



**UNIVERSITY
OF TRENTO**

Dipartimento di Ingegneria e Scienza dell'Informazione
CORSO DI INGEGNERIA DEL SOFTWARE



Ypsilon 2
your own stats

Documento di progetto: Documento di architettura

Gruppo: T05



Sommario

1. Scopo del documento

2. Diagramma delle classi

2.1. Tipi di dato

2.1.1. Q&A

2.1.2. Lingua

2.2. Classi

2.2.1 Gestione autenticazione

2.2.1. Utente

2.2.2. Notifiche

2.2.3. Commenti

2.2.4. Corsi

2.2.5. Appello

2.2.6. Q&A

3. Diagramma delle classi complessivo

4. Codice in Object Constraint Language

4.1. Valutazione commento

4.2. Valutazione QandA

4.3. Valutazione corso

4.4. Corso favorito

4.5. Autenticazione

4.6. Calcolo statistiche

4.7 Collega domanda

4.8 Visualizzazione notifiche

4.9 Univocità appello



1. Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto **Ypsilon 2** attraverso l'utilizzo del diagramma delle classi in Unified Modeling Language (UML) e codice in Object Constraint Language (OCL).

Grazie ad esso sarà quindi possibile comprendere le regole che definiscono il diagramma UML e la logica del comportamento del software.

2. Diagramma delle classi

Nel presente capitolo vengono presentate e descritte le varie classi previste nel progetto **Ypsilon 2**. In particolare, ogni componente presente nel diagramma dei componenti ed ogni attore e sistema esterno presente nel diagramma di contesto verranno rappresentati attraverso l'utilizzo di una o più classi, con la possibilità di essere associate tra di loro. In questo caso verranno fornite informazioni aggiuntive al fine di chiarire al meglio come le classi si relazionano tra loro.

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.

Vengono riportate di seguito le classi individuate a partire dal diagramma di contesto e dal diagramma dei componenti. Ognuna di queste classi viene descritta in base alle funzioni e agli attributi relativi ad essa; in alcuni casi la descrizione viene effettuata tramite codice OCL.

2.1 Tipi di dato


Di seguito una breve descrizione dei tipi di dato utilizzati dal sistema.

2.1.1 Q&A:

Nella pagina di un corso troviamo la sezione Question and Answer relativa, che permette agli utenti di inviare nuove domande ad un docente oppure visualizzare quelle che hanno già ricevuto una risposta.

Per questo motivo è stato creato un tipo di dato **Q&A** caratterizzato da due attributi.

In particolare individuiamo gli attributi *domanda* e *risposta*. Questo tipo di dato viene utilizzato dal sistema per identificare più dinamicamente una certa domanda e la sua relativa risposta.


	<<Enumeration>> QaType
	1: Domanda 2: Risposta

2.1.2. Lingua:

In tutte le pagine principali l'utente ha la possibilità di cambiare la lingua di sistema.

Per questo motivo è stato creato un tipo di dato **Lingua** caratterizzato da due attributi.

In particolare definiamo le due lingue che sono state momentaneamente implementate nel sistema, ovvero **italiano** ed **inglese**.

	<<Enumeration>> Lingua
	1: Italiano 2: Inglese

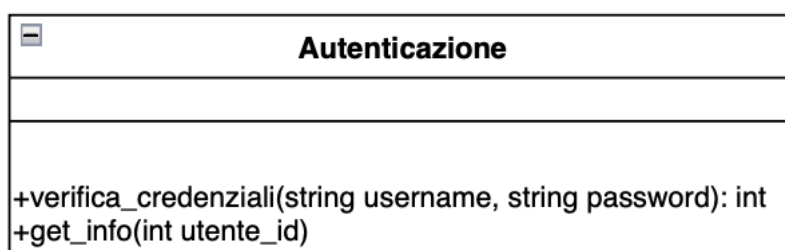
2.2. Classi

2.2.1. Gestione autenticazione

Il diagramma di contesto presenta il sistema esterno **sistema credenziali universitarie**. Questo elemento permette e gestisce l'autenticazione di utenti all'interno della piattaforma. Viene così identificata una classe **Autenticazione**. Il software sviluppato per il progetto Ypsilon3 non gestirà né memorizzerà username e password inseriti dall'utente, ma attraverso questa classe, verranno passati solo i dati di autenticazione ad un sistema peer esterno, il Sistema delle Credenziali di Ateneo.

Quest'ultimo si occuperà di valutare dati e rispondere specificando se le credenziali inserite sono valide.

La classe viene dettagliatamente descritta nell'area sottostante.



Descrizione dei metodi implementati dalla classe **Autenticazione**:

- *verifica_credenziali(username, password)*: richiama l'API di UNITN per eseguire l'accesso con i dati inseriti
- *get_info()*: aggiorna le informazioni dell'utente prendendo le informazioni dalle API di UNITN (ruolo, corso d'insegnamento, data immatricolazione....)

2.2.2. Utente

Il diagramma di contesto presenta due tipologie differenti di utenti: utente docente e utente studente, i quali possono essere definiti anche come utenti autenticati.

Viene quindi definita una super-classe **UtenteAutenticato** e due classi **Docente** e **Studente**, le quali ereditano tutti gli attributi e metodi dalla super-classe sopra citata.

La super-classe **UtenteAutenticato** presenta i seguenti attributi:

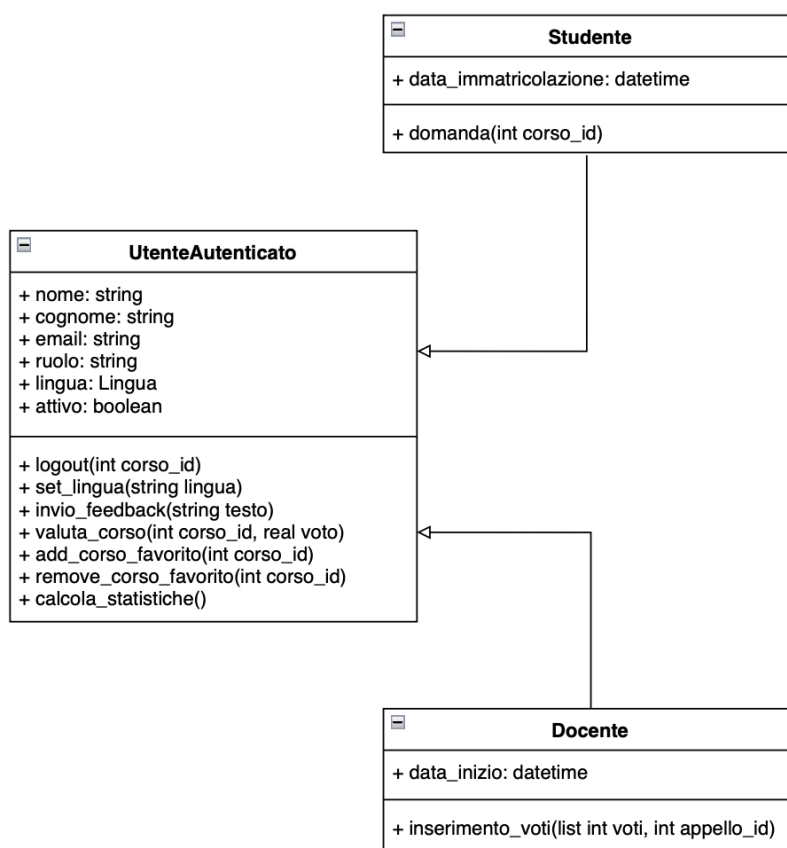
- *Nome, Cognome, Email*: sono i dati personali dell'utente, definiti automaticamente grazie al sistema esterno **sistema di credenziali universitarie**.
- *Ruolo*: definisce il ruolo dell'utente autenticato in utente docente o utente studente, esso viene scelto automaticamente grazie al sistema esterno **sistema di credenziali universitarie**.
- *Lingua*: definisce l'ultima lingua utilizzata dall'utente per visualizzare l'interfaccia.
- *Attivo*: utente 'vecchio' non più utilizzato, ma ne viene tenuta traccia per mantenere i vari collegamenti.

La classe **Docente**, oltre ad ereditare gli attributi della super-classe **UtenteAutenticato**, presenta i seguenti attributi:

- *Data_inizio*: definisce la data assunzione del docente

La classe **Studente**, oltre ad ereditare gli attributi della super-classe **UtenteAutenticato**, presenta i seguenti attributi:

- *Data_immatricolazione*: definisce la data di immatricolazione dell'utente studente, inserita automaticamente grazie al sistema esterno **sistema di credenziali universitarie**.



Descrizione dei metodi implementati dalla super-classe **UtenteAutenticato**:

- *logout(int corso_id)*: permette di effettuare il logout dal sito.
- *set_lingua(string lingua)*: permette di settare/modificare la lingua presente all'interno del sito.
- *invio_feedback(string testo)*: permette di inviare dei feedback, tramite l'utilizzo del sistema esterno **posta elettronica**.
- *valuta_corso(int corso_id, real voto)*: permette di valutare un corso.
- *add_corso_favorito(int corso_id)*: aggiunge il corso come favorito per quell'utente.
- *remove_corso_favorito(int corso_id)*: rimuove il corso come favorito per quell'utente.
- *calcola_statistiche()*: calcola le statistiche dello studente da mostrare nella pagina visualizza profilo.

Descrizione dei metodi implementati dalla classe **Docente**:

- *inserimento_voti(list int voti, int appello_id)*: gestisce l'inserimento voti da parte del docente per un determinato appello di un corso.

Descrizione dei metodi implementati dalla classe **Studente**:

- *domanda(int cors_id)*: gestisce la possibilità di inviare una domanda da parte di uno studente su un determinato corso, nella zona Q&A.

2.2.3. Notifiche

Viene definita una classe **Notifica** con il fine di descrivere gli attributi e i metodi relativi.

La classe **Notifica** presenta i seguenti attributi:

- **Testo:string** : definisce il contenuto effettivo della notifica.
- **Visualizzata:boolean** : serve per tenere traccia se la notifica è stata già visualizzata o meno.
- **Utente_id**: definisce il codice identificativo associato a ciascun utente autenticato.
- **Corso_id**: definisce il numero identificativo associato a ciascun corso.
- **Commento_id**: definisce il numero identificativo associato a ciascun commento.

- **Qa_id**: definisce il numero identificativo associato a ciascuna risposta del Q&A.

Notifica
+ testo: string + visualizzata: boolean + utente_id: int + corso_id: int + commento_id: int + qa_id: int
+ set_visualizzata()

Descrizione dei metodi implementati dalla classe **Notifica**:

- *Set_visualizzata()*: permette di settare come “già visualizzate” le notifiche visualizzate dall'utente.

2.2.4. Commenti

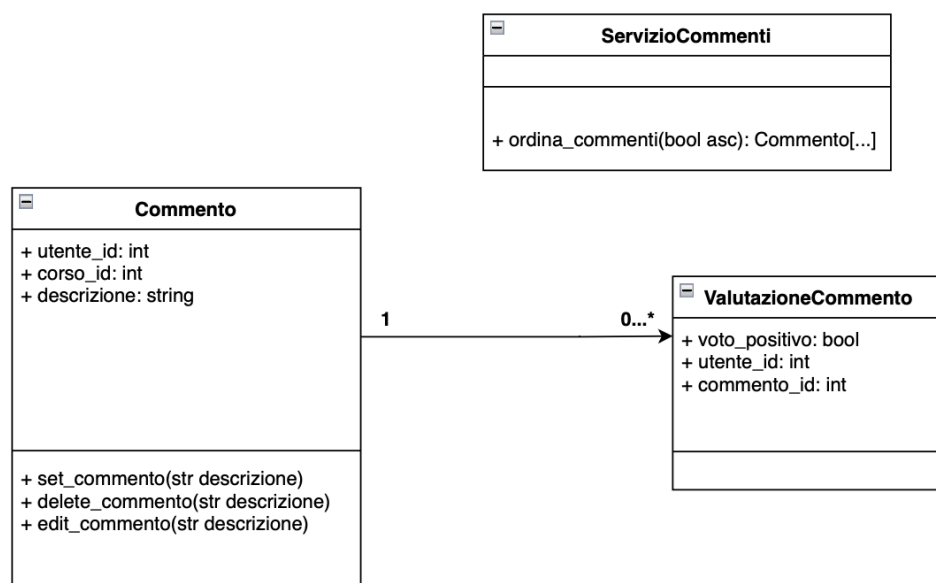
Vengono definite tre classi relative ai commenti: la classe **Commento**, la classe **ValutazioneCommento** e la classe **ServizioCommenti**.

La classe **Commento** presenta i seguenti attributi:

- *Utente_id*: definisce il codice identificativo associato a ciascun utente autenticato.
- *Corso_id*: definisce il codice identificativo associato a ciascun corso.
- *Descrizione*: definisce il contenuto effettivo del commento.

La classe **ValutazioneCommento** presenta i seguenti attributi:

- *Voto_positivo*: definisce la valutazione ad un commento(true = valutazione positiva, false = valutazione negativa).
- *Utente_id*: definisce il codice identificativo associato a ciascun utente autenticato.
- *Commento*: definisce il commento stesso.



Descrizione dei metodi della classe **Commento**:

- `set_commento(str descrizione)`: permette la pubblicazione di un commento.
- `delete_commento(str descrizione)`: permette di eliminare un commento pubblicato in precedenza.
- `edit_commento(str descrizione)`: permette di modificare un commento pubblicato in precedenza.

Descrizione dei metodi della classe **ServizioCommento**:

- `Ordina_commenti(bool asc)`: permette di ordinare la lista dei commenti in base a commenti con più valutazioni positive ricevute o commenti pubblicati più recentemente.

2.2.5. Corsi

Vengono definite quattro classi relative ai corsi: la classe **Corso**, la classe **CorsoPreferito**, la classe **ValutazioneCorso**, la classe **ServizioCorso**.

La classe **Corso** presenta i seguenti attributi:

- *Nome*: definisce il nome del corso.
- *Numero_cfu*: definisce il numero di crediti relativi al corso.
- *Utente_id*: definisce il codice identificativo associato a ciascun utente autenticato.
- *Valutazione_corso*: definisce la valutazione aggiornata relativa al corso.

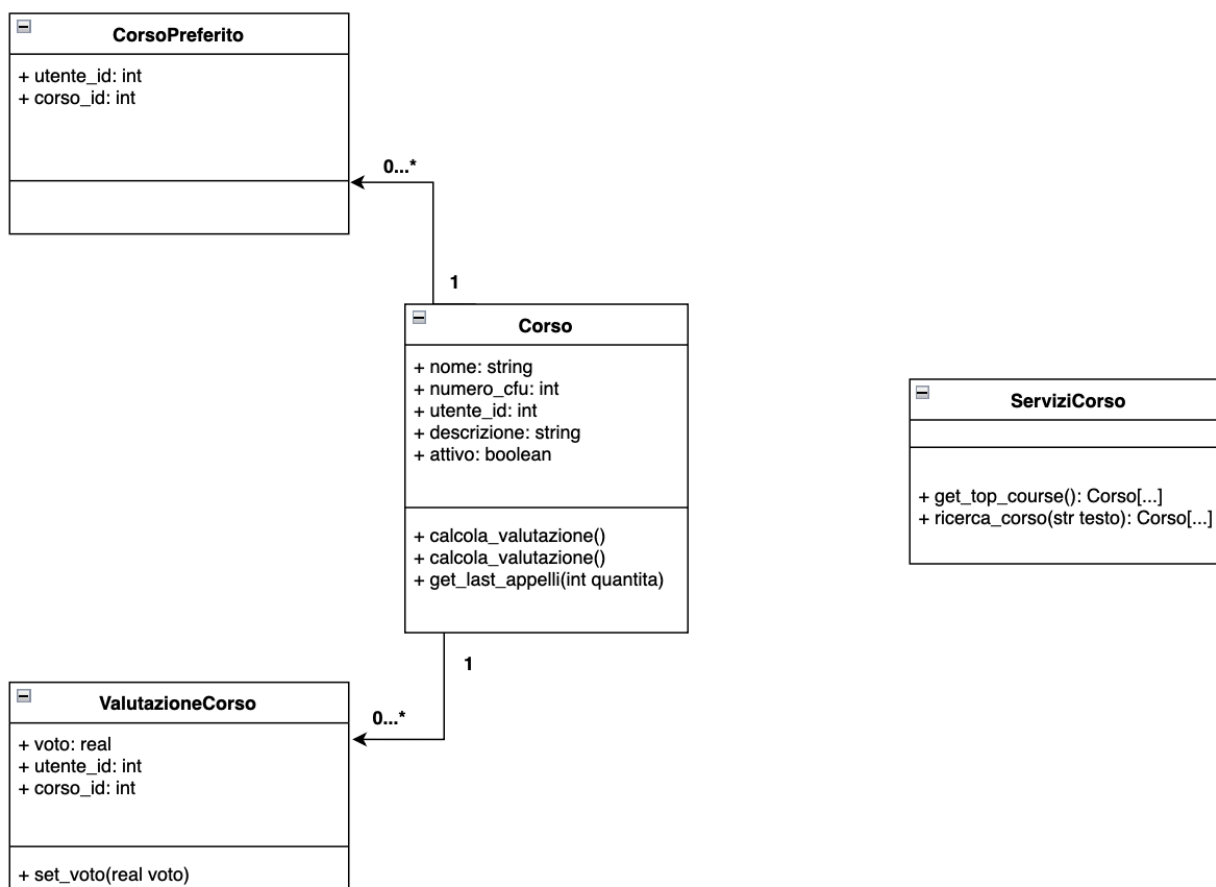
- *Attivo*: corso “vecchio” non più utilizzato, verrà però comunque tenuta traccia per mantenere i vari collegamenti.

La classe **CorsoPreferito** presenta i seguenti attributi:

- *Utente_id*: definisce il codice identificativo associato a ciascun utente autenticato.
- *Corso_id*: definisce l'id associato a ciascun corso

La classe **ValutazioneCorso** presenta i seguenti attributi:

- *Voto*: definisce la valutazione che un utente assegna ad un determinato corso.
- *Utente_id*: definisce il codice identificativo associato a ciascun utente autenticato.
- *Corso_id*: definisce il codice identificativo associato a ciascun corso.



Descrizione dei metodi della classe **Corso**:

- *Calcola_valutazione()*: calcola la valutazione relativa al corso, facendo la media tra tutte le valutazioni ricevute.

Descrizione dei metodi della classe **ValutazioneCorso**:

- *Set_voto(real voto)*: permette l'assegnazione, da parte di un utente, di una valutazione al relativo corso.

Descrizione dei metodi della classe **ServizioCorso**:

- *Get_top_course()*: permette di ottenere la lista dei corsi basata sulla valutazione; quindi dal corso con il voto più alto a quello con il voto più basso.
- *Ricerca_corso(str testo)*: permette la ricerca di un determinato corso da parte di un utente.

2.2.6. Appello

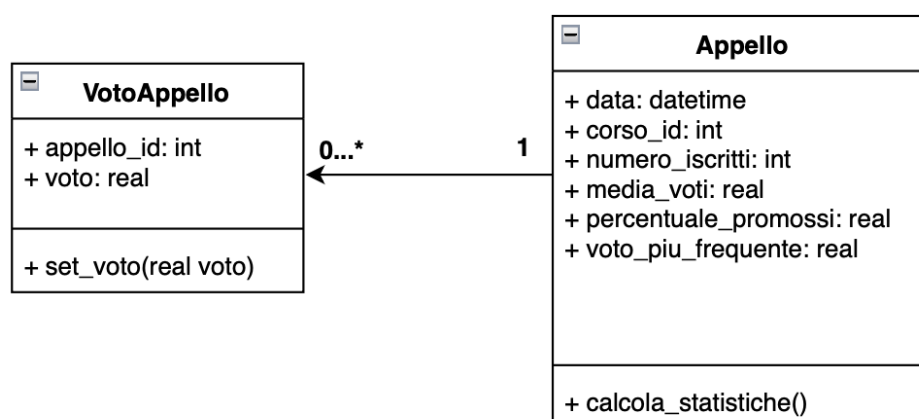
Vengono definite due classi relative ai corsi: la classe **Appello**, la classe **VotoAppello**.

La classe **Appello** presenta i seguenti attributi:

- *Data*: definisce la data in cui è stato svolto l'appello inserito.
- *Corso_id*: definisce il codice identificativo associato a ciascun corso.
- *Numero_iscritti*: definisce il numero di persone che ha svolto l'appello.
- *Media_voti*: definisce la media dei voti dell'appello inserito.
- *Percentuale_promossi*: definisce la percentuale dei promossi dell'appello inserito.
- *Voto_piu_frequente*: definisce il voto più frequente relativo all'appello inserito.

La classe **VotoAppello** presenta i seguenti attributi:

- *Appello_id*: definisce il codice identificativo associato a ciascun appello.
- *Voto*: definisce il singolo voto.



Descrizione dei metodi della classe **Appello**:

- *Calcola_statistiche()*: permette di calcolare le statistiche relative ad un corso utilizzando gli appelli inseriti dall'utente docente.

Descrizione dei metodi della classe **VotoAppello**:

- *Set_voto(real voto)*: permette di aggiungere un singolo voto.

2.2.7. Q&A

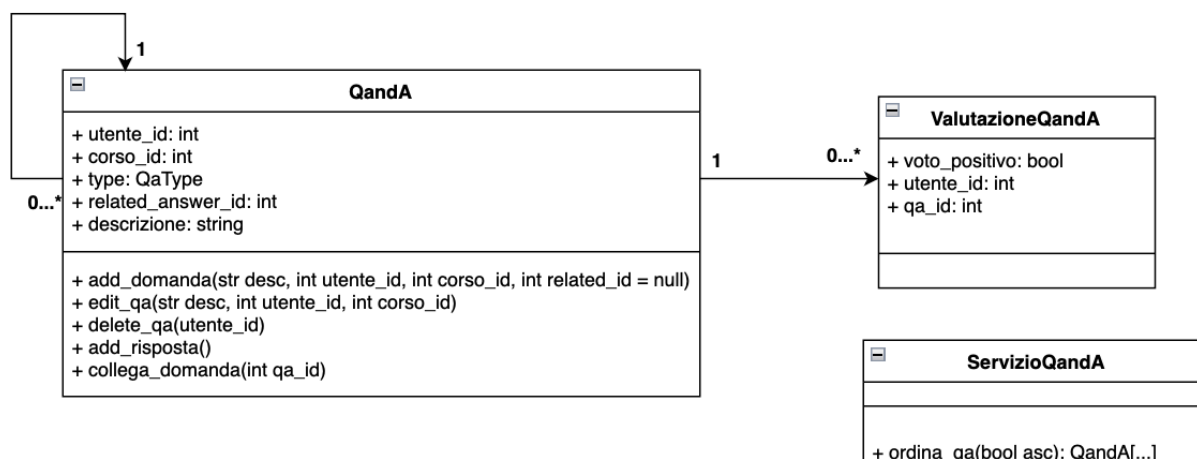
Vengono definite tre classi relative ai corsi: la classe **QandA**, la classe **ValutazioneQandA** e la classe **ServizioQandA**.

La classe **QandA** presenta i seguenti attributi:

- *Utente_id*: definisce il codice identificativo associato a ciascun utente autenticato.
- *Corso_id*: definisce il codice identificativo associato a ciascun corso.
- *type*: basandosi sul dataType Q&A, definisce se è una domanda o una risposta.
- *Related_answer_id*: definisce il caso in cui una domanda viene referenziata ad una risposta già pubblicata.
- *Descrizione*: definisce l'effettivo testo che compone una domanda o una risposta.

La classe **ValutazioneQ&A** presenta i seguenti attributi:

- *Voto_positivo*: definisce la valutazione ad una risposta all'interno della zona Q&A(true = valutazione positiva, false = valutazione negativa).
- *Utente_id*: definisce il codice identificativo associato a ciascun utente autenticato.
- *Qa_id*: definisce il codice identificativo associato a ciascuna risposta nel Q&A





Descrizione dei metodi della classe **QandA**:

- *Add_domanda(str desc, int utente_id, int corso_id, int related_id = null)*: permette l'invio di una domanda da parte di un utente studente nell'area Q&A di un determinato corso.
- *Edit_qa(str desc, int utente_id, int corso_id)*: permette di modificare una risposta pubblicata in precedenza.
- *Delete_qa(utente_id)*: permette di eliminare una risposta pubblicata in precedenza.
- *Add_risposta()*: permette la pubblicazione di una risposta da parte di un utente docente ad una domanda nell'area Q&A, precedentemente inviata da un utente studente.
- *Collega_domanda(int qa_id)*: permette di referenziare una domanda ad una risposta precedentemente pubblicata.

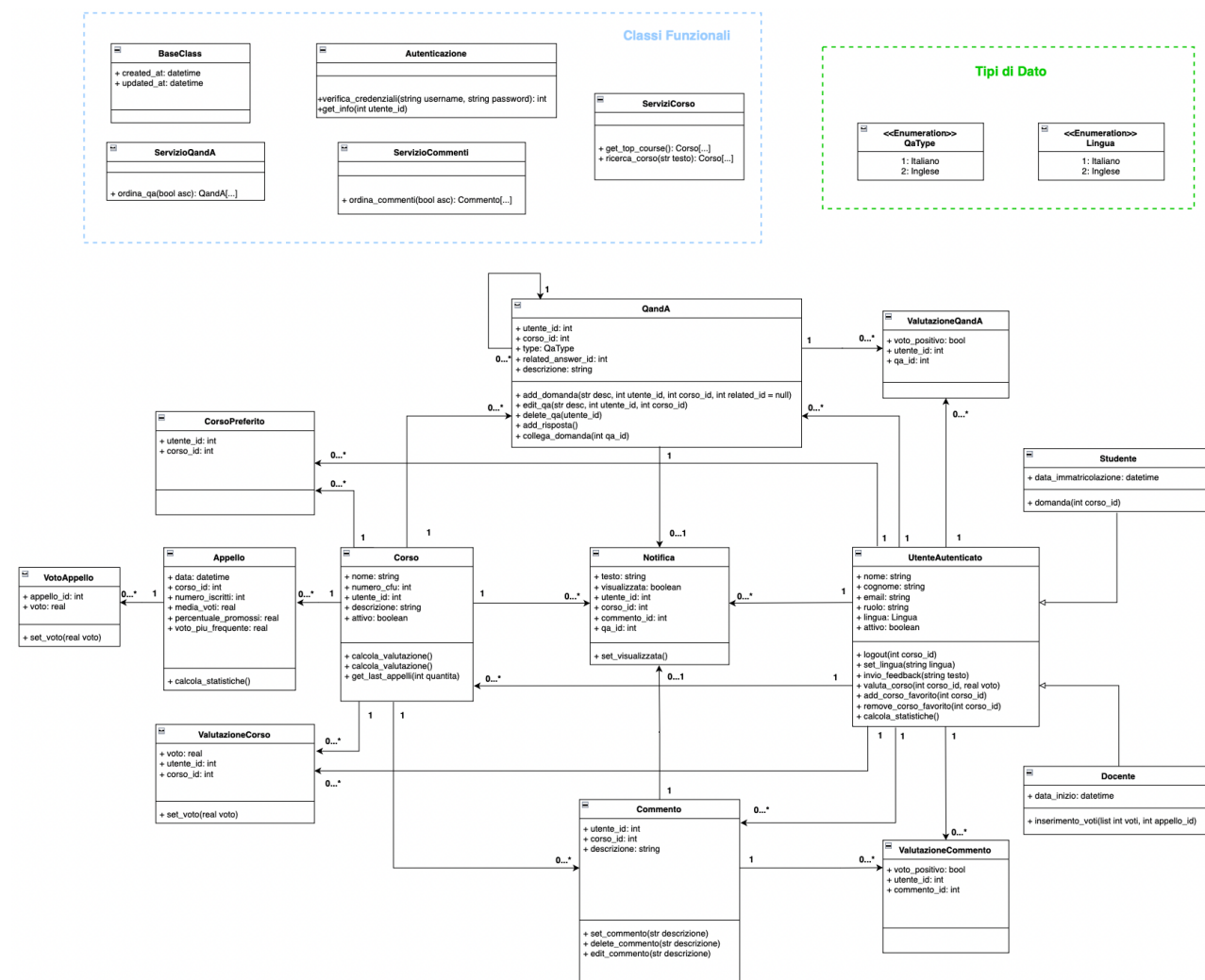
Descrizione dei metodi della classe **ServizioQandA**:

- *Ordina_qa(bool asc)*: permette di ordinare le risposte all'interno del Q&A, esse verranno messe in ordine cronologico, oppure ordinate in base alle valutazioni positive ricevute.



3. Diagramma delle classi complessivo

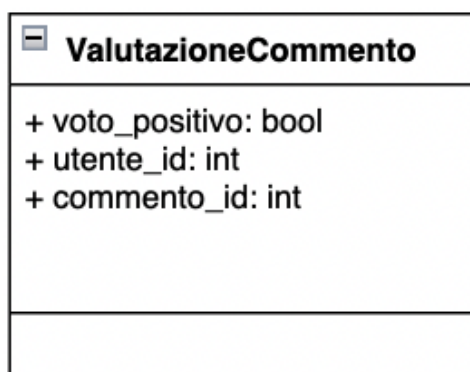
Riportiamo di seguito il diagramma complessivo; esso comprenderà tutte le classi descritte in precedenza. Sono state inoltre aggiunte alcune classi ausiliarie con lo scopo di descrivere eventuali tipi di struttura usati.



4. Codice in Object Constraint Language

In questo capitolo viene descritta in modo formale la logica prevista per le operazioni di alcune classi. Per fare questo, utilizziamo l'Object Constraint Language (OCL), poiché tali concetti non sono rappresentabili in nessun altro modo formale nel contesto UML.

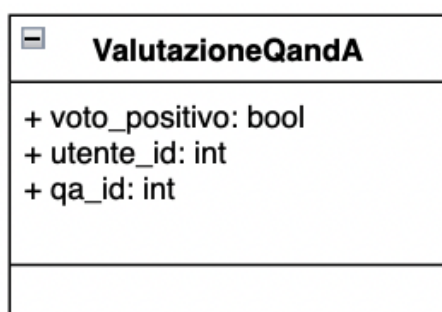
4.1 Valutazione commento



Ogni singolo utente autenticato ha la possibilità di dare una valutazione ad ogni singolo commento pubblicato all'interno di ogni corso. Essa può essere una valutazione positiva, oppure una valutazione negativa. Ciò può essere espresso in OCL in questo modo:

```
context ValutazioneCommento inv:
    ValutazioneCommento.utente_id->isUnique(commento_id)
```

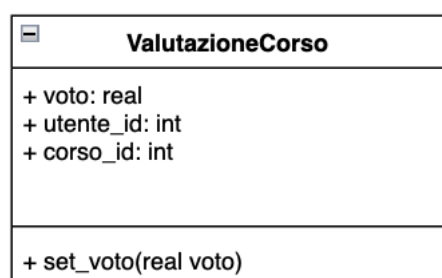
4.2 ValutazioneQandA:



Ogni singolo utente autenticato ha la possibilità di dare una valutazione ad ogni singola risposta pubblicata da un utente docente all'interno di ogni corso. Essa può essere una valutazione positiva, oppure una valutazione negativa. Ciò può essere espresso in OCL in questo modo:

```
context ValutazioneQandA inv:  
ValutazioneQandA.utente_id->isUnique(qa_id)
```

4.3 Valutazione corso



Ogni singolo utente autenticato ha la possibilità di valutare ogni singolo corso. La valutazione è obbligatoriamente unica e singola per ogni corso. Ciò può essere espresso in OCL in questo modo:

```
context ValutazioneCorso inv:  
ValutazioneCorso.utente_id->isUnique(corso_id)
```


4.4 Corso favorito

UtenteAutenticato
+ nome: string + cognome: string + email: string + ruolo: string + lingua: Lingua + attivo: boolean
+ logout(int corso_id) + set_lingua(string lingua) + invio_feedback(string testo) + valuta_corso(int corso_id, real voto) + add_corso_favorito(int corso_id) + remove_corso_favorito(int corso_id) + calcola_statistiche()

CorsoPreferito
+ utente_id: int + corso_id: int

Ogni singolo utente autenticato ha la possibilità di effettuare l'inserimento di un corso alla sua lista personale dei corsi preferiti. Ciò può essere espresso in OCL in questo modo

```
context UtenteAutenticato::add_corso_favorito(corso_id)
post: CorsoFavorito->select(utente_id=self.id and corso_id=corso_id)->size()==1
```

Ogni singolo utente autenticato ha la possibilità di effettuare la rimozione di un corso dalla sua lista personale dei corsi preferiti. Ciò può essere espresso in OCL in questo modo:

```
context UtenteAutenticato::remove_corso_favorito(corso_id)
post: CorsoFavorito->select(utente_id=self.id and corso_id=corso_id)->size()==0
```

4.5 Autenticazione

Autenticazione
+verifica_credenziali(string username, string password): int +get_info(int utente_id)

Dopo essersi autenticato un utente deve avere gli attributi: nome, cognome, email, ruolo, attivo non vuoti, il metodo **get info** può essere eseguito solo se verifica credenziali va a buon fine. Ciò può essere espresso in OCL in questo modo:

```
context Autenticazione::get_info(UtenteAutenticato ua)
pre:
verifica_credenziali(username, password)=ua.id
post:
ua.nome=notEmpty() and
ua.cognome=notEmpty() and
ua.email=notEmpty() and
ua.ruolo=notEmpty() and
ua.attivo=notEmpty()
```

4.6 Calcola statistiche

Appello
+ data: datetime + corso_id: int + numero_iscritti: int + media_voti: real + percentuale_promossi: real + voto_piu_frequente: real
+ calcola_statistiche()

La funzione **Calcola_statistiche** permette di controllare che ci sia almeno un appello presente e in questo caso calcolare varie statistiche. Per ogni singolo appello verranno quindi calcolate: la media dei voti, la percentuale dei promossi e il voto più frequente. Ciò può essere espresso in OCL in questo modo:

```
context Appello::calcola_statistiche()
pre:
VotoAppello->select(appello_id=self.id)->size()>0
post:
Self.numero_iscritti = VotoAppello->select(appello_id=self.id)->count()
self.media_voti = VotoAppello->select(appello_id=self.id)->avg()
self.percentuale_promossi = VotoAppello->select(appello_id=self.id and
voto>=18).voto->sum() / VotoAppello->select(appello_id=self.id).voto->sum()
self.voto_piu_frequente = VotoAppello->select(appello_id=self.id).voto->mode()
```

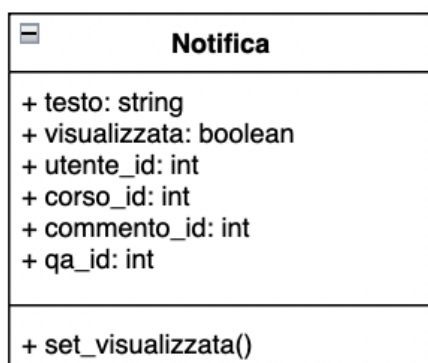
4.7 Referenzia domanda

QandA
+ utente_id: int + corso_id: int + type: QaType + related_answare_id: int + descrizione: string
+ add_domanda(str desc, int utente_id, int corso_id, int related_id = null) + edit_qa(str desc, int utente_id, int corso_id) + delete_qa(utente_id) + add_risposta() + collega_domanda(int qa_id)

L'utente docente ha la possibilità di referenziare una domanda, ricevuta da un utente studente, ad una risposta già pubblicata in precedenza riferitasi ad un'altra domanda. Ciò può essere espresso in OCL in questo modo:

```
context QandA::collega_domanda(int qa_id)
pre:
QandA ->select(id=qa_id).type="Risposta"
post:
Self.related_answare_id=qa_id
```

4.8 Visualizzazione notifica:

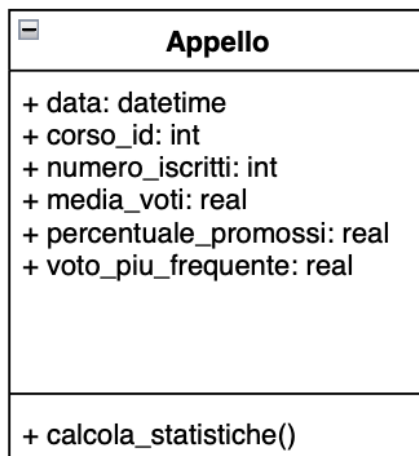


La funzione **set_visualizzato()** permette di segnalare se le notifiche sono state visualizzate o meno. Quando una notifica non è visualizzata la funzione relativa a quella

notifica ritorna false, invece, dopo che un utente autenticato visualizza una notifica, la funzione relativa a quella notifica ritorna true. Ciò può essere espresso in OCL in questo modo:

```
context Notifica::set_visualizzato()  
post:  
Self.visualizzata=True
```

4.9 Univocità appello



Il sistema deve garantire l'univocità di appello. Questo significa che deve fare un controllo sull'appello di un determinato corso in una relativa data, e controllare se quest'ultimo è unico.

In particolare effettuiamo questo controllo poiché non è possibile che vi siano più appelli di un corso in stessa data.

Ciò può essere espresso in OCL in questo modo:

```
context Appello inv:  
Appello.data->isUnique(corso_id)
```

5. Diagramma delle classi con codice OCL

Viene di seguito riportato il diagramma delle classi completo con l'aggiunta del codice OCL precedentemente descritto

