Università degli Studi di Padova

Relazione del Progetto di Programmazione ad Oggetti

Luca Soldera - 1028464

In Ordine

Contents

1 Informazioni

1.1 Ambiente di Sviluppo

- **Sistema Operativo**: Windows 7
- **Compilatore**: MinGW 4.8.2, 32bit
- **Versione Qt Creator**: 3.5.1
- **Versione Qt**: 5.3

1.2 Materiale Consegnato

1.3 Compilazione

Attraverso la shell del terminale posizionarsi all'interno della cartella `ProgettoLucaSoldera` ed eseguire il comando `qmake`. Una volta generato il `Makefile`, eseguire il comando `make`. Questa operazione genererà un file eseguibile `In Ordine` che potrà essere avviato cliccandoci.

2 Scopo del Progetto

Il progetto `In Ordine` ha come scopo la creazione di un software per la raccolta di ordini in un'ipotetica pasticceria. Un ordine è formato dai seguenti elementi:

- **Data e Ora**: corrispondo al giorno e all'ora in cui il cliente desidera ritirare l'ordine;
- **Nome, Cognome e Numero di Telefono**: corrispondono ai dati del cliente;
- **Lista di Prodotti**: ciò che il cliente desidera ordinare;
- **Preventivo**: stima del costo dell'ordine.

La lista dei prodotti potrà contenere fino a **dieci** prodotti tra i seguenti:

- **Torta**: con i seguenti campi:
  - **Peso**: grandezza della torta espressa in Kg;
  - **Gusto**: il gusto che il cliente vorrà per la torta;
  - **Immagine**: booleano che segnala se la torta verrà decorata con un'immagine;
  - **Scritta**: scritta di decorazione.
- **Prodotto Dolce**: prodotto dolce diverso dalla torta (ad esempio pasticcini), con i seguenti campi:
  - **Peso**: quantità espressa in kg;
  - **Tipo**: tipologia del prodotto dolce che il cliente desidera;
  - **Prezzo**: prezzo al chilo del prodotto.
- **Prodotto Salato**: (ad esempio pizzette) con i seguenti campi:
  - **Pezzi**: quantità espressa in numero di pezzi;
  - **Tipo**: tipologia del prodotto salato che il cliente desidera;
  - **Prezzo**: prezzo per ogni pezzo.

`In Ordine` consente di **inserire** i vari campi, **modificarli** ed **eliminarli**. Inoltre consente di **salvare** e **caricare** ordini su/da file **XML**.

3 Classi

Nel progetto si è deciso di aderire il più possibile al design architetturale Model View Controller (MVC) per mantenere separate le classi grafiche dalle classi logiche. Come operatore di RTTI si è utilizzato esclusivamente il `dyinamic_cast`.

3.1 Model

In Ordine vuole rappresentare i dati dei prodotti di una pasticceria quindi si è deciso di creare una gerarchia di quattro classi: la classe base astratta è **Prodotti** che fornisce un metodo virtuale puro `preventivo()` per il calcolo del costo di ogni prodotto e viene implementato dalle classi derivate concrete, inoltre contiene un campo dati `ID` che rappresenta un'id univoco per ogni prodotto.
Torta, **Dolce** e **Salato** sono le tre classi concrete derivate da `Prodotti`, rappresentano rispettivamente una torta, un prodotto dolce o un prodotto salato.
Contenitore è la classe che rappresenta una collezione di prodotti. È una lista di **nodi**, i quali contengono un puntatore ad un prodotto; `Contenitore` fornisce anche una classe interna **Iteratore** per poter scorrere la lista in modo semplificato.
`Contenitore` offre la possibilità di inserire un prodotto in testa alla lista, eliminare un prodotto fornendo l'id del prodotto da eliminare e la possibilità di ottenere i dati di un prodotto fornendo l'id.
Ordine è la classe che rappresenta il model del progetto, infatti dentro questa classe vengono salvati i dati che rappresentano l'ordine che si vuole effettuare. Ordini è formata da tre campi dati privati:

- **Contenitore**: classe collezione di prodotti;
- **QDataTime**: rappresenta la data e l'ora dell'orario dell'ordine;
- **Cliente**: classe che modella un cliente ed è formata da tre stringhe che rappresentano il nome, il cognome e il numero di telefono del cliente.

Sono inoltre state create due classi, **Peso** e **Pezzi** che rappresentano le quantità in cui vengono inseriti i prodotti negli ordini.

3.2 Controller

ControllerOrdine è la classe che rappresenta il controller del progetto e si occupa di ricevere i dati immessi dall'utente nella view e salvarli nel model e riproporli dal model alla view nel caso debbano essere modificati. ControllerOrdine si occupa inoltre di leggere e scrivere da/su file qual'ora si voglia caricare o salvare un ordine. Questo avviene tramite l'utilizzo delle classi `QXMLStreamReader` e `QXMLStreamWriter` che permettono di produrre e leggere file XML (Nell'apertura di un file si assume che il file aperto sia un file generato con l'applicazione `In Ordine`). Questa classe inoltre ha tre campi dati, `torta`, `dolce` e `salato` che servono per creare un prodotto ricevendo i dati dalla view o dalla lettura di un file per poi inserirlo nel model.

3.3 View

Le due principali classi grafiche di `In Ordine` sono:

- **MainWindow**: classe che rappresenta la schermata iniziale del progetto nella quale è possibile decidere se inserire un nuovo ordine o caricarne uno da file;
- **ViewOrdine**: classe che rappresenta la view del progetto e permette di inserire, modificare o eliminare i dati riguardanti il cliente, la data dell'ordine e i dieci prodotti che si vuole ordinare; questa classe fornisce anche il preventivo dell'ordine aggiornato in base ai prodotti aggiunti e la possibilità di salvare quello su cui si sta lavorando.

Sono state inoltre create cinque classi di supporto alla view:

1. **MyWidget**:classe derivata da `QWidget` per rappresentare un prodotto come widget in una lista all'interno della view;
2. **MyCheckBox**: classe derivata da `QCheckBox` per gestire il passaggio dei parametri del checkbox;
3. **MyComboBox**: classe derivata da `QComboBox` per gestire il passaggio dei parametri della combobox;
4. **MyDateTimeEdit**: classe derivata da `QLineEdit` per gestire il passaggio dei parametri della line edit per la data e l'ora;
5. **MyLineEdit**: classe derivata da `QLineEdit` per gestire il passaggio dei parametri delle line edit.

4 Gestione della memoria

Per quanto riguarda la gestione della memoria è stato ridefinito il distruttore di `Contenitore` in modo che elimini il primo `nodo` che a sua volta chiamerà il distruttore di prodotto e quello del nodo seguente, in modo da eliminare automaticamente tutta la collezione; anche il metodo `togli(int)` si occupa di eliminare il nodo da togliere chiamando la `delete` su di esso.// Per quanto riguarda la classe `Prodotti` il distruttore è stato dichiarato virtuale in modo che venga richiamato in automatico quello della classe derivata. Per la parte grafica si è lasciato la gestione delle distruzioni degli oggetti alle classi Qt, impostando tutti gli oggetti creati come figlio della view principale, in questo modo alla chiusura della view vengono automaticamente distrutti anche i figli. è stato inoltre ridefinito il metodo `closeEvent(QQevent*)` in modo che una volta chiusa la view principale si richiami il distruttore del controller che a sua volta richiama automaticamente i distruttori del model e della view.