QDrink-Freedom

QDrink

andrei.petrov@studenti.unipd.it

Relazione Progetto di Programmazione ad Oggetti V1.0.0

Nome del documento	Relazione Progetto di Programmazione ad Oggetti
Versione del Documento	1.0.0
Data Creazione	01/07/2016
Redazione	Petrov Andrei 1003859
Destinatari	*Prof. Francesco Ranzato*

Indice

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Lo scopo del seguente documento è raccogliere e descrivere le attività progettuali e di codifica del prodotto realizzato.

1.2 Scopo del progetto

Come tema del progetto ho scelto di realizzare un piccolo applicativo legato al mondo della "*miscelazione*" e "*beverage*". Da un lato si vuole realizzare un calcolatore automatico della gradazione alcolica del drink in funzione agli ingredienti caratterizzanti e dall'altro lato si vuole realizzare un porta note sulla composizione degli ingredienti interessati alla creazione, descrizione ed analisi di ricette cocktail.

1.3 Glossario

- **Ingrediente**:
  ciascuno delle sostanze che entrano a far parte di un prodotto o risultano necessarie all'esecuzione di una ricetta;
- **Base**:
  ingrediente alcolico che costituisce la base di un miscelato;
- **Aromatizzante**
  ingrediente alcolico caraterizzante l'aroma del miscelato, per esempio il vino bianco aromatizzato Martini Bianco;
- **Colorante**:
  ingrediente alcolico oppure non alcolico con la funzione di offrire colore al miscelato;
- **Drink**:
  bevanda alcolica anonima costituita da una miscela di una oppure più sostanze alcoliche e non;
- **Cocktail**:
  drink a cui viene dato un nome;
- **Ricettario**:
  collezione di Cocktail;
- **Molecular Mixology**:
  tecnica di miscelazione in cui gli ingredienti si presentano in combinazione liquida e molecolare;

1.4 Specifiche progettuali

1.4.1 Vincoli di progetto

Il progetto deve soddisfare i seguenti vincoli:

- Definizione ed utilizzo di una gerarchia G di tipi;
- Definizione ed utilizzo di un opportuno contenitore C, con relativi iteratori;
- Il front-end dell'applicazione deve essere una GUI sviluppata via il framework Qt.

1.4.2 Descrizione

L'applicazione gestisce ricette personali. Una ricetta cocktail permette il salvataggio di una collezione di ingredienti. Questi ultimi sono di tre tipologie: base, aromatizzante e colorante.

1.4.3 Elenco funzionalità

L'applicazione deve offrire le seguenti funzionalità:

- Operazioni CRUD (Create, Read, Update, Delete) sugli elementi del ricettario QDrink;
- Backup:
  - Creazione backup dei dati su file XML;
  - Caricamento backup da file XML;
- Visualizzazione lista ricette e per ogni ricetta la relativa percentuale alcolica;
- Svuota ricettario.

1.4.4 Dettagli ambiente di sviluppo

Per realizzare il progetto vengono utilizzate le seguenti tecnologie:

- Sistema Operativo Fedora versione 22;
- IDE Qt Creator versione 4.0.2;
- Qt Designer, tool integrato con Qt Creator;
- Librerie Qt versione 4.8.0 (64 bit).

Il progetto è stato compilato ed eseguito sulle macchine del Laboratorio Informatico Plesso Paolotti dove compila ed esegue correttamente.

Il caricamento dei dati utili all'applicazione vengono caricati in maniera automatica. Tuttavia è possibile caricare un insieme di dati diversi a run-time.

Il progetto viene separato in due parti: front-end e back-end. Non viene fatto uso dello stile di programmazione Model-View Programming consigliato da Qt. Ho scelto di utilizzare i QWidget Qt perchè ogni tipologia di QWidget offre un modello di dati interno.

1.4.5 Indicazioni per la compilazione

Il progetto viene rilasciato con un file QDrink.pro pre-definito. Per compilare il progetto e successivamente eseguire si devono eseguire i seguenti comandi:

- qmake;
- make;
- make clean;
- ./QDrink.

1.4.5.1 Descrizione comandi di compilazione

Il primo comando genera il Makefile necessario per attuare la compilazione. Tale file include un insieme di direttive Qt. Generato il file necessario, l'esecuzione di make fa partire la compilazione vera e propria sulla macchina. L'istruzione genera un insieme di file temporanei. Per pulire i file non più necessari viene in aiuto il terzo comando. Infine, per eseguire l'applicativo utilizzare il quarto comando.

2 Dettagli progettuali ed architetturali

Lo sviluppo dell'applicazione viene suddiviso in due parti: la parte relativa al dominio applicativo, back-end, e la parte relativa alle componenti grafiche, front-end.

2.1 Back-end

La parte di back-end viene realizzata utilizzando il linguaggio di programmazione C++ puro.

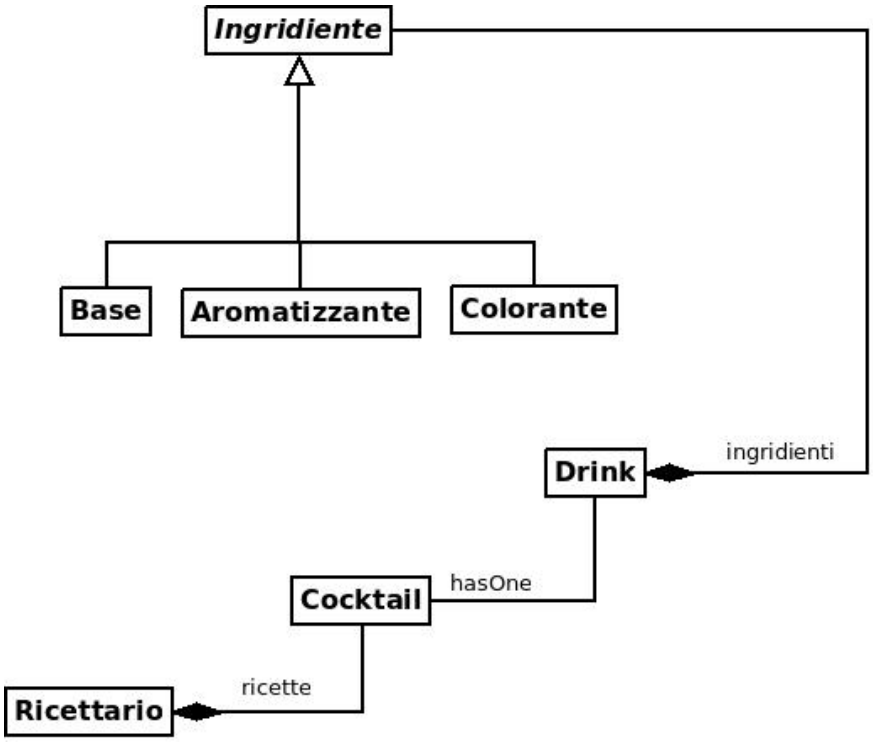

Fig 1: Back-end applicazione QDrink

2.1.1 Descrizione classi

Durante la fase di analisi vengono individuati i seguenti oggetti:

- *Ingrediente*$_g$: unità più piccola costituente un drink;
- *Base*$_g$, *Aromatizzante*$_g$, *Colorante*$_g$: tipologie specifiche di ingrediente;
- *Drink*$_g$: collezione di uno o più ingredienti;
- *Cocktail*$_g$: miscelato completo;
- *Ricettario*$_g$: collezione di zero o più cocktail.

Ogni oggetto è caraterizzato da una precisa responsabilità. La gerarchia interessa la semantica della componente Ingrediente, perchè nella miscelazione l'essenza di un cocktail sono gli ingredienti. La mancanza di una standardizzazione uniforme e severa rende difficile la modellazione e la trattazione delle ricette. Per questo motivo ho scelto un compromesso.

Le sottoclassi della classe Ingrediente offrono una scomposizione logica. Questa scelta è necessaria per alcune tipologie di cocktail come ad esempio nelle ricette della *Molecular Mixology*$_g$ dove non esiste una netta separazione tra ingredienti di tipo base, aromatizzante e colorante.

Di seguito vengono descritte le componenti più significative del back-end. Per agevolare la descrizione delle componenti introduco la seguente convenzione: descrizione, proprietà, metodi.

2.1.1.1 Ingrediente

- Descrizione: classe base astratta e polimorfa, rappresenta proprietà comuni per tutte le tipologie di ingredienti;
- Proprietà:
  - Quantità: *int*;
  - Percentuale Alcolica: *int*;
  - Nome: *string*;
- Metodi:
  - Getters: per ogni campo dati vengono introdotti metodi per recuperare le informazioni per i relativi campi dati;
  - Setters: per ogni campo dati vengono introdotti metodi per aggiornare il contenuto dei campi dati;
  - Statici: nella sezione privata della classe viene introdotto un metodo ausiliario per il calcolo della gradazione alcolica dell'ingrediente in funzione alle informazioni dei propri campi dati;
  - Vituali puri:
    - copia: metodo costante che restituisce un puntatore a ingrediente con la facoltà di creare una copia dell'oggetto di invocazione;
    - getTipo: metodo costante che restituisce una stringa rappresentante il tipo dinamico dell'oggetto d'invocazione.
  - Virtuali: il distruttore viene marcato virtuale per convenzione di classe virtuale.

2.1.1.2 Sottoclassi della classe Ingrediente: Base, Aromatizzante e Colorante

Le derivazioni pubbliche della classe Ingrediente rappresentano implementazioni della classe base. Ogni sottoclasse condivide la stessa struttura logica.

- Descrizione: le sottoclassi rappresentano tipologie diverse della classe generica Ingrediente;
- Proprietà: vengono ereditate dalla classe base;
- Metodi:

  - I metodi virtuali puri vengono implementati rispettivamente;
  - Il rispettivo costruttore a tre parametri che inizializzano i campi dati ereditati;
  - Virtuali: il distruttore è marcato virtuale con il corpo vuoto per convenzione di classe polimorfa.

2.1.1.3 Drink

- Descrizione: la classe rappresenta il miscelato di un cocktail. Ogni ingrediente ha il tempo di vita che viene gestito dalla questa classe;
- Proprietà:
  - Ingridienti: *vector<Ingridiente*>*, la colezione degli ingredienti;
- Metodi:
  - Costruttore di copia: metodo pubblico per attuare la copia profonda di un'istanza della classe Drink;
  - Volume: *int*, il metodo restituisce un intero rappresentante la capacità del drink;
  - GradazioneAlcolica: *double*, il metodo restituisce il tasso alcolico del drink in funzione ai singoli ingredienti;
  - Add: *void*, il metodo permette di aggiungere un ingrediente al drink;
  - Remove: *bool*, il metodo permette di rimuovere un ingrediente per nome;
  - SetDrink: *void*, il metodo offre la capacità di settare il drink di invocazione con un altro drink;
  - GetChild: *Ingridiente**, il metodo permette di restituire l'ingridiente i-esimo nel caso in cui fosse possibile, altrimenti restituisce il puntatore nullo;
  - HowManyElements: *unsigned int*, il metodo restituisce il numero di ingredienti;
  - Distruttore: il metodo distrugge in modo profondo il drink, mediante la distruzione di ogni ingrediente in parte;

2.1.1.4 Cocktail

- Descrizione: la classe ha responsabilità di gestire un miscelato completo di nome, momento di consumo e drink;
- Proprietà:
  - NomeCocktail: *string*;
  - MomentoConsumo: *string*;
  - Drink: *Drink*;
- Metodi:

- Costruttore a tre parametri: il costruttore ha la facoltà di inizializzare eventualmente di default i campi dati;
- Costruttore di copia: ha la facoltà di copiare profondamente le informazioni dell'oggetto passato come riferimento costante campo dati per campo dati;
- Getters: metodi che ritornano informazione in relazione al campo dati desiderato;
- setters: metodi per la modifica a run time dei campi dati associati;
- AddIngrediente: *void*, metodo che aggiunge un ingrediente al cocktail;
- RemoveIngrediente: *bool*, metodo che rimuove un ingrediente se questo esiste dalla lista degli ingredienti;

2.1.1.5 Ricettario

- Descrizione: la classe ha la responsabilità di gestione del tempo di vita di ciascuna ricetta in parte;
- Proprietà:
  ricettario: CList<Cocktail>;
- Metodi:
  - Costruttore di default;
  - Costruttore di copia;
  - Size: *unsigned int*, metodo costante che ritorna il numero di ricette presenti nel ricettario;
  - Insert: *void*, metodo di inserimento di una ricetta;
  - Remove: *bool*, metodo di rimozione di una ricetta dal ricettario, la ricerca della ricetta viene attuata per nome;
  - Find: *Cocktail**, il metodo ricerca una ricetta per nome;
  - GetRicettario: *vector<Cocktail>*, il metodo ritorna una copia della ricetta salvata in un vector di ricette;
  - Load: *bool*, il metodo carica un file XML contenente la lista delle ricette in memoria, il booleano segnala la correttezza di esecuzione;
  - Save: *bool*, il metodo salva su file XML le ricette presenti in memoria;
  - Distruttore: il ricettario viene svuotato.

2.1.1.6 CList: contenitore realizzato

- Descrizione: la classe rappresenta una lista doppiamente lincata e templatizzata. Questa permette l'inserimento e la rimozione in testa e coda con una complessita costante O(1); la ricerca è proporzionale ad una complessità O(n/2) nel caso medio mentre nel caso peggiore è O(n) infine il caso migliore è banalmente costante perchè l'elemento desiderato è in proprio l'elemento in testa oppure coda;

- Proprietà:
  - Classe **nodo**: localizzata nella parte privata è caraterizzata da un campo dati parametrico T ed un puntatore al nodo successivo. Poichè la visibilità della classe è privata i metodi sono tutti pubblici;
  - Classe **iteratore**: classe *friend* e localizzata nella parte pubblica. Vengono offerti i rispettivi costruttori (default e di copia), operatori di: assegnazione, uguaglianza, diverso, pre incremento e post incremento, operator(), infine il metodo di istanza costante isNotNull();
- Metodi:
  - Costruttore di default: inizializza una lista vuota;
  - Costruttore di copia: copia profonda, il metodo utilizza un metodo statico di ausiliario per fare la copia profonda;
  - Operatore di assegnazione: non viene alterata la semantica dell'operatore, esegue un assegnamento profondo;
  - RimuoviElemento: il metodo rimuove l'elemento puntato dall'iteratore passato come parametro riferimento costante;
  - Is_empty: il metodo ritorna un booleano true se la lista è vuota, altrimenti ritorna false;
  - Size: il metodo ritorna un unsigned int indicante il numero di nodi nella lista;
  - Operator[]: il metodo di accesso a membro ritorna l'oggetto puntato dall'iteratore;
  - Push_back: il metodo inserisce un oggetto parametrico in coda alla lista;
  - Push:front: il metodo inserisce un oggetto parametrico in testa alla lista;
  - Pop_front: il metodo rimuovo un oggetto dalla testa della lista.

2.2 Front-end

L'utente interagisce con componenti grafiche che permettono di:

1. Inserire, Cercare, Rimuovere e Modificare ricette Cocktail;
2. Visualizzare per ogni ricetta la gradazione alcolica complessiva;
3. Creare backup delle proprie ricette;
4. Caricare backup precedentemente salvati su disco;
5. Svuotare il ricettario su cui l'utente sta lavorando, permettendoli di iniziare un nuovo ricettario.

Per ogni ricetta l'utente visualizza i seguenti dati:

- Nome Cocktail;
- Momento di Consumo;

- Lista Ingredienti.

Nella finestra relativa alla visualizzazione della ricetta la lista degli ingredienti è una **QTableWidget** la quale permette una visualizzazione tabellare dei dati. Si possono effettuare modifiche sugli elementi della tabella mediante l'evento di *double click* sull'elemento interessato. Per attuare le modifiche è necessario clickare sul bottone salva, altrimenti le modifiche non hanno effetto. Inoltre è possibile aggiungere oppure rimuovere degli ingredienti.

N.B.
Durante l'aggiunta di un nuovo ingrediente è necessario che il tipo del ingrediente sia esplicitamente specificato e il cui valore sia uno dei seguenti: "Base", "Aromatizzante", "Colorante"

Nel caso della visualizzazione delle ricette viene visualizzato una finestra contenente anche in questo caso una **QTableWidget** che visualizza una tabella con due colonne. La prima colonna interessa il nome della ricetta, mentre la seconda colonna interessa una informazione ricavata che delinea la gradazione alcolica complessiva della ricetta cocktail. Qui, inoltre, è possibile che l'utente rimuovi una ricetta oppure ne visualizzi i dettagli della ricetta selezionata.

Per quanto riguarda la funzionalità di ricerca, l'utente deve inserire il nome di una ricetta. Se la ricetta è presente viene visualizzato una finestra con i dettagli della ricetta, altrimenti viene visualizzato un messaggio comunicante la non presenza di una ricetta con tal nome inserito. Nel caso in cui l'utente non inserisca alcun nome l'applicazione si accorge di ciò e comunica all'utente la necessità di qualche stringa testuale.

Le funzionalità di backup sono molto semplici. In un caso viene creato un backup ed all'utente viene chiesto il luogo di salvataggio sul file system. Invece,nel caso di caricamento del backup è richiesto che il file da cui viene effettuato il caricamento sia un file XML; altrimenti viene lanciato un messaggio testuale indicante il contesto.

Infine, la comunicazione tra gli oggetti appartenenti al front-end avviene mediante signals e slots.

Per l'implementazione del front-end viene utilizzato Qt Designer per assistere nella progettazione della GUI. Durante la progettazione della GUI si è osservata la qualità del codice generato automaticamente. Infatti, non è stato necessario intervenire per eliminare informazione accidentale generata dal *uic*, user interface compiler di Qt. Tuttavia è stato necessario intervenire a sistemare dei problemi legati a signal e slot. Si è riscontrato il problema della **CONNECT** automatica da parte del compilatore moc. Questo problema è stato riscontrato a causa di un bacco di Qt Creator versione minore della versione 5.1.

3 Gestione della memoria

Nella fase di analisi ho trovato poco vantaggioso introdure la condivisione di memoria tramite *smart pointers* perchè l'applicazione non ha la necessità di condivisione di memoria. Per questo motivo ho scelto di optare per una gestione di memoria profonda. Gli oggetti grafici senza un padre, secondo la convenzione Qt, vengono allocati nello stack. Mentre ogni componente che ha un padre viene allocato sullo heap, affidando la gestione della memoria al ambiente di gestione della memoria automatica di Qt.