PROGETTO DI INGEGNERIA DEL SOFTWARE

SKI ONLINE

Architettura

Indice dei contenuti

INDICE DEI CONTENUTI 2
SCOPO DEL DOCUMENTO 3
DIAGRAMMA DELLE CLASSI 3
UTENTI DEL SISTEMA 4
Specifica OCL 4
GESTIONE SESSIONE 5
Specifica OCL 5
SKIPASS 6
Specifica OCL 6
GESTIONE MAESTRI 7
Specifica OCL 7
ACQUISTO SKIPASS E PRENOTAZIONE LEZIONI 9
Specifica OCL 9
LIMITAZIONE NUMERO SKIPASS ACQUISTABILI 11
Specifica OCL 11
GESTIONE SKIPASS ASSOCIATI AD UNA UTENZA 12
Specifica OCL 12
WIDGET INFORMATIVO LEZIONI 13
Specifica OCL 13
WIDGET ANNUNCI PUBBLICATI 14
Specifica OCL 14
GESTIONE WIDGET METEOROLOGICO 15
GESTIONE APERTURA E CHIUSURA IMPIANTI 16
Specifica OCL 16
GESTIONE AFFOLLAMENTO E STATO IMPIANTI 17
Specifica OCL 17
GESTIONE CLASSIFICA GLOBALE 18
Specifica OCL 18
GESTIONE STORICO IMPIANTI 19
Specifica OCL 19
GESTIONE CREDENZIALI UTENTE 20
Specifica OCL 20
GESTIONE RECUPERO PASSWORD 22
Specifica OCL 22
GESTIONE ELIMINAZIONE ACCOUNT 23
Specifica OCL 23
DIAGRAMMA DELLE CLASSI COMPLETO 24

Scopo del documento

Il presente documento riporta la definizione dell'architettura del progetto SKI ONLINE 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.

Diagramma delle classi

Nel presente capitolo vengono presentate le classi previste nell'ambito del progetto SKI ONLINE. 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.

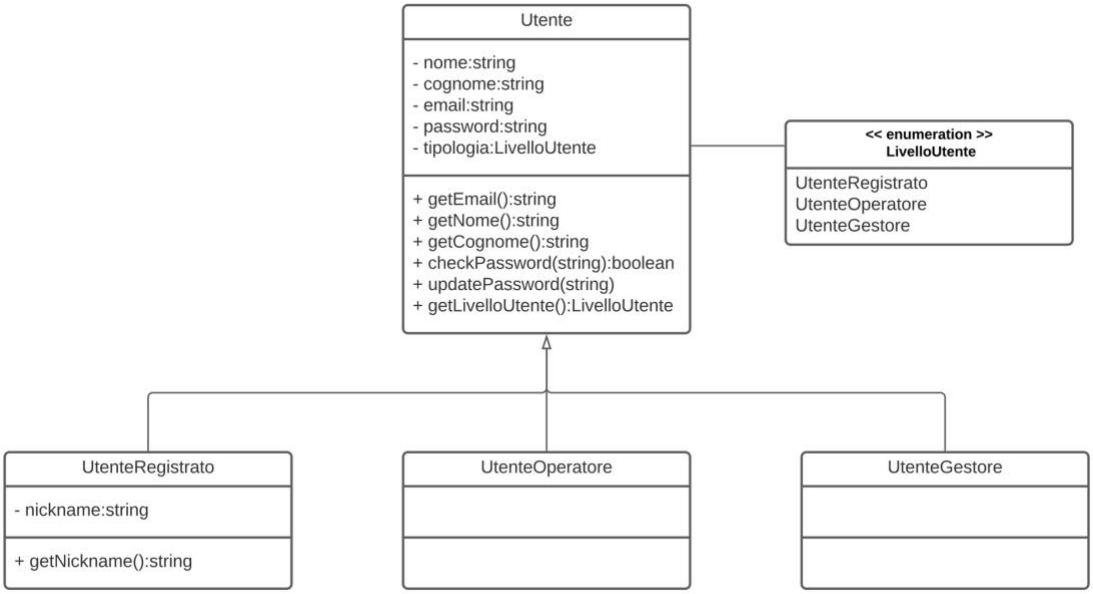
Riportiamo di seguito le classi individuate a partire dai diagrammi di contesto e dei componenti. In questo processo si è proceduto anche nel massimizzare la coesione e minimizzare l'accoppiamento tra classi.

Nella stesura del diagramma delle classi si è utilizzata la seguente convenzione sui nomi dei datatype: i datatype primitivi (interi, decimali, booleani, stringhe, …) sono stati indicati con la lettera iniziale minuscola, mentre i datatype definiti nel diagramma stesso (Data, Esito, …) sono stati indicati con la lettera iniziale maiuscola.

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, in particolare si è utilizzata la notazione utilizzata dalla pagina Wikipedia in italiano su OCL.

Utenti del sistema

Analizzando i vari livelli di utenza che sono previsti nel sistema, ovvero: “Utente Registrato”, “Utente Operatore” e “Utente Gestore”, sono state identificate tre classi corrispondenti ai tre livelli di utenza, accomunate da una classe madre comune “Utente”. Per introdurre un meccanismo di distinzione dei diversi livelli utente, senza fare affidamento su possibili implementazioni in linguaggi che supportano un operatore “instanceof”, è stata definita una enumeration che specifica i tre livelli corrispondenti alle tre classi figlie di Utente. Di seguito il dettaglio di queste classi con i propri attributi e metodi.



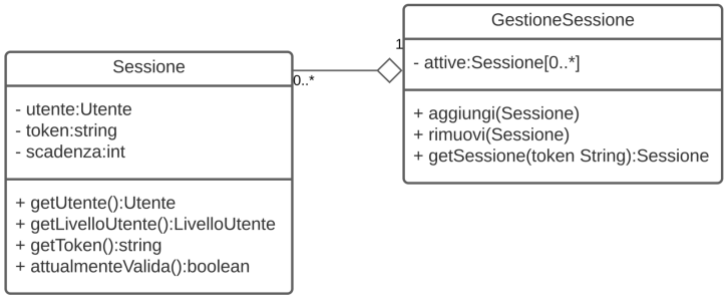
Specifica OCL

I campi email e password della classe “Utente” devono rispettare gli standard di sicurezza richiesti dal RNF 5. Queste condizioni sono espresse in OCL attraverso una invariante con questo codice:

```
context Utente inv: password.size() >= 8 and
password.exists(c | c = [1,2,..,9]) and
password.exists(c | c = [`!"?$%^&*_-+=:;@'~#|\<,>./]) and
email.exists(c = @)
```

Gestione sessione

Analizzando la componente “Gestione sessione” questa è stata scomposta in due classi “Sessione” e “Gestione sessione”, la prima si occupa di memorizzare i dati di una singola sessione mentre la seconda di raccogliere e gestire tutte le sessioni attualmente attive. Di seguito il dettaglio di queste classi con i propri attributi e metodi.



Il datatype “Utente” e l’enumeration “LivelloUtente” fanno riferimento a quelli definiti precedentemente

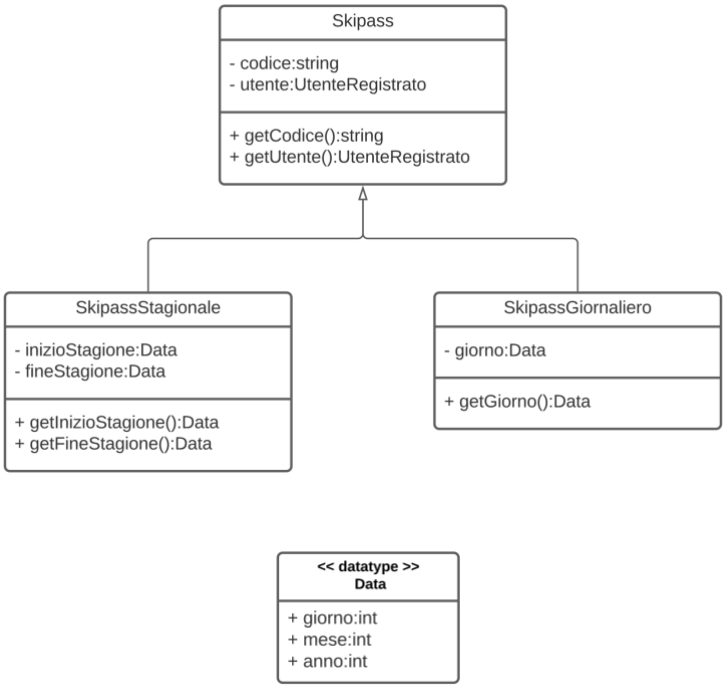
Specifica OCL

Per poter aggiungere una nuova sessione all’elenco delle sessioni attive, questa deve essere valida (ovvero non ancora scaduta). Questa condizione sulla classe “GestioneSessione” è espressa in OCL attraverso una pre-condizione con questo codice

```
context GestioneSessione::aggiungi(s:Sessione)
pre: s.attualmenteValida() = true
```

Skipass

Analizzando le due diverse tipologie di skipass acquistabili, ovvero “Skipass giornaliero” e “Skipass stagionale”, sono state identificate due classi corrispondenti alle due tipologie di skipass, accomunate da una classe madre “Skipass” comune. Di seguito il dettaglio di queste classi con i propri attributi e metodi.



Specifica OCL

Ogni skipass stagionale ha un periodo di validità a partire dall’inizio della stagione fino alla sua fine, per questo motivo una data di fine della stagione antecedente a quella di inizio non avrebbe senso logico. Questa condizione sulla classe “SkipassStagionale” è espressa in OCL attraverso una invariante con questo codice:

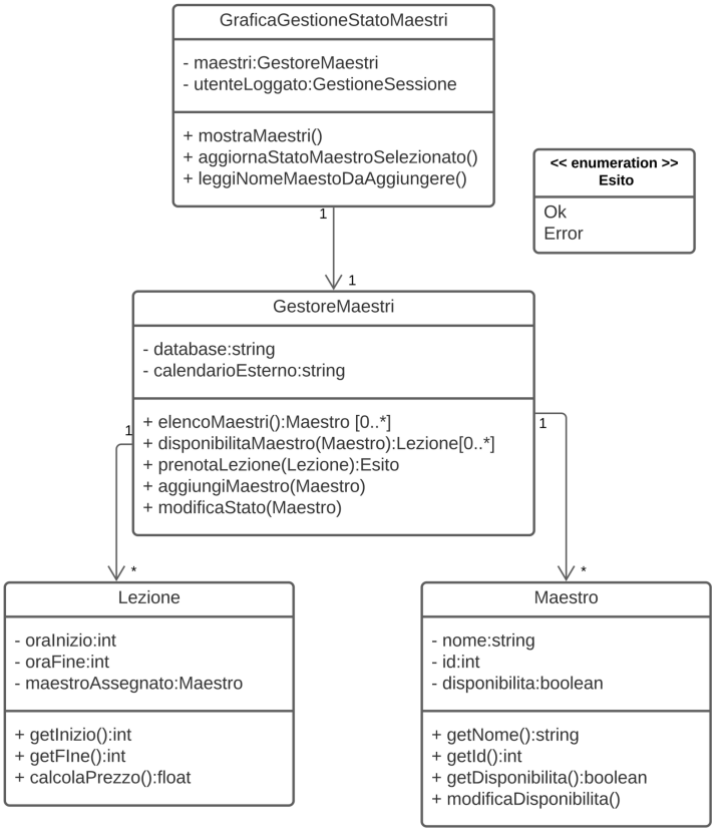
```
context SkipassStagionale inv:
inizioStagione < fineStagione
```

Inoltre, per quanto riguarda il datatype Data, il valore numerico del giorno e del mese devono essere coerenti con i valori ammissibili da calendario. Questa condizione è espressa in OCL attraverso una invariante con questo codice:

```
context Data inv:  1 <= giorno <= 31 AND 1 <= mese <= 12
```

Gestione maestri

Analizzando le componenti “Gestione maestri” e “Grafica gestione stato maestri”, sono state identificate quattro classi, due direttamente correlate alle due componenti, mentre le rimanenti “Lezione” e “Maestro” derivate dalla tipologia di dati che le prime due utilizzano. I due attributi della glasse GestoreMaestri si riferiscono al sistema esterno di database e al sistema esterno di calendario per la memorizzazione degli appuntamenti dei maestri. Di seguito il dettaglio di queste classi con i propri attributi e metodi.



Il tipo di dato “Gestione sessione” si riferisce alla classe definita precedentemente

Specifica OCL

Ogni lezione ha un orario di inizio in cui il maestro e l’allievo si incontrano e un orario di fine nel quale si salutano, per questo motivo un orario di fine antecedente all’orario di inizio non avrebbe senso logico. Questa condizione è espressa in OCL attraverso una invariante con questo codice:

```
context Lezione inv: oraInizio < oraFine
```

Un maestro ha uno stato di disponibilità ad effettuare lezioni rappresentato da un valore booleano, attraverso il metodo modifica disponibilità questo valore viene cambiato passando da disponibile a non disponibile oppure viceversa. Questa condizione è espressa in OCL attraverso una post-condizione con questo codice:

```
context Maestro::modificaDisponibilita()
post: disponibilita = not disponibilita@pre
```

Quando la classe “Gestore maestri” viene incaricata di prenotare una lezione, attraverso il suo metodo “prenotaLezione”, il maestro associato alla lezione deve avere la disponibilità, discussa nel punto precedente, a valore true (ovvero disponibile), al momento della registrazione nel calendario. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice:

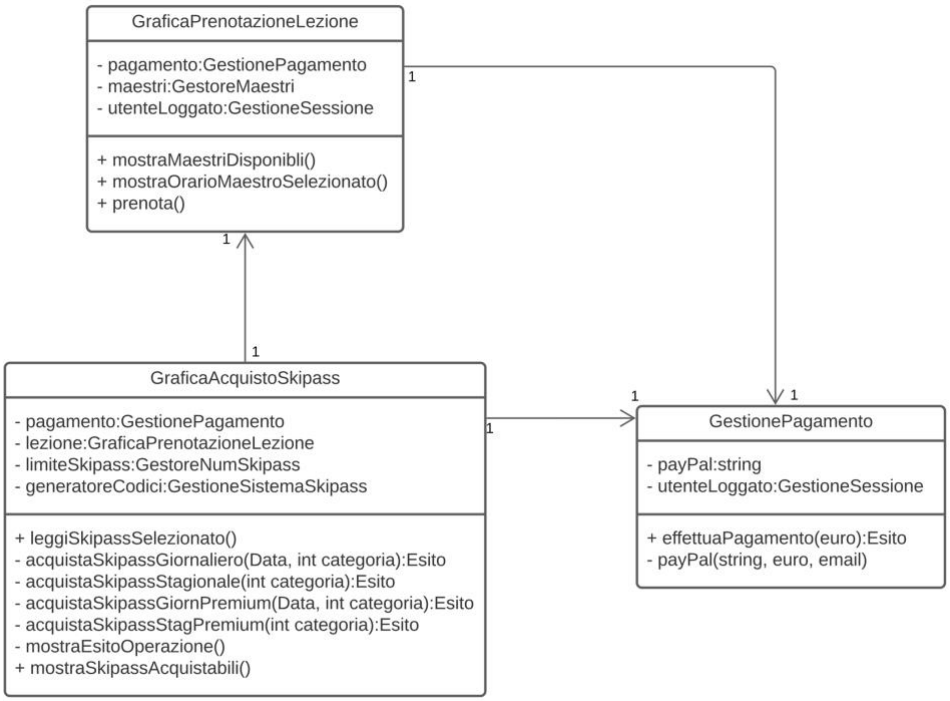
```
context GestioneMaestri::prenotaLezione(l:Lezione) pre:
l.getMaestroAssegnato().getDisponibilita() = true
```

Infine ogni metodo disponibile nella classe “GraficaGestioneStatoMaestri” può essere eseguito solo da un utente di livello “Utente Gestore”. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice, per semplicità è stata riportata la condizione solo su “mostraMaestri”, ma sono da intendersi altre due identiche su “aggiornaStatoMaestroSelezionato” e “leggiNomeMaestroDaAggiungere”:

```
context GraficaGestioneStatoMaestri::mostraMaestri()
pre: utenteLoggato.getSessione(String).getLivelloUtente() =
UtenteGestore
```

Acquisto skipass e prenotazione lezioni

Analizzando le componenti “Grafica acquisto skipass”, “Gestione pagamento” e “Grafica prenotazione lezione”, si è proceduto a identificare tre classi con le medesime funzionalità. La classe GraficaAcquistoSkipass si occupa di ricevere lo skipass scelto dall’utente tramite il metodo leggiSkipassSelezionato ed internamente viene scelto, in base alla scelta fatta dall’utente, quale metodo acquista richiamare. Inoltre, per gli skipass “premium” dopo l’acquisto l’utente verrà reindirizzato, tramite la classe GraficaPrenotazioneLezione, alla pagina per prenotare una lezione di sci. Di seguito il dettaglio di queste classi con i propri attributi e metodi.



I tipi di dato “Gestione sessione”, “Gestione maestri” ed “Esito” si riferiscono alle classi (ed enumeration) definite precedentemente

I tipi di dato “Gestione numero skipass” e “Gestione sistema skipass” si riferiscono a delle classi che verranno definite successivamente

Specifica OCL

Quando la classe “GraficoAcquistoSkipass” viene incaricata di acquistare uno skipass, attraverso uno dei quattro metodi acqustaSkipass, bisogna verificare che la richiesta di pagamento a GestionePagamento sia andata a buon fine. In caso affermativo la classe deve richiamare il metodo per la generazione dello skipass in GestioneNumeroSkipass corrispondente. Inoltre nel caso di acquisto di skipass giornalieri bisogna procedere con l’acquisto solo se non è stato raggiunto il limite di vendite in un dato giorno. Queste condizioni sono espresse in OCL attraverso una pre-condizione e una post-condizione con questo codice:

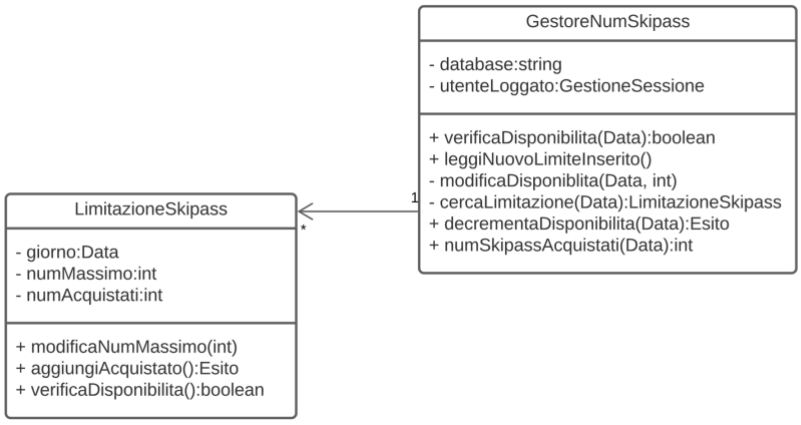
```
context GraficaAcquistoSkipass::acquistaSkipassGiornaliero(d:Data, int)
pre: limiteSkipass.verificaDisponibilita(d) = true
post: if(result = Ok) then generatoreCodici.generaSkipassGiornaliero(d)
context GraficaAcquistoSkipass::acquistaSkipassGiornalieroPremium(d:Data, int)
pre: limiteSkipass.verificaDisponibilita(d) = true
post: if(result = Ok) then generatoreCodici.generaSkipassGiornaliero(d)

context GraficaAcquistoSkipass::acquistaSkipassStagionale(int)
post: if(result = Ok) then generatoreCodici.generaSkipassStagionale()
context GraficaAcquistoSkipass::acquistaSkipassStagionalePremium(int)
post: if(result = Ok) then generatoreCodici.generaSkipassStagionale()
```

Inoltre, ogni metodo disponibile nella classe “GraficaPrenotazioneGestione” può essere eseguito solo da un utente di livello “Utente Registrato”. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice, per semplicità è stata riportata la condizione solo su “mostraMaestriDisponibili”, ma sono da intendersi altre due identiche su “mostraOrarioMaestroSelezionato” e prenota:

```
context GraficaPrenotazioneLezione::mostraMaestriDisponibili()
pre: utenteLoggato.getSessione(String).getLivelloUtente() =
UtenteRegistrato
```

Limitazione numero skipass acquistabili

Analizzando la componente “Gestione numero skipass acquistabili” questa è stata scomposta in due classi “GestoreNumeroSkipassAcquistabili” e “LimitazioneSkipass”, la prima si occupa di aggiungere, rimuovere e modificare limitazioni in diverse giornate, mentre la seconda si occupa di memorizzare la singola limitazione in una giornata.



I tipi di dato “Data”, “Gestione sessione” ed “Esito” si riferiscono alle classi definite precedentemente

Specifica OCL

Nella classe “LimitazioneSkipass” l’attributo “numMassimo” indica il numero massimo di skipass acquistabili nel giorno specificato nell’attributo corrispondente, mentre l’attributo “numAcquistati” indica il numero di skipass già acquistati sempre nel giorno indicato. Per questo motivo un numero massimo di skipass acquistabili minore del numero di skipass effettivamente acquistati genererebbe una evidente contraddizione e non si deve poter verificare. Questa condizione è espressa in OCL attraverso una invariante con questo codice:

```
context LimitazioneSkipass inv:
numMassimo >= numAcquistati
```

Sempre nella classe “LimitazioneSkipass” quando si richiede di incrementare il numero di skipass già acquistati bisogna verificare che la disponibilità sia ancora positiva e il numero deve essere subito aggiornato, onde evitare problemi di accesso concorrente. Queste condizioni sono espresse in OCL attraverso una pre-condizione e una post-condizione con questo codice:

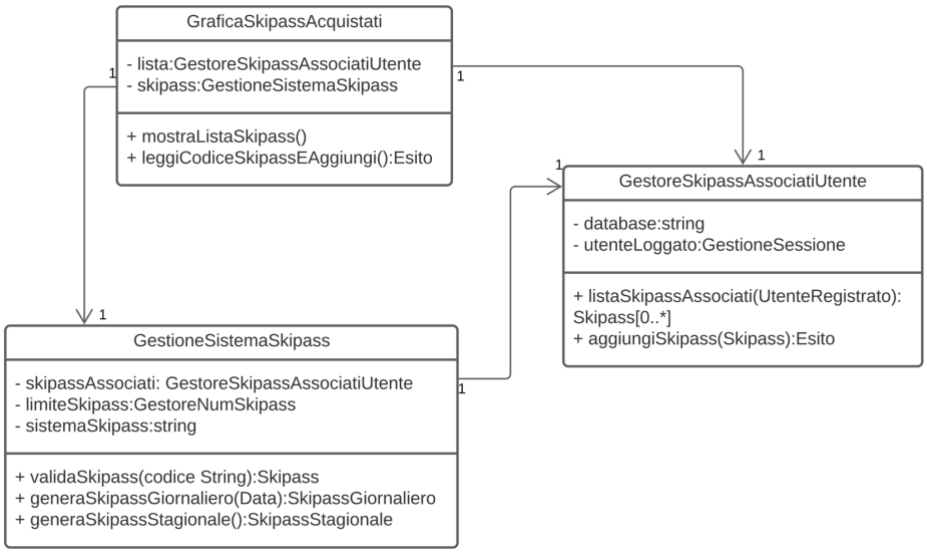
```
context LimitazioneSkipass::aggiungiAcquistato()
pre: verificaDisponibilita() = true
post: numAcquistati = numAcquistati@pre + 1
```

Infine nella classe “GestoreNumSkipass” la modifica della disponibilità del numero massimo di skipass acquistabili in un giorno può essere effettuata solo da un utente di livello “Utente Gestore”, per questo motivo il metodo “leggiNuovoLimiteInserito” (e di conseguenza “modificaDisponibilità” che viene richiamato) può essere eseguito solo da un utente gestore. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice:

```
context GestoreNumSkipass::leggiNuovoLimiteInserito()
pre: utenteLoggato.getSessione(string).getLivelloUtente() = UtenteGestore
```

Gestione skipass associati ad una utenza

Analizzando le componenti “Gestione sistema skipass preesistente”, “Grafica skipass acquistati” e “Gestione skipass associati ad una utenza” si è proceduto a identificare tre classi con le medesime funzionalità. L’attributo sistemaSkipass della classe GestioneSistemaSkipass si riferisce al sistema esterno di controllo skipass già esistente e funzionante utilizzato nella skiarea. Di seguito il dettaglio di queste classi con i propri attributi e metodi.



I tipi di dato “GestoreNumSkipass”, “Gestione sessione”, “Utente Registrato”, “Esito”, “Data”, “Skipass”, “Skipass giornaliero”, “Skipass stagionale” si riferiscono alle classi definite precedentemente

Specifica OCL

Quando la classe “GestioneSistemaSkipass” viene incaricata di validare un codice skipass, oppure di generare uno skipass stagionale, se l’esito della validazione nel primo caso, l’esito della generazione nel secondo, è andato a buon fine è necessario che lo skipass venga aggiunto al database e associato all’utente, attraverso il metodo “aggiungiSkipass” del Gestore Skipass Associati Utente. Questa condizione è espressa in OCL attraverso una post-condizione con questo codice:

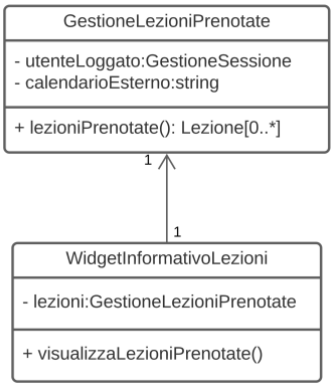
```
context GestioneSistemaSkipass::validaSkipass(string)
post: if(result->size() = 1) then skipassAssociati.aggiungiSkipass(result)

context GestioneSistemaSkipass::generaSkipassStagionale()
post: if(result->size() = 1) then skipassAssociati.aggiungiSkipass(result)
```

Inoltre, sempre la classe “GestioneNumSkipass”, nel caso in cui venga incaricata di generare uno skipass giornaliero, oltre alla post condizione descritta precedentemente lo skipass ottenuto deve essere valido nel giorno richiesto all’utente e la generazione può essere effettuata solo se la richiesta di incremento del numero di skipass già acquistati è terminata correttamente. Queste condizioni sono espresse in OCL attraverso una pre-condizione e una post-condizione con questo codice:

```
context GestioneSistemaSkipass::generaSkipassGiornaliero(d:Data)
pre: limiteSkipass.decrementaDisponibilita(d) = Ok
post: result.getGiorno() = d and if(result->size() = 1) then
skipassAssociati.aggiungiSkipass(result)
```

Widget informativo lezioni

Analizzando le componenti “Widget informativo lezioni” e “Gestione lezioni prenotate” si è proceduto a identificare due classi con le medesime funzionalità. Di seguito il dettaglio di queste classi con i propri attributi e metodi.



I tipi di dato “Lezione” e “Gestione sessione” si riferiscono alle classi definite precedentemente

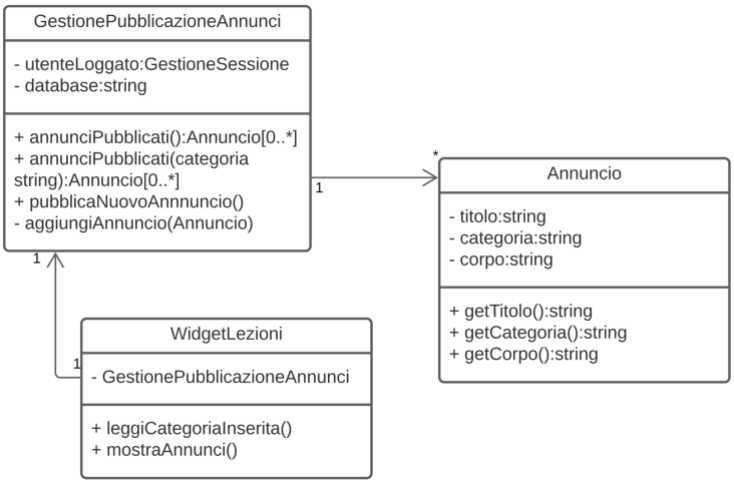
Specifica OCL

Il metodo “lezioniPrenotate” della classe “GestioneLezioniPrenotate” può essere eseguito solamente da un utente di livello “Utente Registrato”. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice:

```
context GestioneLezioniPrenotate::lezioniPrenotate()
pre: utenteLoggato.getSessione(string).getLivelloUtente() = UtenteRegistrato
```

Widget annunci pubblicati

Analizzando le componenti “Widget annunci” e “Gestione pubblicazione annunci” si è proceduto a identificare due classi con le medesime funzionalità e una classe “Annuncio” che memorizza il singolo annuncio. Di seguito il dettaglio di queste classi con i propri attributi e metodi.



Il tipo di dato “Gestione sessione” si riferisce alla classe definite precedentemente

Specifica OCL

Un Annuncio deve contenere per forza un titolo, che lo identifica, e un corpo, mentre il campo categoria è opzionale e quindi non obbligatorio nell’inserimento. Questa condizione è espressa in OCL attraverso una invariante con questo codice:

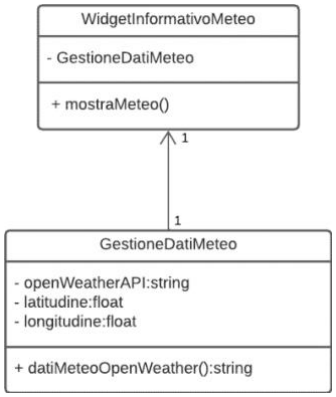
```
context Annuncio inv:
(titolo <> "") and (corpo <> "")
```

Inoltre, il metodo “pubblica nuovo annuncio” (e di conseguenza aggiungiAnnuncio che viene richiamato) può essere eseguito solo da un utente di livello “Utente Gestore”. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice:

```
context GestionePubblicazioneAnnunci::pubblicaNuovoAnnuncio()
pre: utenteLoggato.getSessione(string).getLivelloUtente() = UtenteGestore
```

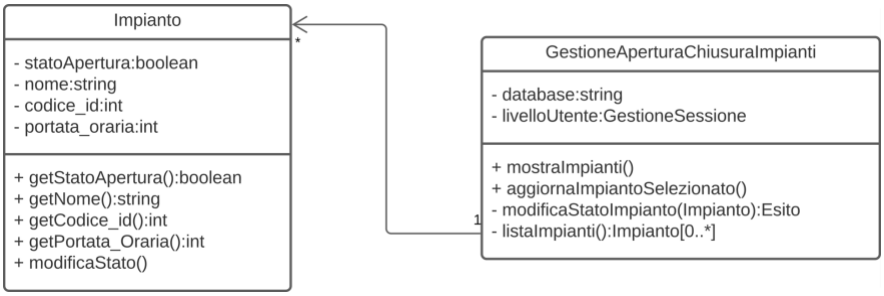
Gestione widget meteorologico

Analizzando le componenti “Widget informativo meteo” e “Gestione dati meteo” queste sono state identificate come classi per la gestione e visualizzazione del widget dedicato alla visione delle condizioni meteorologiche. Di seguito il dettaglio di queste classi con i propri attributi e metodi.



Gestione apertura e chiusura impianti

Analizzando la componente “Gestione apertura e chiusura impianti” si è proceduto ad identificare la relativa classe con le medesime funzionalità. Inoltre, è stata individuata una classe “Impianto” che memorizza un impianto della stazione sciistica. Di seguito tutti i dettagli delle classi con i relativi attributi e metodi.



Il tipo di dato “Gestione sessione” si riferisce alla classe definite precedentemente

Specifica OCL

Un impianto non può avere una portata oraria negativa; lo stesso ragionamento si applica al codice identificativo che non sarà mai un numero intero negativo. Questa condizione è espressa in OCL attraverso una invariante con questo codice:

```
context Impianto inv:
portata_oraria>0 AND codice_id>=0
```

Inoltre, il metodo “modificaStato” al termine della sua esecuzione deve aver modificato lo stato dell’impianto a cui fa riferimento da aperto a chiuso o viceversa. Questa condizione è espressa in OCL attraverso una post condizione con questo codice:

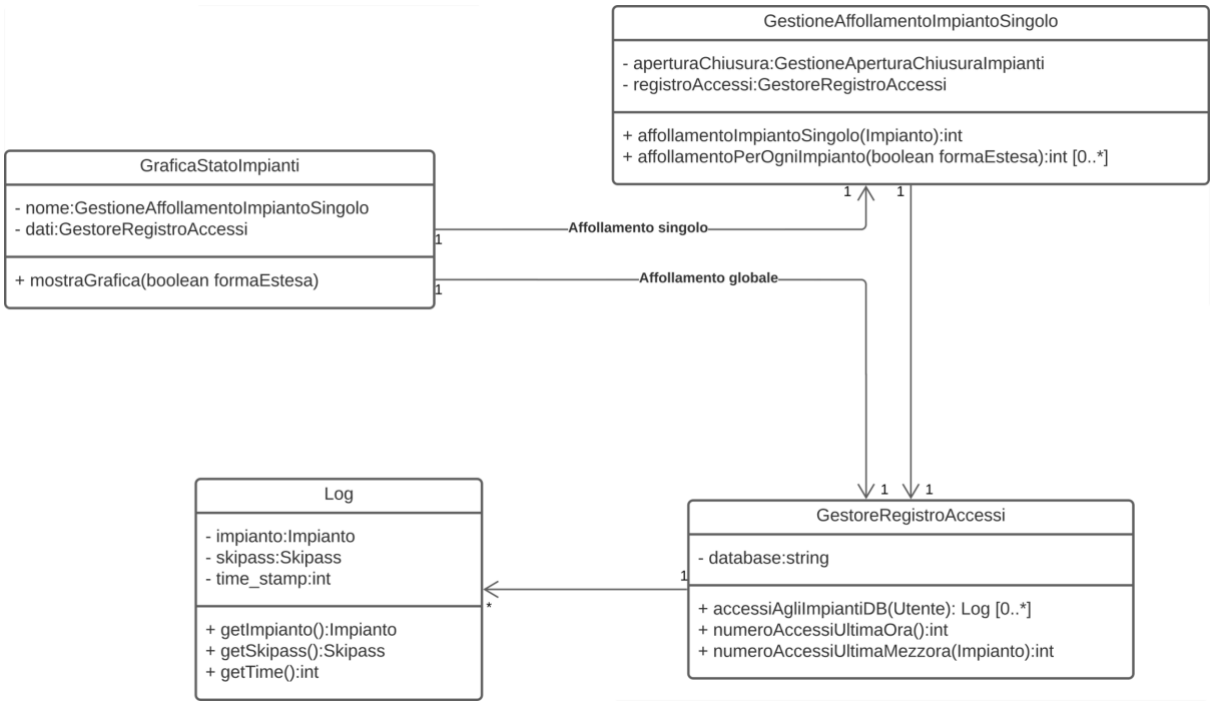
```
context Impianto::modificaStato() post:
statoApertura = not statoApertura@pre
```

Infine, ogni metodo disponibile nella classe “GestioneAperturaChiusuraImpianti” può essere eseguito solo da un utente di livello “Utente Gestore”. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice, per semplicità è stata riportata la condizione solo su “mostraImpianti”, ma è da intendersi identica anche per “aggiornaImpiantoSelezionato”:

```
context GestioneAperturaChiusuraImpianti::mostraImpianti()
pre: livelloUtente.getSessione(string).getLivelloUtente() = UtenteGestore
```

Gestione affollamento e stato impianti

Analizzando le componenti “Gestione affollamento impianto singolo”, “Grafica stato impianti” e “Gestione Registro Accessi” si è proceduto a identificare tre classi con le medesime funzionalità. Inoltre, è stata individuata una classe “Log” che memorizza un accesso ad un impianto sciistico. Di seguito i dettagli riguardanti queste classi con i relativi metodi ed attributi.



I tipi di dato “Skipass”, “Impianti” e “Gestione apertura e chiusura impianti” si riferiscono a classi definite precedentemente

Il tipo di dato int[0..] indica un array di interi il cui indice corrisponde al codice univoco di ciascun impianto; in questo caso, ad esempio, per ottenere l’affollamento per l’impianto 0, allora questo valore è in corrispondenza dell’indice 0 dell’array .*

Specifica OCL

Un Log non può presentare un “time_stamp” con valore negativo. Questa condizione è espressa in OCL attraverso una invariante con questo codice:

```
context Log inv:
time_stamp>0
```

Nella classe “GestoreRegistroAccessi” sia il metodo “numeroAccessiUltimaOra” che “numeroAccessiUltimaMezzora” non posso ritornare un valore negativo. Questa condizione è espressa in OCL attraverso una pre-condizione con questi codici:

```
context GestoreRegistroAccessi::numeroAccessiUltimaOra()
post: result>=0
```

```
context GestoreRegistroAccessi::numeroAccessiUltimaMezzora(Impianto)
post: result>=0
```

Gestione classifica Globale

Analizzando la componente “Gestione classifica Globale” si è proceduto ad identificare la relativa classe responsabile della visualizzazione della classifica globale con le medesime funzionalità. Di seguito tutti i dettagli riguardante la classe con i relativi attributi e metodi.

I tipi di dato “Gestione sessione” e “Gestore registro accessi” si riferiscono a classi definite precedentemente

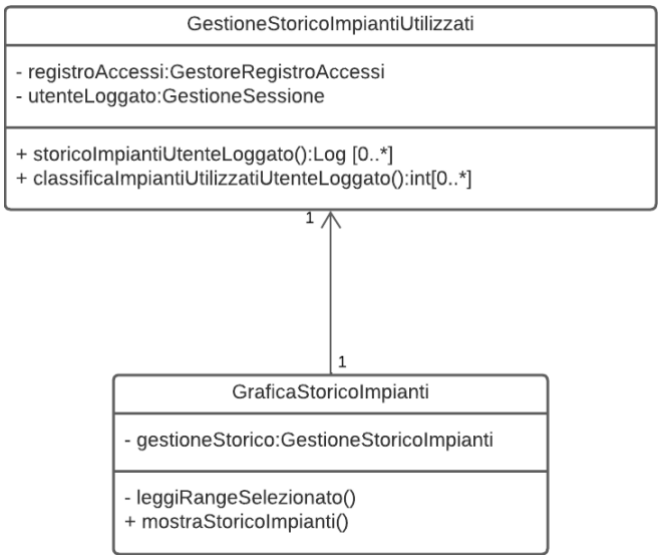
Specifica OCL

Il metodo “elaboraAndMostraClassifica” nella classe “GestioneClassificaGlobale” può essere eseguito solamente da un utente loggato, ovvero la cui sessione non è nulla. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice:

```
context GestioneClassificaGlobale::elaboraAndMostraClassifica()
pre: livelloUtente.getSessione(string)->size()<>0
```

Gestione storico impianti

Analizzando le componenti “Gestione storico impianti utilizzati” e “Grafica storico impianti” si è proceduto ad identificare le relative classi con le medesime funzionalità. Di seguito tutti i dettagli riguardanti le classi e i relativi attributi e metodi.



I tipi di dato “Gestione sessione” e “Log” si riferiscono a classi definite precedentemente

Il tipo di dato int[0..] indica un array di interi il cui indice corrisponde al codice univoco di ciascun impianto; in questo caso, ad esempio, nella posizione 0 dell’array, è presente il numero di volte che l’utente ha preso l’impianto 0.*

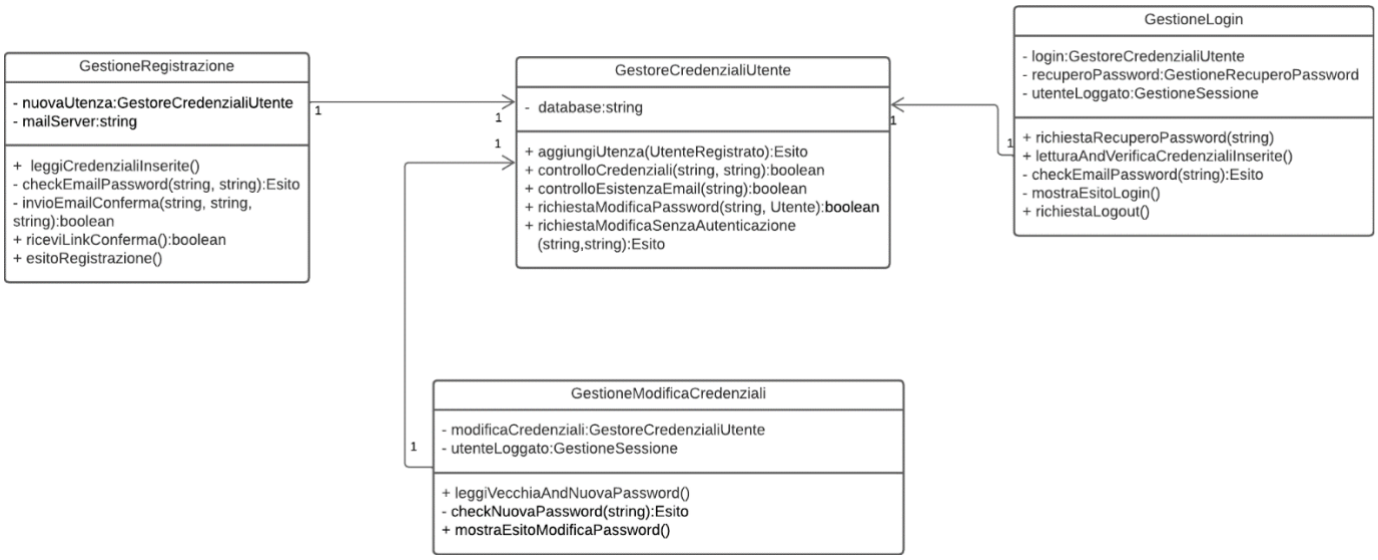
Specifica OCL

Il metodo “mostraStoricoImpianti” nella classe “GestioneStoricoImpiantiUtilizzati” può essere eseguito solo da un utente di livello “Utente Registrato”. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice:

```
context GestioneAperturaChiusuraImpiantiUtilizzati::mostraStoricoImpianti()
pre: utenteLoggato.getSessione(string).getLivelloUtente() = UtenteRegistrato
```

Gestione credenziali utente

Analizzando le componenti “Gestore credenziali utente”, “Gestione registrazione”, “Gestione login” e “Gestione modifica credenziali” si è proceduto ad identificare quattro classi con le medesime funzionalità. Di seguito i dettagli riguardanti queste classi con i relativi metodi ed attributi.



I tipi di dato “Gestione sessione” ed “Esito” si riferiscono alle classi (ed enumeration) definite precedentemente

Specifica OCL

L’email inserita dall’utente verrà inviata al server mail solo nel caso in cui i campi “email” e “password” siano stati inseriti nel formato corretto; perciò, solo se il metodo “checkEmailPassword” della classe “GestioneRegistrazione” restituisce il valore “Ok”, allora viene eseguito il metodo “invioEmailConferma” della stessa classe. Questa condizione è espressa in OCL tra una post-condizione con questo codice:

```
context GestioneRegistrazione::checkEmailPassword(email:string, string)
post: if(result = Ok) then invioEmailConferma(email, string, string)
```

La nuova utenza verrà aggiunta nel database esterno solo nel caso in cui il sistema abbia ricevuto correttamente la conferma di validazione dell’indirizzo email (tramite link); perciò solo se il metodo “riceviLinkConferma” della classe “GestioneRegistrazione” restituisce il valore booleano “true”, allora viene richiamato il metodo "aggiungiUtenza” della classe “nuovaUtenza”. Questa condizione è espressa in OCL tramite una post-condizione con questo codice:

```
context GestioneRegistrazione::riceviLinkConferma()
post: if(result = true) then nuovaUtenza.aggiungiUtenza(UtenteRegistrato)
```

Il metodo “richiestaModificaPassword” della classe “GestoreCredenzialiUtente” può essere eseguito solo da un utente attualmente loggato. Questa condizione è espressa in OCL attraverso una pre-condizione con questo codice:

```
context GestoreCredenzialiUtente::richiestaModificaPassword(string,utente)
pre: utenteLoggato.getSessione(string)->size() = 1
```

Affinché venga eseguito il metodo “controlloCredenziali” della classe “GestoreCredenzialiUtente”, la password e l’email inserite dall’utente devono rispettare il formato previsto dal sistema, ovvero, il metodo “checkEmailPassword” della classe “GestioneLogin” deve restituire il valore “Ok”. Questa condizione è espressa in OCL tra una post-condizione con questo codice:

```
context GestioneLogin::checkEmailPassword(pwd:string)
post: if(result = Ok) then login.controlloCredenziali(string, pwd)
```

La richiesta di recupero password può essere inoltrata solo se viene confermata l’esistenza dell’email con cui l’utente fa la richiesta; perciò il metodo “richiestaRecuperoPassword” della classe “GestioneLogin” viene eseguito solo se il metodo “controlloEsistenzaEmail” della classe “GestoreCredenzialiUtente” restituisce esito positivo, ovvero valore “true”. Questa condizione è espressa in OCL tramite una pre-condizione con questo codice:

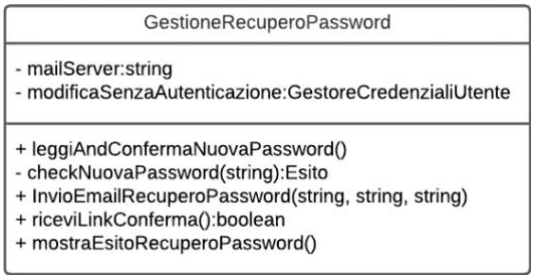
```
context GestioneLogin::richiestaRecuperoPassword(email:string)
pre: login.controlloEsistenzaEmail(email) = true
```

Affinché si possano modificare le credenziali di un utente, questo deve essere innanzitutto loggato, ovvero avere una sessione attiva. Condizione espressa in OCL tramite una pre-condizione.
Inoltre, per modificare la password attuale con una nuova, quest’ultima dovrà rispettare il formato previsto dal sistema, ovvero, solamente se il metodo “checkNuovaPassword” della classe “GestioneModificaCredenziali” ritorna valore “Ok”, allora sarà possibile eseguire il metodo “richiestaModificaPassword” della classe “modificaCredenziali”. Condizione espressa il OCL tramite una post-condizione.
Entrambe le condizioni descritte precedentemente sono espresse nel seguente codice:

```
context GestioneModificaCredenziali::checkNuovaPassword(string)
pre: utenteLoggato.getSessione(string)->size() = 1
post: if(result = Ok) then modificaCredenziali.richiestaModificaPassword(Utente)
```

Gestione recupero password

Analizzando la componente “Gestione recupero password” si è proceduto a identificare questa classe con le medesime funzionalità. Di seguito i dettagli riguardanti questa classe con i relativi metodi ed attributi.



Il tipo di dato “GestoreCredenzialiUtente” ed “Esito” si riferiscono alle classi (ed enumeration) definite precedentemente

Specifica OCL

L’email per il recupero password verrà inviata al server mail solo nel caso in cui la nuova password sia stata inserita nel formato corretto; perciò, solo se il metodo “checkEmailPassword” della classe “GestioneRecuperoPassword” restituisce il valore “Ok”, allora viene eseguito il metodo “invioEmailRecuperoPassword” della stessa classe. Questa condizione è espressa in OCL tra una post-condizione con questo codice:

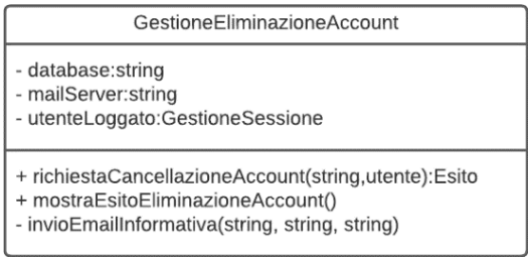
```
context GestioneRecuperoPassword::checkNuovaPassword(string)
post: if(result = Ok) then invioEmailRecuperoPassword(string, string, string)
```

La modifica delle credenziali avviene solo nel caso in cui il sistema abbia ricevuto correttamente la conferma dell’indirizzo email di recupero (tramite link); perciò se il metodo “riceviLinkConferma” della classe “GestioneRecuperoPassword” restituisce il valore “true” allora viene eseguito il metodo "richiestaModificaSenzaAutenticazione” della classe “GestoreCredenzialiUtente”. Questa condizione è espressa in OCL tramite una post-condizione con questo codice:

```
context GestioneRecuperoPassword:riceviLinkConferma()
post: if(result = true) then
modificaSenzaAutenticazione.richiestaModificaSenzaAutenticazione(string, string)
```

Gestione eliminazione account

Analizzando la componente “gestione eliminazione account” si è proceduto a identificare questa classe con le medesime funzionalità. Di seguito i dettagli riguardanti questa classe con i relativi metodi ed attributi.



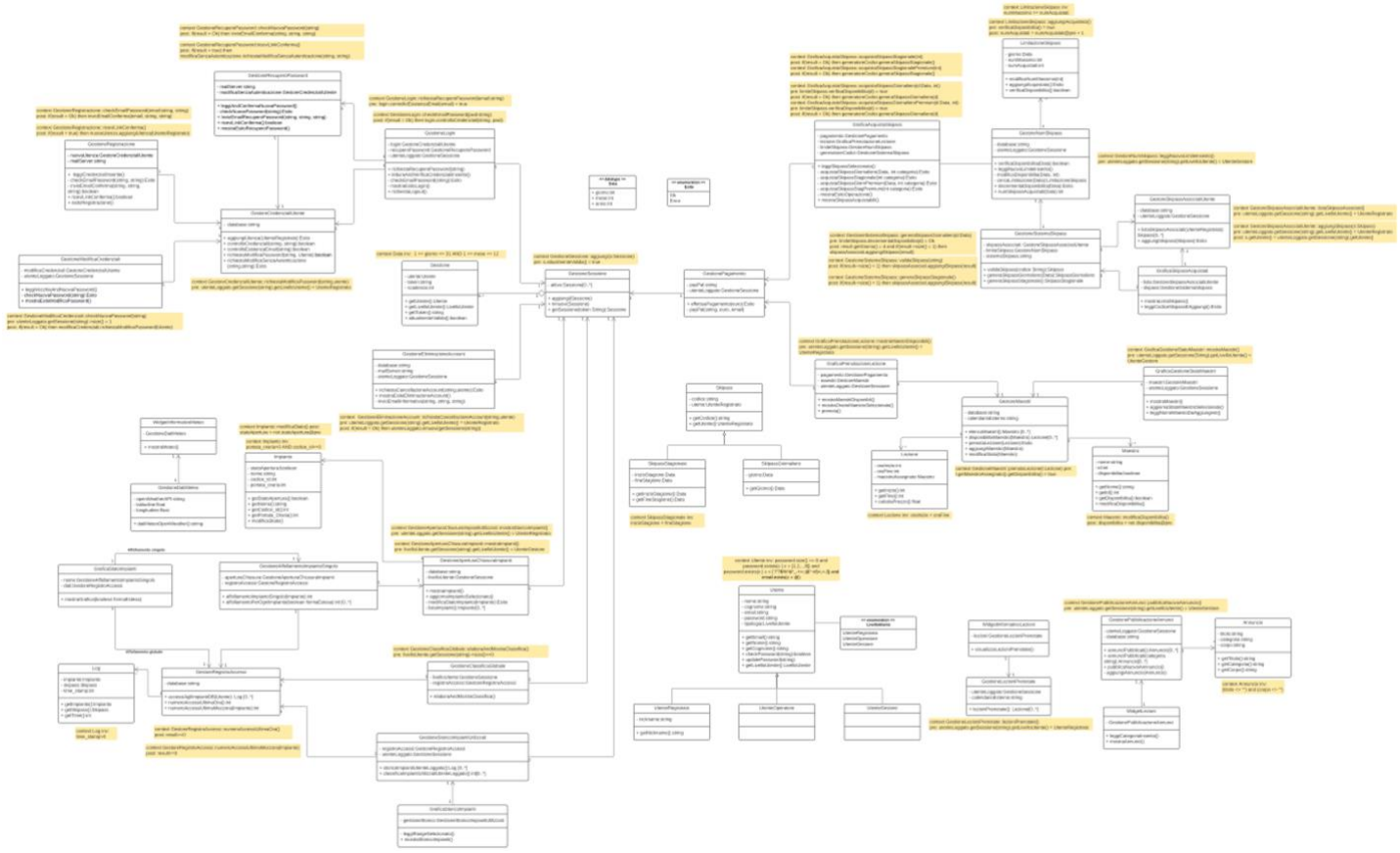
Il tipo di dato “GestioneSessione” ed “Esito” si riferiscono alle classi (ed enumeration) definite precedentemente

Specifica OCL

Il metodo “richiestaCancellazioneAccount” della classe “GestioneEliminazioneAccount” può essere eseguito solamente da un utente di livello “Utente Registrato”. Questa condizione è espressa in OCL attraverso una pre-condizione.
Inolte, in seguito alla cancellazione dell’account deve essere effettuato in automatico il log-out. Questa condizione è espressa in OCL attraverso una post-condizione.
Entrambe le condizioni descritte precedentemente sono espresse nel seguente codice:

```
context  GestioneEliminazioneAccount::richiestaCancellazioneAccount(string,utente)
pre: utenteLoggato.getSessione(string).getLivelloUtente() = UtenteRegistrato
post: if(result = Ok) then utenteLoggato.rimuovi(getSessione(string))
```

Diagramma delle classi completo



Una versione salvata in alta risoluzione del diagramma delle classi è disponibile al link di seguito: https://github.com/ski-online/Deliverable/blob/main/Diagramma%20delle%20classi%20-%20AltaRisoluzione.svg