UNIVERSITY OF TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Progetto:

U-Sushi

Titolo del documento:

Documento di Architettura

INDICE

Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto U-Sushi 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.

1. Diagramma delle classi

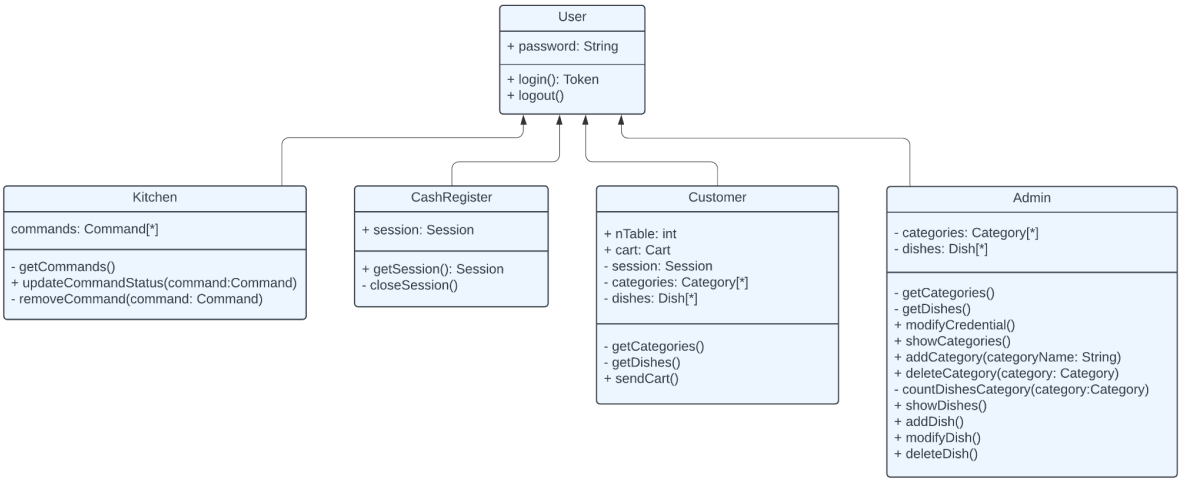
Nel presente capitolo vengono presentate le classi previste nell'ambito del progetto U-Sushi. Ogni componente presente nel diagramma dei componenti diventa una o più classi. Tutte le classi individuate sono caratterizzate da un nome, una lista di attributi che identificano i dati gestiti dalla classe 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. In questo processo si è proceduto anche nel massimizzare la coesione e minimizzare l'accoppiamento tra classi.

1.1. Utenti

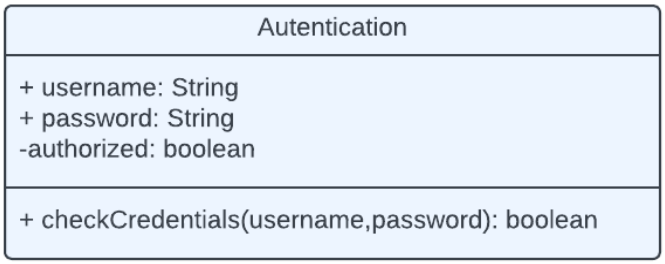
Analizzando il diagramma di contesto realizzato per il progetto U-Sushi si nota la presenza di **4 attori:**
Il "*customer*" è il cliente, ossia colui che, seduto al tavolo, effettua le ordinazioni dei vari piatti; il "*cash register*" è l'addetto che sta alla cassa e si occupa dei pagamenti da parte dei vari clienti una volta terminata la loro consumazione; "*kitchen*" è il colui che riceve le varie ordinazioni dei piatti e aggiorna il loro stato, permettendo ai clienti di sapere a che punto è la preparazione delle loro ordinazioni; infine l' "*admin*" è colui che si occupa di gestire le credenziali dei vari

utenti, al fine di garantire la sicurezza del sistema, oltre a tutte le informazioni contenute nel database.
Tutti questi attori hanno specifiche funzioni e attributi ma hanno anche alcune cose in comune.
Sono state quindi individuate le classi ***Customer***, ***CashRegister***, ***Kitchen*** e ***Admin*** con funzioni e attributi specifici e una classe *User* con funzioni e attributi in comune collegate tramite una generalizzazione.



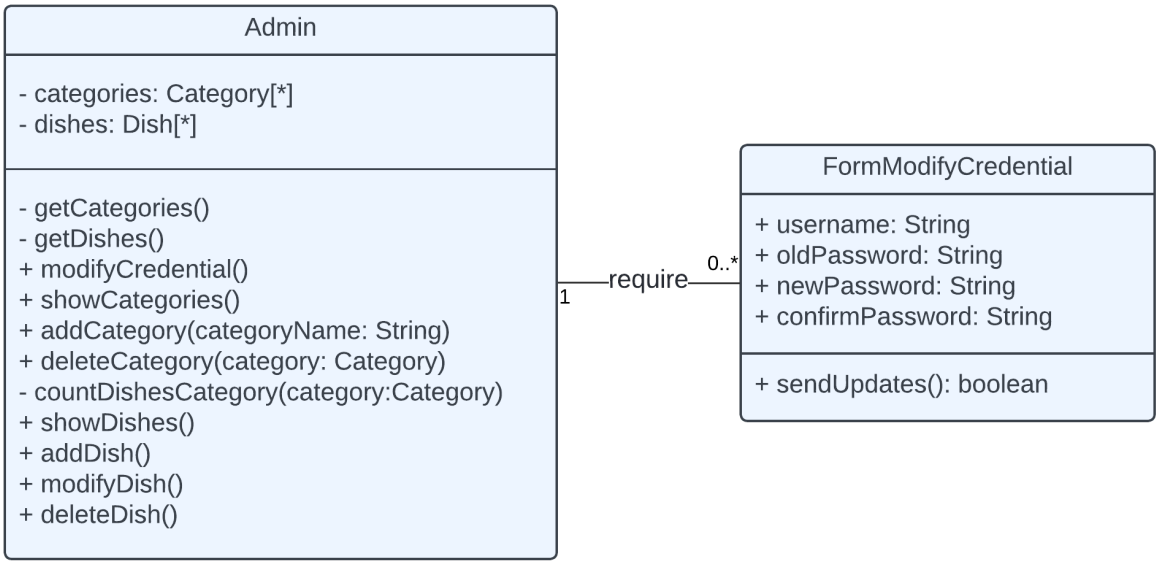
1.2. Gestione autenticazione

Il diagramma dei componenti analizzato presenta un *sistema subordinato* denominato "*Gestione accesso*". Questo elemento rappresenta il meccanismo di autenticazione degli utenti attraverso Auth0, un sistema esterno di gestione delle credenziali. È stata identificata una classe ***Autenticazione***, che si interfaccerà con questo sistema. Il software sviluppato non si occuperà della memorizzazione delle password degli utenti, della loro cifratura e sicurezza, ma solo della loro modifica (operazione servata all'admin).
Il software, al momento dell'autenticazione, tramite questa classe passerà i dati di autenticazione ad un sistema paritario esterno, ossia Auth0, che valuterà i dati e risponderà specificando se le credenziali inserite sono valide o meno. Di seguito il dettaglio di questa classe con i propri attributi e metodi.



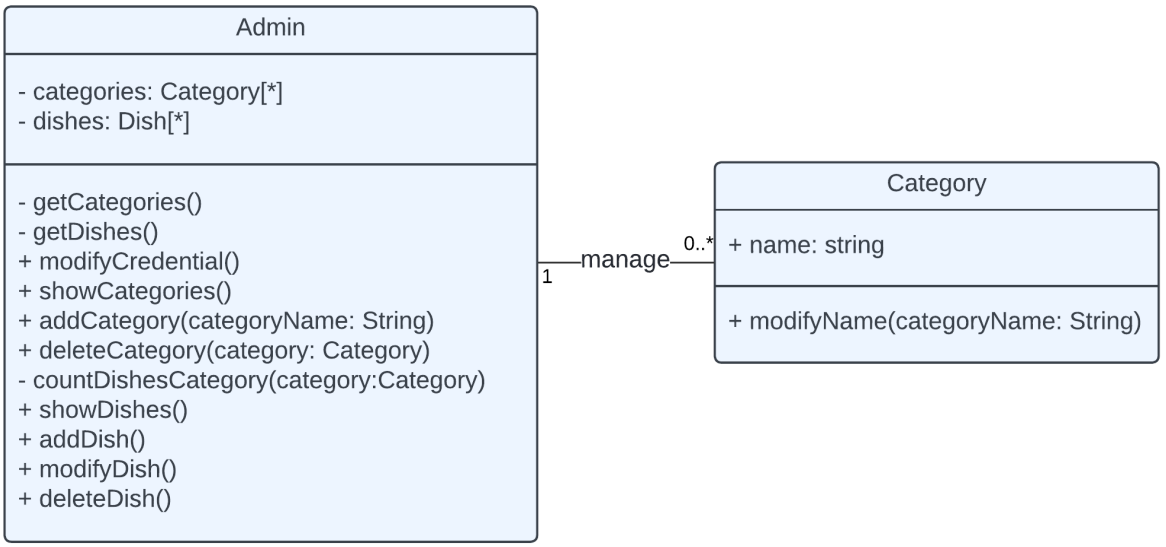
1.3. Modifica delle password

Analizzando il diagramma di contesto, l'operazione di modifica delle password è invece riservata all'admin, il quale dovrà provvedere a cambiarle periodicamente o all'occorrenza, al fine di garantire la sicurezza del sistema.
Per fare ciò, è stata individuata la necessità di creare un'apposita classe ***FormModifyCredential***, che permetta di eseguire in modo corretto e sicuro la modifica della password di un utente da parte dell'admin.
Di seguito il dettaglio di queste classi con i propri attributi e metodi.



1.4. Gestione delle Categorie

Analizzando il componente "*Pagina admin*" nel component diagram si nota che prevede diverse funzioni, tra cui la gestione delle categorie presenti nel database. Dal momento che i piatti sono raggruppati per categorie, si è scelto di creare un'apposita classe ***Category***, anziché rappresentarle come stringhe nei parametri del piatto (Dish).
Questa scelta permette una gestione più semplice delle categorie esistenti e dei piatti appartenenti ad esse, oltre una maggiore consistenza dei dati a livello database. Oltre a evitare errori di battitura da parte dell'admin all'inserimento e modifica dei piatti, che porterebbero ad una visualizzazione errata di alcuni piatti da parte del cliente, è possibile gestire in modo più semplice ed efficiente tutte le categorie esistenti, aggiungendone di nuove o eliminando quelle vecchie e limitando la scelta della categoria di un piatto a quelle create dall'admin. Inoltre, in questo modo, se si volesse modificare il nome di una categoria di piatti, sarebbe sufficiente una singola operazione, anziché dover modificare tutti i piatti di quella categoria, operazione che, oltre a poter essere molto onerosa a livello di tempo, potrebbe facilmente portare a errori di vario genere.

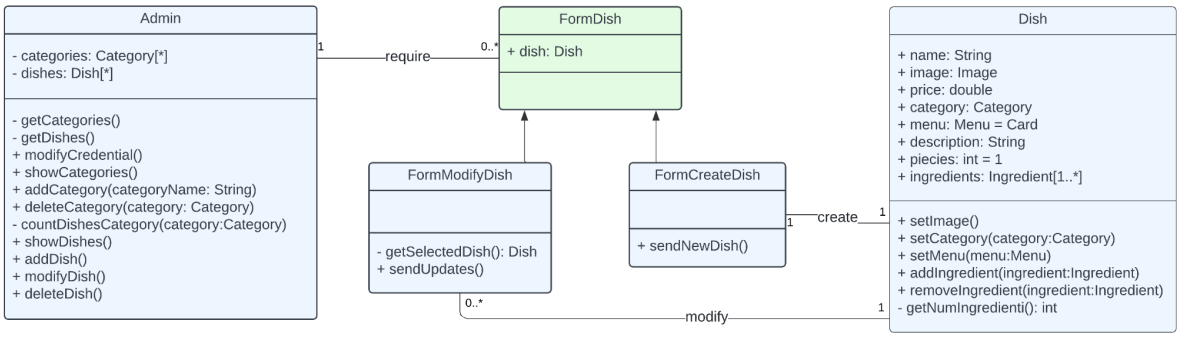


1.5. Piatti, menù e ingredienti

L'entità principale del sistema sono i piatti che il ristorante prepara. Viste le numerose caratteristiche dei piatti è stata identificata la necessità di una classe ***Dish***, la quale permetta una gestione ottimale di questa entità.
Dal momento che gli ingredienti che costituiscono un piatto saranno comuni in diversi piatti, è stata identificata anche una classe ***Ingredient***, la quale conterrà oltre al nome dell'ingrediente anche una lista di allergeni presenti in esso e quindi nel piatto che lo contiene. Quindi anche per gli allergeni, è stata individuata un'apposita classe ***Allergen***. Gli allergeni e gli ingredienti verranno definiti e memorizzati nel database dagli sviluppatori e non saranno modificabili da terze parti; questo al fine di evitare che alcuni allergeni non siano presenti in un ingrediente che li contiene (andando così a minacciare la salute dei clienti) o non siano proprio memorizzati nel database causa dimenticanza di chi di dovere.
Per quanto riguarda i menù, dal momento che ce ne sono solo 3, è stato scelto di rappresentarli tramite un enumeratore ***Menu*** e ogni piatto deve avere un menù di appartenenza. Inoltre è sufficiente memorizzare un singolo menù per piatto, dal momento che i menù hanno una struttura a cipolla, cioè ogni menù contiene, oltre ai propri, anche i piatti dei menù più interni: *Dinner* contiene tutti i piatti di *Lunch* e *Card* contiene tutti i piatti di *Dinner*.

1.6. Gestione dei piatti

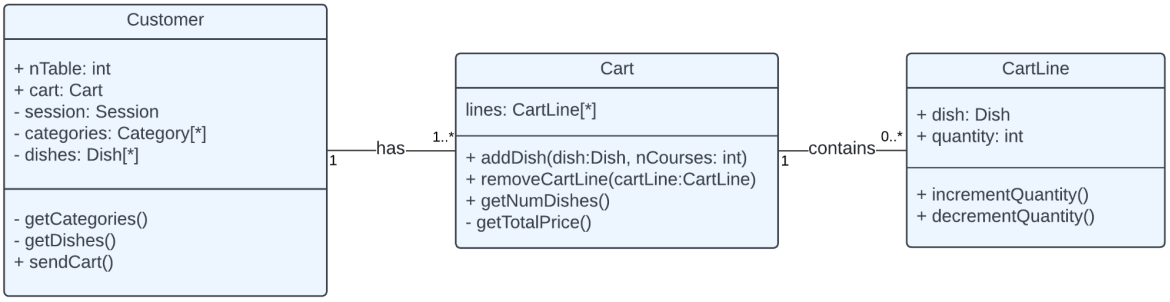
Analizzando nuovamente il componente “**Pagina admin**” si osserva che l’ultima delle funzioni assegnate all’admin è quella di gestire i piatti presenti nel database.
La creazione e modifica dei piatti avviene tramite dei form; entrambi hanno specifiche funzioni e attributi ma hanno anche molto in comune, infatti hanno la stessa struttura e gli stessi campi (corrispondenti agli attributi di un piatto).
Sono state quindi individuate due classi ***FormModifyDish*** e ***FormCreateDish*** con funzioni e attributi specifici e una classe ***FormDish*** con funzioni e attributi in comune collegate tramite una generalizzazione.
Invece, per quanto riguarda l'eliminazione di un piatto, dal momento che si tratta di una singola funzione e non prevede l’inserimento di dati aggiuntivi, si è scelto di implementare tale operazione come singola funzione.



1.7. Gestione del carrello

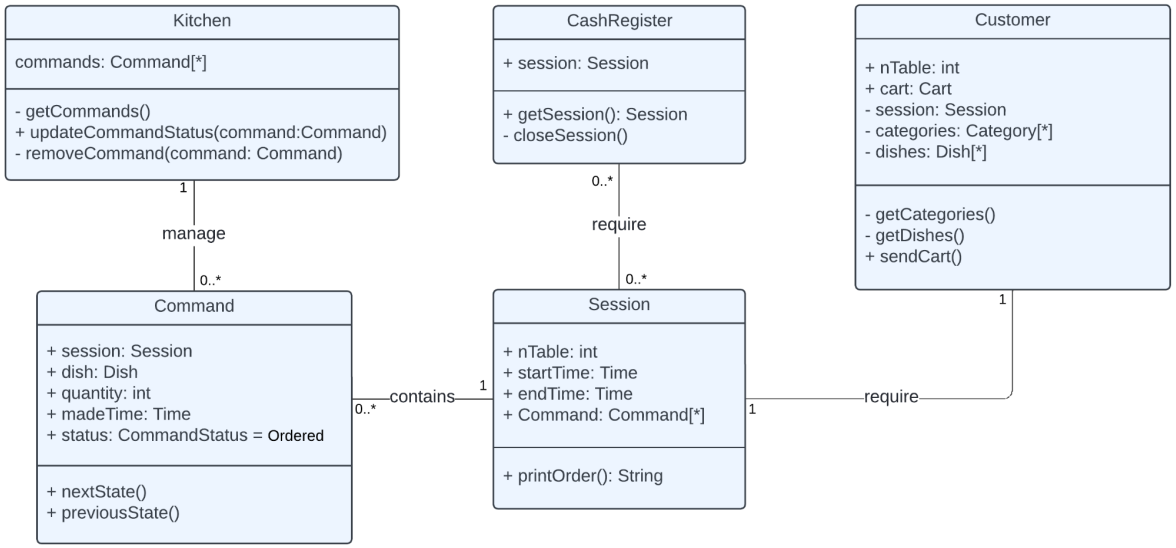
Il diagramma di contesto analizzato presenta un’entità *Carrello* che rappresenta il carrello di un utente, il quale potrà aggiungere o eliminare piatti o modificarne il numero di portate prima di inviare l’intera comanda alla cucina. Questa entità è

locale e permette all'utente di modificare la sua ordinazione in caso di ripensamenti, dal momento che dopo averla inviata questa è definitiva. Inoltre, dal momento che un utente può fare più ordinazioni durante la sua permanenza al tavolo, e quindi riempire diversi carrelli, è stata identificata la necessità di avere una apposita classe ***Cart***, che permetta di gestire in modo semplice ed efficiente i carrelli dei vari utenti, e una classe ***CartLine*** che funge da supporto alla classe precedente.



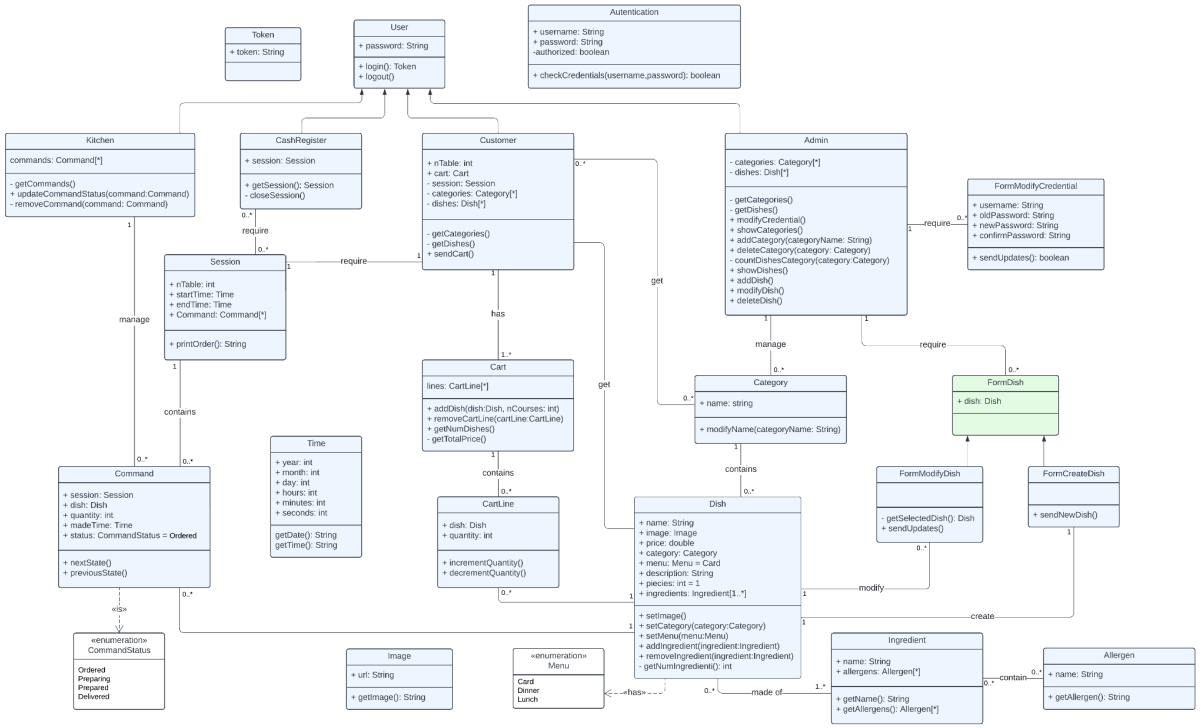
1.8. Invio delle comande e gestione degli ordini

A questo punto, abbiamo identificato due ulteriori necessità. La prima deriva dal fatto che un cliente, anche se può riempire più carrelli e quindi inviare diverse comande durante la sua permanenza al tavolo, alla fine del servizio deve essere associato ad un singolo ordine complessivo, comprendente tutti i piatti che ha ordinato nelle diverse comande. Inoltre, i piatti ordinati devono essere inviati anche alla cucina, la quale potrà modificarne lo stato di preparazione; se un utente ordina lo stesso piatto in carrelli differenti, quei piatti (con la rispettiva quantità) dovranno essere rappresentati come due oggetti differenti all'interno del sommario dell'ordine, dal momento che sicuramente in un determinato momento avranno stati di preparazione differenti.
Abbiamo quindi individuato la necessità di avere 2 ulteriori classi: ***Session*** che identifica la sessione dell'utente e contiene tutti i piatti da lui ordinati, oltre ad una data di inizio e fine sessione, che permette di controllare anche che non vengano ordinati altri piatti dopo la chiusura del conto da parte della cassa; ***Command***, che rappresenta il piatto ordinato da un utente in un carrello e che viene identificato dal codice della sessione e dal suo indice di linea nell'ordine.
La cucina (Kitchen) quindi, riceverà periodicamente le varie comande (Command) e li metterà in coda alle altre ordinazioni e potrà aggiornarne lo stato dei vari.
Per quanto riguarda la sessione del cliente (Session) potrà essere richiesta dal cliente stesso (Customer), per visualizzare uno storico dei piatti ordinati, o dalla cassa (CashRegister), per compilare il conto e poi chiudere l'ordine.
Di seguito il dettaglio di queste classi con i propri attributi e metodi.



1.9. Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi con tutte le classi fino ad ora presentate. Oltre alle classi già descritte, sono state inserite classi ausiliarie, ad esempio **`Time`**. Queste classi servono per descrivere eventuali tipi strutturati usati, ad esempio, negli attributi delle altre classi.

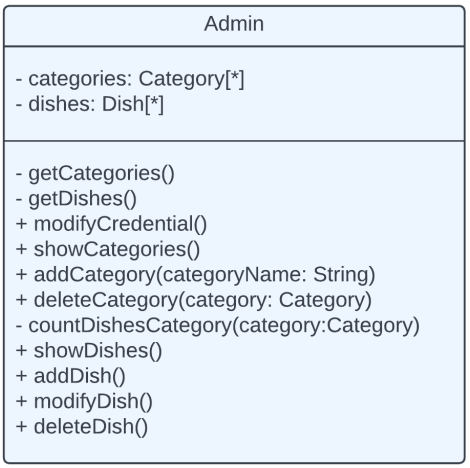


2. Codice in Object Constraint Language

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.

2.1. Eliminazione di una categoria

L'admin non può eliminare una categoria se all'interno di questa ci sono presenti dei piatti. Questa condizione su questa classe:

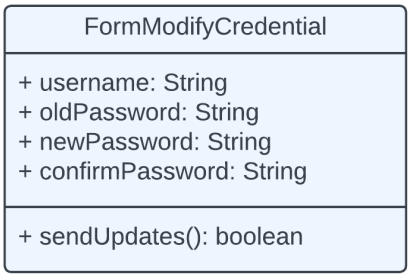


è espressa in OCL attraverso una precondizione con questo codice:

```
context Admin::deleteCategory(category:Category)
pre : countDishesCategory(category) = 0
```

2.2. Invio modifica delle credenziali

Dopo aver compilato la form per la modifica delle credenziali, l'admin potrà inviare la richiesta di modifica solo se i campi del form "newPassword" e "confirmPassword", nel quale si ripete la nuova password al fine di evitare possibili errori di battitura, sono uguali. Questa condizione su questa classe:

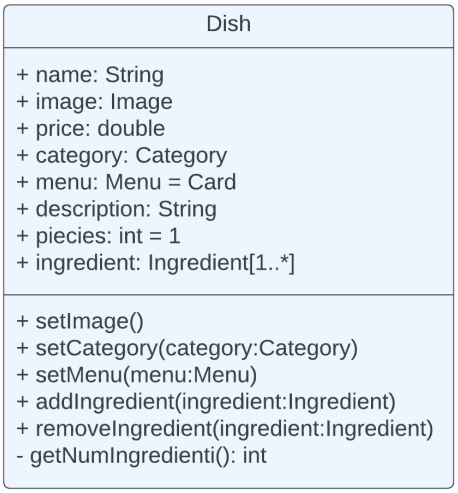


è espressa in OCL attraverso una precondizione con questo codice:

```
context FormModifyCredential::sendUpdates()
pre : self.newPassword = self.confirmPassword
```

2.3. Vincoli sui parametri del piatto

Ogni piatto deve sempre avere avere un nome, un'immagine, una categoria di appartenenza. Inoltre il prezzo di un piatto non deve mai avere valore negativo. Queste quattro condizioni su questa classe:



e sono espresse in OCL attraverso un'invariante con questo codice:

```
context Dish inv:
name != ""
image != null
category != null
price >= 0
```

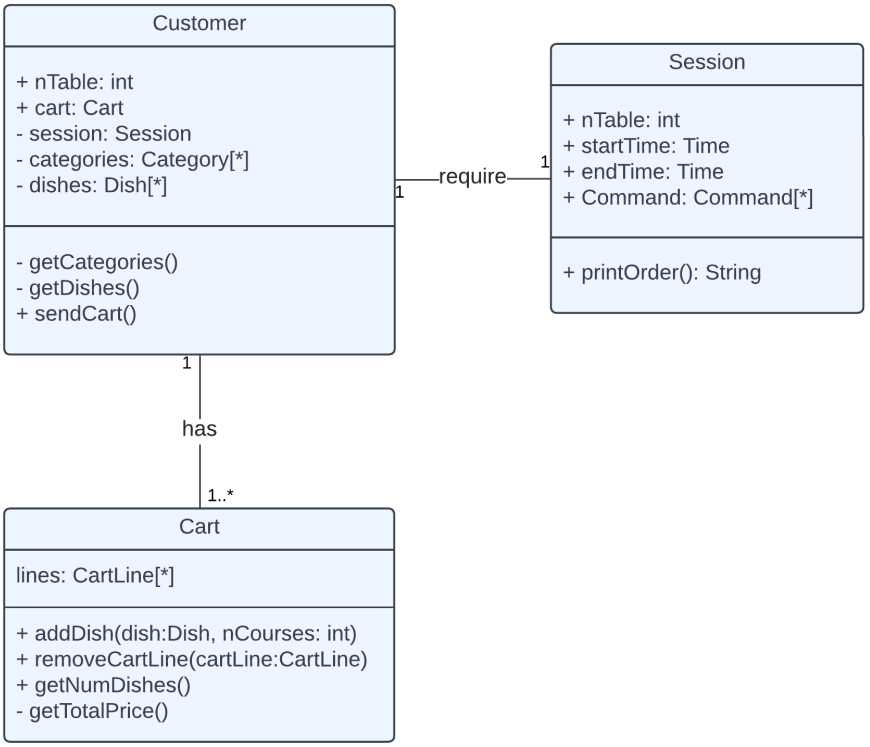
2.4. Un piatto non può essere fatto d'aria

Nella classe Dish sopra riportata è presente anche il metodo removeIngredient. Questo metodo può essere eseguito solo se il numero di ingredienti di cui è composto il piatto, presenti nel vettore ingredients, sono almeno 2, dato che ogni piatto deve essere composto da almeno un ingrediente. Il metodo getNumIngredienti restituisce il numero di elementi presenti nel vettore. Questa condizione è espressa in OCL attraverso una precondizione con questo codice:

```
context Dish::removeIngredient(ingredient:Ingredient)
pre : self.getNumIngredienti() >= 2
```

2.5. Invio del carrello di un cliente

Un cliente può inviare un carrello solo se questo contiene almeno un piatto, dal momento che non avrebbe senso inviare un'ordinazione vuota. Inoltre, la sessione del cliente non deve ancora essere stata chiusa, quindi deve avere l'orario di fine sessione non impostato. Questa condizione su queste classi:



è espressa in OCL attraverso una precondizione con questo codice:

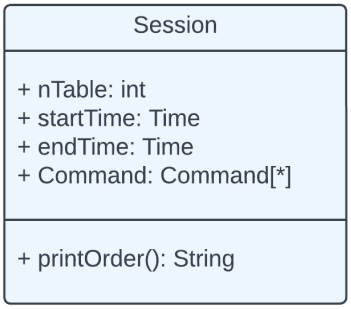
```
context Customer::sendCart()
pre : self.cart.getNumDishes() > 0
and session.endTime = null
```

Inoltre, una volta inviato il carrello, l'utente avrà a disposizione un nuovo carrello vuoto. Questa condizione è espressa in OCL attraverso una postcondizione con questo codice:

```
context Customer::sendCart()
post : self.cart.getNumDishes() = 0
```

2.6. Finestra temporale della sessione di un cliente

La sessione di un cliente deve avere un orario di fine sessione, se questo è impostato, maggiore del suo orario di inizio. Questa condizione su questa classe:

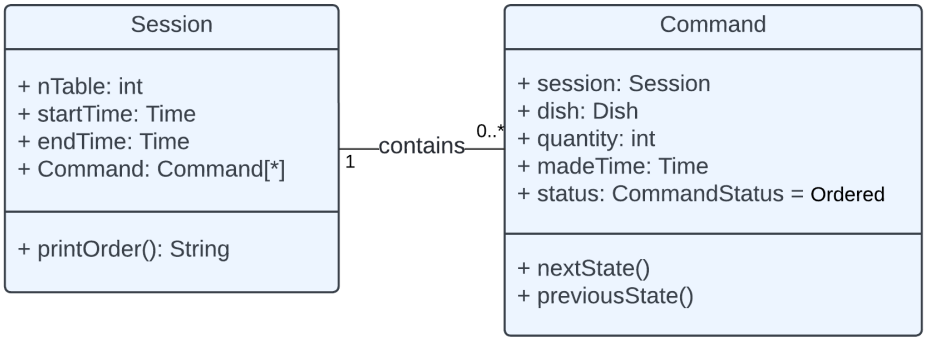


è espressa in OCL attraverso una precondizione con questo codice:

```
context Session inv:
(self.endTime = null) or (self.endTime  > self.startTime)
```

2.7. Orario delle comande

Dal momento che un utente può inviare nuove comande solo fino a quando la sua sessione è aperta, ogni comanda deve avere un orario di invio che sia compreso tra l'orario di inizio della sessione e l'orario di chiusura della stessa. Questa condizione su queste classi:

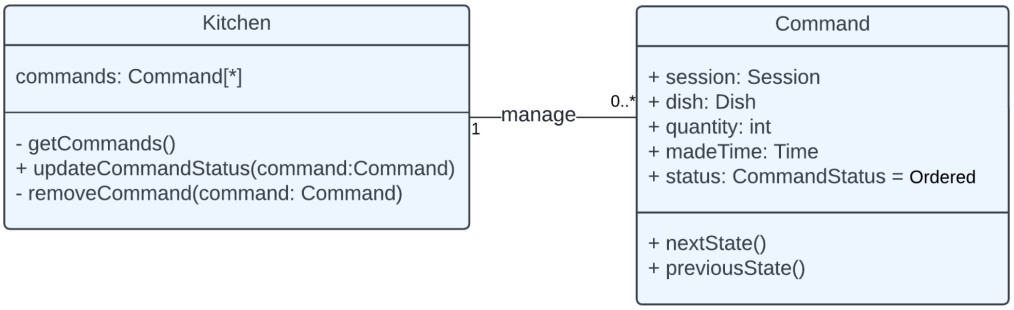


è espressa in OCL attraverso un'invariante con questo codice:

```
context Command inv:
(self.madeTime > self.session.startTime) and
(self.madeTime < self.session.endTime)
```

2.8. Rimozione comande dalla lista della cucina

Un addetto può eliminare un piatto dalla coda delle comande solo dopo che questo è stato servito. Quindi il piatto può essere rimosso solo se si trova nello stato dei “Delivered”. Questa condizione sulla classe:



è espressa in OCL attraverso una precondizione con questo codice:

```
context Kitchen::removeCommand(command: Command)
pre : command.status = Delivered
```

3. 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.