sé l'XML è facilmente comprensibile dalle persone e la libreria Qt mette a disposizione metodi e classi per la gestione e l'utilizzo di tale linguaggio. Come scelta progettuale si è scelto di implementare i metodi di lettura e scrittura (read e write) nel file in una classe apposita chiamata "xmlio" per mantenere separate le varie funzionalità del codice.

La read è implementata nel seguente modo, apre il file in modalità lettura, verifica che sia stato letto il tag radice <root> se sì, vengono letti poi i giusti tag dell'oggetto Arma (ogni oggetto arma inizia e finisce con il tag <Arma> al cui interno ha un attributo in cui viene indicata la sottoclasse di Arma di cui fa parte l'oggetto) e caricati nel container allocandoli, una volta arrivati alla fine del ciclo si salta al prossimo oggetto arma se esiste altrimenti termina il ciclo.

La write è implementata nel seguente modo, apre il file in modalità scrittura, accede al container attraverso un iteratore e scorre tutti i vari oggetti salvati, salvando i vari attributi di ogni oggetto come campi nel file XML.

Si ha quindi il seguente frammento di codice XML:

```
<root>

  <Arma Classe =" classe dell'oggetto">

    <Campo_dati1> dato1 </Campo_dati1>

    .....

    <Campo_datiN> datoN </Campo_datiN>

  </Arma>

  .....

</root>
```

Estensibilità e libreria Qt

Come richiesto dalle specifiche è stato rispettato il requisito di produrre codice estendibile, infatti la gerarchia può essere ulteriormente ampliata aggiungendo nuove classi. È stato usato in largo uso il polimorfismo all'interno dell'applicazione, sono state separate le varie classi in modo tale da poter essere riutilizzate, infatti sono state create delle sottoclassi di QComboBox e di QTableView in modo tale da avere degli oggetti grafici che possano essere riutilizzati anche per nuove implementazioni di funzionalità future. Un ulteriore esempio di estensibilità è come è stata implementata la funzionalità di ordinamento, sono stati riutilizzati i metodi utilizzati per la ricerca (per ordinare solo certi oggetti che avevano certi campi per esempio "calibro") applicando il metodo per l'ordinamento fornito da Qt, permettendo così di ordinare riutilizzando del codice già presente. Fondamentale perciò è stato utilizzare i metodi e classi fornite dalla libreria Qt per poter implementare le varie funzionalità dell'applicazione, infatti per visualizzare i dati e tenerli sempre aggiornati dopo operazioni inserimento, modifica, ricerca e ordinamento si è usato la QtableView di Qt. Per poter funzionare si è utilizzato un delegato predefinito e un modello custom Qintermedio che è sottotipo di QSortFilterProxyModel. Questa classe ha lo scopo di fare da ponte con il modello custom QGestioneTabella che deriva dalla classe astratta QAbstractTableModel. Quando si effettua una modifica di un oggetto Arma grazie alle classi di Qt, la modifica si propaga nella giusta riga del QtableView, analogamente per la ricerca e per l'eliminazione. Perciò QGestioneTabella mette a disposizione tutti i metodi necessari per poter visualizzare, eliminare, inserire e modificare i dati. Altri utilizzi della libreria Qt si hanno nella gestione dell'immagine.

Conteggio ore:

- Analisi preliminare del problema ≈ 2 h
- Progettazione modello ≈ 2 h
- Progettazione GUI ≈ 2 h
- Apprendimento libreria Qt ≈ 10 h
- Codifica modello ≈ 12 h
- Codifica GUI ≈ 14 h
- Debugging ≈ 2 h
- Testing ≈ 5 h
- Stesura relazione ≈ 3 h

Note:

L'intero progetto per evitare conflitti dovuti a diverse versioni di Qt, è stato sviluppato nella macchina virtuale fornita durante il corso di "programmazione ad oggetti", perciò il sistema operativo che è stato utilizzato per la produzione dell'applicazione è stato Ubuntu 18.04.2 LTS, con Qt creator versione 5.9.5.