Restaurant - Freedom

Mattia Zecchinato - Matricola 1006435

01 Febbraio 2016

Sommario

Relazione del progetto didattico del corso di Programmazione ad oggetti A.A. 2015/2016 tenuto dal professor Ranzato Francesco presso l'Università degli Studi di Padova

Capitolo 1

Progetto

1.1 Introduzione

Il progetto modella la gestione di una comanda di un ristorante.
Alla comanda è permesso aggiungere solo piatti presenti nel menu, possono essere cibi e bevande. Ad ogni elemento aggiunto alla comanda è possibile applicare delle modifiche in aggiunta o rimozione, si è deciso che ad ogni modifica in aggiunta/rimozione venga aumentato/diminuito il prezzo per una cifra pari a 0.20€.
Al pagamento del conto l'ordine viene chiuso e ne viene aperto uno nuovo.
Il menu è invece salvato in un database utilizzando JSON, è permessa l'aggiunta, la rimozione e la modica di un piatto.
Il menu a differenza dell'ordine è persistente nel tempo.
Il menu vine gestito da un container realizzato basandosi sulla struttura di una lista doppiamente linkata.
Nello sviluppo si è cercato di rispettare il pattern MVC (Model-View-Controller), dividendo la struttura, l'interfaccia grafica e il controllo.

1.2 Model

Sono presenti 2 gerarchie una con superclasse la classe astratta `Food` l'altra la classe concreta `Order`.
La gerarchia `Food` è estendibile in quanto non presenta numerosi vincoli, anche la gerarchia `Order` è estendibile avendo solo vincoli alla gerarchia `Food` attraverso puntatori alla classe base `Food` (polimorfismo).

1.2.1 Food

La classe `Food` è astratta in quanto presenta il distruttore come metodo virtuale puro.

Descrive in modo generale un cibo, i suoi campi dati sono: *name, price* e *category*.

1.2.2 Dish

`Dish` è una classe concreta sottotipo di `Food`, aggiunge ai campi dati ereditati una `GList<QString>`, chiamata *ingredients* che elenca gli ingredienti presenti nel piatto. Si è scelta un contenitore lista in quanto gli ingredienti vengono, nella maggior parte dei casi, letti tutti in successione per la stampa della lista o avvengono inserimenti in coda in caso di aggiunta.
Presenta i metodi di scrittura e lettura nel database di un oggetto `Dish`.

1.2.3 Beverage

`Beverage` è l'altra classe concreta sottotipo di `Food`, l'unico campo che viene aggiunto è *capacity* che descrive la quantità volumetrica di bevanda. Descrive in modo generico una bevanda e come per la classe `Dish` presenta innumerevoli possibilità di essere estesa, per esempio una suddivisione tra alcolici e analcolici. Sono presenti i metodi di scrittura e lettura nel database.

1.2.4 Order

`Order` è la classe costituita da una lista di `OrderItem` e descrive ogni singolo componente dell'ordine.
Si è scelta una lista in quanto l'inserimento viene fatto sempre in coda e non c'è bisogno di estrarre elementi specifici.

1.2.5 OrderItem

`OrderItem` descrive ogni piatto ordinato e oltre ad un puntatore `Food*` possiede i campi *id, quantity, changes* e *varPrice*, che rispettivamente rappresentano, un identificativo per l'item, la quantità ordinata, le modifiche apportate al piatto (aggiunte o rimozioni) e la variazione di prezzo.
Le variazioni sono di tipo `QString` contenenti il nome della modifica, il primo carattere è `+` o `-` e viene impostato automaticamente in base all'operazione scelta.
Per non modificare il prezzo, visto che non viene fatta una copia del piatto ma viene usato un puntatore, viene salvata la differenza di prezzo da quello originale a quello con le modifiche. Per semplicità si è deciso che ad ogni aggiunta e rimozione corrisponda un aumento o diminuzione di 0.20€.

1.3 View

Sono presenti diverse view le precedenti sono quelle di modifica dell'ordine e di modifica del menu.
La parte view è a volte non totalmente separata dal controller.

Per non creare interferenze si è deciso di disattivare i widget padri all'apertura di quelli creati per alcune funzionalità.

1.3.1 MainWidget

`MainWidget` è la prima schermata che si trova all'avvio del programma.
Nella barra dei menu, e solo in questo widget è presente la possibilità di modifica, rimozione e aggiunta di un cibo al menu.
Il widget è diviso in due pannelli, quello di destra contiene i pulsanti per l'aggiunta dell'ordinazione e in basso quello per il pagamento del conto. A sinistra invece è presente un'area scrollabile in cui saranno elencate le ordinazioni che comporranno l'ordine, con ognuna un pulsante per eliminarle.
Siccome non ha senso pagare per un conto vuoto, in caso di nessun elemento nel conto, il tasto per pagare viene disabilitato.

1.3.2 OrderItemWidget

Rappresenta il layout con cui viene mostrato ogni singolo elemento dell'ordine nella schermata `MainWidget`, mostra il nome, il prezzo unitario e il prezzo totale, inoltre contiene il pulsante che permette di eliminare l'elemento dall'ordine.

1.3.3 ItemWidget

Viene chiamata per aggiungere un elemento all'ordine e permette la scelta di un cibo (chiamata a `MenuWidget`), la quantità e l'aggiunta di modifiche (chiamata a `AddChange`). Ogni modifica può essere eliminata cliccando sul rispettivo pulsante `X`.

1.3.4 MenuWidget

Mostra in una finestra interamente scrollabile la lista di tutti i cibi presenti nel menu, vengono mostrati raggruppati per categorie e posti in ordine alfabetico sia le categorie che i cibi in esse contenuti.

1.3.5 MenuItemWidget

Rappresenta il layout con cui viene mostrato ogni singolo elemento dell'ordine nella schermata `MenuWidget`, mostra il nome, il prezzo e il pulsante che permette di eliminare l'elemento dall'ordine.
A differenza che il cibo sia un `Dish*` o un `Beverage*`, aggiunge sotto alle informazione sopra esposte rispettivamente la lista degli ingredienti o la capacità.
In base alla funzione necessaria viene passato un intero che provvede a modificare la funzione dei pulsanti di ogni singolo piatto. Le tre funzioni sono quelle di selezione, nel caso di visualizzazione del menu per scelta, il che ritornerà un

puntatore al piatto; di rimozione che elimina il cibo definitivamente dal menu e anche dal database e la modifica che apre un nuovo widget con tutte le informazioni del piatto.

1.3.6 NewFoodWidget

Questo widget è richiamato per aggiungere al menu o modificare un piatto. Al termine dell'operazione, il piatto aggiunto o modificato viene salvato in modo permanente nel database, per non perdere l'aggiornamento del menu.
In caso di aggiunta mostra una scelta sul tipo di piatto da inserire e poi una schermata con tutte le informazioni richieste.
Sono presenti dei controlli a video in caso non vengano completati tutti i campi richiesti.

1.3.7 BillWidget

Viene aperto se dal `MainWidget` si preme il pulsante per pagare e mostra il totale del conto e la possibilità di confermare il pagamento. In caso positivo l'ordine viene cancellato e ricreato, ritornando alla schermata chiamante con un ordine nuovo.

1.4 Controller

Il controller è gestito dalla classe `Actor` che contiene il *menu* come `GList<Food*>` e un `Order`.

1.4.1 Actor

La classe `Actor` presenta i metodi per la modifica del menu, con inserimento, rimozione e ricerca di un cibo, inoltre presenta i metodi per il salvataggio e la scrittura nel database.

1.5 Container

Come contenitore si è deciso di usare una lista doppiamente linkata, in quanto le maggiori operazioni eseguite sono inserimenti in coda o comunque la lettura in ordine degli elementi della lista.

1.5.1 GList

`GList` è stata creata seguendo le linee base si una lista doppiamente linkata.
Presenta quindi i metodi: `begin, end, size, empty, insert, erase, push, pop, clear, merge, sort`.
I suoi nodi sono formati da puntatori smart che tengono conto del numero

di riferimenti all'oggetto, in questo modo è permessa anche condivisione di memoria.

1.6 Database

Per il salvataggio del menu si è scelto di usare JSON, consigliato nella documentazione Qt.
La libreria Qt infatti offre la gestione dei dati in modo semplice e veloce. Il salvataggio avviene utilizzando metodi richiamati a cascata presenti nelle classi del model.
Il database viene totalmente gestito a monte dalla classe `Actor`, attraverso i metodi di scrittura e lettura del database e di modifica dei cibi.

Capitolo 2

Esecuzione

2.1 Ambiente di sviluppo

- Compilatore: GCC 4.6.3
- versione QMake: 3.0
- cersione Qt: 5.3.2
- Sistema Operativo: Ubuntu 12.04.1 LTS

2.2 Materiale consegnato

La cartella "mzecchin_pao1516_freedom" è così suddivisa:

- cartella "restaurant" contente i file `.h` e `.cpp` e i file `.pro` e `.pri` per una corretta compilazione; è inoltre presente un file "`menu.json`" che contiene il database JSON.
- "relazione.pdf" che contiene la relazione del progetto (file corrente)

2.3 Compilazione ed esecuzione

Attraverso la shell del terminale posizionarsi all'interno della cartella "restaurant" ed eseguire il comando `qmake`. Una volta generato il Makefile, sempre all'interno della cartella "restaurant", avviare il comando `make`. Questa operazione genererà un file eseguibile restaurant che potrà essere avviato tramite un doppio click.