Relazione del progetto di PMO

"SIMULAZIONE RISTORANTE"

matricola : Rossi Francesco Pio
Numero matricola : 298356

1. Specifica del software.

L'applicazione simula il comportamento di un cliente in un Ristorante.
Ogni giorno il ristorante prevede "Il menù del giorno" il quale è composto da 3 portate : un primo, un secondo e un dolce scelti casualmente dal menù.
All'inizio della simulazione il cliente può scegliere se ordinare il menù del giorno oppure ordinare le varie portate singolarmente.
Le 3 portate del "menù del giorno" hanno un piccolo sconto rispetto alle portate ordinate singolarmente.
I piatti ordinati dal cliente vengono serviti nell'ordine classico : primi, secondi, e infine dolci.
Ogni volta che il cliente termina di mangiare una pietanza, può chiamare il cameriere per ordinare qualcos'altro.
In cucina ci sono tre cuochi : uno per i primi, uno per i secondi e uno per i dolci. Il cuoco dei primi è il più lento, il cuoco dei secondi è il più veloce, e il cuoco dei dolci è il più gentile, infatti offre sempre una fetta di torta al cliente ogni volta che ordina un dolce.
Il cliente passa quindi attraverso 4 stati : Ordinazione -> attesa -> consumazione -> pagamento.
Durante l'attesa i cuochi cucinano. Una volta che una portata arriva al cliente, esso inizia a mangiare entrando nello stato "Consumazione", dopodichè ritorna nello stato attesa, nel quale può scegliere se chiamare il cameriere per ordinare altro o attendere semplicemente le altre portate ordinate precedentemente.
Il software termina quando tutti i piatti sono stati serviti, viene stampato il conto e il cliente paga.

2.Studio del problema

Primo punto critico : la gestione del Menù.

Viene implementata una classe Menù la quale ha un attributo interno "args" che contiene i parametri di ingresso della funzione Main(), cioè il nome dei 4 file di testo che contengono la lista dei primi, secondi, dolci e bevande con i rispettivi prezzi.
L'attributo "args" della classe Menù viene fissato tramite una funzione fissaArgs(string[] args) che permette di fissare args una volta per tutte senza la possibilità di modificarlo, in questo modo il Menù non può variare durante l'esecuzione.
La classe menù contiene 4 dizionari, i quali contengono la lista di primi, secondi, dolci e bevande : al nome di ogni pietanza viene associato il prezzo.
I dizionari vengono riempiti con i dati presenti nei file.

Secondo punto critico : l'ordine in cui i piatti vengono serviti.

Punto fondamentale dell'applicazione è che i piatti devono essere serviti in questo ordine : bevande, primi , secondi, dolci.
Questo requisito viene gestito fornendo ad ogni cuoco la lista dei piatti che deve cucinare : Il cuoco dei primi ha la lista con tutti i primi ordinati dal cliente, stessa cosa il cuoco dei secondi e il cuoco dei dolci.
Quando il cliente va nello stato attesa, uno dei cuochi inizia a cucinare : per prima cosa si controlla se la lista del cuoco dei primi contiene almeno un elemento, in quel caso il cuoco dei primi cucina il primo elemento della lista. Se invece ci sono zero elementi si passa a controllare la lista dei secondi, e infine quella dei dolci.
Quando un ordine viene cucinato, viene rimosso dalla lista e servito al cliente.
In questo modo viene gestita anche la terminazione del programma : quando tutte e tre le liste sono vuote, il cliente può passare a pagare il conto.

Terzo punto critico : come fanno i cuochi a cucinare?

I cuochi cucinano rimuovendo il primo elemento della lista dei piatti da cucinare dalla lista stessa. L'azione di cucinare viene simulata attraverso una sleep : Il cuoco dei primi è il più lento, impiegando 4 secondi (il metodo Sleep prende come argomento 4000, che sta per millisecondi), il cuoco dei secondi impiega 2 secondi e il cuoco dei dolci ne impiega 3.
Una volta cucinata la pietanza, viene servita al cliente dal cameriere.
Con lo stesso meccanismo della Sleep si simula il cliente che mangia la portata che gli è appena stata servita.

Quarto punto critico : il cliente chiama il cameriere per ordinare altro.

Una volta che il cliente termina di mangiare una portata, può scegliere se chiamare il cameriere per ordinare qualcos'altro o andare avanti con la simulazione.
Se va avanti con la simulazione, viene cucinata e servita la prossima portata che aveva ordinato, oppure, se tutte le portate sono state servite, viene chiesto il conto.

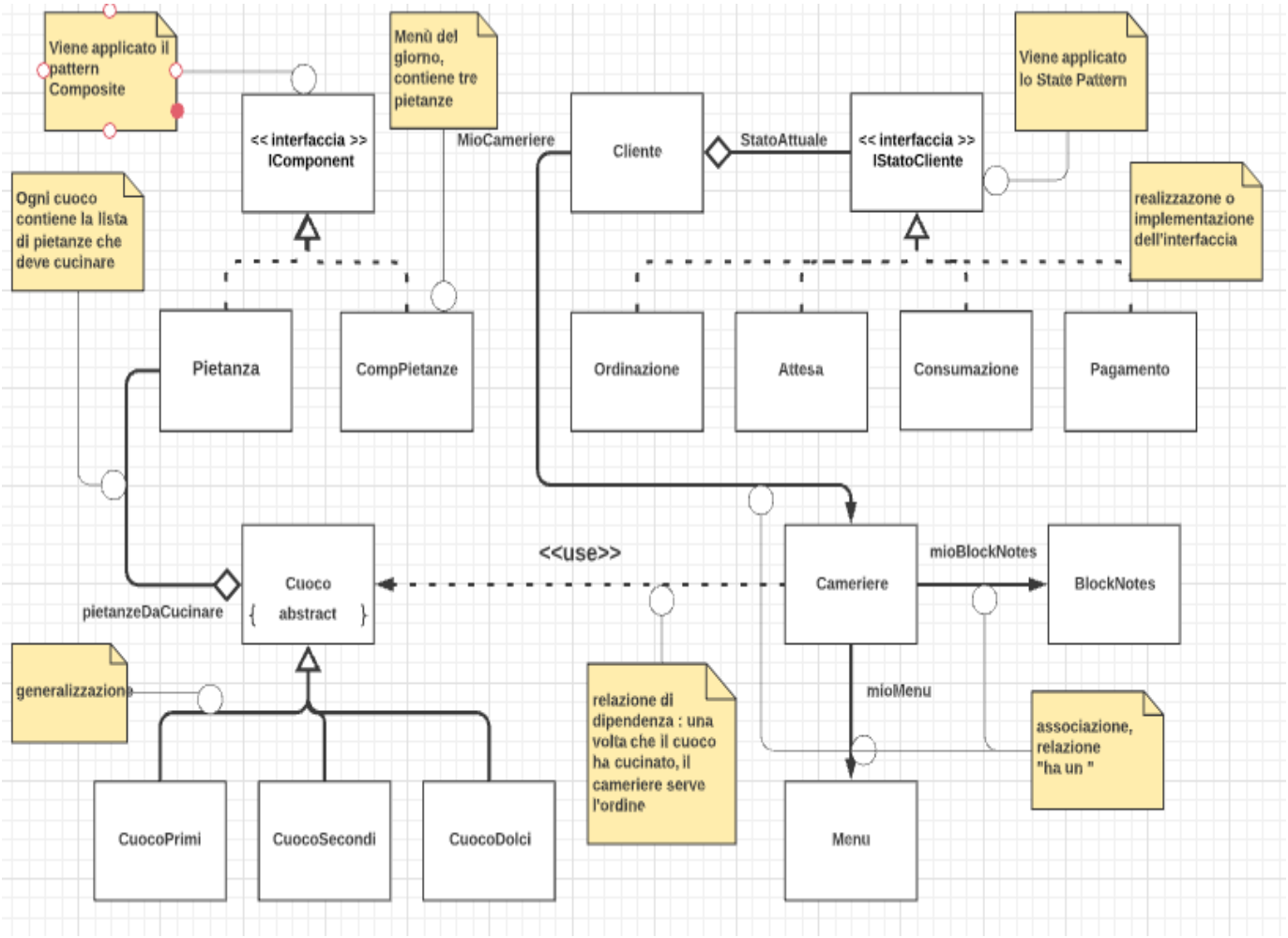
Quinto punto critico : calcolo del conto.

Il cameriere ha un blocknotes. La classe BlockNotes contiene una lista con tutti gli ordini effettuati dal cliente, e tramite un metodo interno è possibile calcolare il conto visitando ogni elemento della lista e sommando i vari prezzi.

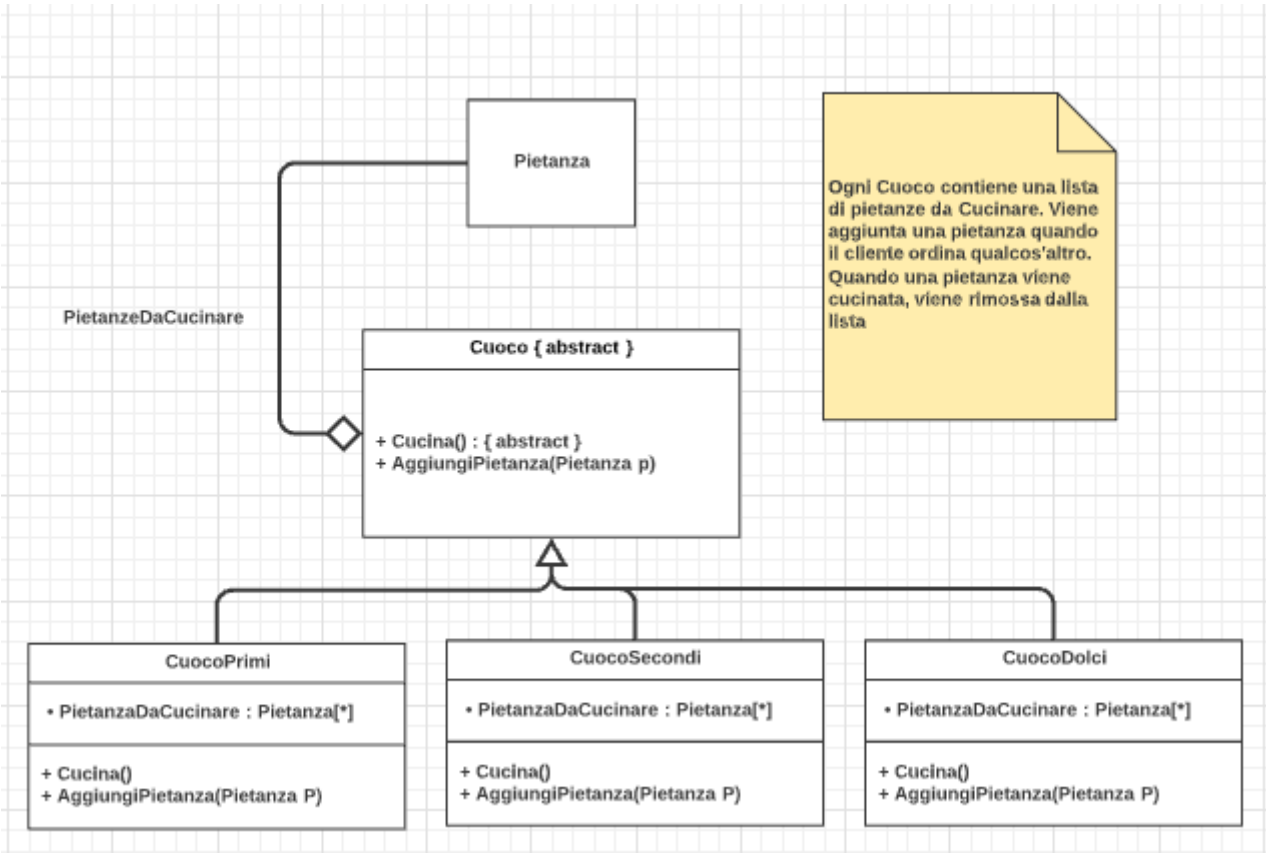
La lista può contenere sia il menù del giorno che pietanze singole, e tramite il Composite pattern tutti gli elementi vengono trattati allo stesso modo chiamando per ogni elemento il metodo OttieniPrezzo().

3.Scelte architetturali

L'architettura completa del software con tutte le classi e le loro dipendenze è rappresentata dal seguente diagramma delle classi :

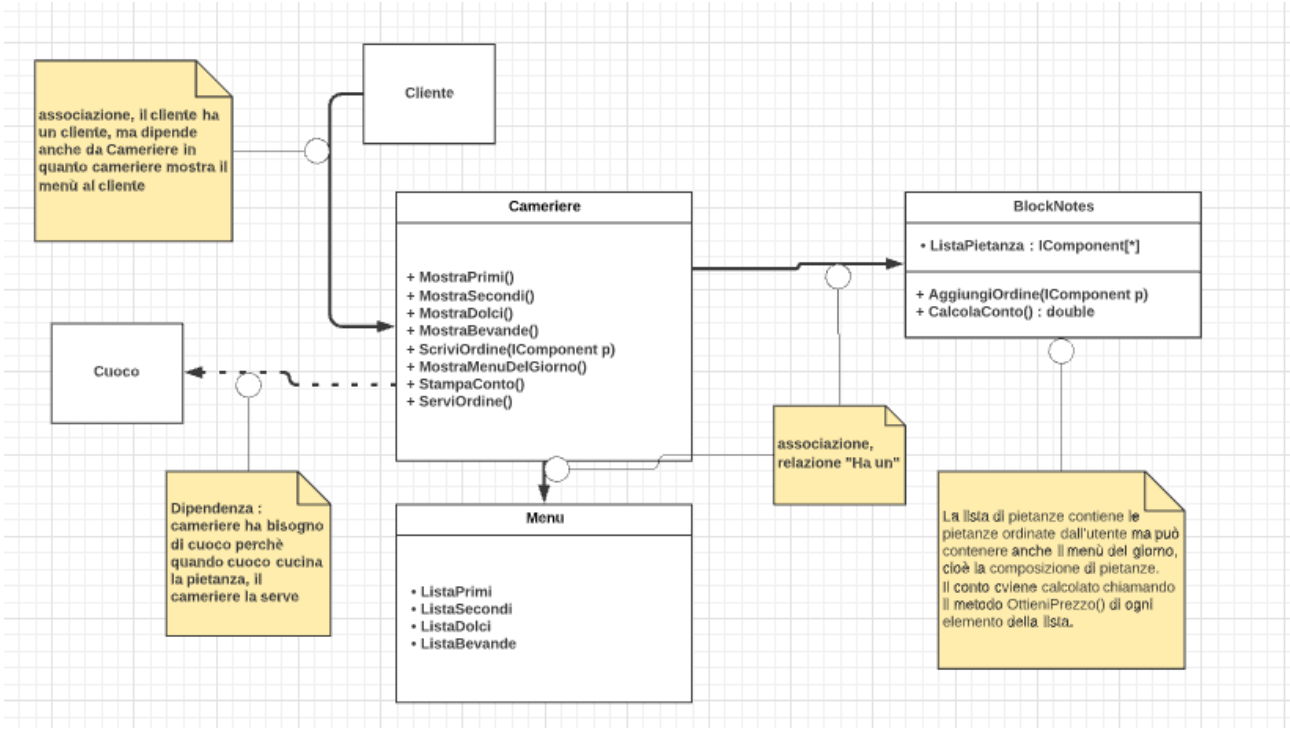


Andiamo a vedere nel dettaglio alcune relazioni del sistema.



Notiamo che *Cuoco* è una classe astratta, che non può essere istanziata.
La classe *CuocoPrimi*, *CuocoSecondi* e *CuocoDolci* ereditano i membri di *Cuoco*, quindi c'è una relazione di generalizzazione. Le 3 sotto classi fanno l'override del metodo *Cucina()*, metodo astratto della classe padre.
C'è una relazione di aggregazione fra la classe *Cuoco* e la classe *Pietanza* in quanto ogni cuoco possiede una lista di pietanze da Cucinare.
Quando il cliente ordina una nuova pietanza, viene aggiunta tramite il metodo AggiungiPietanza(), per poi essere rimossa quando quella pietanza viene cucinata.

Di seguito viene mostrata un'altra parte dell'architettura del sistema.



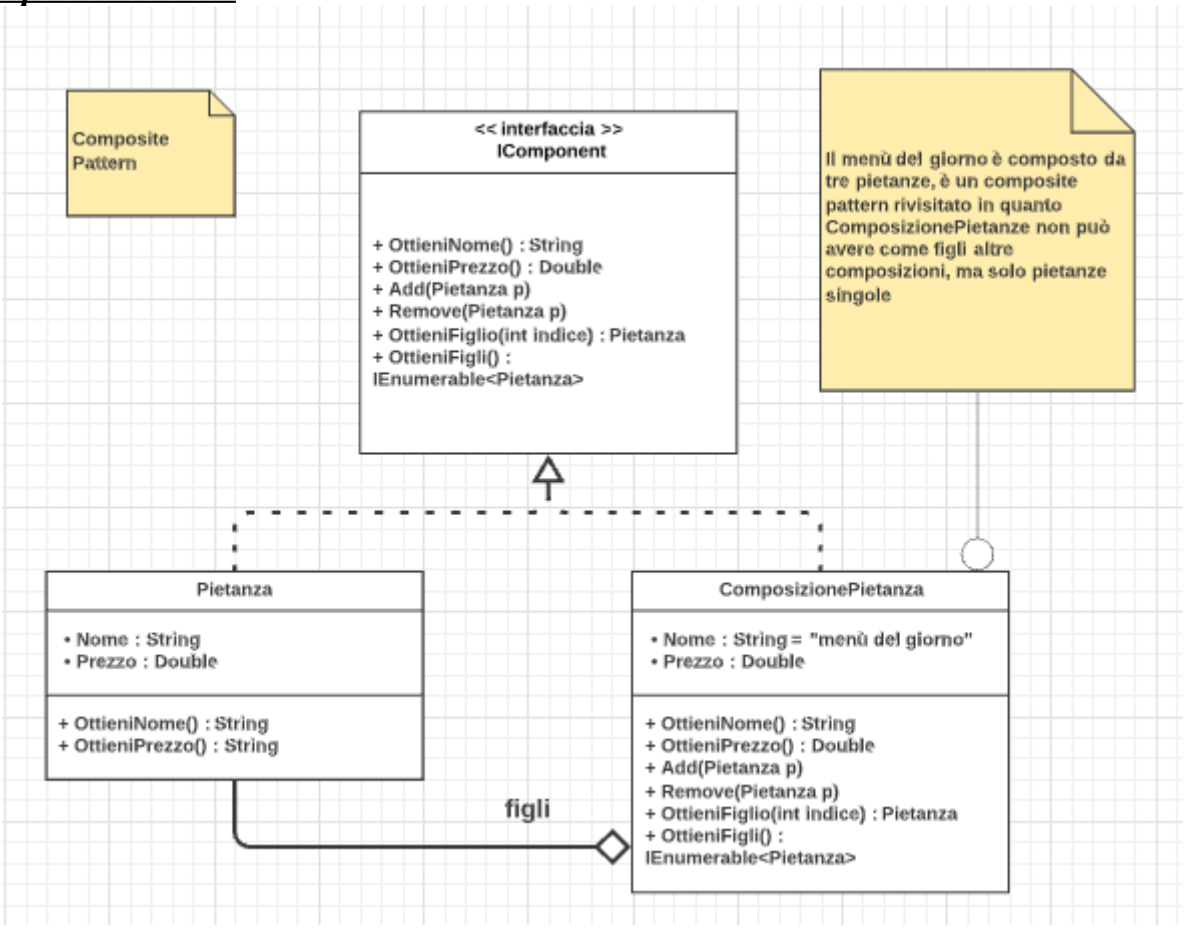
Il *Cliente* ha una relazione di associazione con il *Cameriere*, essendoci una relazione del tipo "*ha-un*".
La classe *Cameriere* ha una relazione di associazione con la classe *BlockNotes* e la classe *Menu* : anche in questo caso c'è una relazione del tipo "*ha-un*".
C'è una relazione di dipendenza tra la classe *Cameriere* e la classe *Cuoco*, perchè Il cameriere necessità del *Cuoco* per essere implementato e non appena il *Cuoco* termina di cucinare un piatto, il *Cameriere* lo serve.
Per questo motivo la relazione di associazione fra *Cliente* e *Cameriere* può essere vista anche come una dipendenza in quanto il cliente ordina tramite il cameriere che mostra le pietanze del Menù.

DESCRIZIONE E MOTIVAZIONE DEI PATTERN UTILIZZATI

Singleton

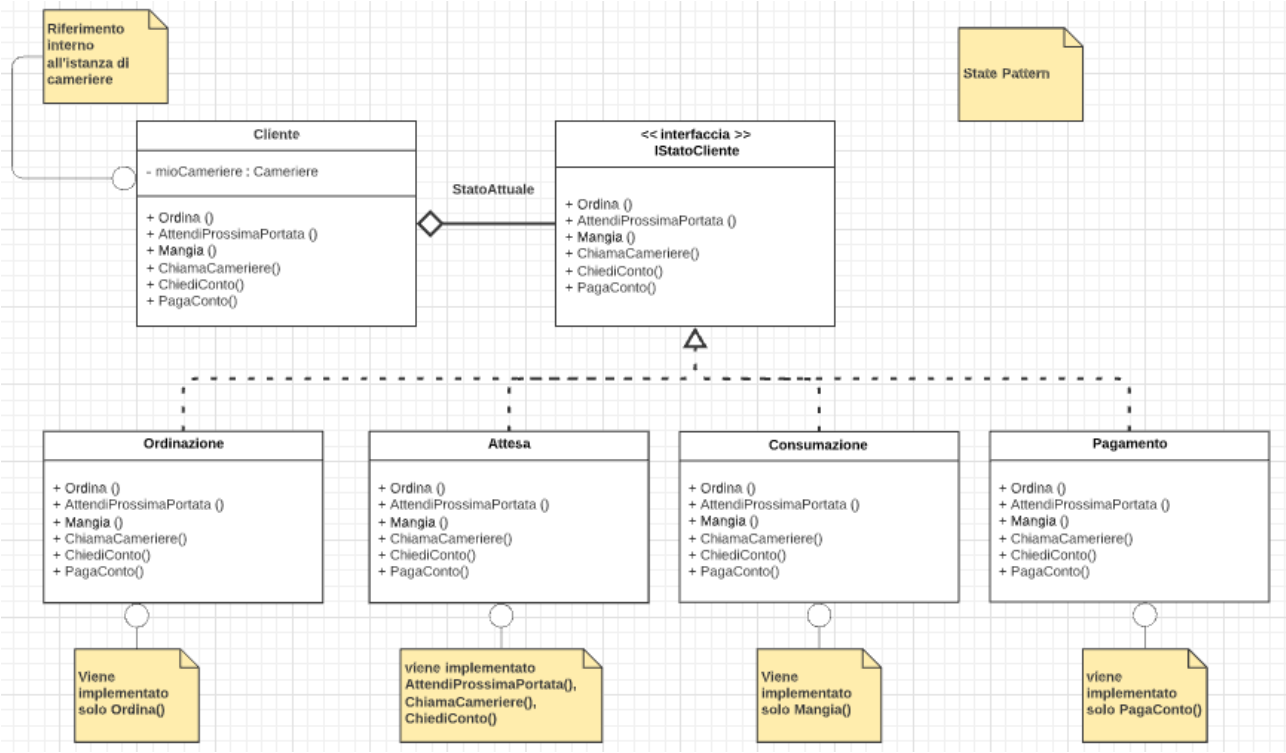
Viene utilizzato il pattern Singleton per le classi : Cliente, Cameriere, BlockNotes, Menù, CuocoPrimo, CuocoSecondo e CuocoDolci.
Ciò è motivato dal fatto che la simulazione non prevede altri clienti e altri camerieri, per questo si evita la possibilità di crearne altri.
Il menù è un Singleton perchè il ristorante, come ovvio che sia, ha solo una lista di pietanze su cui basarsi e non avrebbe senso creare più di un menù.

Composite Pattern

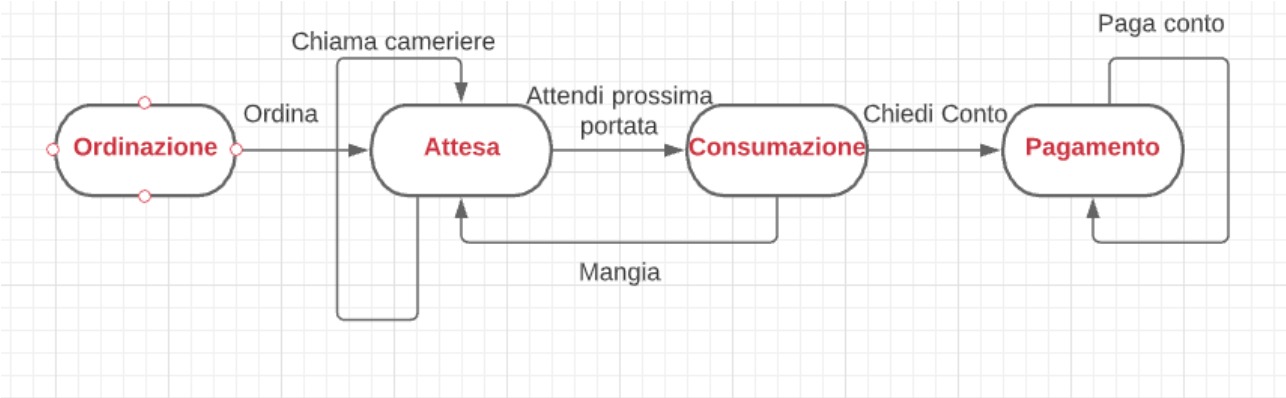


Viene applicato il composite pattern per permettere di trattare le pietanze singole e il menù del giorno allo stesso modo.
Così è più facile calcolare il conto in quanto la lista degli ordini effettuati dal cliente presente nel BlockNotes può contenere sia pietanze singole che menù del giorno, e su entrambi i tipi viene richiamato il metodo OttieniPrezzo() per calcolare il conto.
E' un composite pattern rivisitato poiché ComposizionePietanze (menù del giorno) può avere come figli solo pietanze singole e non altre composizioni. In particolare il menù del giorno contiene un primo, un secondo e un dolce.

State pattern



Viene applicato lo state pattern perchè il comportamento del cliente dipende dal suo stato il quale cambia durante l'esecuzione.
Per esempio il metodo Mangia() non può essere eseguito se il cliente si trova in uno stato diverso da Consumazione().
Il cliente può passare attraverso 4 stati e il passaggio da uno stato all'altro è rappresentato dal seguente diagramma :



In questo modo la logica relativa allo stato si sposta in una classe separata e viene rispettato il principio di responsabilità singola.

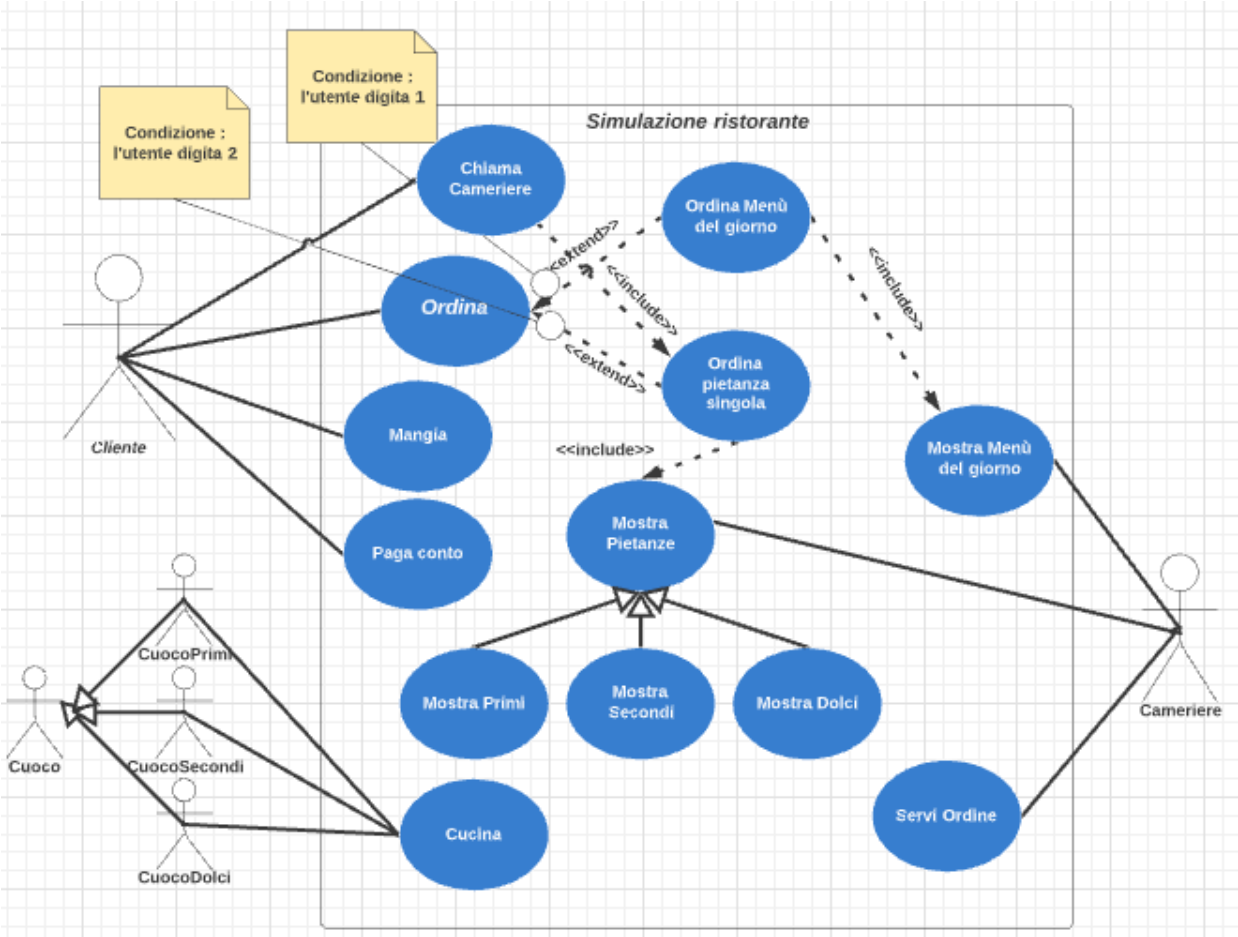
4. Documentazione sull'utilizzo.

Quando il software viene eseguito, il metodo Main() prende come parametri di ingresso i nomi di 4 file : primi.txt, secondi.txt, dolci.txt e bevande.txt.
Questi argomenti sono comunque fissati nel file launchSettings.json.
E' possibile modificare i 4 file in qualsiasi momento per cambiare le pietanze presenti nel menù. Quando si vuole aggiungere una nuova pietanza in uno dei 4 file bisogna rispettare queste regole :

- Nessun spazio vuoto all'inizio della riga.
- Inserire all'inizio della riga il nome della pietanza. Se è composta da più parole, bisogna separarle tramite un sottotratto _ .
- Dopo aver scritto il nome del piatto, lasciare uno spazio e inserire il prezzo.

5. Casi d'uso con relativo schema UML

Ecco lo schema dei casi d'uso in UML



Il caso d'uso più significativo è *Ordina.*
Lo scopo principale del software infatti è quello di permettere al cliente di ordinare. Il caso d'uso viene esteso da "Ordina Menù del giorno" e "Ordina pietanza singola" a seconda della scelta effettuata dal cliente : se il cliente digita 1 nella simulazione, ordina il menù del giorno, se digita 2, ordina le pietanze singolarmente.
Il cameriere ha il compito importante di mostrare le pietanze del menù e il menù del giorno affinchè il cliente possa effettuare la sua scelta.
Il caso d'uso "Mostra Pietanze" generalizza "Mostra Primi", "Mostra Secondi" e "Mostra dolci".
Il cliente può anche chiamare il cameriere per ordinare qualcos'altro, quindi questo caso d'uso include quello nel quale il cliente ordina le pietanze singolarmente.