RELAZIONE DEL PROGETTO DI PROGRAMMAZIONE AD OGGETTI
A.A. 2018/2019

Mattia Gottardello-1142520

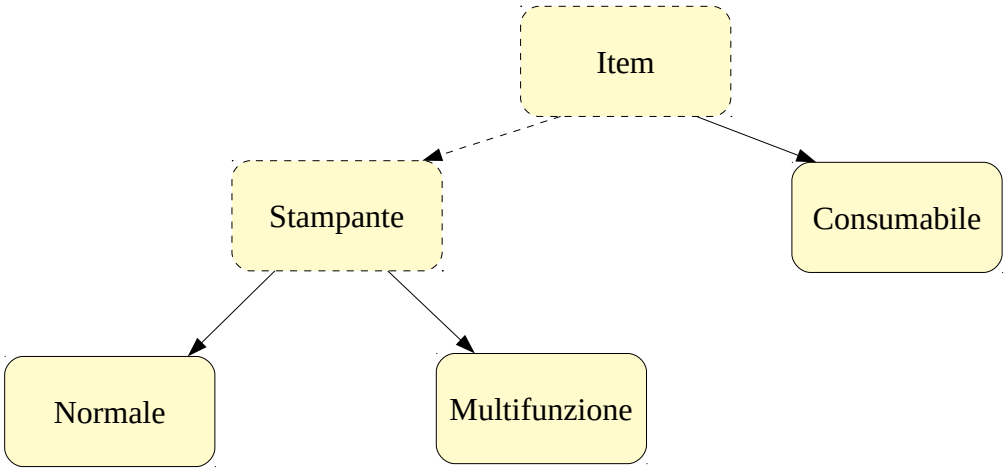
Agosto 2019

Progetto Qontainer

- **Abstract**

Lo scopo di questo progetto, è quello di realizzare un preventivatore per una ditta di noleggio e compravendita di stampanti, dando la possibilità al dipendente di preventivare al cliente una spesa totale e un parziale del materiale noleggiato (parziale perché si va semplicemente a sommare il prezzo al giorno dei vari articoli noleggiati ma non viene fatto un conteggio totale in base ai giorni di noleggio).
Il programma utilizza anche due carrelli i quali contengono appunto le quantità e gli oggetti (presi dal catalogo) che il cliente intende ottenere.
Grazie al preventivatore FastPreventive, il dipendente ha la possibilità di crearsi dei cataloghi personalizzati che può utilizzare in occasioni diverse in base alle esigenze e può in tempo reale salvare in formato PDF (quindi in formato stampabile) tutte le richieste del cliente in modo da poterne fornire anche una copia.
Il progetto è stato pensato altresì per una futura espansione della ditta in quanto la struttura permette che la stessa, in futuro, possa vendere altri articoli.

- **Descrizione Gerarchia**



Rappresentazione grafica della gerarchia

Come detto nel primo punto, la struttura della gerarchia può permettere l'inserimento di altri articoli grazie alla classe base astratta Item che tramite gli attributi Venditore, Modello e Costo generalizza l'inserimento degli elementi dando un alto valore all'espandibilità della gerarchia.
Essa ha due classi figlie:

- Consumabile (classe per gli articoli come cartucce, i suoi attributi sono dimensione (numero di pagine di stampa), colore, se la cartuccia è originale e se è stata rigenerata)
- Stampante (classe astratta che generalizza il concetto di stampante, i suoi attributi sono il costo al giorno in caso di noleggio, la possibilità di avere Wifi, fronte-retro, di stampare foto, di essere usata e di poter stampare a colori)

La classe astratta Stampante è stata ideata per suddividere i vari tipi di stampante in base alle caratteristiche delle stesse.
Essa ha due classi figlie:

- Normale (classe per le stampanti senza fronte-retro, senza la possibilità di stampare fotografie ma che può rappresentare anche plotter. Per mia decisione ho deciso di inserire i plotter nella categoria stampanti normali idealizzando il fatto che questo tipo di stampanti, in nessun modo, possano stampare fotografie e fronte-retro.)
- Multifunzione (classe per le stampanti che hanno tutte le caratteristiche, in più possono avere scanner e fax)

- **Descrizione del Container**

Il contenitore è stato pensato basandosi su un catalogo e per questo motivo la sua struttura è quella di una lista singolarmente linkata.
La lista contiene la classe interna privata nodo(con campi info e next) che definisce i singoli nodi della lista, sono stati definiti costruttore a due parametri (oggetto, next), costruttore di copia e distruttore.
Il container contiene un campo first che indica il primo nodo della lista e oltre a costruttore vuoto, costruttore ad un parametro, costruttore di copia e distruttore ha i seguenti metodi:
Metodi statici privati di utilità:

- copy: crea una copia di tutti i nodi in modo ricorsivo;
- destroy: distrugge il singolo nodo richiamando la delete;
- search: cerca da un nodo in poi in modo ricorsivo se esiste l'oggetto che viene fornito nei parametri formali dicendo se c'è o no;
- isEqual: utilizza i due nodi passati nei parametri formali per verificare in modo ricorsivo che le due liste/sottoliste siano uguali.

Metodi non statici e pubblici:

- pushInOrder: crea all'interno della lista in nuovo nodo inserendolo in ordine assoluto
- removeOneItem: rimuove esattamente l'elemento nella lista che corrisponde a quello passato nei parametri formali, senza perdere la lista successiva al quel nodo;
- removeOneAtIndex: rimuove esattamente l'elemento all'indice specificato, senza perdere la parte restante della lista;
- searchIntoList: cerca nella lista l'elemento specificato dicendo se c'è o no;
- searchAtIndex: ritorna il campo info dell'elemento all'indice specificato;
- getIndex: ritorna l'indice dell'elemento specificato;
- replaceAtIndex: rimpiazza il vecchio nodo rimuovendolo all'indice indicato e inserendo quello nuovo in ordine;
- getFirst: ritorna il primo nodo della lista;
- is_empty: dice se la lista è vuota o no;
- size: ritorna la dimensione della lista;
- itemsCounter: ritorna il numero esatto di oggetti uguali a quello indicato contenuti nella lista;
- begin: ritorna l'iteratore al primo elemento della lista;
- end: ritorna l'iteratore al paste the end della lista;
- constBegin: ritorna l'iteratore costante al primo elemento della lista;
- constEnd: ritorna l'iteratore costante al past the end della lista;

É stato fatto l'overload dei seguenti operatori:

- operator=
- operator==
- operator!=

Sono stati implementati inoltre entrambi gli iteratori (con puntatore a nodo e campo pte che indica il past the end), sia quello normale che quello const.
Relativamente agli iteratori sono stati implementati costruttore a due parametri privato (puntatore a nodo e pte) e costruttore di copia pubblico poi:

- operator ++ prefisso;
- operator*;
- operator→;
- operator==;
- operator!=;
- hasNext: indica se il nodo successivo a quello puntato esiste;

- **Descrizione del Deep Pointer**

Il Deep Pointer (DeepPtr) è stato realizzato pensando ad una variante dei semplici puntatori, sono stati implementati i seguenti metodi:

- costruttore ad un parametro opzionale di default;
- costruttore di copia;
- distruttore;

Operatori:

- operator=;
- operator==;
- operator!=;
- operator<;

- operator>;
- operator*;
- operator→;

- **Descrizione GUI**

mainwindow.h

La GUI contiene, nella parte superiore, due menù:
- uno per per la gestione dei file(caricamento e salvataggio dei file del catalogo), per il salvataggio del preventivo in formato PDF e uscita del programma;
- uno per l'aggiunta, modifica e rimozione di elementi all'interno del catalogo.

É prevista la possibilità di inserire il nome della ditta o il nome del cliente, la partita iva e la data del giorno in cui viene effettuato il preventivo (dettagli utilizzati solo nel salvataggio in PDF).
La lista di sinistra è quella del catalogo, le due liste dei carrelli di destra sono rispettivamente da sinistra a destra quelle di noleggio e compra. Sopra alle liste di destra abbiamo i dettagli dell'oggetto selezionato e il logo.
Sotto alle liste abbiamo i due pulsanti di rimozione dai rispettivi carrelli dell'oggetto selezionato negli stessi e sotto ai due pulsanti abbiamo il conteggio del totale comprato e del parziale noleggiato.
In basso a sinistra abbiamo la barra di ricerca dinamica che aggiorna la lista del catalogo man mano che si scrive e sopra di essa ci sono i pulsanti per comprare/ noleggiare l'oggetto selezionato nel catalogo con la quantità indicata sopra ai pulsanti.

insertionwindow.h

Interfaccia per l'inserimento di elementi in un catalogo che parte con la possibilità di inserire il venditore (es. Canon,HP, Epson, etc…), modello e costo (sempre e solo in formato 00.0 o 00.00) dell'oggetto che si sta inserendo e poi si parte con le specifiche indicando se è una stampante o un consumabile, nel caso di consumabile vengono chiesti dimensione, colore e se è originale o rigenerata, nel caso di stampante viene chiesto se è normale o multifunzione e poi tutta la serie di specifiche come costo al giorno, se ha wifi, se ha il doubleside, se è a colori, se è usata, se si possono stampare foto e in base alla selezione normale o multifunzione compare la richiesta se è un plotter, se ha lo scanner e/o il fax.
Questa interfaccia avvisa anche il dipendente che in caso di inserimento di oggetti con le stesse caratteristiche, una copia di essi esiste già in catalogo.
Si è cercato di creare un interfaccia dinamica, che in caso di cambio di tipi ecc…, resetta i campi successivi a quella selezione precedentemente selezionati.
In caso di inserimento di dati sbagliati quindi un costo o un costo al giorno che non rispettano la forma 00.0 o 00.00 e una dimensione (relativa al numero di pagine) diversa da un intero, viene attivato un messaggio di errore relativo all'input errato.

modifywindow.h

Interfaccia per la modifica di elementi all'interno della lista, è una classe figlia dell'insertionWindow e prevede il caricamento dei dati dell'oggetto selezionato in modo da poter vedere cosa è stato selezionato precedentemente e quando si conferma la modifica, viene eliminato il vecchio oggetto e viene inserito in ordine quello modificato. La struttura è la stessa della classe madre insertionWindow.

smartlistview.h

Classe che estende la QlistWidget, creata per permettere l'implementazione di alcuni metodi di utilità utilizzati nella GUI.

- **Filtro di ricerca**

L'organizzazione degli oggetti nella GUI (parlando di elementi del catalogo) è stata affrontata utilizzando gli indici relativi e assoluti.
Nel catalogo gli oggetti possiedono una determinata posizione data dall'inserimento in ordine degli stessi, quando viene effettuata una ricerca alcuni elementi spariscono nella GUI per permettere il filtraggio, per non perdere gli elementi al di fuori del filtro è stata utilizzata una mappa che collega gli indici assoluti a quelli relativi filtrati.
Una volta resettato il filtro, la mappa torna a mostrare tutti gli elementi del catalogo.

- **Classe ausiliaria carrello**

La classe ausiliaria carrello è stata ideata prendendo spunto dai normali carrelli dei siti di e-commerce, gli oggetti che vi sono inseriti sono ricavati dal catalogo con l'ausilio dalla mappa contenente gli indici.
Il carrello contiene l'oggetto e la quantità comprata o noleggiata di quell'oggetto.

- **Codice polimorfo**

Avendo utilizzato in modo forte il DeepPtr, vengono trattati oggetti di tipo DeepPtr<Item>, il container utilizza oggetti di questo tipo quindi la maggior parte delle operazioni nello stesso effettuano azioni polimorfe, per esempio l'operator==, l'operator!=, la clone (metodo utilizzato per la copia profonda), il metodo getType (restituisce il tipo dell'oggetto), la print (che restituisce in formato di stringa tutti i dettagli dell'oggetto) e perfino la serializzazione viene fatta polimorficamente.
I DeepPtr<Item> vengono utilizzati anche nei carrelli, la loro struttura però non è quella del catalogo e quindi quella della lista, ma sfruttano le Qmap per permettere l'utilizzo dei DeePtr stessi come chiave e il valore associato è la quantità.
É polimorfa anche l'operazione di confirm all'interno dell'insertionWindow e nella modifyWindow, questa scelta è data dal fatto che la struttura è la stessa ma nella prima classe i dati vengono inseriti dall'utente, mentre nella seconda i dati vengono mostrati ed eventualmente modificati e questo particolare mi ha fatto decidere di creare una micro-gerarchia di window solo per l'inserimento e la modifica.

- **Formato del file**

Riguardo al formato del file è stato scelto il .xml, in quanto esso ha un struttura e una realizzazione molto semplice. Sono state usate le librerie dedicate all'xml e per avere un file con alcuni Item gia inseriti si consiglia di guardare la cartella "Cataloghi" nella directory del progetto, essa contiene un file che si chiama PrimoCatalogo.xml.

- **Creazione del file di preventivo**

É stata implementata la funzione che permette il salvataggio dei dati derivanti dai due carrelli in formato PDF, questo file non è utilizzabile all'interno del programma ma è di utilità al dipendente per salvarsi e fornire una copia delle richieste del cliente.
Un esempio di questo file lo si può trovare all'interno della cartella "Preventivi" nella directory del progetto.

- **Manuale utente**

Il programma all'apertura risulta privo di dati, questo permette all'utente di creare sin da subito un nuovo catalogo inserendo gli elementi tramite il pulsante di inserimento nel menù modifica. Nel caso l'utente abbia già creato un catalogo può caricarlo all'interno del programma tramite il pulsante di caricamento nel menù file.
Il nome utente e la partita iva (solo caratteri numerici, la lunghezza è stata lasciata appositamente variabile perché in caso di partita iva estera le cifre non sono più 12 ma 14 dovute all'aggiunta del prefisso) e la data possono essere modificate in qualsiasi momento.
Una volta caricato il catalogo basta selezionare l'elemento desiderato, decidere la quantità e decidere se noleggiarlo o comprarlo tramite gli appositi pulsanti(gli item consumabili non possono essere noleggiati, il noleggio di un item con quantità pari a 0 non comporta azioni), è possibile filtrare gli item tramite l'apposita barra di ricerca. Nel caso il cliente cambi idea relativamente a qualcosa di comprato o noleggiato, si possono rimuovere gli item tramite gli appositi pulsanti di rimozione. I totali si aggiornano automaticamente come i dettagli dell'oggetto selezionato.
Una volta creato il catalogo è possibile salvarlo grazie al pulsante salva contenuto nel menù file.
Una volta creato il preventivo e inserito il nome, e la p. Iva è possibile esportarlo in formato PDF tramite il pulsante export nel menù file.
É possibile uscire dal programma tramite il classico tasto x della finestra oppure con il pulsante esci contenuto nel menù file.

- **Analisi delle tempistiche**

Il monte ore è stato sforato di circa 4/5 ore, impiegate per individuare e progettare la soluzione dei carrelli e impiegate inoltre per capire il funzionamento del salvataggio PDF.

- 12/13 ore per progettazione, analisi del problema e creazione della gerarchia
- 15 ore per progettazione e creazione della GUI
- 20 ore per implementazione pattern MVC
- 7/8 ore per test e debug

Nelle ore di progettazione e implementazione è compreso l'apprendimento delle relative librerie per la buona realizzazione del progetto.

- **Istruzioni per la compilazione e l'esecuzione**

Spostarsi con il terminale fino alla directory del progetto contenente il file .pro, ed eseguire in successione i comandi qmake, make e ./FastPreventive.