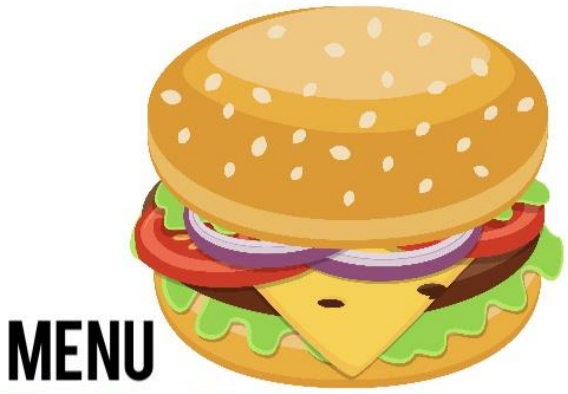


OBJECT DESIGN DOCUMENT

SOMMARIO

1. INTRODUZIONE

1.1 SVANTAGGI NELLA PROGETTAZIONE DEGLI OGGETTI

Comprensibilità vs Tempo: Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Al fine di rispettare queste linee guida il codice sarà integrato da commenti volti a migliorarne la leggibilità; tuttavia questo richiederà una maggiore quantità di tempo necessario per lo sviluppo del nostro progetto.

Prestazioni vs Costi: Dal momento che il budget allocato è spendibile principalmente in risorse umane e non consente l'acquisto di tecnologie proprietarie specifiche verranno utilizzati template open source e componenti hardware di nostra proprietà.

Interfaccia vs Usabilità: Verrà realizzata un'interfaccia grafica chiara e concisa, usando form e pulsanti predefiniti che hanno lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Linee guida per la documentazione dell'interfaccia

Gli sviluppatori dovranno seguire precise linee guida per la stesura del codice:

Naming Convention:

Per la documentazione delle interfacce bisognerà utilizzare nomi:

- Descrittivi;
- Pronunciabili;
- Di lunghezza medio-corta;
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9).

Variabili:

- I nomi delle variabili dovranno iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola;
- In ogni riga dovrà esserci un'unica variabile dichiarata, eventualmente allineata con quelle del blocco dichiarativo;
- In determinati casi, è possibile utilizzare il carattere underscore "_": il caso principale previsto è quello relativo alla dichiarazione di costanti oppure di proprietà statiche.

Metodi:

- I nomi dei metodi dovranno iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola, secondo la "Camel Notation";
- Il nome del metodo sarà costituito da un verbo che ne identifica l'azione seguito da un sostantivo, eventualmente aggettivato;
- Il nome dei metodi accessori e modificatori seguirà, rispettivamente, i pattern getNomeVariabile e setNomeVariabile;
- Ai metodi verrà aggiunto un commento JavaDoc, il quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo.

Classi Java e pagine JSP:

- I nomi delle classi e delle pagine dovrà iniziare con la lettera maiuscola, così come le parole successive all'interno del nome;
- I nomi delle classi e delle pagine dovrà corrispondere alle informazioni e le funzioni fornite da quest'ultime;
- Le classi saranno strutturate prevedendo rispettivamente:
  1. Dichiarazione della classe pubblica;
  2. Dichiarazione di costanti;
  3. Dichiarazioni di variabili di classe;
  4. Dichiarazioni di variabili di istanza;
  5. Costruttore;
  6. Metodi.

Packages:

- I nomi dei packages dovranno essere scritti in minuscolo concatenando insieme diversi sostantivi o sigle, separate dal carattere ".";
- Non saranno ammessi caratteri speciali.

1.2 Definizioni, acronimi e abbreviazioni

RAD = Requirement Analysis Document

SDD = System Design Document

ODD = Object Design Document

1.3 Riferimenti

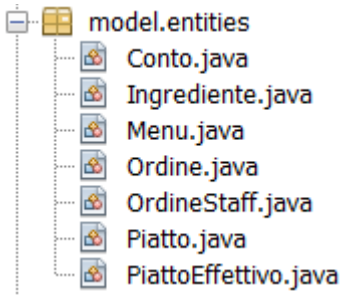
- RAD_MENUMAXI
- SDD_MENUMAXI

2. Packages

La parte server del nostro sistema è divisa orizzontalmente come segue:

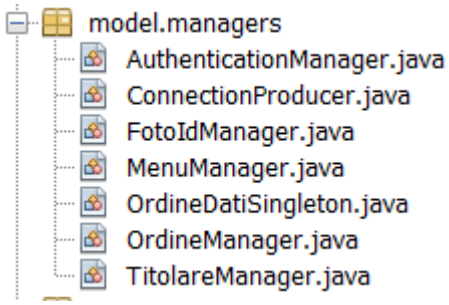
- model.entities
- model.managers
- service
- servlet
- WebPages

2.1 MODEL.ENTITIES



Classe	Descrizione
Conto	Classe che rappresenta le informazioni relative ad una richiesta di conto effettuata da un cliente.
Ingrediente	Classe che rappresenta le informazioni relative ad un ingrediente.
Menu	Classe che rappresenta le informazioni relative al menu.
Ordine	Classe che rappresenta le informazioni relative ad un ordine effettuato da un cliente.
OrdineStaff	Classe che rappresenta le informazioni relative all'ordine che lo staff riceve.
Piatto	Classe che rappresenta le informazioni relative al piatto.
PiattoEffettivo	Classe che rappresenta le informazioni relative ad un piatto ordinato da un cliente (quindi con le relative modifiche).

2.2 MODEL.MANAGERS

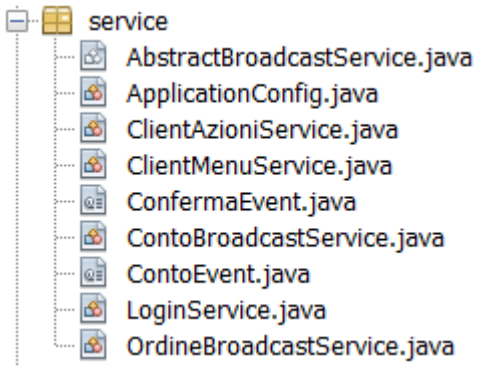


Classe	Descrizione
AuthenticationManager	Classe che implementa le operazioni riguardanti l'autenticazione.
ConnectionProducer	Classe responsabile di fornire connessioni al database.

FotoIdManager	Classe che genera id univoci per la memorizzazione delle foto dei piatti.
MenuManager	Classe che si occupa di fornire i dati relativi al menu.
OrdineDatiSingleton	Classe che consente di tenere costantemente in memoria le liste di ordini attivi, ordini da completare e richieste conti.
OrdineManager	Classe che permette di salvare un ordine una volta che il conto è stato richiesto e offre un'interfaccia alle operazioni del singleton nascondendone l'utilizzo.
TitolareManager	Classe che implementa le operazioni effettuabili dal titolare: richieste dei dati relativi a guadagni e popolarità delle portate e operazioni di aggiornamento del menu.

2.3 SERVICE

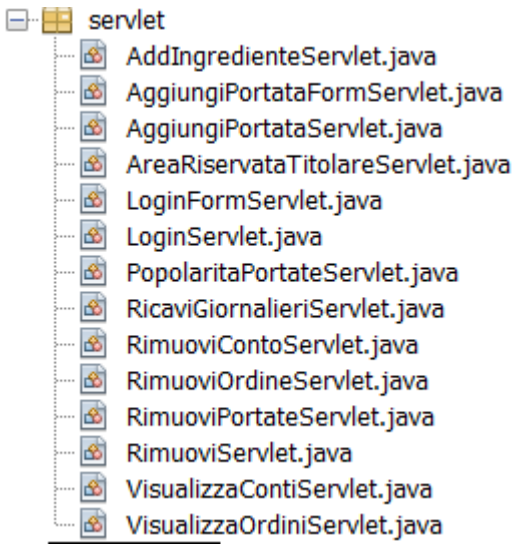
Il package service offre un interfaccia ad alcune delle operazioni offerte dai managers pubblicandole all'esterno come Rest-Api.



Classe	Descrizione
AbstracrBroadcastService	Classe astratta usata come base di un web service che invia eventi ai client inscritti al servizio .
ApplicationConfig	Classe standard per la pubblicazione di web services.
ClientAzioniService	Classe che offre un'interfaccia alle operazioni che può intraprendere un client: aggiunta ordine e richiesta conto.
ClientMenuService	Classe che offre l'intero menu.
ConfermaEvent	Interfaccia qualifier.
ContoBroadcastService	Classe che permette a un client di iscriversi per ricevere le nuove richieste conto in arrivo, e di notificarle in broadcast a tutti i client inscritti.
ContoEvent	Interfaccia qualifier.

LoginService	Classe che offre il servizio di autenticazione all'esterno.
OrdineBroadcastService	Classe che permette a un client di iscriversi per ricevere le nuove i nuovi ordini in arrivo, e di notificarli in broadcast a tutti i client inscritti.

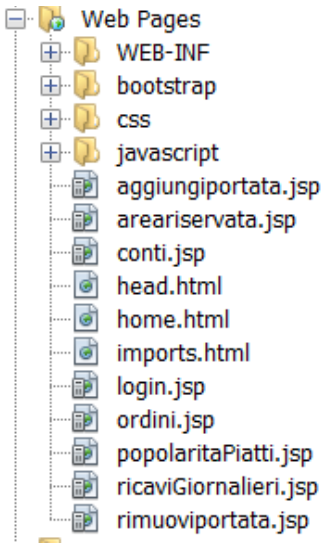
2.4 SERVLET



Classe	Descrizione
AddIngredienteServlet	Servlet che gestisce l'aggiunta di un nuovo ingrediente da parte del titolare.
AggiungiPortataFormServlet	Servlet che gestisce i campi da visualizzare staticamente nel form per l'aggiunta di una nuova portata al menu.
AggiungiPortataServlet	Servlet che gestisce l'aggiunta di una nuova portata al menu da parte del titolare.
AreaRiservataTitolareServlet	Servlet che permette la visualizzazione della jsp riguardante l'area riservata del titolare.
LoginFormServlet	Servlet che permette la visualizzazione della jsp riguardante il form di login.
LoginServlet	Servlet che permette di effettuare il login.
PopolaritaPortateServlet	Servlet che gestisce la visualizzazione della popolarità delle portate da parte del titolare.
RicaviGiornalieriServlet	Servlet che gestisce la visualizzazione dei ricavi giornalieri da parte del titolare.
RimuoviContoServlet	Servlet che gestisce la stampa delle richieste di conto effettuate dai clienti
RimuoviOrdineServlet	Servlet che gestisce la rimozione di un ordine completato tra quelli effettuati dai clienti.

RimuoviPortateServlet	Servlet permette la visualizzazione della jsp per la rimozione di una portata dal menu da parte del titolare.
RimuoviServlet	Servlet che gestisce la rimozione di una portata dal menu da parte del titolare.
VisualizzaContiServlet	Servlet che gestisce la visualizzazione delle richieste di conto effettuate dai clienti.
VisualizzaOrdiniServlet	Servlet che gestisce la visualizzazione delle ordinazioni effettuate dai clienti.

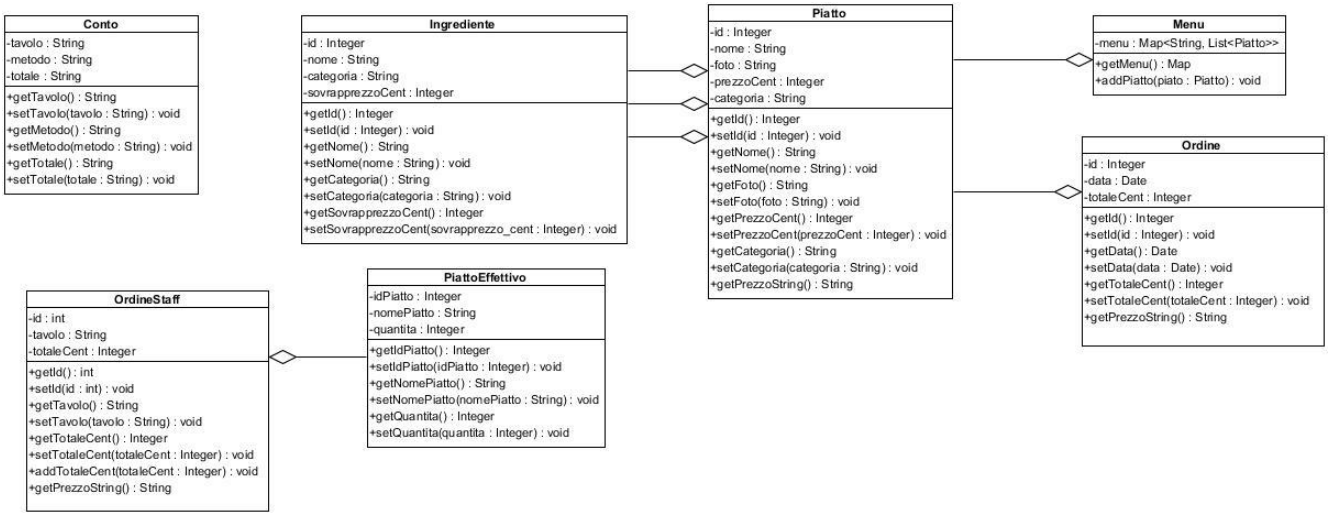
2.5 WEBPAGES



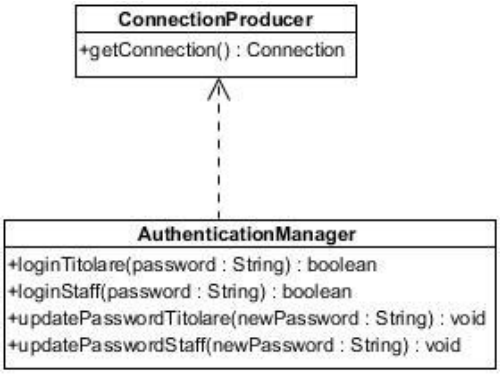
Classe	Descrizione
aggiungiportata	Sezione in cui il titolare può aggiungere una nuova portata al menu.
areariservata	Sezione che rappresenta l'area riservata del titolare.
conti	Sezione in cui lo staff può visualizzare le richieste di conto effettuate dai clienti.
home	Sezione in cui è possibile scegliere l'area per il quale effettuare il login.
login	Sezione in cui è possibile immettere la password per effettuare il login all'apposita area.
ordini	Sezione in cui lo staff può visualizzare le ordinazioni effettuate dai clienti.
popolaritaPiatti	Sezione in cui il titolare può visualizzare la popolarità delle portate per un dato mese.
ricaviGiornalieri	Sezione in cui il titolare può visualizzare i ricavi giornalieri della sua attività.
rimuoviportata	Sezione in cui il titolare può rimuovere una portata dal menu.

3. INTERFACCE DI CLASSE

3.1 INTERFACCE ENTITIES

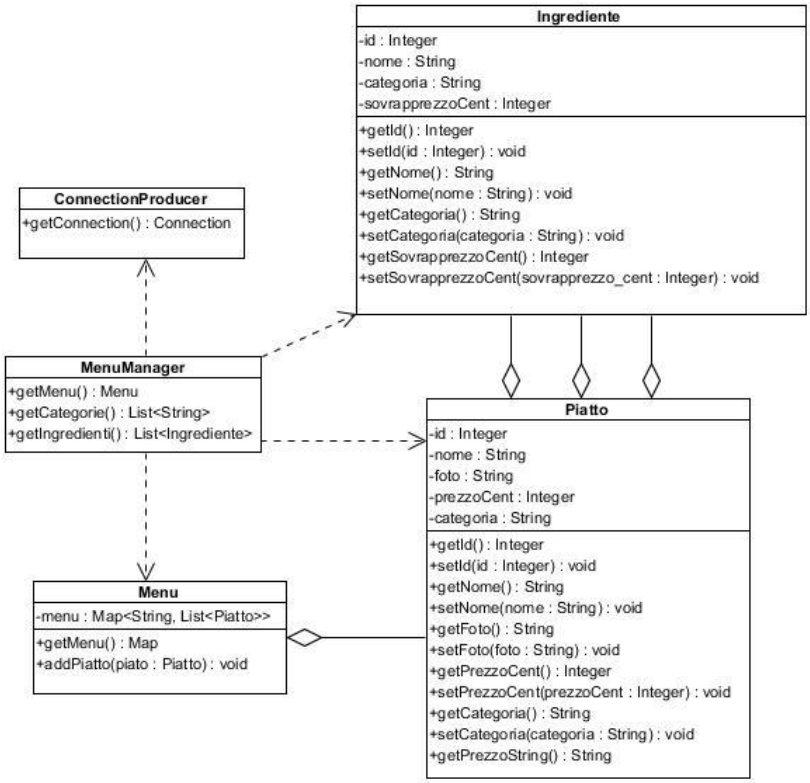


3.2 AUTHENTICATIONMANAGER



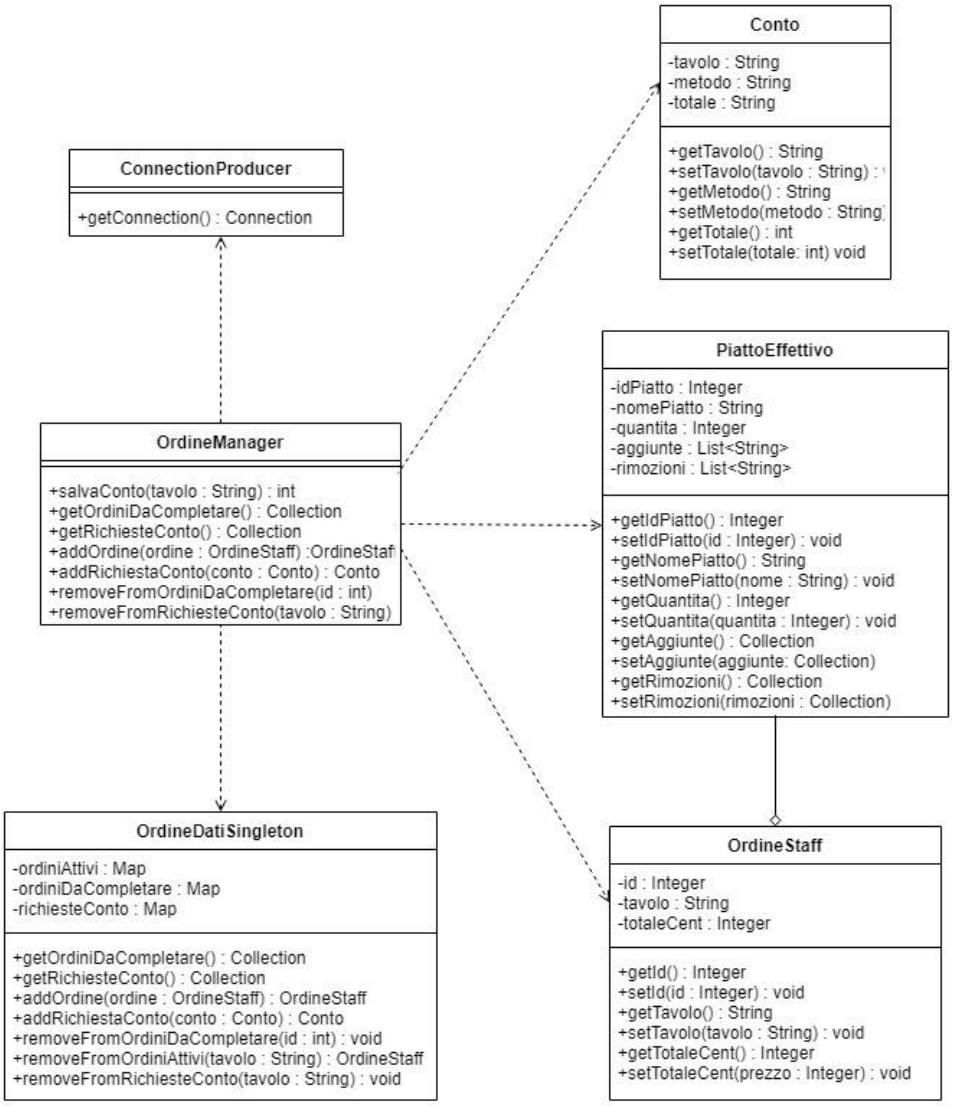
Nome Classe	AuthenticationManager
Pre-condizione	• **context** AuthenticationManager:: loginTitolare(password: String): boolean **pre:** password!=null; • **context** AuthenticationManager:: loginStaff(password: String): boolean **pre:** password!=null; • **context** AuthenticationManager:: updatePasswordTitolare(newPassword: String): void **pre:** newPassword!=null; • **context** AuthenticationManager:: updatePasswordStaff(newPassword: String): void **pre:** newPassword!=null;
Post-condizione	
Invarianti	

3.3 MENUMANAGER



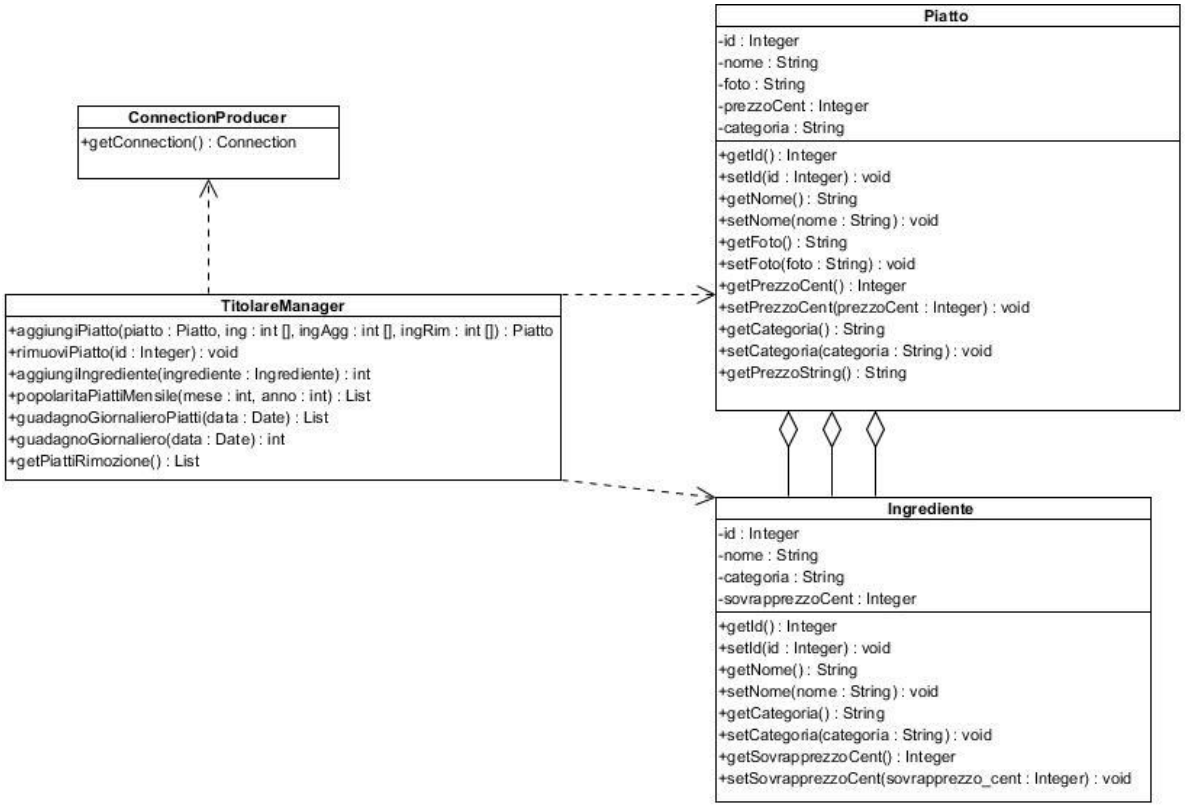
Nome Classe	**StaffManager**
Pre-condizione	
Post-condizione	• **context** MenuManager:: getMenu(): Menu m **post:** m!=null; • **context** MenuManager:: getCategorie(): List<String> c **post;** c!=null; • **context** MenuManager:: getIngredienti(): List<Ingredienti> i **post:** i!=null;
Invarianti	

3.4 ORDINEMANAGER



Nome Classe	**AccountTitolareManager**
Pre-condizione	• **context** OrdineManager:: salvaConto(tavolo: String): String **pre:** tavolo!=null && OrdineDatiSingleton::getRichiesteConto().contains(tavolo); • **context** OrdineManager::addOrdine(ordine : OrdineStaff) **pre:** ordine != null • **context** OrdineManager::addRichiestaConto(conto :Conto) **pre:** conto != null && OrdineDatiManager::getOrdiniAttivi().contains(conto.tavolo)
Post-condizione	
Invarianti	

3.5 TITOLAREMANAGER

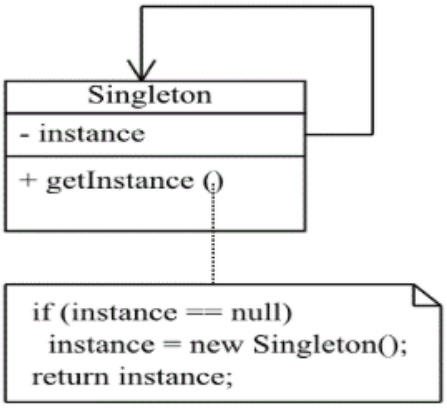


Nome Classe	TitolareManager
Pre-condizione	• **context** TitolareManager:: aggiungiIngrediente(ingrediente: Ingrediente): int **pre:** ingrediente!=null; • **context** TitolareManager:: popolaritaPiattiMensili(mese: int, anno:int): List **pre:** 1<=mese<=12 && 2019<=anno<=OGGI.anno && !(anno =OGGI.anno && mese>OGGI.mese) • **context** TitolareManager::guadagnoGiornaliero(date: Date): List **pre:** date!=null; • **context** TitolareManager:: guadagnoGiornaliero(date:Date): int **pre:** date!=null;
Post-condizione	
Invarianti	

4. DESIGN PATTERN

4.1 SINGLETON

Il singleton è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.



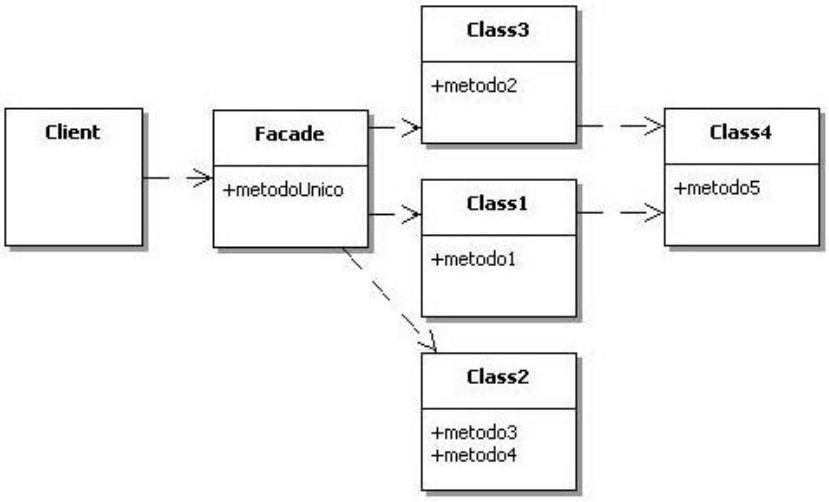
Utilizzo:

Il Singleton Pattern è stato scelto per la classe **OrdiniDatiSingleton** perché contiene dei dati che devono essere mantenuti in memoria per tutta la durata del ciclo di vita dell'applicazione; è stato poi scelto per implementare le classi **Broadcaster** che devono tenere traccia per tutta la durata del ciclo di vita dell'applicazione dei client inscritti per ricevere la notifica tramite evento. Inoltre verrà utilizzato per il **fotoIdManager** che fornisce id univoci per il salvataggio della foto.

4.2 FAÇADE

Il Façade Pattern è un design pattern di tipo **Strutturale** che nasconde la complessità di un sistema e offre una interfacca ai client che vogliono accedere ad esso.

In italiano si traduce con "**Facciata**" perchè è proprio questo il suo obiettivo: semplificare l'accesso ad un sistema complesso mostrando una facciata.



Utilizzo:

Nel progetto Menu Maxi il façade è stato scelto per offrire delle Rest-api nelle classi del package service e per nascondere il singleton OrdineDatiSingleton nel sottosistema OrdineManager.