Progetto di Programmazione ad Oggetti

A.A. 2019/2020

SpesaBottega

De Grandi Samuele, 1146136

INDICE

1. ABSTRACT

SpesaBottega è un'applicazione che permette di gestire dati che riguardano i generi alimentari tipici di una piccola bottega di quartiere e nello specifico la loro vendita. L'applicazione rende possibile inserire, modificare ed eliminare un genere alimentare all'interno di un carrello della spesa. In particolare, è possibile selezionare un panino, ottimo ad esempio per una pausa pranzo veloce, in base alle caratteristiche dei suoi ingredienti.
I dati modificati vengono poi salvati automaticamente all'interno di un file in modo da garantirne il caricamento in un momento successivo. Inoltre, è possibile caricare da file un carrello della spesa già esistente, magari con una lista di generi alimentari preferita e/o ricorrente.

2. DESCRIZIONE DELLA GERARCHIA

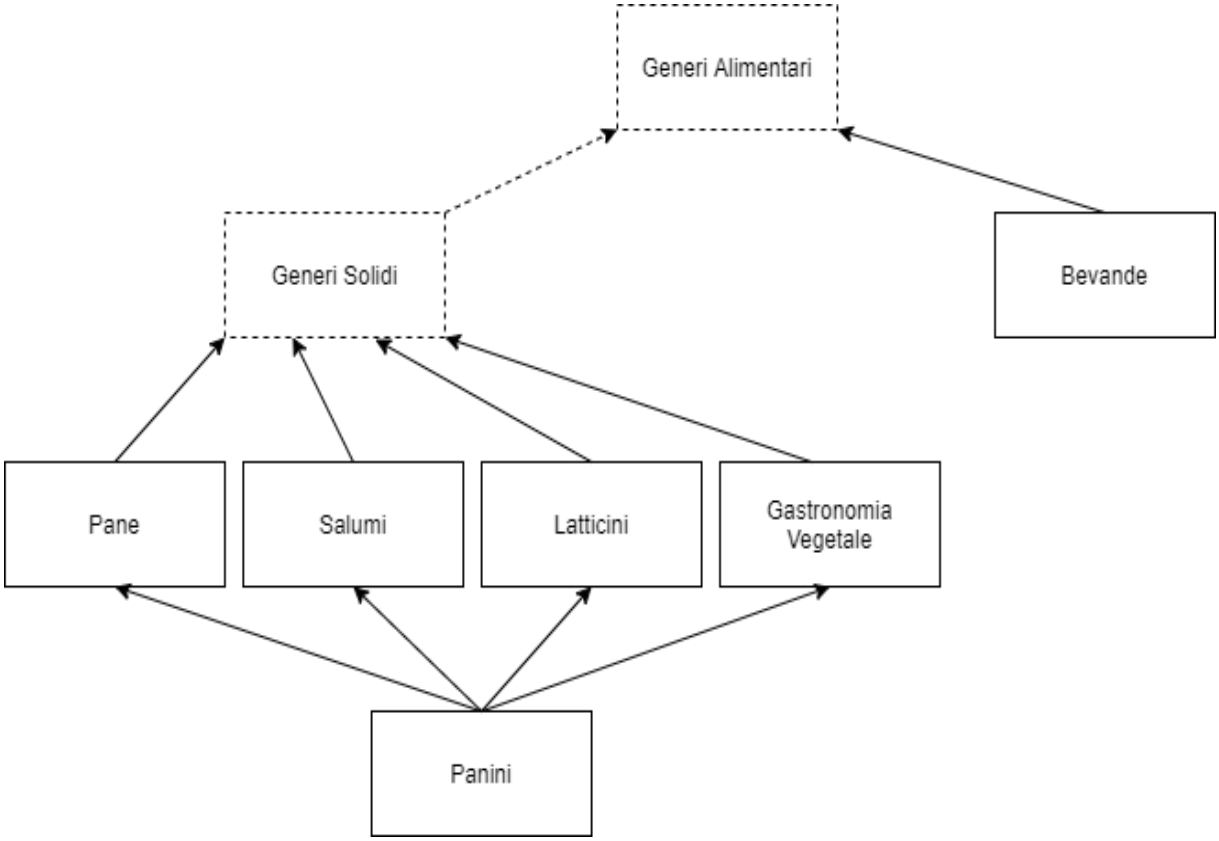

Figura 2: Gerarchia

Nell'immagine sopra indicata è presente l'intera gerarchia riguardante la parte logica del progetto, in particolare le classi tratteggiate sono classi polimorfe astratte, mentre le restanti sono concrete.

Alla base della gerarchia è presente la classe base astratta polimorfa GeneriAlimentari, che rappresenta un generico alimento o bevanda. La classe è composta da campi dati caratteristici

di un genere alimentare e da una serie di metodi virtuali, che verranno poi implementati nelle sottoclassi concrete. Inoltre, la classe è fornita di costruttori, operatori virtuali, distruttore virtuale e metodi get/set relativi agli attributi.
Ci sono due sottoclassi che derivano direttamente e sono una astratta Generi Solidi e una concreta Bevande. Sono presenti quattro sottoclassi concrete polimorfe al livello successivo della gerarchia di Generi Solidi: Pane, Salumi, Latticini e GastronomiaVegetale. Inoltre, è presente un ulteriore livello di gerarchia rappresentato dalla sottoclasse concreta polimorfa Panini, che deriva dalle quattro classi precedenti e va a chiudere il diamante della gerarchia con ereditarietà multipla appena descritta.
Tutte le classi concrete implementano i metodi virtuali puri di GeneriAlimentari e Generi Solidi. Ciascuna di queste sottoclassi aggiunge un attributo tipico dell'alimento a quelli già presenti nella classe base GeneriAlimentari e nella derivata Generi Solidi: per la classe Pane viene aggiunto un campo consistenza, per Salumi uno che ne indica la tipologia di conservazione, per Latticini viene descritta la digeribilità, per Gastronomia Vegetale il contenuto di vitamine e fibre, infine per la classe Panini viene aggiunto un attributo ingredienti, che permette di capire di quali alimenti è composto un panino e quindi stabilire le sue caratteristiche.

3. UTILIZZO DEL POLIMORFISMO

Il polimorfismo è stato utilizzato nella gerarchia grazie ai metodi virtuali presenti nelle classi. Oltre al distruttore virtuale, sono stati definiti metodi per la clonazione polimorfa, operatori virtuali di uguaglianza e disuguaglianza e i metodi getInfo() e getType(), virtuali per tutta la gerarchia. Questi ultimi due in particolare sono utili per conoscere la descrizione e il tipo di un genere alimentare, ad esempio, nella prima fase di ricerca, nel calcolo del numero di prodotti e del prezzo del carrello o nelle fasi di caricamento e salvataggio dei dati. Inoltre, getType() è marcato virtuale puro, così come il metodo di clonazione polimorfa, mentre il distruttore virtuale, utile per evitare di lasciare memoria allocata, è esplicitamente marcato di default.
Grazie all'utilizzo del polimorfismo è possibile estendere la gerarchia aggiungendo altre tipologie di generi alimentari, sia alla classe GeneriSolidi, sia alla classe Bevande, evitando di riprogettarla.

4. CONTENITORE

Il modello logico dei dati sfrutta un opportuno contenitore creato per l'applicazione. Si tratta di un array dinamico utilizzato per contenere i puntatori polimorfi GeneriAlimentari*, sottoforma di template, con le principali funzionalità e gli operatori necessari a garantirne un utilizzo ideale. La scelta è ricaduta su questa tipologia di contenitore per far fronte in modo efficiente alle numerose operazioni di accesso in posizione arbitraria e di inserimento in coda (infatti la prima operazione avviene in tempo costante e la seconda in tempo ammortizzato costante).
Si possono trovare tre diverse istanze dei contenitori Container<GeneriAlimentari*> nella classe Model: una per contenere la lista completa dei generi alimentari, nel Catalogo, la seconda per contenere una copia del primo dal quale verranno eliminati i generi alimentari che non soddisfano i vincoli di ricerca selezionati (per la sezione Cerca appunto) e infine l'ultima per contenere i generi alimentari aggiunti nel carrello, dopo la fase iniziale di ricerca.

5. GUI

Per quanto riguarda la GUI, l'applicazione è composta da due interfacce principali: Cerca, dedicata alla ricerca di determinati generi alimentari e Carrello dove si possono aggiungere i

generi alimentari precedentemente trovati. Sono state quindi definite quattro classi. La prima, ListWidget, viene utilizzata per le operazioni di ricerca e ridefinisce alcuni metodi ereditati da QListWidget. La seconda, ListView, deriva da QListWidgetItem e aggiunge un parametro GeneriAlimentari* al costruttore, che serve per ottenere le informazioni da modificare e in caso aggiornare in seguito. La terza, ModificaGeneriAlimentari, crea una finestra QDialog e attraverso uno QSpinbox permette di scegliere la quantità desiderata, che grazie alla classe ListView, aggiorna anche il peso e il prezzo del singolo genere alimentare selezionato. Riguardo la quarta classe, FilterPanini, appena nell'interfaccia Cerca viene selezionata la classe Panini, viene creata una finestra QDialog, con i vincoli presi dalle sue superclassi dirette (consistenza, conservazione salume, digeribilità e contenuto di vitamine e fibre), che si possono applicare alla ricerca del panino.
E' presente inoltre una classe ConvertImage, utile per la conversione di stringhe in codifica a 64 bit delle immagini, in file png.

6. FORMATO DEI FILE PER L'I/O

L'applicazione permette il caricamento e il salvataggio dei dati attraverso l'utilizzo di file in formato XML. Ho scelto questo formato per via della sintassi intuitiva e in quanto Qt mette a disposizione una libreria per la gestione dell'I/O, in particolare le classi QXmlStreamReader e QXmlStreamWriter. Il compito di caricamento e salvataggio dei file è affidato alle funzioni load e save, rispettivamente di Carrello e Catalogo, presenti nel Model.

7. ORE UTILIZZATE

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

- Analisi preliminare del problema: 3 ore
- Progettazione modello: 3 ore
- Progettazione GUI: 3 ore
- Apprendimento libreria Qt: 10 ore
- Codifica modello: 15 ore
- Codifica GUI: 15 ore
- Debugging: 3 ore
- Testing: 3 ore

8. COMPILAZIONE

La cartella archivio.zip è composta dai files .h e .cpp contenenti il codice sorgente, un file relazione.pdf, due file XML, una immagine logo.png, un foglio di stile e il file resources.qrc necessario per visualizzare l'immagine e il css.
Per la compilazione è necessario entrare nella directory del progetto, eseguire il comando qmake -project "QT += widgets", qmake e successivamente lanciare il comando make. Infine per l'esecuzione è sufficiente l'istruzione ./SpesaBottega.

9. AMBIENTE DI SVILUPPO

L’intero progetto è stato sviluppato su sistema operativo Windows 10 Home 10.0.18362 64-bit, Qt Creator 5.9.5 e compilatore MinGW 5.3.0 32-bit.