Progetto Gruppo T46

Chad Gym

Documento D3-T46

Documento di architettura

Membri: Andrea Perrone, Michael Mattiolo e Mario Vaccaro.

Sommario

0. Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto Chad Gym 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.

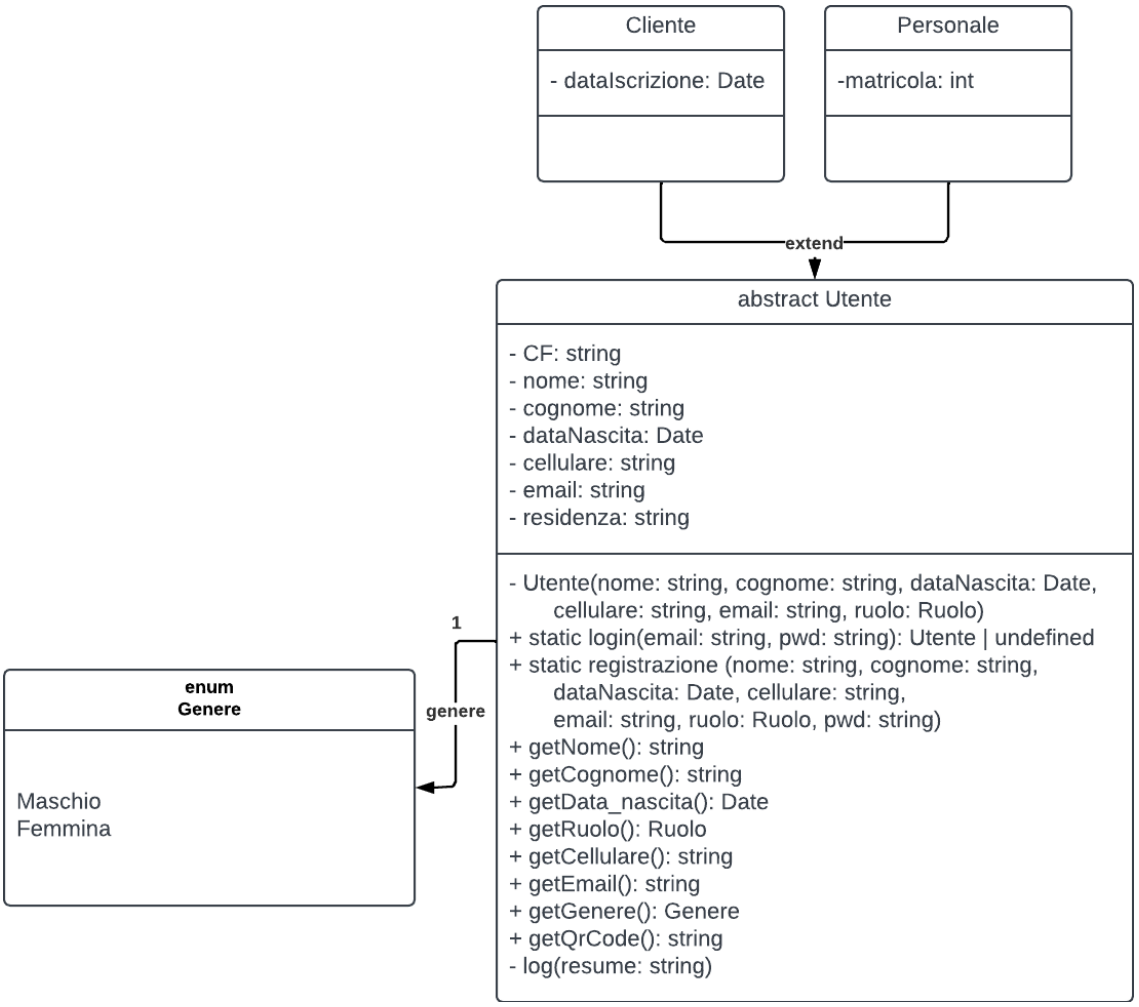
1. Diagramma delle classi

Utente:

Tutti gli utenti del sistema sono rappresentati da una classe astratta chiamata "Utente" che conterrà come attributi privati i dati in comune delle varie tipologie di utenti che accedono alla palestra. La classe in questione conterrà anche dei metodi pubblici per ottenere informazioni sugli attributi delle varie istanze di "Utente". È presente anche un metodo costruttore privato per creare le varie istanze della classe.
Per istanziare la classe si usa il metodo login che, in base alle credenziali di accesso inserite, restituisce un oggetto di tipo Cliente o Personale.
Se l'utente non è presente nel sistema, allora si ricorre al metodo registrazione che, presi come parametri i dati dell'utente, ne crea una tupla nel database.
È infine presente un enum Genere che andrà a contenere i possibili generi attribuibili ad un utente.
La classe Utente verrà estesa da:

1. **Cliente**:
   Estende la classe aggiungendo un proprio attributo riguardante la data di iscrizione;

2. **Personale**:
   Estende la classe aggiungendo un attributo riguardante la matricola.

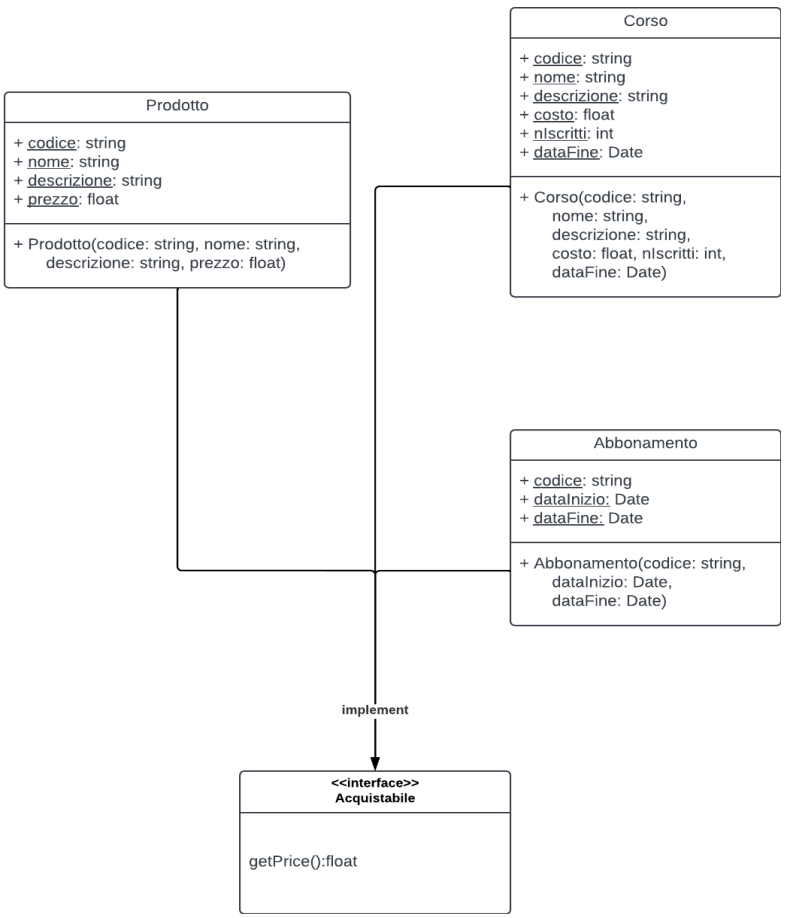


Acquistabile:

L'interfaccia "Acquistabile" viene implementata dalle classi "Prodotto", "Corso" e "Abbonamento" che ricevono successivamente gli attributi e i costruttori a loro associati e dovranno successivamente contenere il metodo getPrice che permette agli elementi di esprimere il prezzo una volta all'interno del carrello.

1. **Prodotto**:
   La classe prodotto contiene gli attributi finali e pubblici relativi alle informazioni del prodotto.
   Gli attributi sono:
   a. codice: string
   b. nome: string
   c. descrizione: string
   d. prezzo: float

2. **Corso**:
   La classe corso contiene degli attributi finali pubblici che ne specificano le caratteristiche affinché ogni cliente possa identificare il corso che più preferisce tramite le caratteristiche specificate.
   Gli attributi sono:
   a. codice: string
   b. nome: string
   c. descrizione: string
   d. costo: float
   e. nIscritti: int
   f. dataFine: Date
3. **Abbonamento:**
   La classe abbonamento è identificata dai suoi attributi finali pubblici che ne identificano le caratteristiche.
   Gli attributi sono:
   a. codice: string
   b. dataInizio: Date
   c. dataFine: Date



Shop:

La classe shop è implementata una volta per ogni uso da parte di un utente e si occupa di gestire il carrello, la lista degli articoli acquistabili, i metodi di pagamento registrati e la procedura di pagamento.
Gli attributi presenti sono:

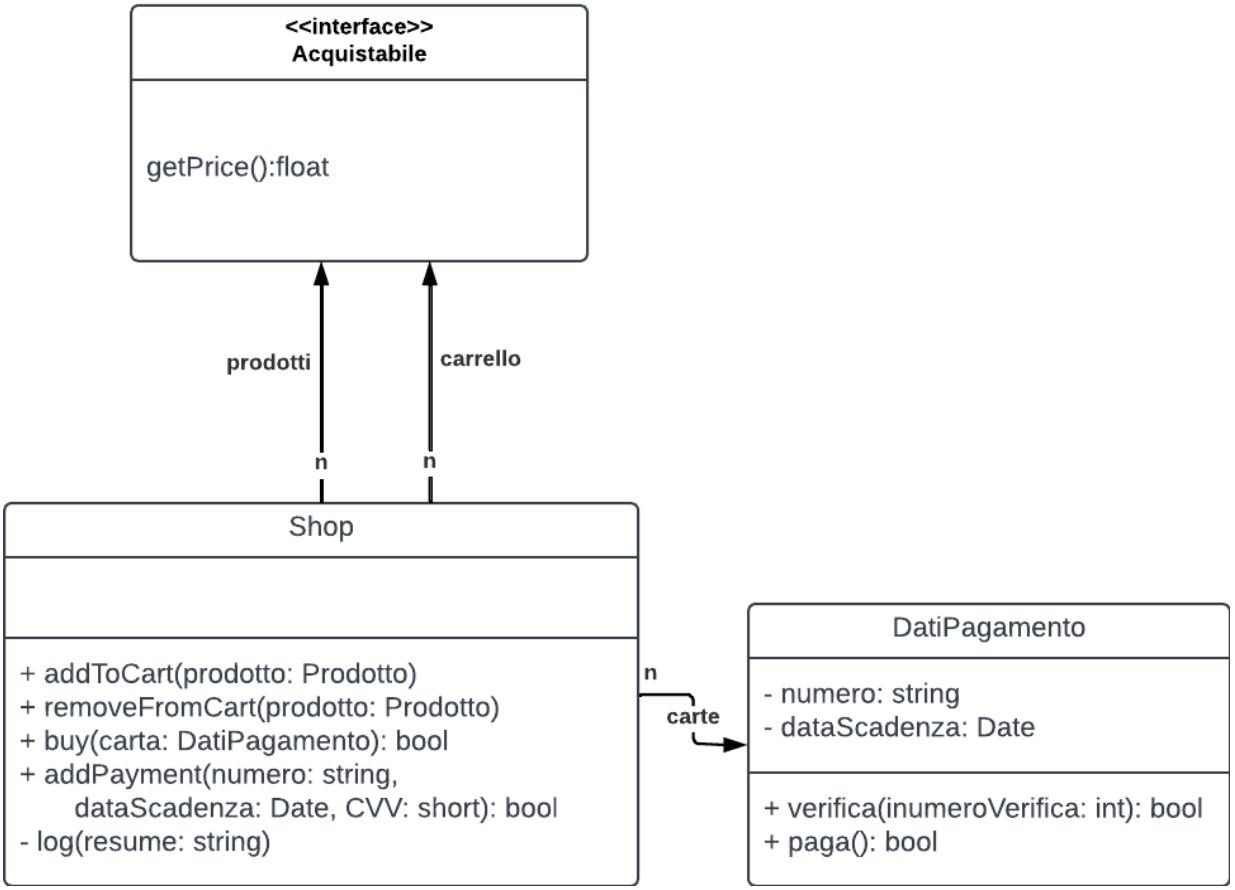
a) prodotti: Acquistabile[]
   La lista dei elementi visualizzabili e acquistabili tramite lo shop;
b) carrello: Acquistabile[]
   La lista degli elementi aggiunti al carrello dall'utente e pronti ad essere acquistati;
c) carte: DatiPagamento[]
   La lista dei metodi di pagamento registrati all'account dell'utente autenticato

Dati pagamento
La classe dati pagamento contiene i dati relativi ad una carta di pagamento, quali:

a) numero: string
b) dataScadenza: Date

Si è deciso di non salvare il CVV della carta per garantire la sicurezza e la riservatezza dei dati sensibili dell'utente.
I metodi implementati sono invece:

a) verifica(numeroVerifica: int): bool necessario a verificare la carta prima dell'uso per assicurarsi che l'acquirente sia l'effettivo proprietario.
   Il valore restituito è l'esito della verifica;
b) paga(): bool necessario ad utilizzare la carta per il pagamento. Questo metodo userà il metodo verifica per confermare l'uso della carta e successivamente si interfaccerà al sistema bancario per effettuare il pagamento.
   Il valore restituito è l'esito del pagamento che sarà **true** se il pagamento si conclude con successo e **false** se il pagamento non si conclude per qualsiasi motivo come carta bloccata, credito insufficiente e verifica della carta invalida.



Classi statiche:

Il diagramma UML contiene 3 classi statiche utilizzate da sottosistemi singoli o per il quale non serve avere un maggior numero di istanze.
Le classi statiche sono:

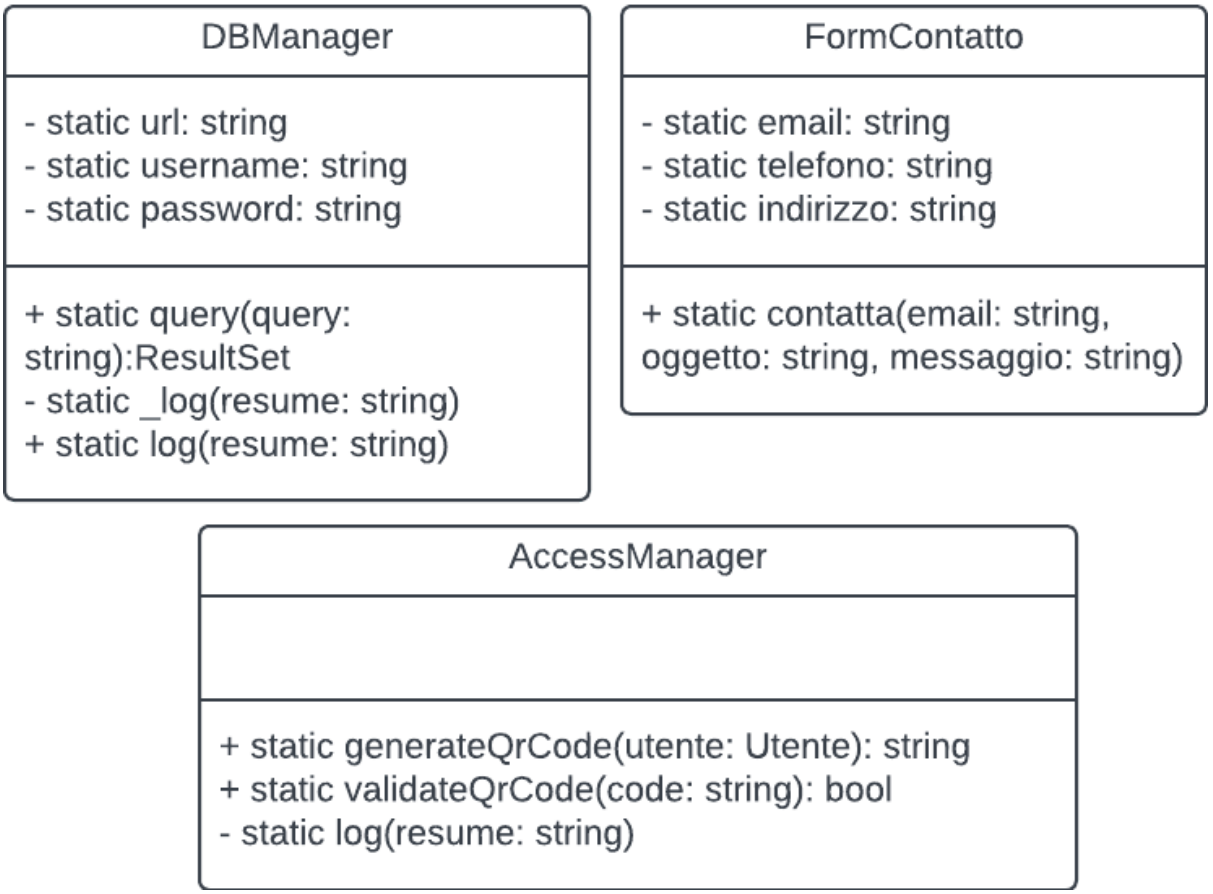
AccessManager:

La classe AccessManager, a differenza della classe Utente la quale già presenta i metodi di login e registrazione, è la classe usata dal sistema di ingressi alla struttura per generare i codici QR di accesso e verificarli.
Presenta quindi entrambi i metodi appena descritti oltre log utilizzato per registrare le operazioni effettuate nel database.

FormContatto:

Il FormContatto è la classe che contiene i dati che saranno successivamente mostrati nel sito come email, telefono e indirizzo.
Presenta inoltre un metodo statico “contatta”, che permette l’uso della funzione di contatto presente nella stessa pagina del sito.
Il metodo contatta richiede email, oggetto e corpo del messaggio come parametri, e permetterà la comunicazione con la palestra.

DBManager:
La classe DBManager è la classe utilizzata dal sistema intero per comunicare con il database, inviando le query di richiesta dei vari dati.
Gli attributi privati della classe sono url, username e password utilizzate dalla classe per connettersi al DBMS.
La classe presenta anche tre metodi.
Il primo di questi è il metodo query che presa la stringa query in input la esegue, e ritorna i risultati di questa se presenti.
Gli ultimi due metodi presenti sono log e "_log", e sono rispettivamente il metodo pubblico usato dalle altre classi per scrivere i log all'interno del database, e il metodo usato dalla classe stessa per scrivere i log delle query effettuate.



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 Ingresso nella palestra

Nella classe `Utente` è presente il metodo `getQrCode()::string`. Per permettere l'ingresso nell'edificio controlliamo quindi che il qr code generato dall'`AccessManager` sia lo stesso e ne controlliamo la validità. Queste condizioni sono espresse in OCL attraverso una precondizione con questo codice:

```
context Utente::getQrCode()::string
(Utente.getQrCode() == AccessManager.generateQrCode(Utente)
AND
(AccessManager.validateQrCode(Utente.getQrCode()) == TRUE
```

2.2 Inizio nuovo abbonamento

Nella classe `Abbonamento` è presente il costruttore `Abbonamento(string, Date, Date)`. Dove la stringa rappresenta il codice e le due date rispettivamente quella di inizio e quella di fine dell'iscrizione. Alla registrazione controlleremo la data di inizio dell'abbonamento che coincida a quella attuale. Questa condizione è espressa in OCL attraverso una postcondizione con questo codice.

```
context Abbonamento::Abbonamento(string,
Date,Date)
post: Abbonamento.dataInizio=self.data_attuale
```

2.3 Verifica utente per pagamenti

Nella classe `DatiPagamento` è presente il metodo `verifica(int)`. Un utente per effettuare un qualsiasi pagamente sarà richiesto che sia loggato, per questo controlleremo che lo username e la password siano presenti. Queste condizioni sono espresse in OCL attraverso una precondizione con questo codice.

```
context DatiPagamento::verifica(int)
pre: (DBmanager.username!=NULL) AND
(DBmanager.password!=NULL)
```

Nella classe `DatiPagamento` è presente il metodo `paga()`. Questo presuppone che la verifica del codice per il pagamento sia approvata. Questa condiziona è espressa in OCL attraverso una precondizione con questo codice.

```
context DatiPagamento::paga()
pre: verifica(int) == true;
```

2.4 Verifica date Abbonamento

Nella classe `Abbonamento` sono presenti due attributi per identificare la data di inizio (`Abbonamento.dataInizio`) e di fine dell'abbonamento (`Abbonamento.dataFine`). Dovranno quindi essere controllate le date in modo che la data di fine abbonamento sia successiva a quella di inizio.

```
context Abbonamento
inv: Abbonamento.dataFine>Abbonamento.dataInizio
```

2.5 Verifica data iscrizione

Nella classe `Cliente` è presente la data d'iscrizione (`Cliente.dataIscrizione`). La data attuale dovrà per forza essere uguale o successiva a quella d'iscrizione. Questa condizione viene espressa in OCL attraverso questo codice.

```
context Cliente
inv: Cliente.dataIscrizione <= Date.now();
```

2.6 Controlli sui prodotti

Nella classe `Prodotto` tutti gli attributi (Prodotto.codice, Prodotto.nome, Prodotto.descrizione) non dovranno essere vuoti o nulli per far si che esista tale prodotto e il Prodotto.prezzo deve essere superiore a 0.00 . Tutti questi controlli vengono espressi in OCL attraverso questo codice.

```
context Prodotto
inv: !codice.isNullOrEmpty() AND
!nome.isNullOrEmpty() AND
!descrizione.isNullOrEmpty() AND
prezzo>=0.00
```

2.7 Controllo prezzo prodotto/abbonamento/corso

E' presente l'interfaccia `Acquistabile` che possiede il metodo `getPrice()` ritornante il prezzo di uno dei servizi offerti dalla palestra. L'output di questo metodo non dovrà mai essere un numero negativo e questo controllo viene fatto tramite questa espressione in OCL.

```
context Shop::getPrice()
post: return >= 0.00
```

2.8 Controlli sull'acquisto

La classe `Shop` possiede un metodo per rimuovere un acquistabile del carrello (`removeFromCart`). Prima di fare ciò, verrà effettuato un controllo per assicurarsi che nel carrello ci sia effettivamente qualcosa e che quindi esista.

Esiste inoltre un metodo log privato il quale prima dell'uso controlla che la stringa di log da inserire sia valida e che non sia quindi una stringa vuota, o riconducibile ad essa.

```
context Shop::log(string)
pre: !resume.isNullOrEmpty()

context Shop::removeFromCart(Prodotto)
pre: carrello.exists(prodotto)
```

2.9 Controllo Form Contatto

La classe `FormContatto` permette di mettersi in contatto direttamente con la palestra per ricevere informazioni di qualunque tipo. Per fare ciò basterà compilare i campi del Form (`email, oggetto, messaggio`) con le informazioni richieste. Verrà effettuato un controllo per assicurarsi che i campi del form siamo tutti compilati e non vuoti.

```
context FormContatto::contatta(string, string, string)
pre: !messaggio.isNullOrEmpty() AND
!oggetto.isNullOrEmpty() AND
!email.isNullOrEmpty()
```

2.10 Controlli sulle caratteristiche dell'Utente

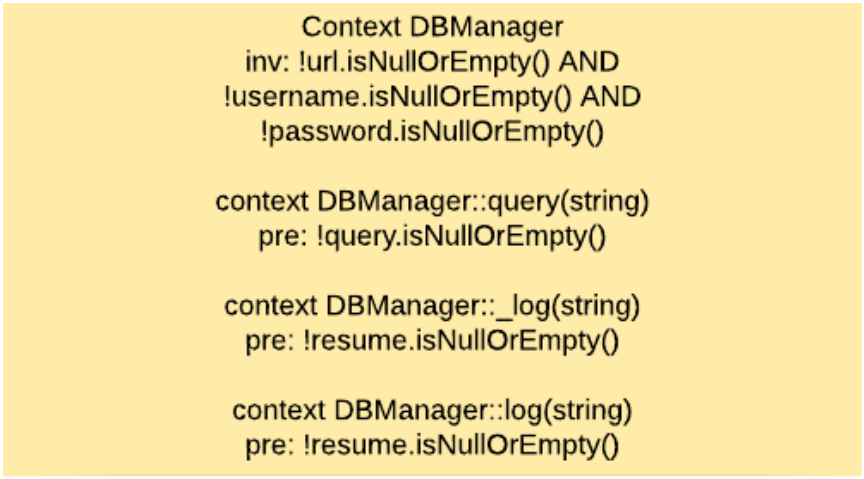
La classe `Utente` contiene i dati principali degli utenti che si registrano nel sistema. Verranno effettuati dei controlli sugli attributi in modo da evitare che qualche dato non sia presente. Saranno fatti inoltre dei controlli sui metodi per evitare anomalie nella restituzione delle informazioni richieste.

Esiste inoltre un metodo log privato il quale prima dell'uso controlla che la stringa di log da inserire sia valida e che non sia quindi una stringa vuota, o riconducibile ad essa.

```
context Utente
inv: !CF.isNullOrEmpty() AND
!nome.isNUllOrEmpty() AND
!cognome.isNullOrEmpty() AND
!cellulare.isNullOrEmpty() AND
!email.isNullOrEmpty() AND
!residenza.isNullOrEmpty() AND
dataNascita < Date.now()

context Utente::getNome()
post: !return.isNullOrEmpty()

context Utente::getCognome()
post: !return.isNullOrEmpty()

context Utente::getDataNascita()
post: return < Date.now()

context Utente::getCellulare()
post: !return.isNullOrEmpty()

context Utente::getEmail()
post: !return.isNullOrEmpty()

context Utente::getResidenza()
post: !return.isNullOrEmpty()

context Utente :: getQrCode()::string
post: (Utente.getQrCode() == AccessManager.generateQrCode(Utente) AND (AccessManager.validateQrCode(Utente.getQrCode()) == TRUE)

context Utente::log(string)
pre: !resume.isNullOrEmpty()
```

2.11 Controlli sul DBManager

La classe `DBManager` contiene i metodi principali per eseguire le query tra cui anche la creazione dei log.
Questa classe dovrà quindi essere regolata e ben controllata essendo uno dei nodi centrali dell'intero sistema.
Dovranno essere effettuati controlli per assicurarsi che gli attributi username, password e url non siano mai vuoti o riconducibili a tali, per permettere la connessione al DBMS.
Verranno inoltre effettuati controlli sui metodi per accertarsi che i parametri non siano stringhe vuote.
I metodi controllati saranno quindi log, "_log" e query.



2.12 Controlli sull'AccessManager

La classe `AccessManager` è un altro dei nodi principali del sistema in quando si occupa di permettere o negare gli accessi ai clienti della palestra.
Sarà necessario che la classe esegua dei log corretti per avere un maggiore controllo sulla situazione del sistema, di conseguenza sarà controllato che la stringa resume passata come parametro al metodo log non sia vuota o riconducibile a tale.

3. Diagramma delle classi con codice OCL

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