

INGEGNERIA DEL SOFTWARE

SOUNDMATCH

DOCUMENTO DI ARCHITETTURA

Alessandro Fontana, Davide Pedrotti, Nicolas Torriglia

Università di Trento

Indice

Scopo del documento	3
1 Diagramma delle classi	3
1.1 Utente Autenticato, Artista e Ascoltatore	3
1.2 Brano e MusicPlayer	4
1.3 Genere e Community	5
1.4 Pagamento	6
1.5 Chat e Messaggio	6
1.6 Gestore delle Ricerche	7
1.7 Gestore Verificati e Mostra Verificati	8
1.8 Autenticazione e Registrazione	9
1.9 Email sender	9
2 Codice in Object Constraint Language	11
2.1 Messaggi tra artisti	11
2.2 Modifica brano	13
2.3 Gestione verificati	14
2.4 Manda richiesta collaborazione	15
2.5 Recupero mail	16
2.6 Completa registrazione	17
2.6.1 Invia mancia	18
2.6.2 Ascolta brano	19
2.6.3 Iscrizione a community	20
2.6.4 Scrivere nella community	21
2.6.5 Ricerca brano	21
3 Diagramma delle classi con codice OCL	22

Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto SoundMatch 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 digramma 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 SoundMatch. 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 Utente Autenticato, Artista e Ascoltatore

Osservando il diagramma di contesto, si nota la presenza di due attori: utente autenticato e artista. Per ognuno di questi due attori introduciamo una classe. La prima, ovvero

UtenteAutenticato, definisce l'utilizzatore della piattaforma una volta che egli ha eseguito il login, ovvero dal momento che si è identificato tramite password e email. Esso può essere di due tipi: artista o ascoltatore. Essendo l'artista in grado di eseguire tutte le operazioni dell'ascoltatore, definiamo la classe Artista come estensione della classe appena descritta, che coincide con l'ascoltatore. Dato che un utente non autenticato si può autenticare tramite login oppure tramite registrazione, creiamo la classe UtenteNonAutenticato.

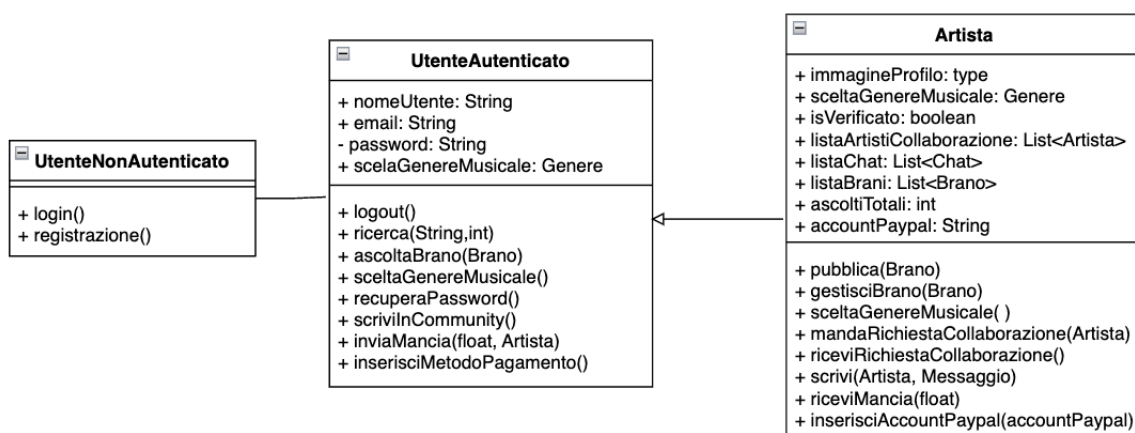


Figura 1: Classi per Utente non Autenticato, Utente Autenticato e Ascoltatore

1.2 Brano e MusicPlayer

Analizzando il diagramma dei componenti realizzato per il progetto SoundMatch, si nota la presenza di un riproduttore musicale il quale consente l'ascolto dei brani. Perciò è stata identificata una classe MusicPlayer che permetterà all'utente autenticato di riprodurre brani e di navigare tra di essi. Inoltre attraverso la classe MusicPlayer verrà incrementato il numero di ascolti di ogni brano presente in SoundMatch. Le altre due classi identificate dal diagramma dei componenti che si interfacciano a MusicPlayer sono: Brano e ModificaBrano. La classe Brano contiene gli attributi che personalizzano e caratterizzano il brano stesso. Essa estende ModificaBrano, la quale permette di modificare vari attributi di un brano dopo che esso è stato pubblicato.

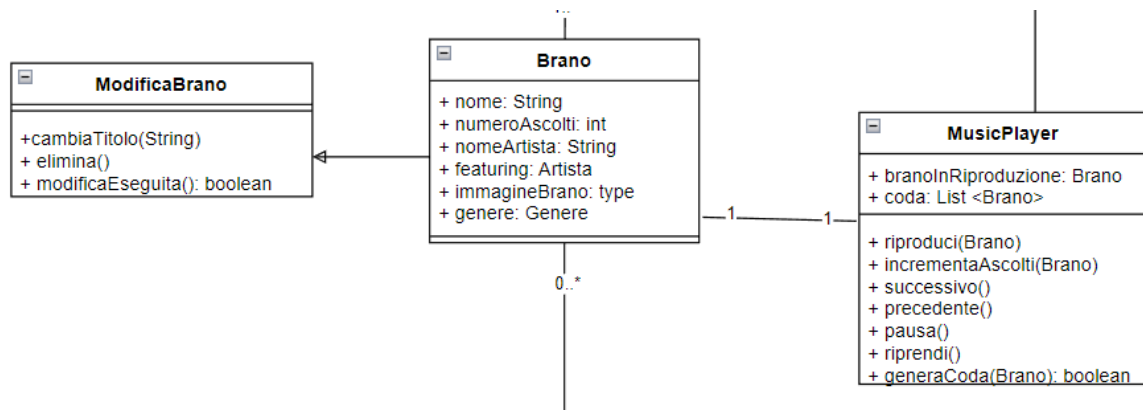


Figura 2: Classi per Brano e MusicPlayer

1.3 Genere e Community

In seguito all'analisi del diagramma dei componenti denotiamo la presenza del componente Gestore Community. A partire da esso definiamo la classe **Community**, con lo scopo di gestire tutte le dinamiche riguardanti la chat globale e la visualizzazione di artisti e brani appartenenti ad un determinato genere. Infatti, la community è costituita da utenti ai quali piace lo stesso genere musicale. Definiamo, di conseguenza, la classe **Genere** che viene poi estesa da **Community**.

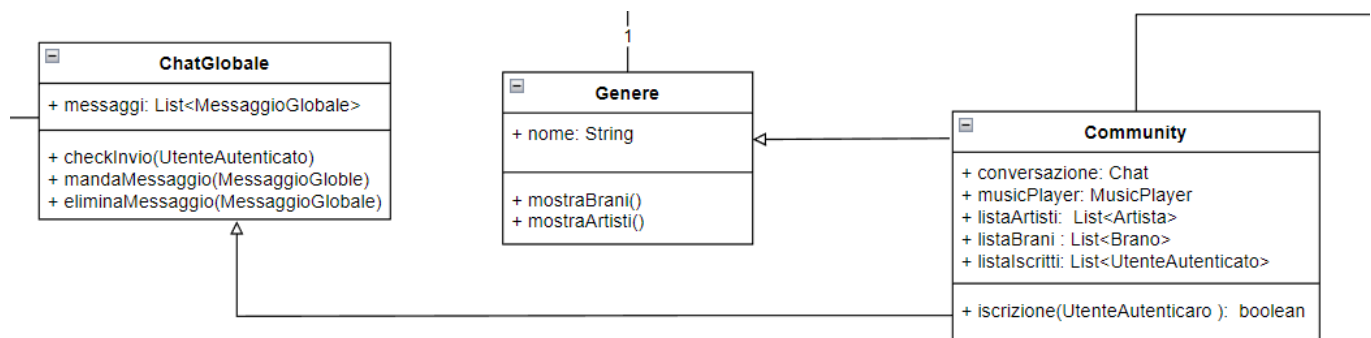


Figura 3: Classi per Genere e Community

1.4 Pagamento

Il diagramma di contesto di SoundMatch presenta un sistema subordinato denominato "Sistema di Pagamento", che rappresenta il meccanismo con cui un utente autenticato può inviare una mancia ad un artista attraverso un sistema di pagamento esterno (PayPal). Dal diagramma è stata identificata una classe: "Pagamento". Essa ha come attributi mittente, destinatario e importo. I dati in questione non verranno salvati dalla classe ma tramite questi ci si interfacerà col sistema di pagamento esterno il quale si occuperà dell'addebito e dell'accredito e di controllare se il saldo è sufficiente per l'operazione.

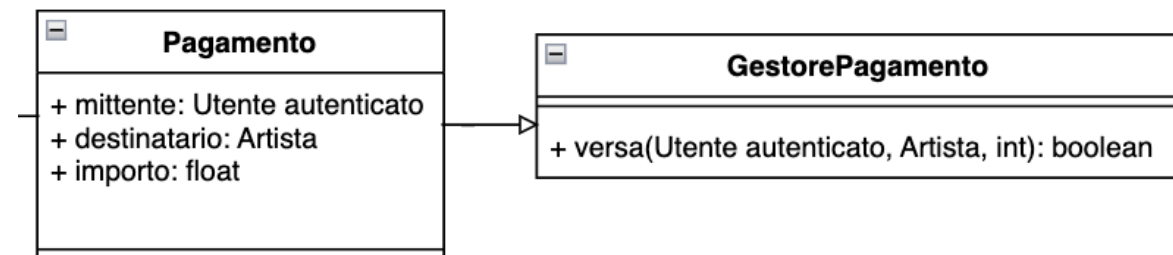


Figura 4: Classi per Chat e Messaggio

1.5 Chat e Messaggio

Attraverso i componenti Pagina Chat e Gestione Chat sono state identificate tre classi che li rappresentano: Chat, GestioneChat e Messaggio. La classe Chat rappresenta una conversazione tra due o più utenti autenticati. Contiene la lista dei messaggi e il destinatario; attraverso i suoi metodi vengono rese disponibili le proprietà di inviare, leggere ed eliminare i messaggi. Questa classe estende GestioneChat, la quale opera da supporto svolgendo la funzione di back-up. Come citato precedentemente, nella classe Chat è presente un'istanza della classe Messaggio che comprende come attributi la data di invio, il testo e il mittente.

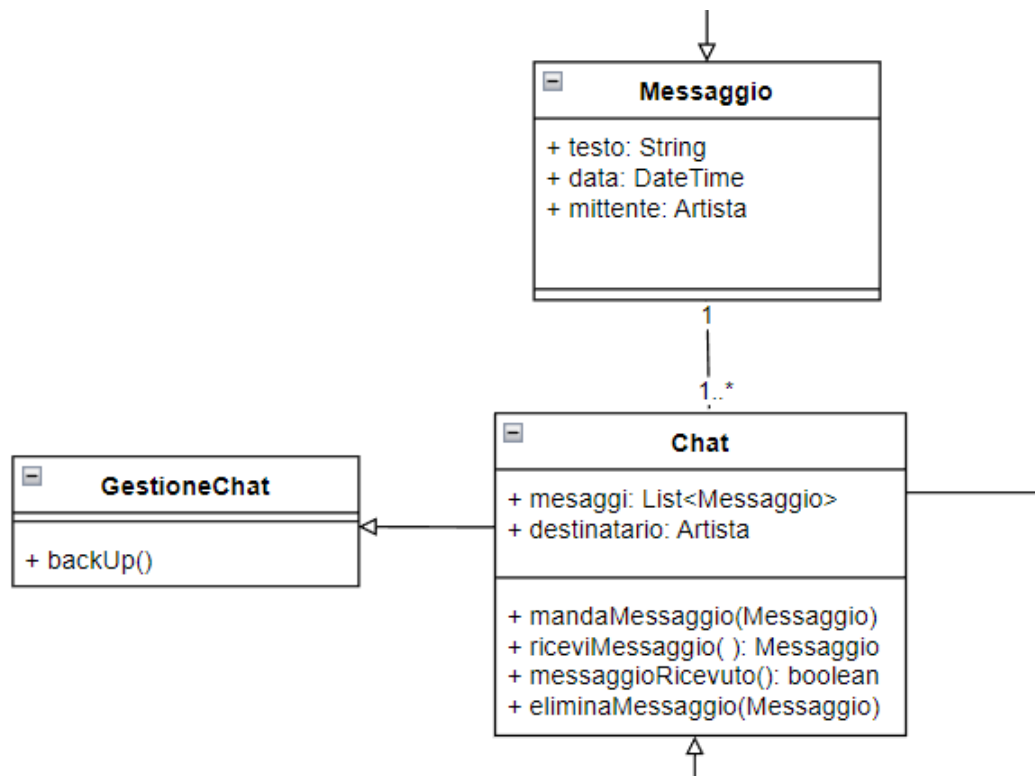


Figura 5: Classi per Chat e Messaggio

1.6 Gestore delle Ricerche

Dall'unione dei componenti Gestore Ricerca e Pagina Ricerca, nasce la classe GestoreRicerche. L'utente ha la possibilità di effettuare delle ricerche di artisti, brani o generi. Questa classe ha proprio il compito di mostrare dei suggerimenti a seguito di un inserimento di testo. Infatti, a partire da una stringa inserita dall'utente e di un secondo parametro che indica tra quale delle tre categorie effettuare tale ricerca, si produce come risultato una lista di elementi affini.

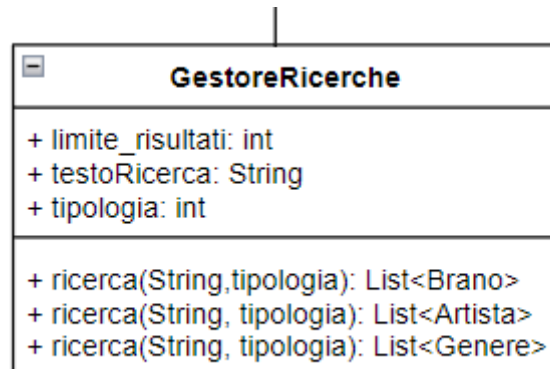


Figura 6: Classe per GestoreRicerche

1.7 Gestore Verificati e Mostra Verificati

Analizzando il diagramma dei componenti si nota la presenza del componente GestoreVerificati. È stata dunque identificata la classe GestoreVerificati la quale contiene nel campo degli attributi una lista di artisti. Inoltre la classe contiene il metodo "check()" che verifica se un artista appartiene alla lista dei verificati e il metodo "setVerificato()", che si occupa di assegnare all'artista il badge verificato una volta raggiunti i requisiti. Un'altra classe individuata nel diagramma dei componenti che si interfaccia e estende Gestore Verificati è Mostra Verificati. Questa classe ha come compito quello di mostrare ad un utente autenticato una lista di giornata degli artisti e del loro brano più popolare.

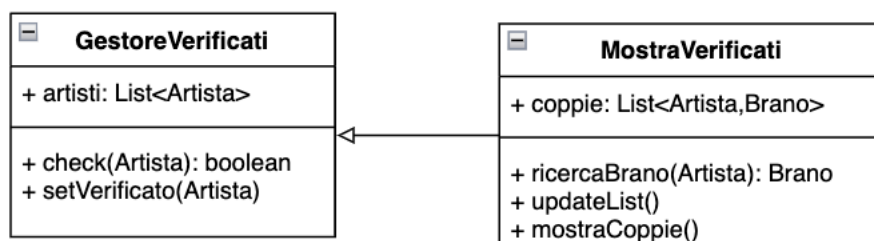


Figura 7: Classi per Gestore Verificati e Mostra Verificati

1.8 Autenticazione e Registrazione

Successivamente all'analisi del diagramma dei componenti e del diagramma di contesto, denotiamo la necessità da parte dell'utente non autenticato di registrarsi o accedere alla piattaforma, in modo tale che possa ottenere i privilegi dell'utente autenticato o addirittura dell'artista se questa è la tipologia di account scelta. La classe Autenticazione ha il compito di verificare password e email di un utente già iscritto a SoundCloud o, alternativamente, gestisce l'accesso con Google se esso è stato scelto come metodo di registrazione. Analogamente, la classe Registrazione gestirà le due modalità di iscrizione.

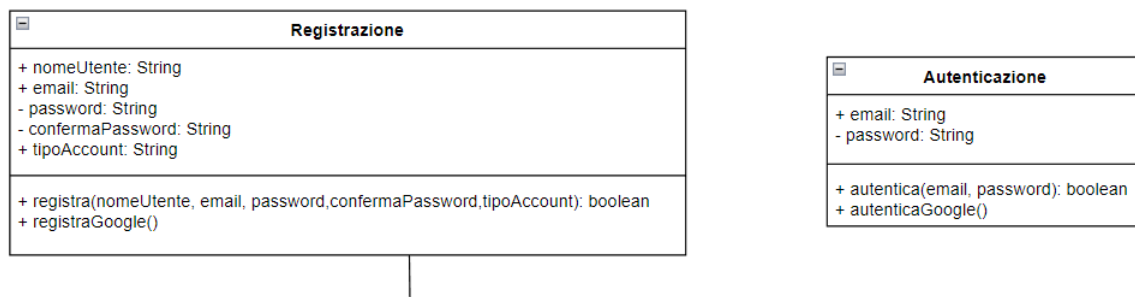


Figura 8: Classi per Autenticazione e Registrazione

1.9 Email sender

Definito come attore nel diagramma di contesto, il servizio di posta svolge un ruolo fondamentale in diverse azioni della piattaforma. Nella fase di registrazione è necessario confermare l'indirizzo di posta tramite email, in modo da non inserirne uno falso o errato. Questo procedimento è essenziale perchè verrà mandata un'email nel caso di smarrimento password o ogni qual volta un'artista richiederà una collaborazione ad un altro artista. Questa è la motivazione che ci ha spinto a definire la classe EmailSender la quale si interfaccia con un servizio di posta esterno. Questa classe si occupa di fornire a tale servizio le informazioni necessarie quali indirizzi email del destinatario e del mittente, oggetto e corpo del messaggio, sia esso testuale o html.

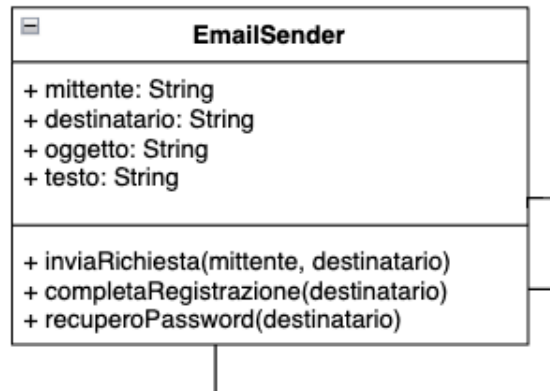


Figura 9: Classi per l'email sender

Diagramma delle classi complessivo

Riportiamo di seguito il diagramma delle classi con tutte le classi fino ad ora presentate. Oltre alle classi già descritte, è stata inserita la classe ausiliara Datatime

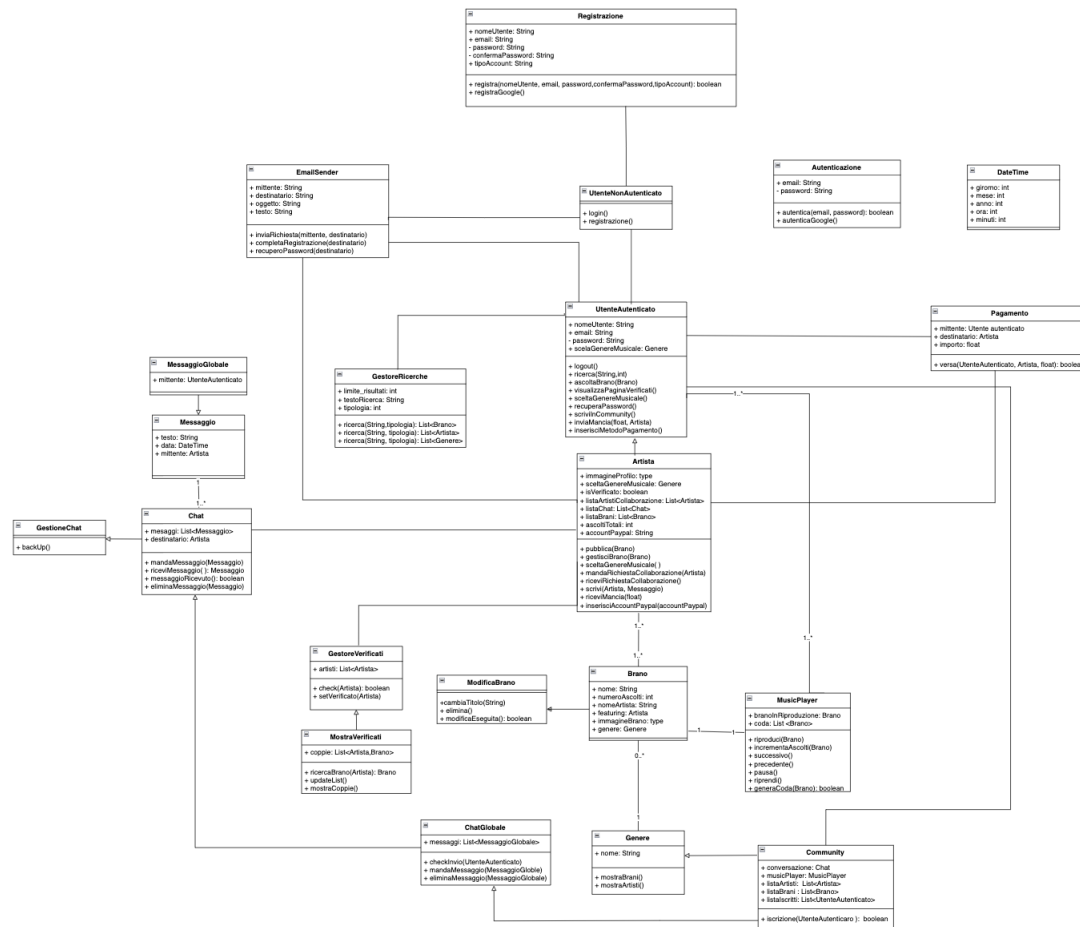


Figura 10: Diagramma delle classi di SoundMatch

2 Codice in Object Constraint Language

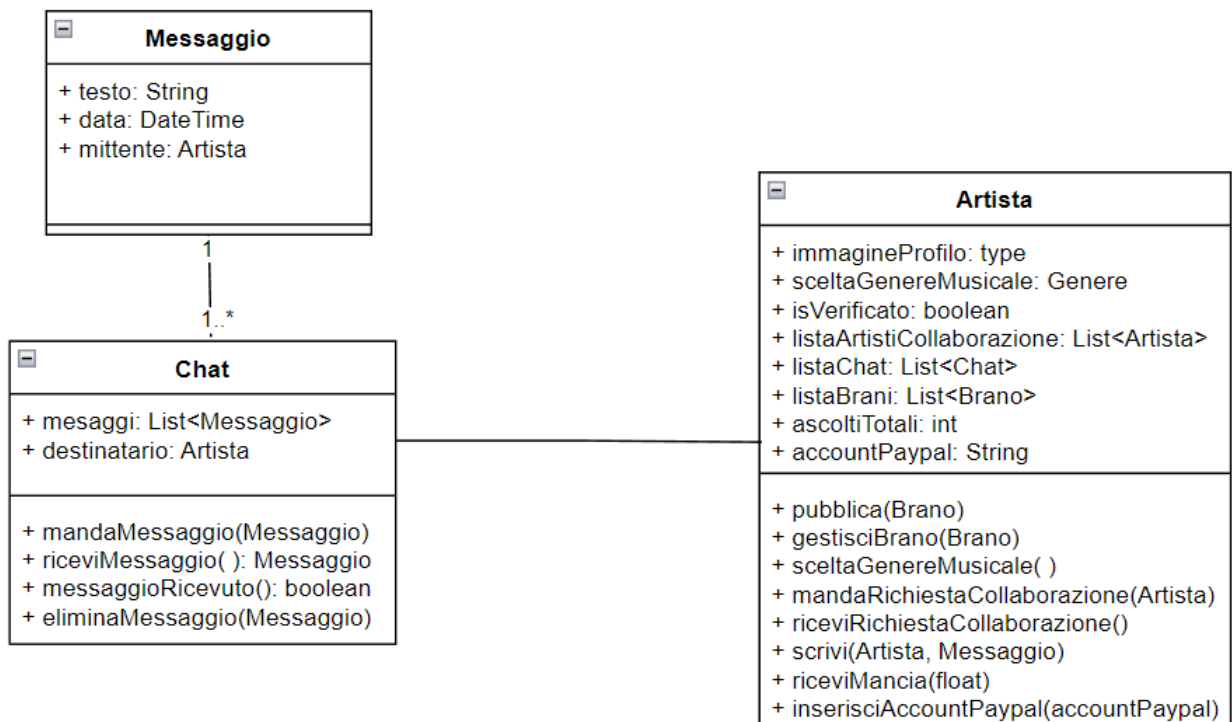
In questo capitolo è descritta in modo formale la logica prevista nell’ambito di alcune operazioni di alcune classi. Tale logica viene descritta in Object Constraint Language (OCL) perché tali concetti non sono esprimibili in nessun altro modo formale nel contesto di UML.

2.1 Messaggi tra artisti

L'artista ha la possibilità di scrivere ad un altro artista soltanto se esso è presente nella "listaArtistiCollaborazione". Inoltre, l'esecuzione di questo metodo porta il metodo `messaggioRicevuto()` a ritornare il valore `true`. Questa condizione è espressa in OCL tramite

precondizione e postcondizione con questo codice:

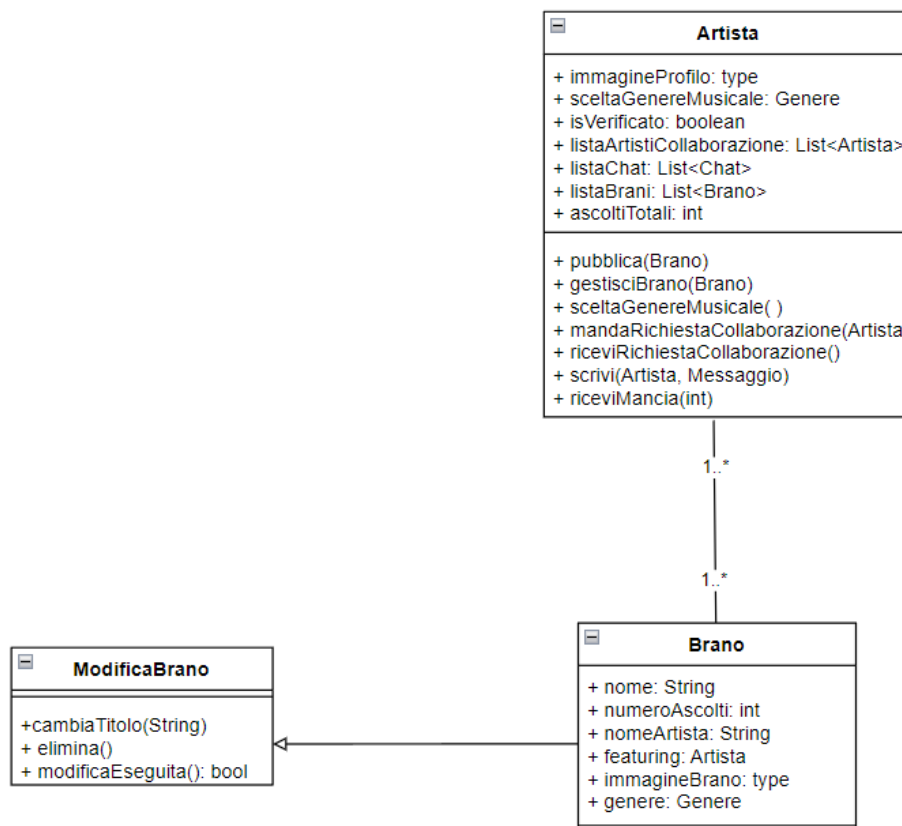
```
context: Artista::scrivi(destinatario,Messaggio)
pre: listaArtistiCollaborazione->includes(destinatario)
post: Chat::messaggioRicevuto() = true
```



2.2 Modifica brano

Nella classe "Artista" è presente il metodo `gestisciBrano(Brano)`, il quale permette all'artista di modificare o eliminare il proprio brano attraverso i metodi della classe "ModificaBrano". Questo metodo può essere eseguito soltanto se il brano che si vuole modificare è presente nella lista dei brani dell'artista. In seguito all'esecuzione di questo metodo, il metodo `modificaEseguita()` ritornerà valore `true`. Questa condizione è espressa in OCL tramite preconditione e postcondizione con questo codice:

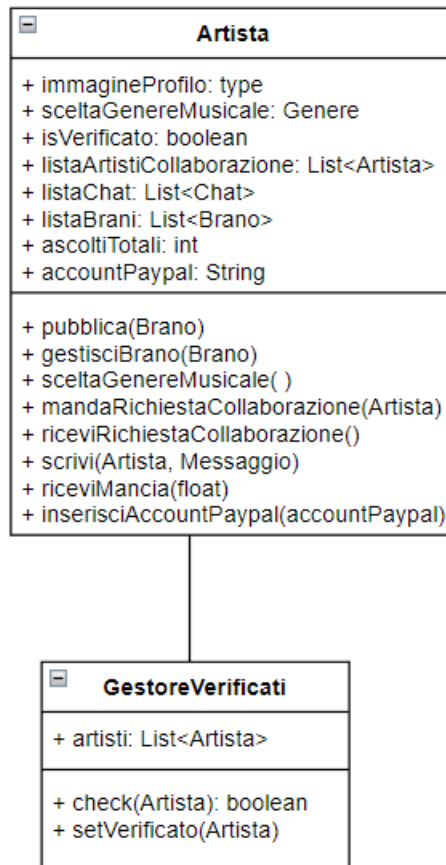
```
context: Artista::gestisciBrano(Brano)
pre: listaBrani -> includes(Brano)
post: modificaEseguita() = true
```



2.3 Gestione verificati

Nella classe "GestoreVerificati" è presente il metodo `setVerificato(Artista)`, il quale si occupa di assegnare all'artista il badge "verificato". Questo metodo richiede che il metodo `check(Artista)` (il quale verifica l'idoneità dell'artista) sia uguale a `true`. Questa condizione è espressa in OCL tramite precondizione con questo codice:

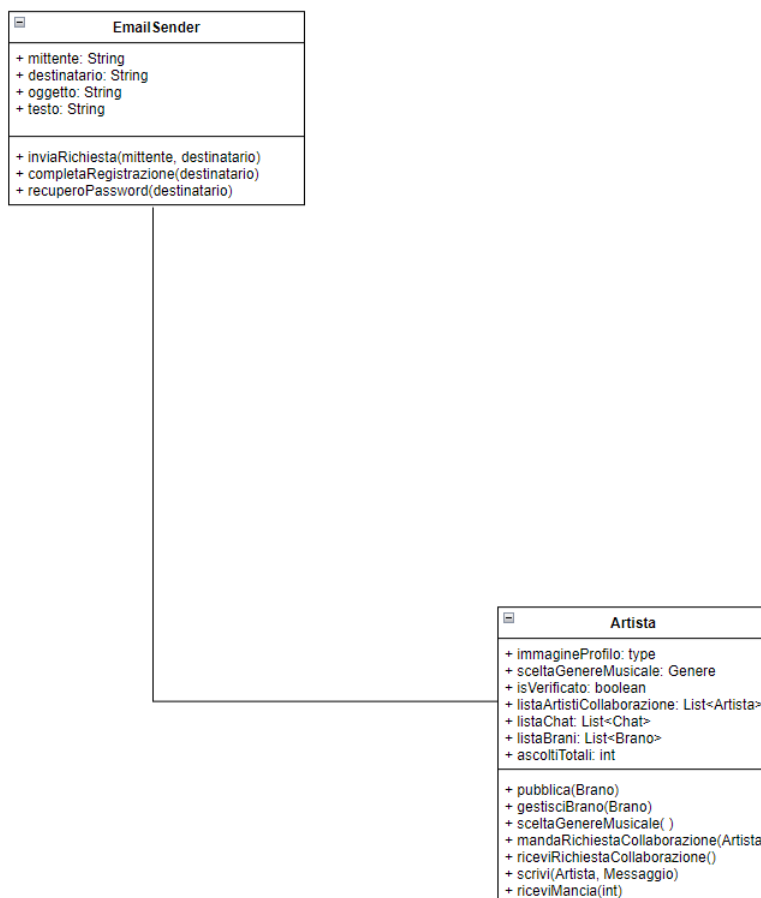
```
context: GestoreVerificati::setVerificato(Artista)
pre: check(Artista) = true
```



2.4 Manda richiesta collaborazione

Nella classe "Artista" è presente il metodo `mandaRichiestaCollaborazione(Artista)`, il quale consente agli artisti di mandare richieste di collaborazione ad altri artisti. In seguito all'esecuzione di questo metodo, la classe "EmailSender" invierà all'email del destinatario la richiesta di collaborazione. Una volta che il destinatario avrà aperto l'hyperlink contenuto nella mail, il metodo `riceviRichiestaCollaborazione()` verrà eseguito. Questo metodo è espresso in OCL tramite postcondizione con questo codice:

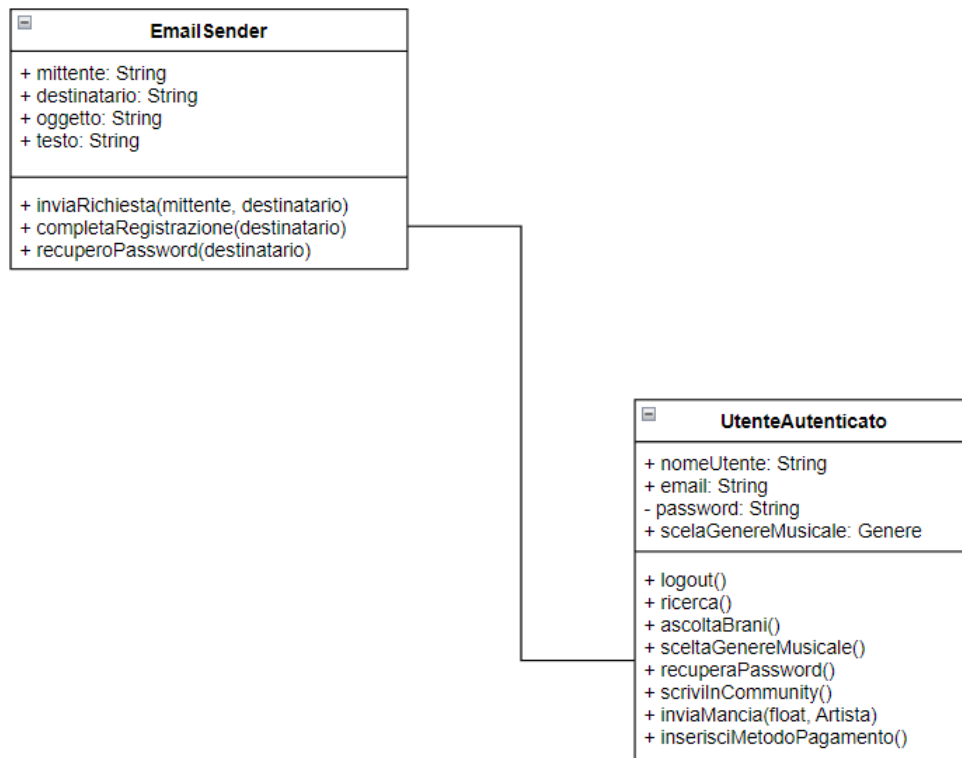
```
context: Artista::mandaRichiestaCollaborazione(Artista)
post: EmailSender::inviaRichiesta(self.email,Artista.email) AND
      Artista::riceviRichiestaCollaborazione()
```



2.5 Recupero mail

Nella classe `UtenteAutenticato` è presente il metodo `recuperaPassword()`, l'esecuzione di questo metodo comporta l'esecuzione del metodo `recuperoPassword(destinatario)` presente nella classe `EmailSender`. Questo metodo è espresso in OCL tramite postcondizione con questo codice:

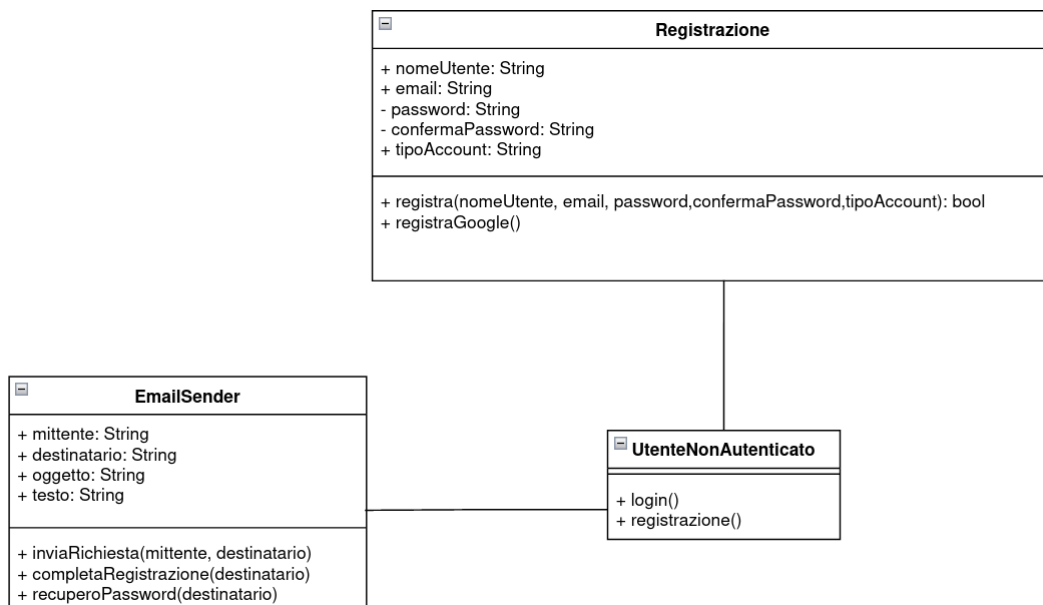
```
context: UtenteAutenticato::recuperaPassword()  
post: EmailSender::recuperoPassword(destinatario)
```



2.6 Completa registrazione

La classe `UtenteNonAutenticato` contiene il metodo `registrazione()`, il quale comporta la successiva esecuzione del metodo `registra(nomeUtente,email,password,confermaPassword,tipoAccount)` presente nella classe `Registrazione`. Successivamente, se il metodo `registra(...)` ritorna valore `true`, verrà eseguito il metodo `completaRegistrazione(destinatario)` della classe `EmailSender`. Questo metodo è espresso in OCL tramite postcondizione con questo codice:

```
context: UtenteNonAutenticato::registrazione()
post: Registrazione::registra(nomeUtente,email,password,confermaPassword,tipoAccount)
AND if (registra(nomeUtente,email,password,confermaPassword,tipoAccount)=true) then
EmailSender::completaRegistrazione(destinatario)
```



2.6.1 Invia mancia

La classe `UtenteAutenticato` comprende il metodo `inviaMancia(float,Artista)`. Questo metodo necessita che il valore `accountPaypal` sia diverso da `null`. In seguito all'esecuzione di tale metodo, l'utente verrà portato alla pagina di login di PayPal, effettuato il login potrà confermare il pagamento. A pagamento effettuato correttamente il metodo `versa(UtenteAutenticato,Artista,float)` ritornerà valore `true`. Se il metodo `versa(...)` ritorna `true`, il metodo `Artista::riceviMancia(float)` verrà eseguito. Questo metodo è espresso in OCL tramite precondizione e postcondizione con questo codice:

```

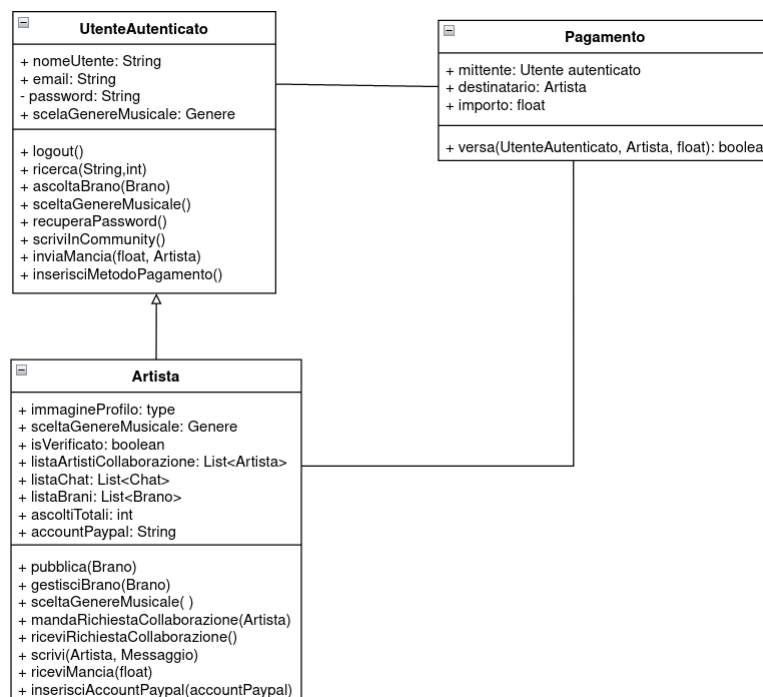
context: UtenteAutenticato::inviaMancia(float, Artista)

pre: Artista.accountPaypal != null

post: versa(self, Artista, float) AND

if (versa(self, Artista, float) = true) then Artista::riceviMancia(float)

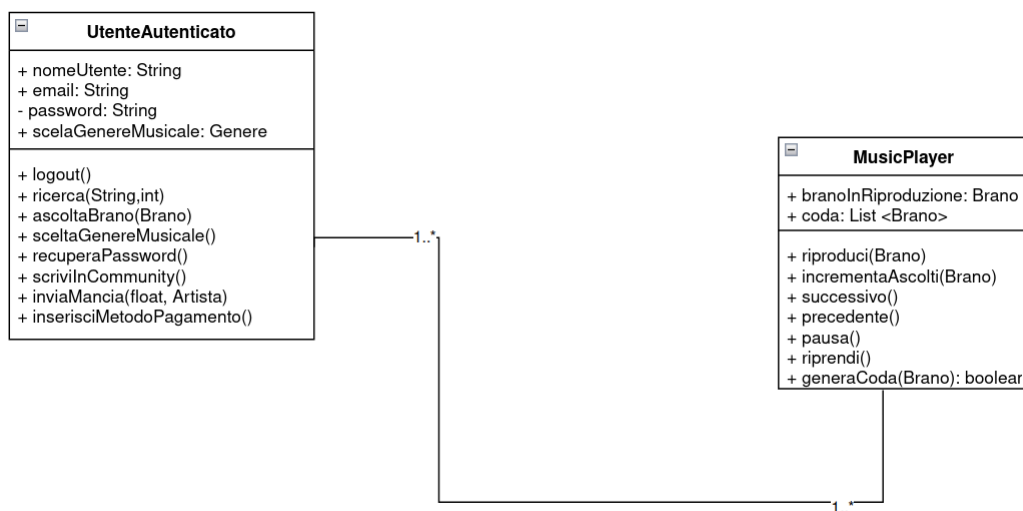
```



2.6.2 Ascolta brano

La classe `UtenteAutenticato` comprende il metodo `ascoltaBrano(Brano)`, in seguito all'esecuzione di tale metodo, il metodo `MusicPlayer::riproduci(Brano)` verrà eseguito, inoltre se `MusicPlayer::generaCoda(Brano)` ritorna valore `true`, il brano verrà inserito in coda. Questo metodo è espresso in OCL tramite postcondizione con questo codice:

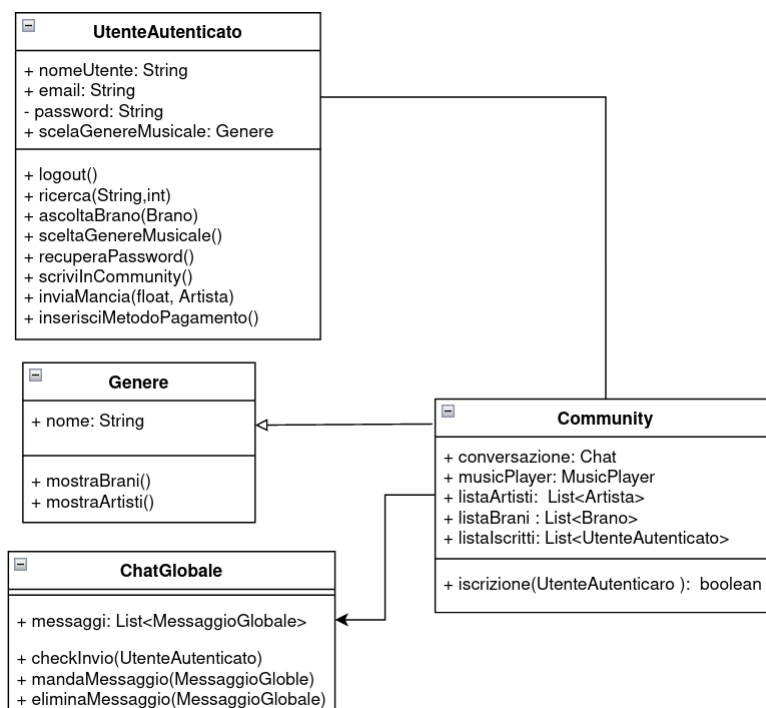
```
context: UtenteAutenticato::ascoltaBrano(Brano)
post: MusicPlayer::riproduci(Brano)
AND if(MusicPlayer::generaCoda(Brano)) then coda->includes(Brano)
```



2.6.3 Iscrizione a community

Nella classe `Community` è presente il metodo `iscrizione(UtenteAutenticato)` il quale viene eseguito soltanto se `UtenteAutenticato` non è presente nella `listaIscritti`, attributo di `Community`. In seguito all'esecuzione del metodo `iscrizione(UtenteAutenticato)`, l'`UtenteAutenticato` sarà presente nella `listaIscritti`. Questo metodo è espresso in OCL tramite preconditione e postcondizione con questo codice:

```
context: Community::iscrizione(UtenteAutenticato)
pre: self.listaIscritti->includes(UtenteAutenticato) = false
post: self.listaIscritti->includes(UtenteAutenticato) = true
```



2.6.4 Scrivere nella community

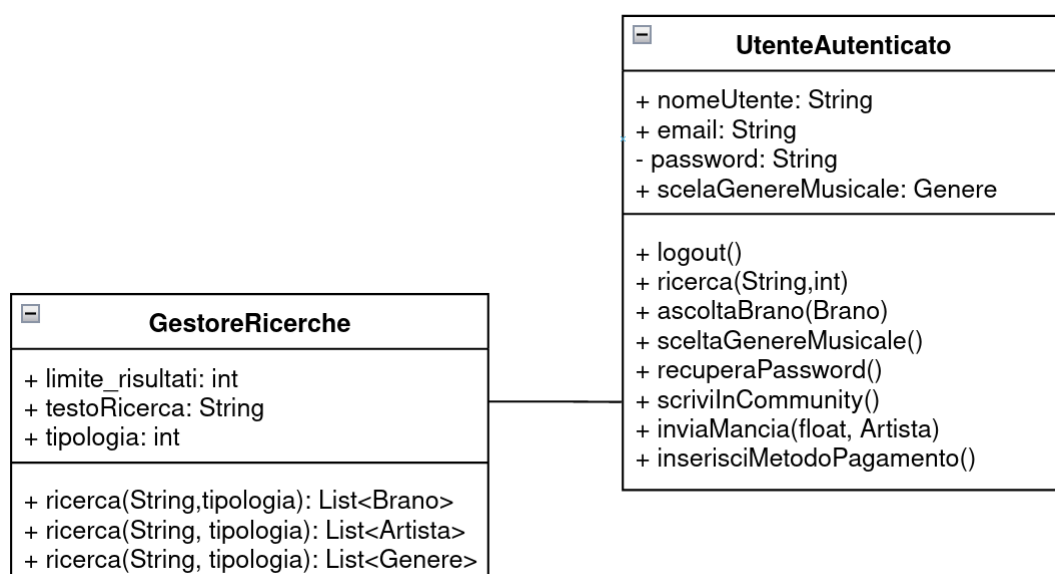
Nella classe `UtenteAutenticato` appena riportata è presente il metodo `scriviInCommunity()`, per poter essere eseguito l'`UtenteAutenticato` deve essere presente nella `listaIscritti`, attributo di `Community`. Questo metodo è espresso in OCL tramite preconditione con questo codice:

```
context: UtenteAutenticato::scriviInCommunity()
pre: Community::listaIscritti->includes(self)
```

2.6.5 Ricerca brano

Nella classe `UtenteAutenticato` è presente il metodo `ricerca(String,int)`. L'esecuzione di questo metodo comporta l'esecuzione del metodo `ricerca(String,tipologia)` presente nella classe `GestoreRicerche`. Questo metodo è espresso in OCL tramite postcondizione con questo codice:

```
context: UtenteAutenticato::ricerca(String,int)
post: GestoreRicerche::ricerca(String,tipologia)
```



Riportiamo dunque il diagramma delle classi con relativo codice OCL

