

## Ingegneria del Software

UniTN - Dipartimento di Ingegneria e Scienza dell'Informazione  
Anno accademico 2022 - 2023  
Bevilacqua L. - Sartore A. - Tecchio L.



**UNIVERSITY  
OF TRENTO**

**Nome del progetto:**

# Hungry Everywhere

**Titolo del documento:**

# Documento di architettura

# INDICE

<b>1. SCOPO DEL DOCUMENTO</b>	<b>4</b>
<b>2. DIAGRAMMA DELLE CLASSI</b>	<b>5</b>
2.1 Utenti e sistemi esterni	6
Figura 1. Classi per utenti e sistemi esterni	6
2.2 Gestione autenticazione	7
Figura 2. Classe per gestione autenticazione	7
2.3 Sistema di pagamento	8
Figura 3. Classe per il sistema di pagamento	8
2.4 Google Maps API	9
Figura 4. Classe per Google Maps API	9
2.5 Pagina di Login	10
Figura 5. Classe per Pagina di Login	10
2.6 Homepage	11
Figura 6. Classe per Homepage	11
2.7 Pagina dei ristoranti	12
Figura 7. Classe per Pagina dei ristoranti	12
2.8 Pagina del ristorante	13
Figura 8. Classe per Pagina del ristorante	13
2.9 Carrello	14
Figura 9. Classe per la pagina Carrello	14
2.10 Diagramma delle classi complessivo	15
<b>3. CODICE IN OBJECT CONSTRAINT LANGUAGE</b>	<b>16</b>
3.1 Gestione Ordine	16
3.2 Numero di ristoranti	17

---

3.3 Esistenza del ristorante	18
3.4 Esistenza ordine	19
3.5 Applica sconto Cliente	20
3.6 Applica sconto Pietanza	21
3.7 Esistenza Cliente	22
3.8 Esistenza Pietanza	23
3.9 Esistenza Utente Generico	24
<b>4. DIAGRAMMA DELLE CLASSI CON CODICE OCL</b>	<b>25</b>

# 1. SCOPO DEL DOCUMENTO

Il presente documento riporta la definizione dell'architettura del progetto Hungry Everywhere usando diagrammi delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL). Nel precedente documento è stato presentato il diagramma degli use case, il diagramma di contesto e quello dei componenti. Ora, tenendo conto di questa progettazione, viene definita l'architettura del sistema dettagliando da un lato le classi che dovranno essere implementate a livello di codice e dall'altro la logica che regola il comportamento del software. Le classi vengono rappresentate tramite un diagramma delle classi in linguaggio UML. La logica viene descritta in OCL perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

---

## 2. DIAGRAMMA DELLE CLASSI

Nel presente capitolo vengono presentate le classi previste nell'ambito del progetto Hungry Everywhere. Ogni componente presente nel diagramma dei componenti diventa una o più classi. Tutte le classi individuate sono caratterizzate da un nome, una lista di attributi che identificano i dati gestiti dalla classe e una lista di metodi che definiscono le operazioni previste all'interno della classe. Ogni classe può essere anche associata ad altre classi e, tramite questa associazione, è possibile fornire informazioni su come le classi si relazionano tra loro. Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti.

## 2.1 Utenti e sistemi esterni

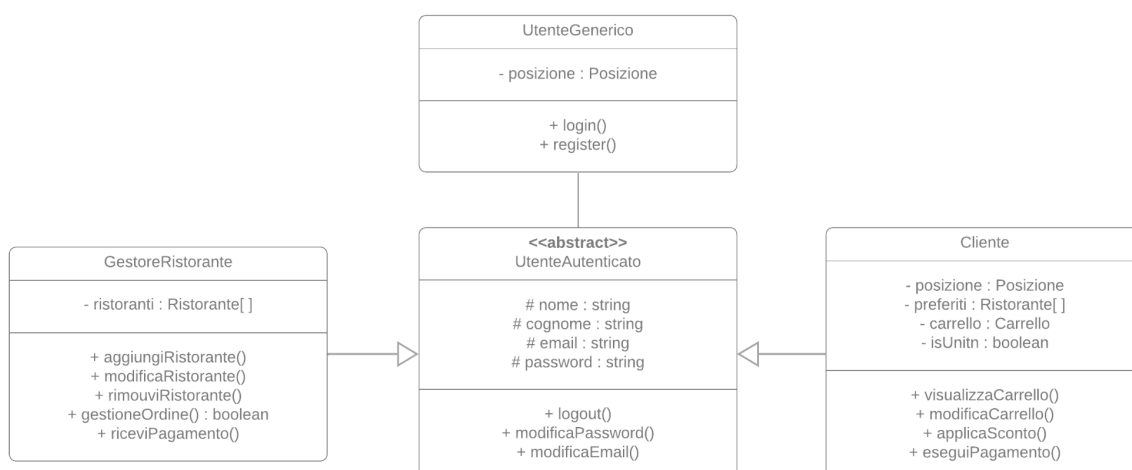
Analizzando il diagramma di contesto realizzato per il progetto Hungry Everywhere si nota la presenza di **due attori “Utente”** e **“Gestore”**. L'attore “Utente” è colui che utilizza il sito per visualizzare i ristoranti vicini a sé e, una volta autenticato, diventa “Cliente” e può ordinare le pietanze. Il “Gestore” è colui che aggiunge, modifica e rimuove i propri ristoranti ed entra in relazione con il Cliente attraverso la conferma o il rifiuto dell'ordine. Sono state dunque individuate 4 classi: *UtenteGenerico*, *UtenteAutenticato*, *GestoreRistorante* e *Cliente*.

La classe **UtenteGenerico** rappresenta un qualsiasi utente non autenticato che usufruisce del sito. Presenta due funzioni per registrarsi o autenticarsi.

La classe **UtenteAutenticato** è una classe astratta ed indica un qualsiasi utente dopo aver eseguito il login. In questa classe sono state aggiunte le operazioni verso il **sistema esterno “Gestore credenziali”** e **“Google Maps API”** anch'essi presenti nel diagramma del contesto.

La classe **Cliente** è una generalizzazione di *UtenteAutenticato*. In questa classe sono state aggiunte le operazioni verso il **sistema esterno “Sistema di pagamento”** anch'esso presente nel diagramma del contesto.

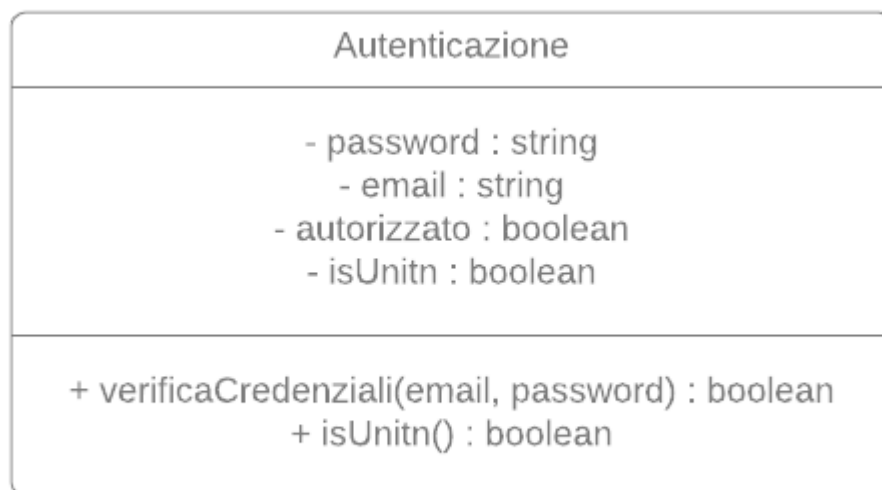
La classe **GestoreRistorante** è una generalizzazione di *UtenteAutenticato*. In questa classe sono state aggiunte le operazioni verso il **sistema esterno “Sistema di pagamento”** anch'esso presente nel diagramma del contesto.



**Figura 1. Classi per utenti e sistemi esterni**

## 2.2 Gestione autenticazione

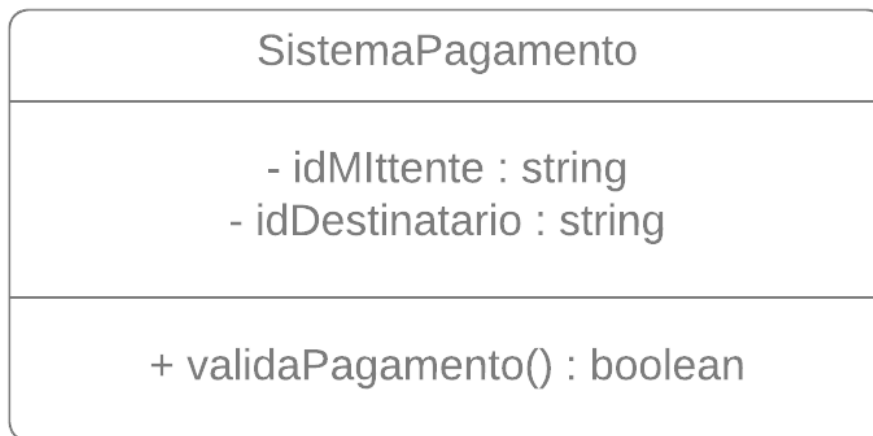
Il diagramma di contesto analizzato presenta un **sistema paritario** denominato “**gestore credenziali**” che si occupa di fornire l’autorizzazione per l’accesso al sistema per gli attori “Utente” e “Gestore”. Il diagramma di contesto presenta anche un **sistema subordinato** denominato “**Login con Google**” che fornisce l’autorizzazione per l’accesso al sistema per l’attore “Utente”. La classe identificata per gestire tali sistemi è detta “**Autenticazione**”, che si occuperà di validare o meno i dati che gli verranno passati dall’applicazione Hungry Everywhere e di notificare l’esito di tale operazione. Nella classe è presente un attributo “**isUnitn**” che serve al sistema per capire se l’utente ha eseguito il login attraverso Google ed è uno studente di UniTn.



**Figura 2. Classe per gestione autenticazione**

## 2.3 Sistema di pagamento

Il diagramma di contesto analizzato presenta un **sistema subordinato** denominato **“Sistema di pagamento”** che si occupa di gestire le richieste di pagamento che vengono effettuate dagli utenti e gli abbonamenti pagati dai gestori. La classe che si occuperà di tale compito è indicata come **“SistemaPagamento”**, la quale valuterà se approvare o rifiutare le transizioni degli acquisti.

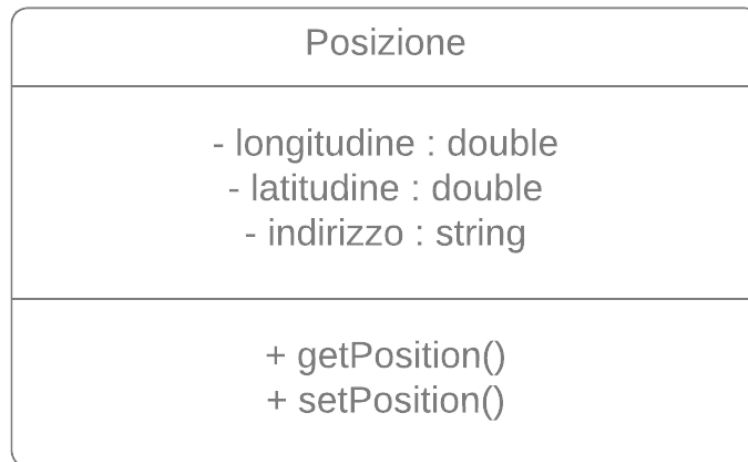


**Figura 3. Classe per il sistema di pagamento**



## 2.4 Google Maps API

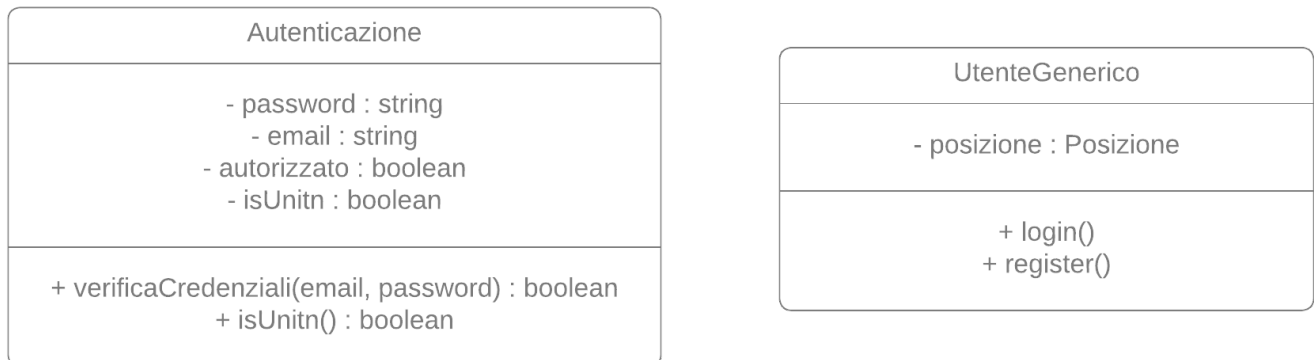
Il diagramma di contesto analizzato presenta un secondo **sistema subordinato** denominato “**Google Maps API**”, del quale è stata individuata la classe *Posizione* che si occuperà di fornire all'applicazione la posizione degli utenti in base alle coordinate che gli vengono fornite.



**Figura 4. Classe per Google Maps API**

## 2.5 Pagina di Login

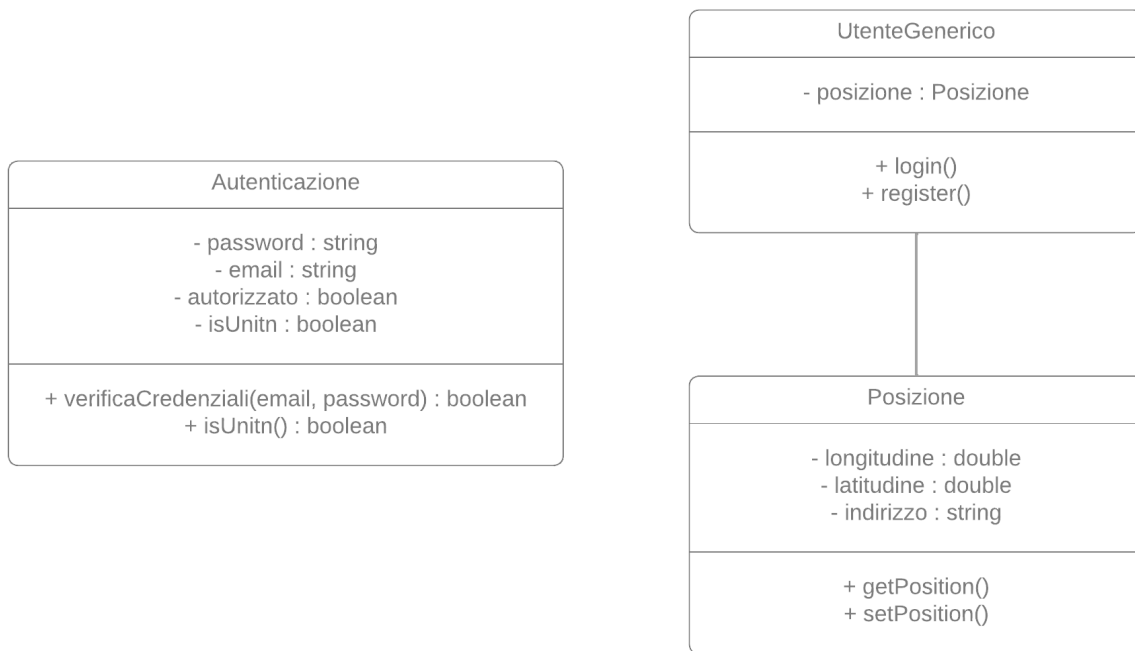
Il componente **“Pagina di Login”** presente nel diagramma dei componenti gestisce la richiesta di autenticazione o di registrazione da parte di un utente generico. Definiamo le classi **“UtenteGenerico”** e **“Autenticazione”** che descrivono l'utente generico con le sue funzionalità e come viene gestita l'autenticazione, descritta al punto [2.2](#).



**Figura 5. Classe per Pagina di Login**

## 2.6 Homepage

Nel diagramma dei componenti è presente il componente **Homepage** che è il primo elemento con cui l'utente si interfaccia e da cui sono state identificate tre classi. La classe **“Posizione”** si occupa di richiedere la posizione ad un utente generico, come descritto nel [2.4](#), mentre la classe **“Autenticazione”** permettere ad un utente generico di registrarsi o di effettuare il login [2.5](#).



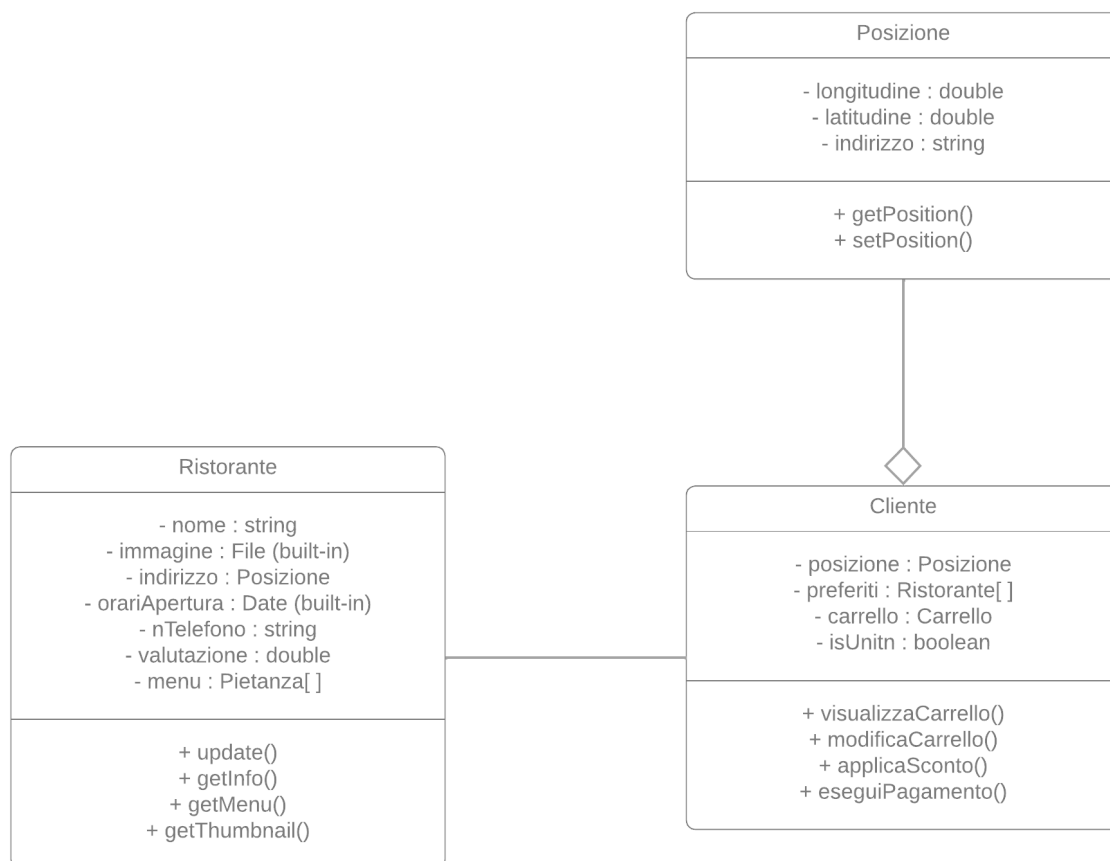
**Figura 6. Classe per Homepage**

## 2.7 Pagina dei ristoranti

Nel diagramma dei componenti è presente il componente **“Pagina dei ristoranti”** che fornisce tutti i ristoranti vicini alla posizione fornita dall'utente. Le informazioni dei singoli ristoranti visualizzate su questa pagina vengono fornite dal metodo **“getThumbnail”** della classe **Ristorante**.

La classe **Posizione** fornisce la posizione dell'utente come descritto nel punto [2.4](#).

La classe **Cliente** ha un attributo **“preferiti”** che permette all'utente di visualizzare e modificare i propri ristoranti preferiti.



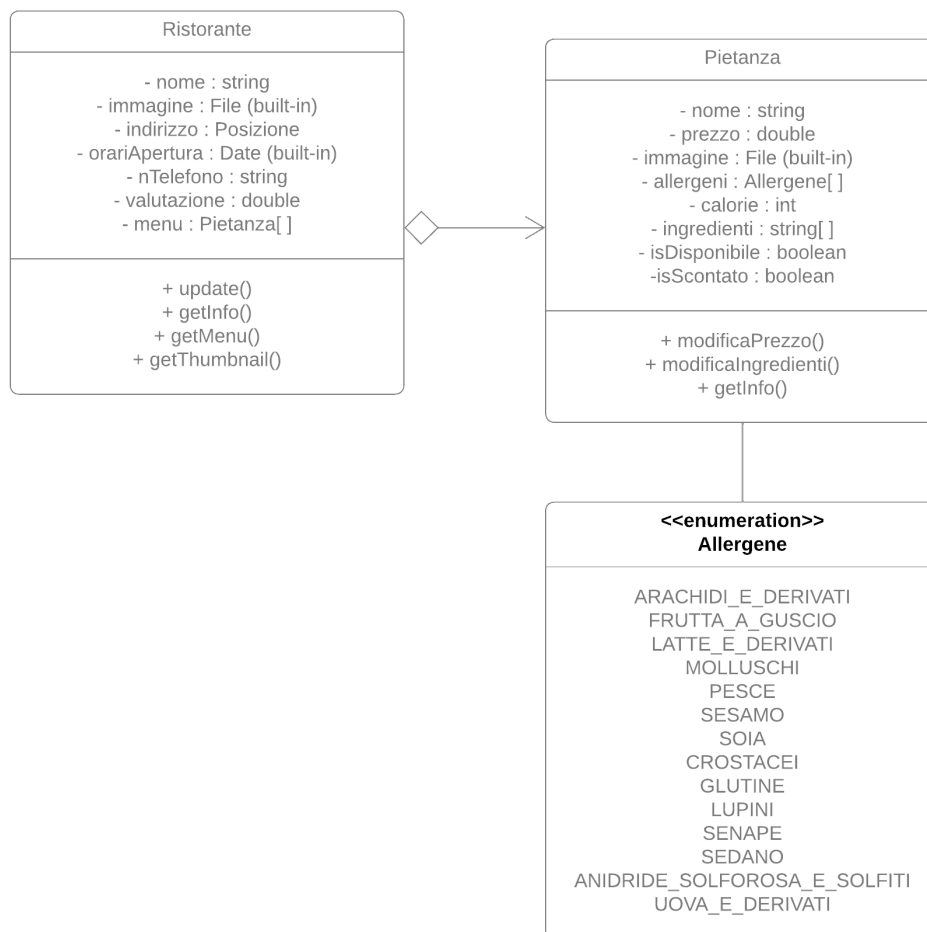
**Figura 7. Classe per Pagina dei ristoranti**

## 2.8 Pagina del ristorante

Nel diagramma dei componenti è presente il componente **“Pagina del ristorante”** che fornisce i dettagli del ristorante scelto dall’utente e mostra il menù con tutte le pietanze. Analizzando il componente in questione sono state identificate le seguenti due classi: *Ristorante* e *Pietanza*.

La classe **“Ristorante”** contiene tutti gli attributi necessari per descrivere i dettagli del ristorante e ha l’attributo menù che contiene diverse pietanze.

La classe **“Pietanza”** descrive i dettagli delle varie pietanze presenti nel menù. È presente una *enumeration* per descrivere tutte i possibili allergeni di quella determinata pietanza.



**Figura 8. Classe per Pagina del ristorante**

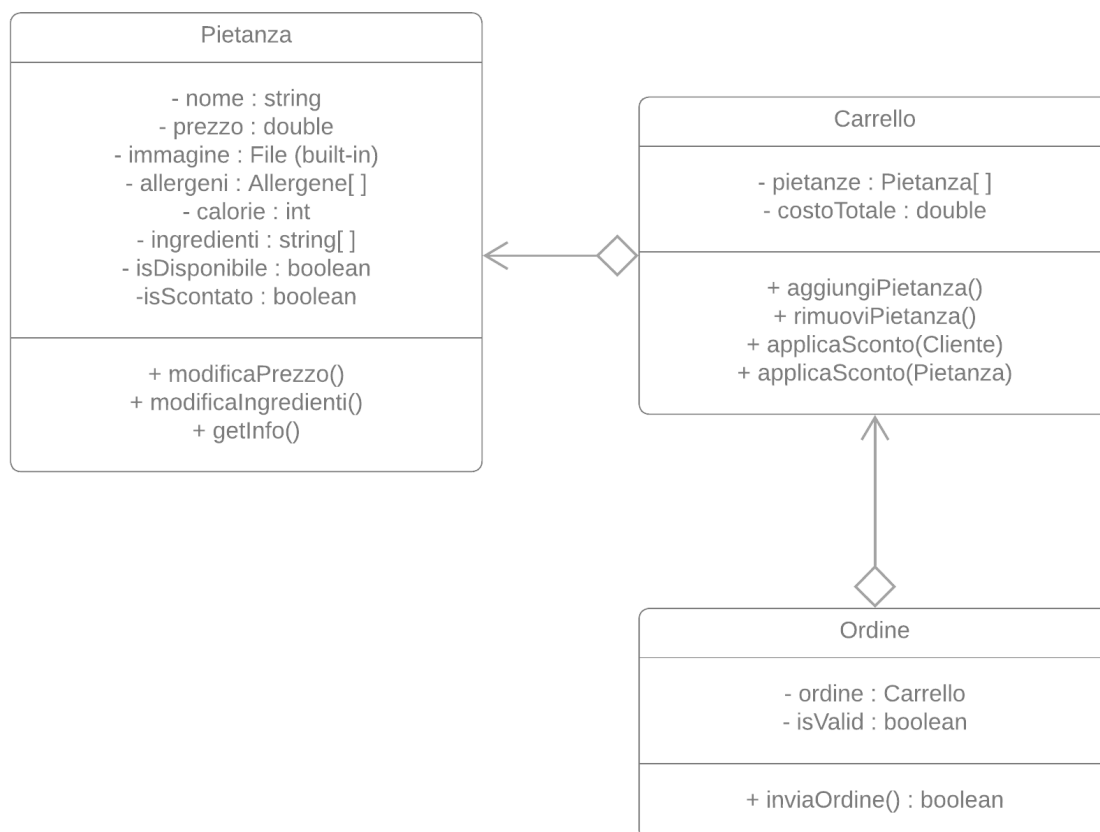
## 2.9 Carrello

Nel diagramma dei componenti è presente il componente **“Carrello”** che descrive le operazioni che vengono eseguite all’interno del carrello di ogni cliente. Analizzando il componente in questione vengono individuate le seguenti classi: *Pietanza*, *Carrello*, *Ordine*.

La classe **“Pietanza”** indica come viene descritta la pietanza all’interno del carrello del Cliente. Può essere modificata rispetto a quella presente nel menù del ristorante tramite la funzione **“modificaIngredienti()”**.

La classe **“Carrello”** descrive il carrello del cliente. L’attributo **“pietanze”** è un vettore di *Pietanza* che indica l’ordine complessivo del Cliente. L’attributo **“costoTotale”** indica il costo totale dell’ordine e la funzione **“applicaSconto()”** ne modifica il valore nel caso ci siano degli sconti applicabili.

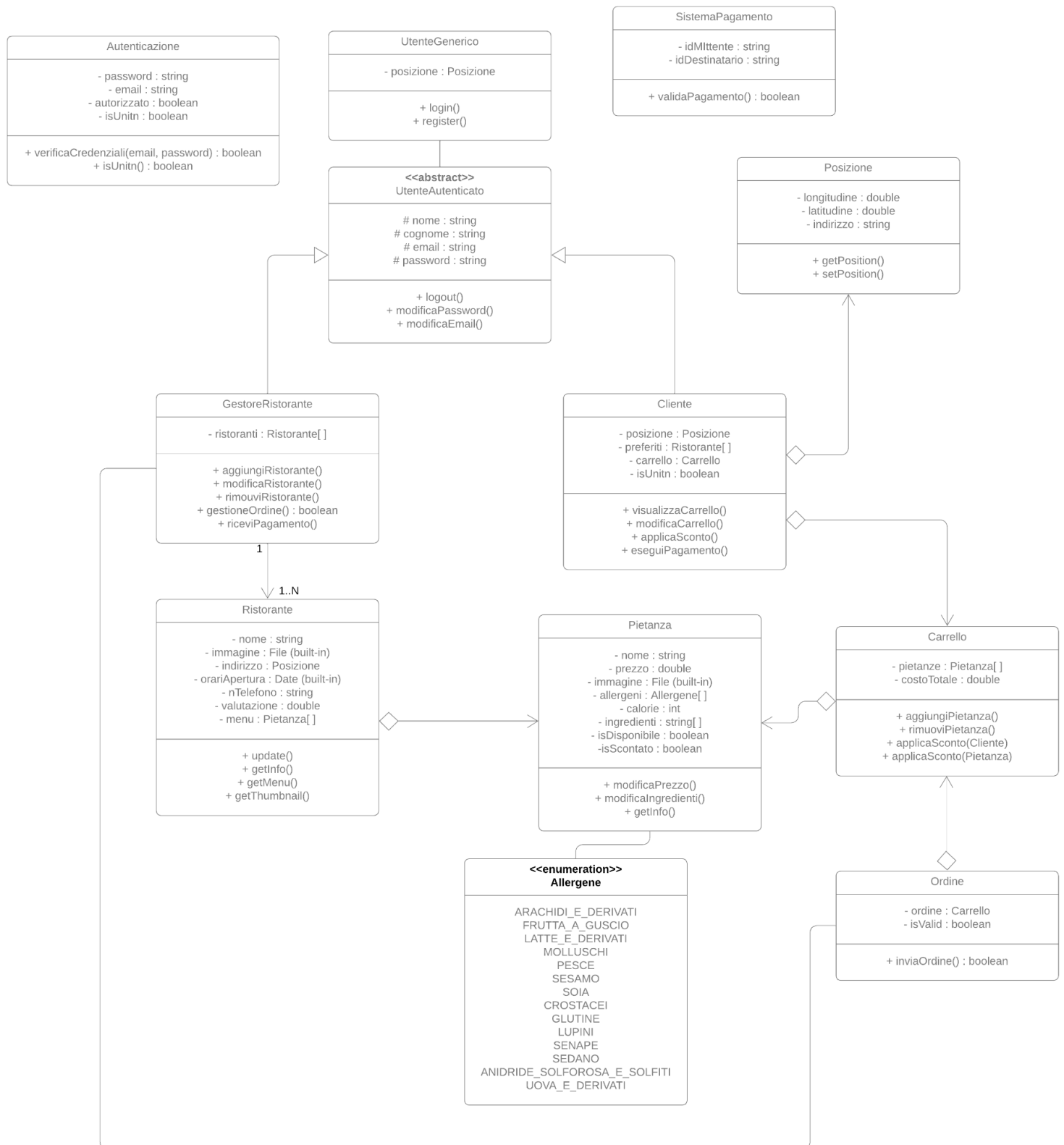
La classe **“Ordine”** descrive l’ordine del Cliente con la relativa conferma o meno da parte del Gestore.



**Figura 9. Classe per la pagina Carrello**

## 2.10 Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi con tutte le classi fino ad ora presentate.



## 3. CODICE IN OBJECT CONSTRAINT LANGUAGE

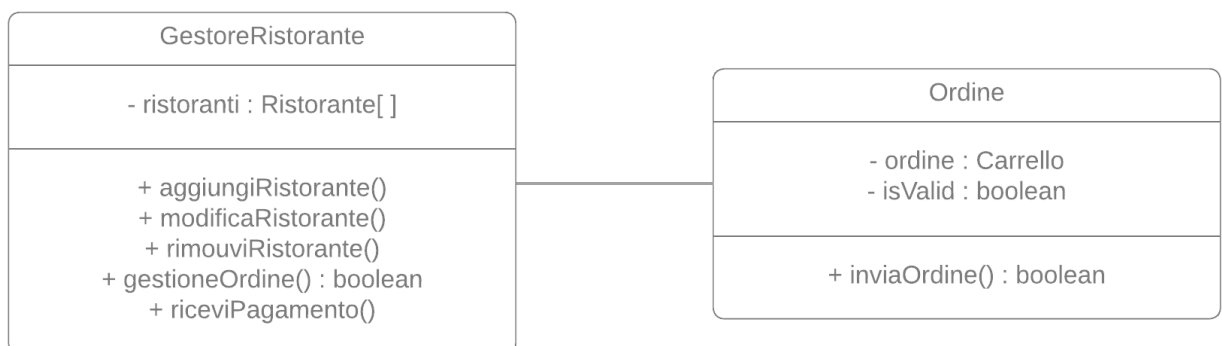
In questo capitolo è descritta in modo formale la logica prevista nell'ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

### 3.1 Gestione Ordine

La gestione dell'ordine avviene in due classi diverse: *GestoreRistorante* e *Ordine*. La classe **Ordine** tramite la funzione "inviaOrdine()" invia la richiesta dell'ordine alla classe **GestoreRistorante** la quale controlla l'ordine e, in base alla risposta (boolean), si imposta l'attributo *isValid* di **Ordine**. Tutto ciò è tradotto dal seguente codice OCL:

```
context Ordine::inviaOrdine(GestoreRistorante) post:
if GestoreRistorante::gestioneOrdine(self) = true
then self.isValid = true else self.isValid = false
endif
```

Riferito alle seguenti classi:



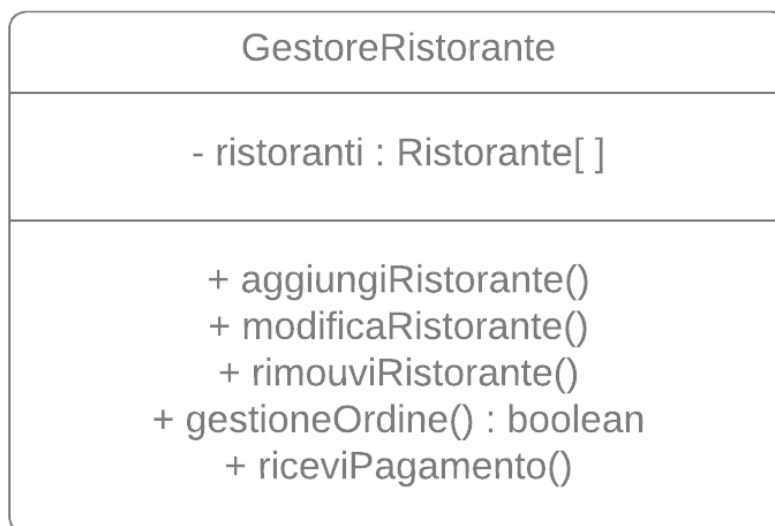


### 3.2 Numero di ristoranti

È presente una condizione che deve sempre essere verificata affinché la classe **GestoreRistorante** possa esistere, ovvero il numero di ristoranti deve essere maggiore di 0. Ciò è espresso dal seguente codice OCL attraverso l'uso di una invariante:

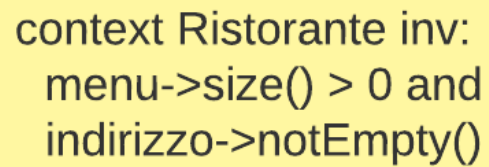
```
context GestoreRistorante inv:  
ristoranti->size() > 0
```

Riferito alla seguente classe:



### 3.3 Esistenza del ristorante

Sono presenti due condizioni che devono sempre essere verificate affinché la classe **Ristorante** possa esistere, ovvero il menù deve avere almeno una pietanza e il campo indirizzo non deve essere vuoto. Ciò è espresso dal seguente codice OCL attraverso l'uso di una invariante:



```
context Ristorante inv:  
  menu->size() > 0 and  
  indirizzo->notEmpty()
```

Riferito alla seguente classe:

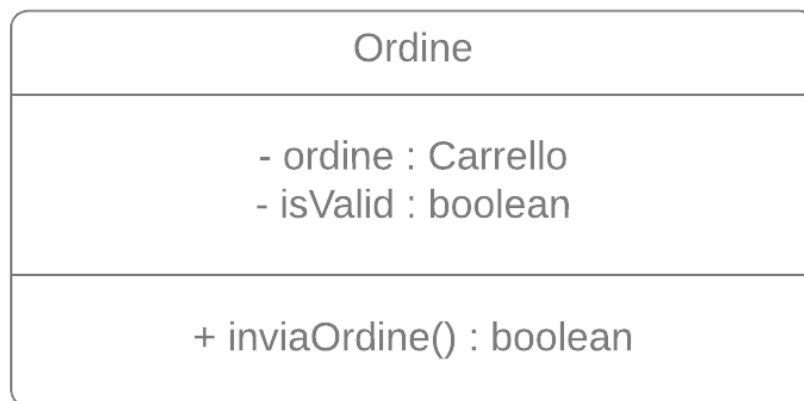


### 3.4 Esistenza ordine

È presente una condizione che deve sempre essere verificata affinché la classe **Ordine** possa esistere, ovvero l'attributo ordine non deve essere vuoto. Ciò è espresso dal seguente codice OCL attraverso l'uso di una invariante:

```
context Ordine inv:  
ordine->notEmpty()
```

Riferito alla seguente classe:

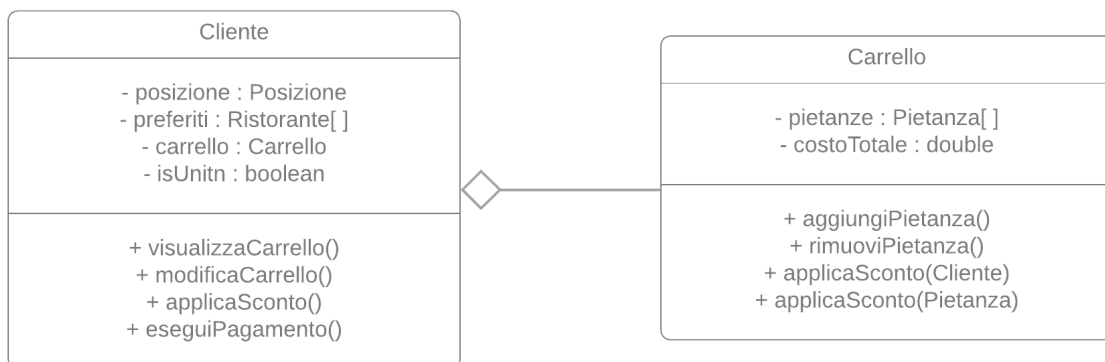


### 3.5 Applica sconto Cliente

La gestione dello sconto applicato al cliente avviene in due classi diverse: *Carrello* e *Cliente*. La classe **Carrello** tramite la funzione “*applicaSconto()*” con argomento un oggetto di classe **Cliente** verifica che il Cliente sia autenticato tramite dominio UniTN (*isUnitn*) e controlla che le pietanze nel suo carrello abbiano uno sconto applicabile. Alla fine attraverso una postcondizione si modifica l'attributo *costoTotale* della classe **Carrello** con la funzione “*modificaPrezzo()*” di **Pietanza** ([3.6](#)). Tutto ciò è tradotto dal seguente codice OCL:

```
context Carrello::applicaSconto(Cliente)
pre: (Utente.isUnitn = true) and
      (Utente.carrello.pietanze.isScontato = true)
post: Carrello.costoTotale = Carrello.costoTotale -
      Pietanza::modificaPrezzo()
```

Riferito alle seguenti classi:

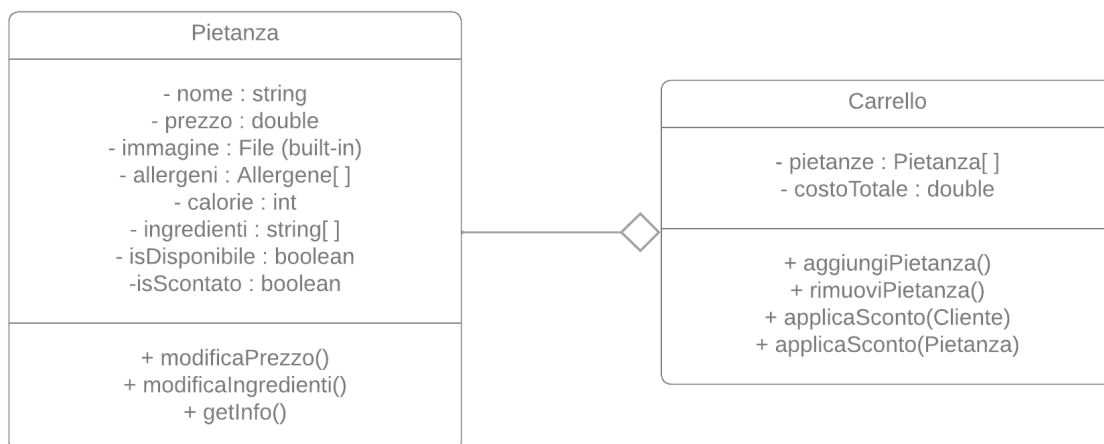


### 3.6 Applica sconto Pietanza

La gestione dello sconto applicato alla pietanza avviene in due classi diverse: *Carrello* e *Pietanza*. La classe **Carrello** tramite la funzione “*applicaSconto()*” con argomento un oggetto di classe **Pietanza** verifica che la pietanza sia disponibile e controlla che la stessa abbia uno sconto applicabile. Alla fine attraverso una postcondizione si modifica l’attributo *costoTotale* della classe **Carrello** con la funzione “*modificaPrezzo()*” di **Pietanza** che ritorna 0 nel caso in cui non ci siano sconti applicabili. Tutto ciò è tradotto dal seguente codice OCL:

```
context Carrello::applicaSconto(Pietanza)
pre: Pietanza.isScontato = true and Pietanza.isDisponibile = true
post: Carrello.costoTotale = Carrello.costoTotale -
Pietanza::modificaPrezzo()
```

Riferito alle seguenti classi:

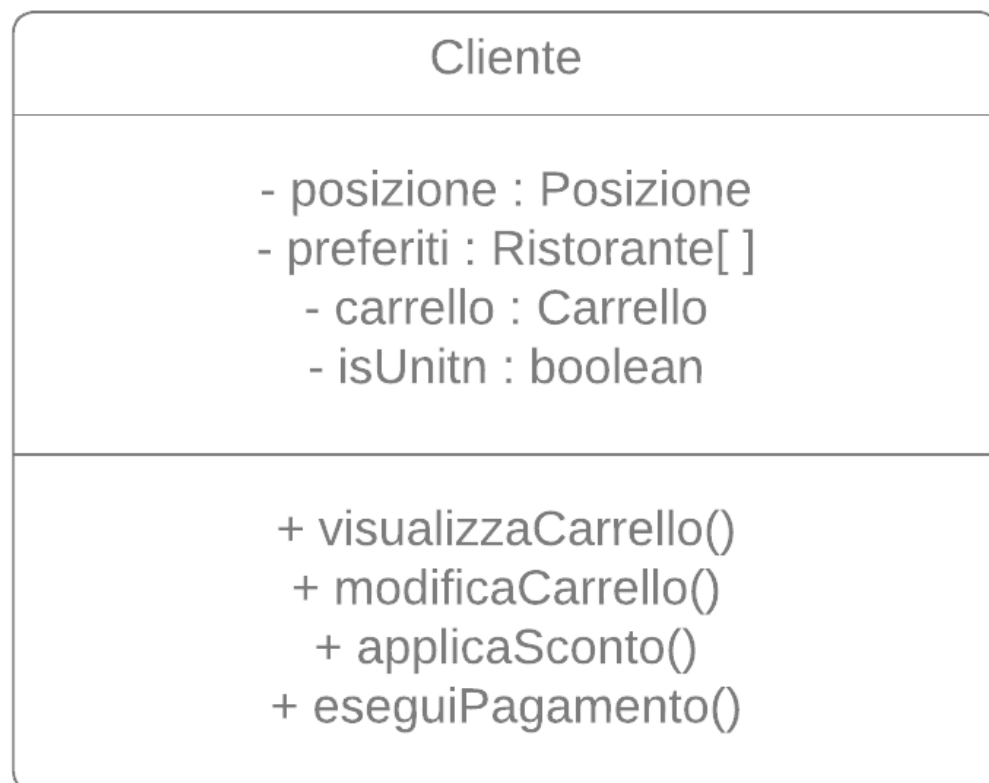


### 3.7 Esistenza Cliente

È presente una condizione che deve sempre essere verificata affinché la classe **Cliente** possa esistere, ovvero l'attributo posizione non deve essere vuoto. Ciò è espresso dal seguente codice OCL attraverso l'uso di una invariante:

```
context Cliente inv:  
posizione->notEmpty()
```

Riferito alla seguente classe:

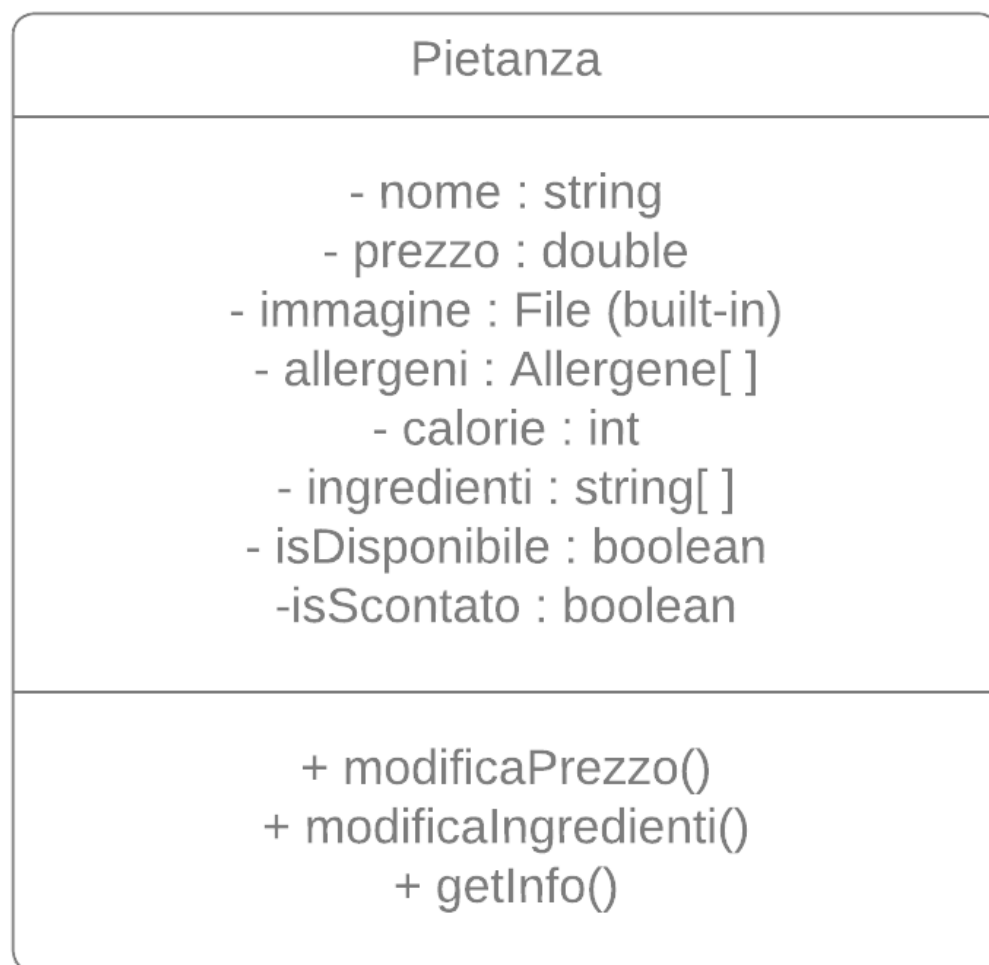


### 3.8 Esistenza Pietanza

Sono presenti due condizioni che devono sempre essere verificate affinché la classe **Pietanza** possa esistere, ovvero l'attributo prezzo deve essere maggiore di 0.0 e la lista degli ingredienti non deve essere vuota. Ciò è espresso dal seguente codice OCL attraverso l'uso di una invariante:

```
context Pietanza inv:  
prezzo > 0.0 and  
ingredienti->size() > 0
```

Riferito alla seguente classe:

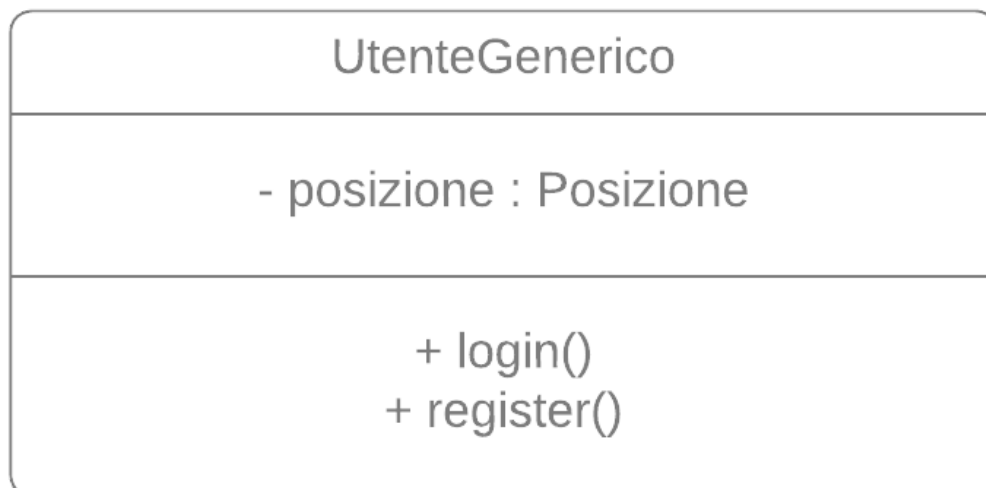


### 3.9 Esistenza Utente Generico

È presente una condizione che deve sempre essere verificata affinché la classe **UtenteGenerico** possa esistere, ovvero l'attributo **posizione** non deve essere vuoto. Ciò è espresso dal seguente codice OCL attraverso l'uso di una invariante:

```
context UtenteGenerico inv:  
posizione->notEmpty()
```

Riferito alla seguente classe:





## 4. DIAGRAMMA DELLE CLASSI CON CODICE OCL

Riportiamo infine il diagramma delle classi con tutte le classi fino ad ora presentate ed il codice OCL individuato.

