Progetto di Programmazione a Oggetti

Agatea Riccardo

a.a. 2018/2019

1 Introduzione

L'applicazione consiste in un sistema di gestione delle ordinazioni di un ristorante. L'applicazione permette l'immissione di nuove ordinazioni, la loro modifica e rimozione, il loro salvataggio su file e caricamento da file in formato XML, e la loro ricerca secondo diversi parametri. Inoltre, gestisce la separazione fra ordinazioni in attesa e completate.

2 Descrizione aspetti progettuali

2.1 Gerarchia di tipi

La gerarchia modella le ordinazioni inviate alla cucina del ristorante. Radicata nella classe polimorfa astratta `Order`, si dirama in due direzioni, `Food` e `Drink`, associate rispettivamente a ordinazioni di piatti e bevande. Il polimorfismo è dato da alcuni metodi virtuali, descritti in seguito, che forniscono funzionalità di copia, move e confronto polimorfi, e permettono di ricavare informazioni esplicite sul tipo degli oggetti. La gerarchia è fortemente estensibile sia "in orizzontale", aggiungendo nuovi sottotipi alle classi base presenti, sia "in verticale", fornendo sottotipi alle classi più derivate della gerarchia.

Grafico della gerarchia

`Order` La classe `Order` incapsula le caratteristiche comuni a tutte le ordinazioni, quali numero del tavolo e nome della pietanza ordinata. I campi dati, tutti privati, sono:

- `unsigned int table`: rappresenta il numero del avolo da cui proviene l'ordinazione
- `std::string item`: rappresenta il nome della pietanza ordinata. Non sono effettuati controlli di coerenza con un eventuale listino o menu: si assume che tali controlli siano effettuati dall'utente della gerarchia.
- `static char separator`: alcuni metodi espongono all'utente stringhe formate dalla concatenazione di dettagli diversi dell'oggetto di invocazione; per mantenere la separazione delle varie caratteristiche viene utilizzato questo carattere.
- `static std::string empty`: gli stessi metodi possono indicare un valore "vuoto" per alcune caratteristiche attraverso questa stringa.

I metodi sono:

- `protected`:
  - `std::string getType() const`: metodo virtuale puro che per un'invocazione `p->getType()` ritorna il tipo di `*p` sottoforma di stringa. È pressochè equivalente a `typeid(*p).name()`, garantendo però che la stringa ritornata non sia implementation defined.
- `public`:
  - `Order(unsigned int, const std::string &)`: costruttore. Il primo parametro rappresenta il numero del tavolo, il secondo la pietanza ordinata.
  - `Order(const Order &)`: costruttore di copia (standard).
  - `Order(Order &&)`: costruttore di move (standard).
  - `~Order()`: distruttore (standard). Ridefinito perchè sia virtuale.
  - `Order& operator=(const Order &)`: operatore di assegnazione di copia (standard).
  - `Order& operator=(Order &&)`: operatore di assegnazione di move (standard).
  - `Order *clone() const`: metodo virtuale puro che implementa la "costruzione di copia polimorfa".
  - `Order *move() = 0`: metodo virtuale puro analogo al precedente che implementa la "costruzione di move polimorfa".

- `std::string recap() const`: metodo che per un'invocazione `p->recap()` espone all'utente lo stato dell'oggetto di invocazione. La stringa ritornata riporta, nell'ordine, il tipo di `*p`, i dettagli (o eventualmente `empty`), il nome della pietanza ordinata, e il numero del tavolo, separati da `separator`. Non è virtuale, ma esegue chiamate a metodi virtuali.
- `std::string getDetails() const`: metodo virtuale puro che per un'invocazione `p->getDetails()` ritorna i dettagli dell'ordinazione `*p`, qualunque essi siano, in una stringa formattata in modo analogo a quella ritornata dal metodo recap().
- `void setDetails(const std::string &)`: metodo virtuale puro che per un'invocazione `p->setDetails()` modifica i dettagli dell'ordinazione `*p` in base al parametro passato al metodo.
- `bool operator==(const Order &) const`: operatore di uguaglianza, virtuale, `o1==o2` ritorna `true` se e solo se i due oggetti sono dello stesso tipo e i campi dati corrispondenti dei due oggetti sono uguali. Fornisce un'implementazione che esegue il confronto fra i tipi (dinamici) dei due oggetti confrontati, e fra i campi dati `table` e `item`.
- `bool operator!=(const Order &) const`: operatore di disuguaglianza, non virtuale ma chiama metodi virtuali, `o1!=o2` ritorna `true` se e solo se `!(o1==o2)` ritorna `true`.

Sono inoltre fornite specializzazioni al tipo `Order` per i due template di funzione `template<typename T> T *clone(const T &)` e `template<typename T> T *clone(T &&)` definiti del namespace `PolyConstruct`. Per una reference `const Order &lref` e una reference a r-value `Order &&rref`, le chiamate `clone(lref)` e `clone(rref)` coincidono rispettivamente con `lref.clone()` e `rref.move()`.

`Food` La classe `Food`, astratta, rappresenta le ordinazioni di cibo (in opposizione alle bevande). L'unico campo dati è:

- `std::string without`: rappresenta eventuali rimozioni richieste dal cliente rispetto alla ricetta usuale

Gli unici metodi aggiuntivi sono il costruttore e i cinque metodi standard forniti dal compilatore. Sono forniti overriding per:

- `std::string getDetails() const`: ritorna il contenuto di `without`.
- `void setDetails(const std::string &)`: assegna a without il valore del parametro.
- `bool operator==(const Order &) const`: dopo aver invocato esplicitamente `Order::operator==(const Order &)`, confronta i campi dati `without` dei due oggetti confrontati.

2.2 Chiamate polimorfe

2.3 Formato dei file di salvataggio e caricamento

3 Note tecniche

3.1 Istruzioni di compilazione

Per la compilazione è fornito il file `progetto.pro`.

3.2 Ambiente di sviluppo

- Sistema operativo di sviluppo: Windows 10 Home 64-bit
- Compilatore: g++ (i686-posix-dwarf-rev0, Built by MinGW-W64 project) 5.3.0
- Qt framework: Qt 5.12.0
- IDE di sviluppo: Qt Creator 4.8.1

3.3 Ripartizione ore

- Analisi preliminare del problema: 1
- Progettazione modello: 2
  - Progettazione template di classe `Container`: 0.5
  - Progettazione gerarchia di classi: 1.5
- Progettazione GUI
- Apprendimento libreria Qt
- Codifica modello: 17
  - Codifica template di classe `Container`: 8
  - Codifica gerarchia di classi: 9
- Codifica GUI
- Debugging
- Testing