UNIVERSITÀ DEGLI STUDI DI PADOVA

MY WINE SELECTA
Progetto di programmazione ad oggetti

Studente:
FRANCESCO BARI (1142849)

Docente:
Prof. FRANCESCO RANZATO

ANNO ACCADEMICO 2018-2019

INDICE

INTRODUZIONE

My Wine Selecta è una applicazione che permette di gestire un catalogo personalizzato di vini. È stata pensata come applicazione per gestire particolari tipi di vino, facenti parte della cosiddetta categoria dei "vini speciali".

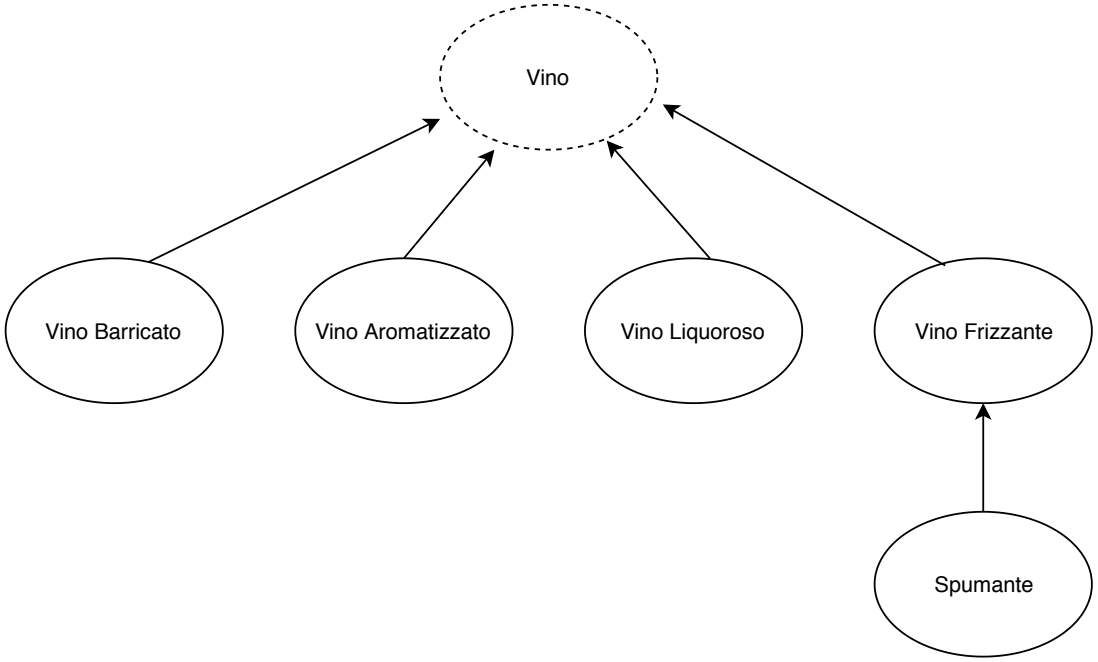
È possibile creare da zero il proprio catalogo, così come è possibile caricare da file un catalogo già esistente. Ovviamente è possibile salvare i propri dati su file. Ogni file in questo senso rappresenta un contenitore sul quale sono permesse operazioni di inserimento, ricerca, modifica e rimozione di oggetti.

DESCRIZIONE CLASSI E GERARCHIA

Alla base della gerarchia troviamo la classe base astratta polimorfa Vino, che rappresenta un generico vino. La classe è composta da campi dati caratteristici di un vino e da una serie di metodi virtuali, puri e non, che verranno poi implementati nelle sottoclassi concrete. L'argomento verrà poi ripreso nella sezione "POLIMORFISMO". La classe è inoltre fornita di costruttori, operatori virtuali, distruttore virtuale e metodi getter e setter relativi agli attributi.

Ci sono quattro sottoclassi concrete polimorfe al livello successivo della gerarchia, VinoBarricato, VinoAromatizzato, VinoLiquoroso e VinoFrizzante. Inoltre è presente un ulteriore livello di gerarchia rappresentato dalla sottoclasse concreta polimorfa Spumante, che deriva da VinoFrizzante.

Tutte le classi concrete implementano tutti i metodi virtuali puri di Vino. Ognuna di queste classi aggiunge uno o più attributi tipici della propria tipologia di vino a quelli già presenti nella classe base Vino (per esempio per VinoLiquoroso viene aggiunto un campo liquoreAggiunto e cosi via per le altre classi). La classe Spumante aggiunge un attributo residuoZuccherino, tipico della classificazione degli spumanti, rispetto alla classe VinoFrizzante dal quale appunto deriva.



In caso di necessità, sarà possibile estendere la gerarchia sopradescritta aggiungendo sottoclassi sia alla classe base Vino che alle sottoclassi già presenti, aumentando così la profondità della gerarchia. Ovviamente saranno da implementare opportunamente i metodi virtuali per ogni nuova sottoclasse.

CONTAINER

Il modello logico dei dati sfrutta un opportuno contenitore creato per l'occasione. Si tratta di un array dinamico utilizzato per contenere i puntatori polimorfi Vino*, sfruttando l'accesso in posizione arbitraria in tempo costante e l'inserimento e la rimozione in coda in tempo ammortizzato costante.

L’applicazione consente anche la rimozione in posizione arbitraria. In questo caso, l’uso di una lista doppiamente concatenata sarebbe stata più efficiente (infatti consente rimozione ed inserimento in posizione arbitraria in tempo costante contro il tempo lineare ammortizzato dell’array dinamico). Ho comunque preferito implementare un array dinamico per le seguenti motivazioni:

- Gli accessi in posizione arbitraria sono molti di più rispetto alle rimozioni
- Per rimuovere un elemento in posizione arbitraria potremmo non avere a disposizione l’iteratore corrispondente a tale elemento, ma solamente la posizione rappresentata da un numero intero. In questo caso non avremmo nessun vantaggio dato dall’uso di una lista doppiamente concatenata.

GUI

Per quanto riguarda la GUI, l’applicazione è composta da tre interfacce principali: il Catalogo, contenente tutti i vini della nostra selezione, e due interfacce dedicate Inserisci e Ricerca, utilizzate rispettivamente per l’inserimento e per la ricerca di determinati vini.

Sono state quindi definite due classi: la prima, VinoListWidget, viene utilizzata per le operazioni di inserimento e ricerca (utilizzando i vincoli più tipici di una ricerca di vini, scelti tra l’insieme degli attributi), ridefinendo ad hoc alcuni metodi ereditati da QListWidget; la seconda, VinoListItem viene utilizzata per le operazioni di modifica di alcuni campi dati, eredita da QListWidgetItem aggiungendo un parametro Vino* al costruttore che servirà per ottenere le informazioni da rappresentare e in caso modificare in seguito. (Nel nostro caso il prezzo e l’immagine del vino, per aggiornare il Catalogo nel tempo e inserire eventuali sconti).

POLIMORFISMO

Il polimorfismo è stato utilizzato nella gerarchia con l'utilizzo di vari metodi virtuali. Oltre al distruttore virtuale, sono stati definiti metodi per la clonazione polimorfa, operatori virtuali di uguaglianza e disuguaglianza e i metodi sopradescritti getInfo e getTipo virtuali per tutta la gerarchia. Quest'ultimo è marcato virtuale puro, così come il metodo di clonazione polimorfa. Il distruttore virtuale è esplicitamente marcato di default (=default).

Inoltre è stato marcato esplicitamente come non disponibile (=delete) l'assegnazione, precedentemente marcata come virtuale, rompendo così la regola dei tre, permettendo però istruzioni del tipo Vino* x = y.clone(); grazie alla clonazione polimorfa e alla covarianza.

FORMATO DEI FILE

L'applicazione consente il salvataggio e l'apertura di file che rappresentano il contenuto del Container. Per fare questo utilizza file in formato XML che ben si prestano ad essere utilizzati da Qt, che mette a disposizione vari strumenti e metodi per la loro manipolazione. Inoltre questo tipo di file presenta il vantaggio di essere facilmente comprensibile dalle persone.

Il compito di apertura, caricamento e salvataggio dei file è lasciato alle funzioni load e save presenti nel Model.

ESTENSIBILITÀ

Per quanto riguarda l'estensibilità del programma, per aggiungere un nuovo tipo di vino, basterà estendere la gerarchia e fare le opportune modifiche nel widget relativo al ComboBoxTipo (creato ad hoc per permettere la selezione del tipo di vino da inserire o ricercare) e nei metodi di save e load presenti nel Model.

Per aggiungere invece nuove funzionalità al programma basterà modificare opportunamente le classi widget precedentemente descritte.

SCELTE PROGETTUALI

Per quanto riguarda le scelte progettuali, si è deciso di aderire al design pattern MVC, ossia Model, View, Controller. Il Model (come descritto precedentemente) include quindi il modello logico dell'applicazione, la View la parte grafica dei widget ereditati e ridefiniti per il nostro scopo e il Controller, posto come intermediario e delegato tra le due parti, si pone il compito di definire tutti gli slot per le funzionalità del Model, connessi tramite connect ai vari signal derivanti da operazioni svolte sulla View.

Una importante scelta progettuale è stata quella di utilizzare due diverse istanze dei contenitori (Container<Vino*>) nella classe Model, uno per contenere i dati veri e propri, rappresentati quindi nel Catalogo, l'altro per contenere una copia del primo dal quale verranno eliminati i vini che non soddisfano i vincoli di ricerca scelti (per la sezione Ricerca appunto) .

CONTEGGIO DELLE ORE

Il tempo impiegato per lo sviluppo dell'applicazione è all'incirca di 55 ore, così distribuite:

- Analisi del progetto: 3 ore
- Progettazione modello: 4 ore
- Progettazione GUI: 5 ore
- Apprendimento libreria Qt: 14 ore (comprensive di tutorato)
- Codifica modello + Container: 10 ore
- Codifica GUI: 10 ore
- Testing + debugging: 6 ore
- Stesura relazione: 3 ore

NOTE

Il file .pro che è stato consegnato è stato modificato aggiungendo QMAKE_CXXFLAGS += -std=c++11 .

I comandi utilizzati sono stati qmake; make;

È stato aggiunto un file data.xml che può eventualmente essere aperto dall'applicazione per consentire alcuni test con dei dati già pronti all'uso.

L'intero progetto è stato sviluppato su sistema operativo macOS Mojave 10.14.4, Qt Creator 5.9 e clang-1000.10.44.4.