UNIVERSITÀ DEGLI STUDI DI SALERNO

INGEGNERIA DEL SOFTWARE

Object Design Document (ODD)

Coordinatori del Progetto
Prof.ssa Filomena Ferrucci
Liliana Annunziata
Raffaele Costantino

Partecipanti
Alessandro Kevin Barletta
Edoardo Carpentiero
Stefano Cirillo
Andrea De Maio
Gianmarco Mucciariello
Antonio Pizza
Alberto Sergio
Gianmaria Trezza

Revision History

Data	Versione	Descrizione	Autori
07/12/2015	0.5	Introduzione Linee guida per l'implementazione Design pattern Packages	Antonio Pizza Stefano Cirillo Andrea De Maio Alberto Sergio Gianmaria Trezza
09/12/2015	1.0	Packages Interfacce	Antonio Pizza Stefano Cirillo Andrea De Maio Alberto Sergio Gianmaria Trezza
10/12/2105	1.1	Aggiunta Package e Interfaccia gestione autenticazione	Andrea De Maio Alberto Sergio Gianmaria Trezza
11/12/2015	1.2	Modifica Package e Interfaccia gestione account	Andrea De Maio Alberto Sergio Gianmaria Trezza
22/12/2015	2.0	Revisione ODD	Andrea De Maio
07/01/2016	3.0	Revisione ODD	Edoardo Carpentiero
22/01/2016	3.1	Controllo ODD	Andrea De Maio

Sommario

1. Introduzione

Questo documento, usato come supporto dell'implementazione, ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutti i servizi individuati nelle fasi precedenti. In particolare definisce le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre nei paragrafi successivi sono specificati i trade-off, le linee guida e i design pattern per l'implementazione.

1.1 Tradeoff nell'Object Design

● Comprensibilità vs Tempo
La comprensibilità del codice è un aspetto molto importante soprattutto per la fase di testing. Ogni classe e metodo deve essere di facile interpretazione anche per chi non ha collaborato al progetto.
Nel codice si useranno i commenti standard per rendere il codice sorgente sempre più comprensibile.
Ovviamente questa caratteristica aggiungerà dei costi allo sviluppo del nostro progetto.

● Prestazioni vs Costi
Non avendo a disposizione alcun budget, utilizziamo materiale open source per la realizzazione del software Pr.D con lo scopo di rendere il sistema performante ed efficiente.

● Sicurezza vs Efficienza
La gestione della sicurezza viene affidata all'utilizzo del login iniziale in quanto va ad autenticare l'utente al quale sarà visualizzata solo la parte del software che gli appartiene, evitando così incongruenze di dati. Il login funziona in una ricerca all'interno del database nella sezione account dove il sistema cercherà se esiste il nome utente e se questo corrisponde alla password inserita. Questa politica di permessi, permette di non appesantire eccessivamente il software ed è un buon compromesso tra sicurezza ed efficienza.

● Tempo di risposta vs Spazio di memoria
La scelta di utilizzare un DB relazionale è scaturita dai diversi vantaggi che se ne derivano:
- Gestione consistente dei dati;
- Tempo di risposta basso rispetto all'utilizzo del file system;
- Accesso veloce e concorrente ai dati;

Lo svantaggio nell'utilizzo di un DB relazionale è che richiede il triplo dello spazio di memoria rispetto ad un file system e si trovano difficoltà nel momento in cui si vogliono gestire enormi quantità di dati.

● **Costi vs Mantenimento**
L’utilizzo di materiale open source rende il sistema facilmente modificabile e manutenibile. Trattandosi di un sistema distribuito i costi di manutenzione saranno alti.

1.2 Riferimenti

- Pr.D_RAD_2.1
- Pr.D_SDD_1.0
- Pr.D_DBD_1.0

2. Linee guida per l’implementazione

2.1 Nomi di file

La pagina web Pr.D. utilizza i seguenti suffissi per i file:

- Per i sorgenti JavaScript è .js
- Per i sorgenti PHP è .php
- Per i sorgenti HTML è .html
- Per i sorgenti CSS è .css

2.2 Organizzazione dei file

Un file consiste di sezioni che dovrebbero essere separate da linee bianche e un commento opzionale che identifica ogni sezione. File più lunghi di 2000 linee sono ingombranti e devono essere evitati.

2.2.1 File sorgenti

Ogni file sorgente PHP contiene una singola classe pubblica, o un’interfaccia. Quando ci sono classi e interfacce private associate con la classe pubblica, è possibile inserirle nello stesso file sorgente della classe pubblica. La classe pubblica deve essere la prima classe o interfaccia nel file.

2.3 Classi e interfacce PHP

Nella scrittura di codice per le classi PHP ci si atterrà agli standard della programmazione Object Oriented nella sua interezza focalizzando l'attenzione su:

- Convenzioni sui nomi
- Struttura dei file
- Accesso alle variabili
- Commenti speciali
- Utilizzo degli spazi bianchi

2.3.1 Commenti

In PHP, come in altri linguaggi è possibile inserire commenti al codice.
I commenti si rivelano di importanza decisiva quando si tratta modificare programmi sviluppati da altri, oppure da noi stessi, soprattutto se è passato qualche tempo dalla realizzazione. I commenti svolgono un ruolo fondamentale in questa fase di manutenzione del codice, in quanto possono facilitare di molto la comprensione di passaggi apparentemente oscuri.

I commenti in PHP possono essere rappresentati nel codice in 3 modi diversi:

1. Caratterizzati da due barre:

```php
<?php
    // Commento
?>
```

2. Contraddistinti dall'uso del cancelletto:

```php
<?php
    # Commento

?>
```

3. L'uso di commenti multiriga:

```php
<?php

        /*
        Questo è un commento
        multiriga

         */

?>
```

2.3.2 Istruzioni

Le istruzioni in php sono rappresentate:

- in file .php incorporate in codice html con il tag <SCRIPT>
    - <SCRIPT language="php">
      </SCRIPT>
- in file .php incorporate in codice html con la sintassi:
    - <BODY>
      <?php
      ?>
      </BODY>
- in file .php con la sintassi:
    - <?php
      ?>

2.3.2.1 Istruzioni di include e require

Le prime linee dello script in PHP contengono le istruzioni di **include** e di **require** che hanno lo scopo di evitare le ripetizioni di codice scrivendo le istruzioni una sola volta all'interno di un file che sarà poi incluso all'interno di tutti gli script che necessitano di quel codice.

2.3.3 Convenzioni di Nomi

La pagina web utilizza per file contenenti solo codice PHP e per file con codice PHP incorporato in codice HTML l'estensione **.php .**

2.3.3.4 Classi

Nella scrittura di codice per le classi PHP, ci si atterrà allo standard nella sua interezza, con particolare attenzione a:

1. Convenzioni sui nomi
2. Struttura dei file
3. Accesso alle variabili
4. Commenti speciali
5. Utilizzo degli spazi bianchi

Si praticano, inoltre, le seguenti restrizioni e variazioni:
All'inzio di ogni file devono essere presenti alcune informazioni strutturate come segue:

```
/**
        Gestione

        Descrizione Classe

        Author: Nome Cognome
        Version : 1.0
        2015 - Copyright by Pr.D Project - University of Salerno
*/
```

2.3.3.5 Interfacce

Le interfacce, sono delle classi speciali che non possono essere istanziate ma soltanto implementate. Esse delineano la struttura di una classe. Qualsiasi classe implementi un'interfaccia, avrà le costanti definite automaticamente e dovrà a sua volta i metodi dell'interfaccia, i quali sono tutti astratti e vanno tutti implementati. Le interfacce sono dichiarate:

```
<?php
    // definizione dell'interfaccia
    interface NomeInterfaccia{
            //codice
     }
?>
```

Nelle interfacce vengono definiti soltanto i metodi astratti e le costanti.

2.3.3.6 Metodi

I metodi nelle classi PHP devono rispettare lo standard del PHP Object Oriented:

```
class Classe{
        public function nomeFunzione() {
            //corpo funzione
        }
}
```

2.3.3.7 Variabili

La variabile è una porzione di memoria in cui viene memorizzato un dato che può variare durante l'esecuzione di un programma. Le variabili non devono essere necessariamente dichiarata prima del suo utilizzo. Questo consente, quindi, di creare variabili immediatemente al momento in cui se ne ha bisogno

$variabile=5;

2.3.3.8 Costanti

Una costante è un identificatore per un valore. Come si può intuire, tale valore non può cambiare durante l'esecuzione dello script. E' case-sensitive per default ed è convenzione comune che i nomi di costanti siano sempre maiuscoli. Per "etica" il nome di una costante inizia quasi sempre con una lettera o underscore seguita da un numero qualsiasi di caratteri alfanumerici o underscore.

define("_MAX", "100");
define("_MIN", "5");

2.4 Pagine HTML

2.4.1 Commenti

I commenti possono essere considerati una strategia per rendere il nostro codice più leggibile nei punti più significativi. Si tratta, infatti, di indicazioni significative, ma invisibili al browser, che inserite in punti specifici del documento ci permette di mantenere l'orientamento anche in file molto lunghi e complessi.
La sintattica è la seguente:

```
<!-- questo è un commento -->
```

2.4.2 Istruzioni

Al browser bisogna sempre dire dove comincia e dove finisce il file html, e questo si ottiene con le semplici istruzioni: <HTML> e </HTML> . Inoltre il file html contiene due parti. la testa (head) e il corpo (body) e, naturalmente, bisogna dire al browser dove iniziano e dove finiscono queste parti. Si fa così: <HEAD> </HEAD> e <BODY> </BODY> . Pertanto un file html contiene sempre, obbligatoriamente queste istruzioni:

- <HTML>
    - < HEAD>
    - </HEAD>
    - <BODY>
    - </BODY>
- </HTML>

Normalmente ci si mette il titolo della pagina, che comparirà in quella striscia in alto nello schermo, al di sopra dei vari menù e tasti cliccabili. E' possibile fare ciò usando i tags: <TITLE> </TITLE> . Esempio:

- <HTML>
    - <HEAD>
        - <TITLE>IL MANUALE ITALIANO PER IL PRINCIPIANTE HTML</TITLE>
    - </HEAD>
    - <BODY>
    - </BODY>
- </HTML>

2.4.3 Convenzioni di nomi

All'inizio di ogni file devono essere presenti alcune informazioni strutturate come segue:

```
<!--
        Gestione

        Descrizione Classe

        Author: Nome Cognome
        Version : 1.0
        2015 - Copyright by Pr.D Project - University of Salerno
-->
```

2.5 Script Javascript

JavaScript è uno dei linguaggi di programmazione più usati al mondo.
JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da *eventi* innescati a loro volta in vari modi dall'utente sulla pagina web in uso (mouse, tastiera ecc...).
Il codice Javascript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.

2.5.1 Commenti

JavaScript prevede delle sequenze di caratteri per **inserire commenti nel codice**.
Tutto ciò che viene marcato come commento non viene preso in considerazione dall'interprete JavaScript. Possiamo inserire due tipi di commenti nel codice:

- commento per singola riga;
- commento multiriga

Il **commento a singola riga** inizia con i caratteri // (doppio slash), come mostrato nel seguente esempio:

```
// Questo è un commento
var x = 5;
```

Tutto ciò che si trova a destra dei caratteri // fino alla fine della riga verrà considerato commento. Questo ci consente di scrivere anche commenti a fianco alle istruzioni:

```
var x = 5; // Questo è un commento
```

A differenza di quanto accade per i punti e virgola, possiamo utilizzare i commenti inline senza preoccuparci troppo di come sarà miniaturizzato il codice: un buon minificatore tipicamente elimina direttamente tutti tipi di commenti.

Il **commento multiriga** (multiline) prevede la sequenza iniziale /* (slash e asterisco) e si conclude con */ (asterisco e slash). Tutto ciò che viene inserito all'interno di questa coppia di terminazioni sarà considerato un commento, indipendentemente dal numero di righe utilizzate, in classico stile
C-like:

```
/* Questo è un commento sulla prima riga
  Questo è un commento sulla seconda riga
  Questa è l'ultima riga del commento */
var x = 5;
```

I documenti Javascript devono essere iniziati da un commento analogo a quello presente nei file Java.

2.5.2 Istruzioni

Per utilizzare Javascript incorporato è necessario il tag <SCRIPT> </SCRIPT>.
La cosa sicuramente più semplice di uno script javascript è descriverne la struttura.
Gli script possono essere sia interni (scritti all'interno della pagina HTML) che esterni (in fogli .js), un pò come gli style o css.
Tutti i browser web in grado di interpretare Javascript, riconoscono l' inizio di uno script tramite il tag HTML <SCRIPT>. Comunque c' e' un altra ragione per usare il tag <SCRIPT> : devi comunicare al browser quale versione di Javascript stai utilizzando.

2.5.3 Convenzioni di Nomi

All'inizio di ogni file devono essere presenti alcune informazioni strutturate come segue:

```
/**
*       Gestione
*
*       Descrizione Classe
*
*       Author: Nome Cognome
*       Version : 1.0
*       2015 - Copyright by Pr.D Project - University of Salerno
*/
```

2.5.3.1 Metodi

I metodi rappresentano attività che un oggetto puo compiere.Da un punto di vista sintattico, la definizione di un metodo per un oggetto è abbastanza simile alla definizione di una propietà.
I metodi possono prevedere degli argomenti come avviene in una normale funzione.

function nome([arg0] [,arg2] .....[,arg254]) {
istruzioni;
[return valorediritorno;] }

2.5.3.2 Variabili

Vige la **convenzione** che i nomi delle variabili inizino con una lettera minuscola e, qualora formati da più parole concatenate, tutte le parole successive alla prima siano capitalizzate e non vengano usati i simboli _ e $ nonostante siano ammessi.

var nomevar [= valore] [, nomevar2 [= valore] .....]

2.5.3.3 Costanti

Le **costanti** servono a memorizzare dei valori che non possono cambiare durante l'esecuzione dello script. **Javascript**, storicamente, non ha mai supportato le costanti.
Da un punto di vista prettamente convenzionale, molti sviluppatori Javascript sono soliti definire i valori costanti all'interno di comuni variabili caratterizzate da un nome in maiuscolo.

var **MIACOSTANTE** = 456;

2.6 Fogli di stile CSS

Tutti gli stili non inline devono essere collocati in fogli di stile separati.

Ogni regola CSS deve essere formattata come segue:

1. I selettori della regola si trovano a livello 0 di indentazione, uno per riga;
2. L'ultimo selettore della regola è seguito da parentesi graffa aperta ({);
3. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
4. La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

Le proprietà e le regole poco chiare dovrebbero essere precedute da un commento esplicativo. Le regole possono essere divise in blocchi concettualmente legati, preceduti da commenti, che ne spiegano lo scopo.

2.7 Database SQL

Le tabelle del database dovrebbero essere in terza forma normale di Codd (3NF). Qualora non lo fossero, la cosa deve essere documentata e motivata nello script di creazione del database.

I nomi delle tabelle devono seguire le seguenti regole:

1. la prima lettera del nome della tabella deve essere maiuscola;
2. se il nome è costituito da più parole, queste sono separate da un underscore (_);
3. il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

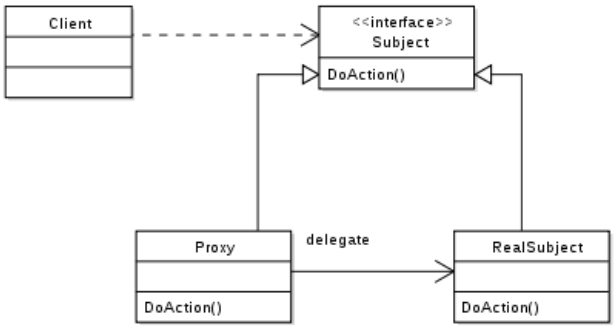
I nomi dei campi devono seguire le seguenti regole:

1. la prima lettera deve essere maiuscola;
2. se il nome è costituito da più parole, queste sono separate da un underscore (_);
3. il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

3. Design Pattern

3.2 Design Pattern Proxy

Il design pattern **Proxy** è usato per fornire un oggetto sostitutivo o segnaposto che fa riferimento ad un oggetto di base reale, di cui l'oggetto proxy è il surrogato.



Questo pattern può risultare utile se l'oggetto reale:

- Richiede molte risorse computazionali.
- Richiede molto tempo per caricarsi.
- Non viene mantenuto in cache ma viene generato ad ogni richiesta.
- Non definisce delle politiche di sicurezza e consente un accesso indiscriminato.

Il Proxy viene utilizzato:

- Per evitare spreco di risorse di calcolo e di rete.
- Per diminuire l'occupazione di memoria da parte degli oggetti istanziati.
- Per controllare accessi a un oggetto.
- Per nascondere agli utilizzatori il fatto che un oggetto risieda in uno spazio di indirizzamento diverso.

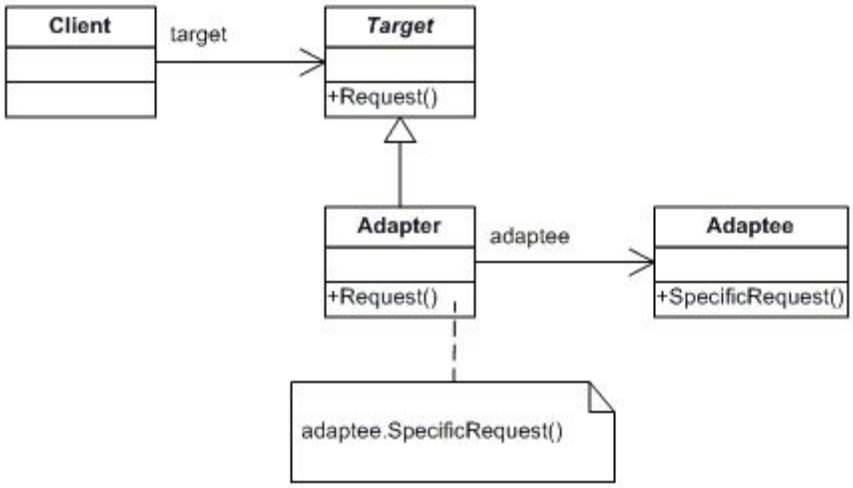
Esistono 5 casi d'uso:
Proxy Remoto: viene fornita una rappresentazione locale dell'oggetto remoto.
Proxy Virtuale: consente di creare oggetti leggeri al posto di quelli pesanti, che vengono creati soltanto quando sono effettivamente necessari.
Proxy di protezione: fornisce diversi livelli di accesso a seconda dei diritti di accesso su un client.
Proxy per un riferimento intelligente: svolge azioni aggiuntive quando si accede ad un oggetto.
Test di unità. Se si ha la necessità di testare un codice che accede ad una risorsa non sempre disponibile o ad una classe che è in fase di sviluppo ma ancora incompleta, è possibile creare un proxy per la risorsa o per la classe, il quale fornisce l'interfaccia completa ma con degli stub per le funzionalità che mancano.

Questo pattern è stato impiegato per poter interfacciarsi con il database. Qualunque classe interagisca con il database, dovrà passare attraverso il Proxy. Il suo file è situato nello Storage Layer.

3.3 Design Pattern Lazy Initialization

La tattica di stanziare un oggetto, inizializzare una variabile, effettuare un calcolo ed eseguire un processo solo nel momento in cui tale operazione è richiesta.
E' una tecnica di programmazione usata in tutte le classi che riguardano la gestione.

3.4 Design Pattern Adapter



Con il nome **adapter**, o adattatore si denota un design pattern utilizzato nella programmazione orientata agli oggetti. A volte viene chiamato **wrapper** (ovvero involucro) per il suo schema di funzionamento.

Il fine dell'adapter è di fornire una soluzione astratta al problema dell'interoperabilità tra interfacce differenti. Il problema si presenta ogni qual volta nel progetto di un software si debbano utilizzare sistemi di supporto (come per esempio librerie) la cui interfaccia non è perfettamente compatibile con quanto richiesto da applicazioni già esistenti. Invece di dover riscrivere parte del sistema, compito oneroso e non sempre possibile se non si ha a disposizione il codice sorgente, può essere comodo scrivere un adapter che faccia da tramite.

L'Adapter è un **pattern strutturale** che può essere basato sia su classi che su oggetti.

L'uso del pattern Adapter risulta utile quando interfacce di classi differenti devono comunque poter comunicare tra loro. Alcuni casi possono includere:

- L'utilizzo di una classe esistente che presenti un'interfaccia diversa da quella desiderata;
- La scrittura di una determinata classe senza poter conoscere a priori le altre classi con cui dovrà operare, in particolare senza poter conoscere quale specifica interfaccia sia necessario che la classe debba presentare alle altre.

Questo pattern è stato impiegato per poter utilizzare la stampa in PDF. Il file è situato nell’Application Layer

4. Componenti off-the-Shelf

Librerie e componenti esterne utilizzate :

- **MySQLi**: libreria usata nel linguaggio di programmazione PHP per fornire un'interfaccia con database MySQL. Questa libreria:
    1. introduce la possibilità di sfruttare un approccio basato sul paradigma object oriented;
    2. permette l'utilizzo di un protocollo binario per la comunicazione che assicura prestazioni più elevate;
    3. fornisce il supporto per le prepared statements cioè istruzioni SQL precedentemente utilizzate e mantenute in cache per successive chiamate, con innegabili vantaggi a carico di ottimizzazione e prestazioni;
    4. mette a disposizione funzionalità avanzate per il debugging;
    5. permette di utilizzare stored procedures, query multiple e transazioni.

- **Chart Javascript**: libreria usata per introdurre grafici d'impatto o diagrammi all'interno di siti o applicazioni. Chart .js supporta la generazione di ben 6 tipi di grafici differenti, il disegno del grafico si avvale del Canvas di html5 e risponde anche nelle interazioni utente. adattandosi in modo "responsive" al device su cui viene prodotto.
  La libreria è stata usata all'interno del sistema per la visualizzazione del grafico del monte ore di un docente

- **Bootstrap**: è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, bottoni e navigazione, e altri componenti dell'interfaccia, così come alcune estensioni opzionali di JavaScript.
  La libreria è stata usata per la creazione del template del sito.

- **jQuery:** è una libreria JavaScript per applicazioni web. Nasce con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché implementare funzionalità AJAX.
  La libreria è stata usata per supporto alla libreria bootstrap

- **Datatables**: è un plug-in per il jQuery libreria Javascript. Si tratta di uno strumento estremamente flessibile, basata su fondamenta di progressive enhancement, e aggiungerà controlli di interazione avanzate a qualsiasi tabella HTML.Impaginazione, ricerca istantanea e più colonne ordinazione e supporta quasi qualsiasi fonte di dati:
    - DOM, Javascript, Ajax e l'elaborazione lato server.

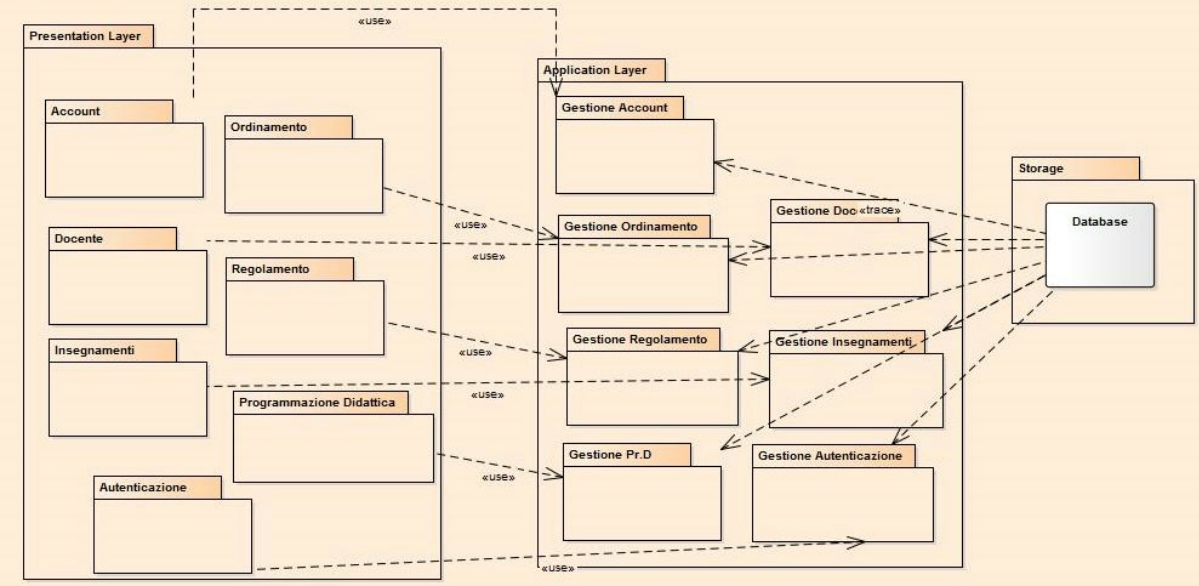
  Libreria utilizzata per la creazione delle tabelle.

- **Fpdf**: La libreria FPDF è un estensione alle funzionalità del linguaggio php (è un insieme di classi preconfezionate) che permette la creazione di documenti pdf nei vari dettagli. E' una libreria gratuita molto semplice da utilizzare, il cui unico requisito è la presenza di un ulteriore libreria, la Zlib (praticamente inserita de facto ovunque si utilizzi php). FPDF permette, in maniera semplice:
    - Supporto ai più diffusi formati di immagine.
    - Gestione dei colori.
    - Gestione dei collegamenti ipertestuali.
    - Supporto per codifica.
    - Possibilità di effettuare compressioni delle pagine.
    - Possibilità di scegliere l'unità di misura, il formato della pagine e dei suoi margini.
    - Possibilità di inserire intestazioni e note a piè di pagina.
    - Supporto per il cambio di pagina automatico.
    - Ritorno a capo automatico e la giustificazione del testo.

  La libreria è stata usata per la stampa in PDF.

5. Packages

Il Package Pr.D è il livello di astrazione più alto del prodotto software. Questo contiene al suo interno ogni sottopackage implementato come descritto nell' SDD. A seguito dell'analisi del carico applicativo richiesto dalla realizzazione dei singoli sottopackage, si è scelto di implementare i seguenti sottopackage:



Nel **presentation layer** sono presenti i package:

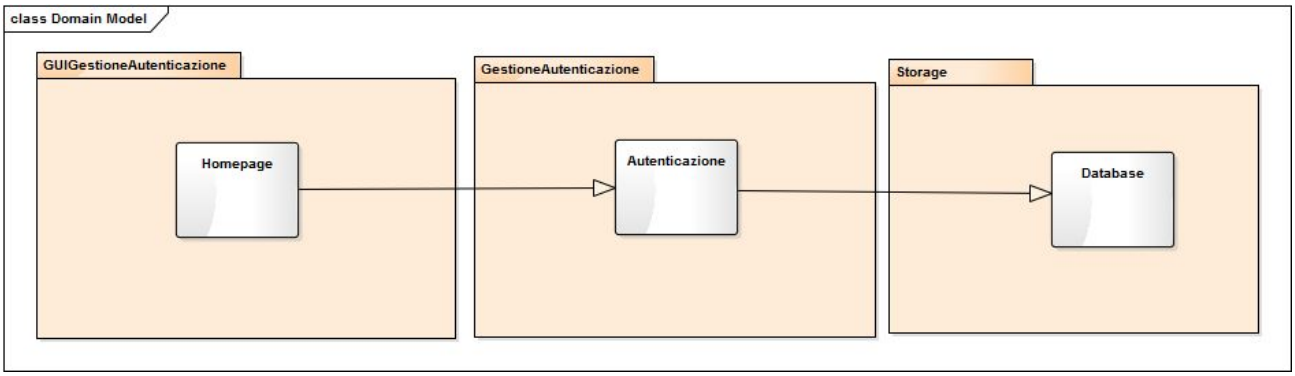
1. Account,Ordinamento,
2. Docente,Regolamento,
3. Programmazione Didattica,Autenticazione,
4. Insegnamento

Nel **application Layer** sono presenti i package:

1. Gestione Account, Gestione Ordinamento,
2. Gestione Regolamento ,Gestione Insegnamento,
3. Gestione Programmazione Didattica,
4. Gestione Autenticazione ,Gestione Regolamento

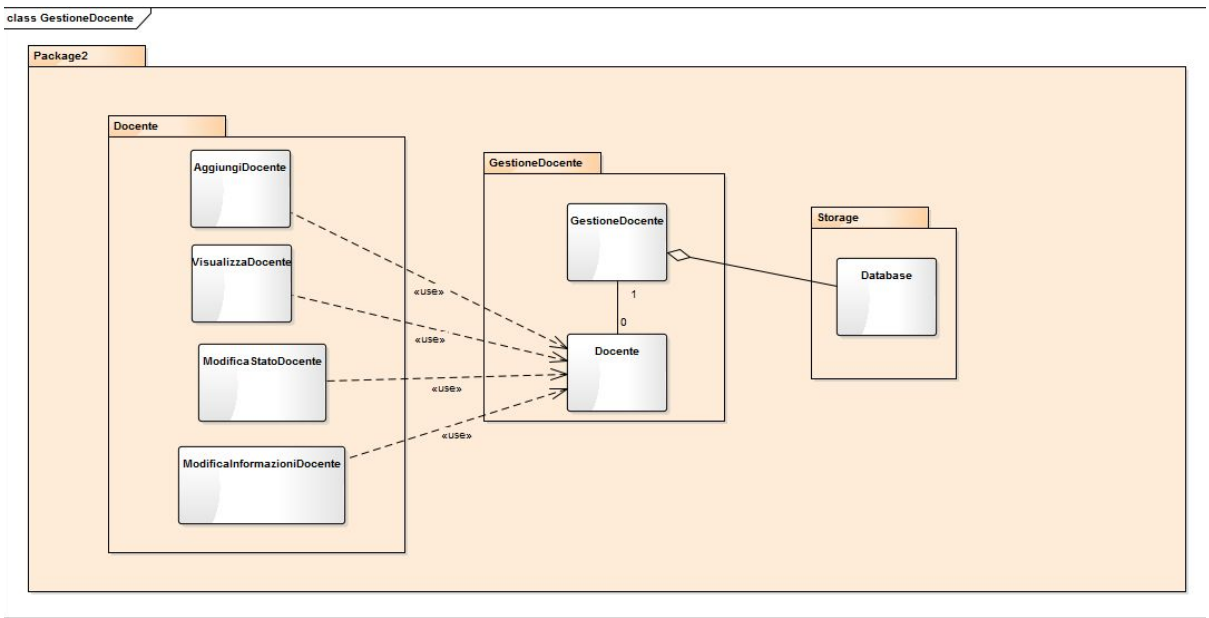
Nello **storage Layer** è presente il database

5.1 Gestione Autenticazione



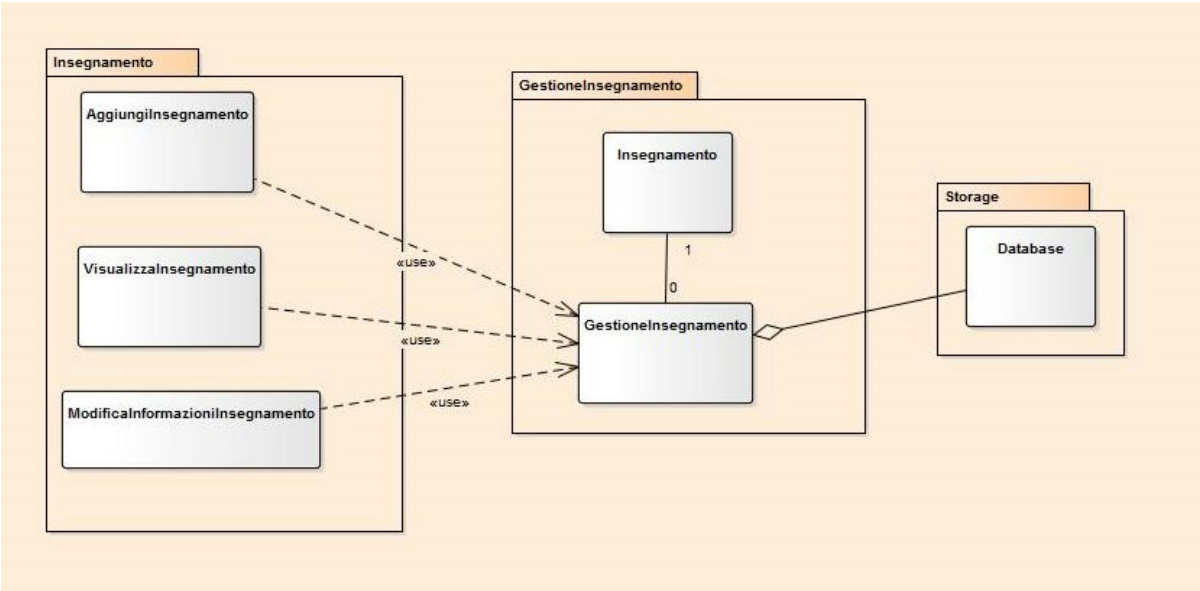
Il package GuiGestioneAutenticazione si trova nel **presentation layer** del server dell'applicazione Pr.D e include le operazioni che si possono effettuare per autenticarsi con il sistema. Il package GestioneDocente si trova nel **application Layer** del server dell'applicazione Pr.D e include le operazioni sulla gestione dell'autenticazione.

5.2 Gestione Docente



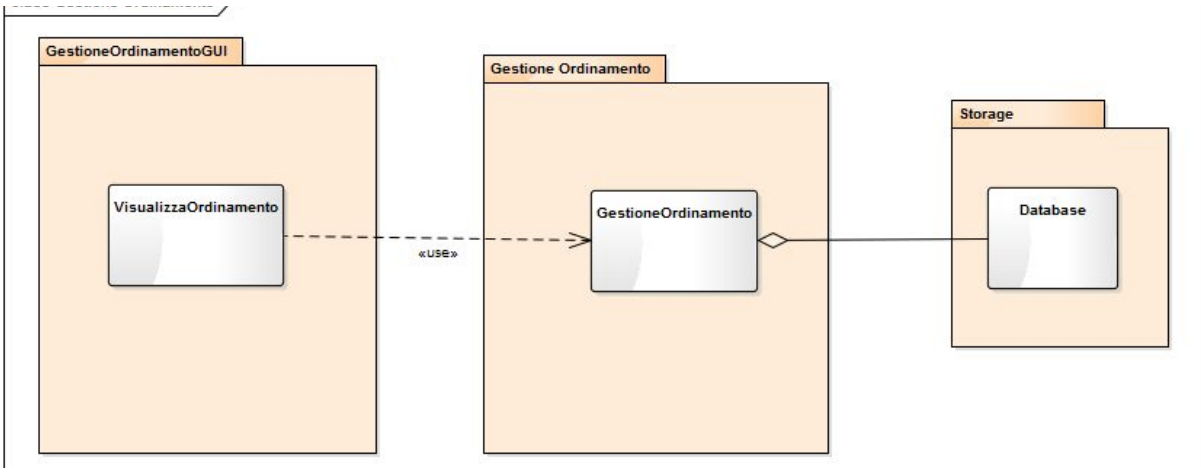
Il package Docente si trova nel **presentation Layer** del server dell'applicazione Pr.D e include le operazioni che si possono effettuare sui docenti presente nel sistema
Il package GestioneDocente si trova nel **application Layer** del server dell'applicazione Pr.D e include le operazioni sulla gestione dei docenti

5.3 Gestione Insegnamento



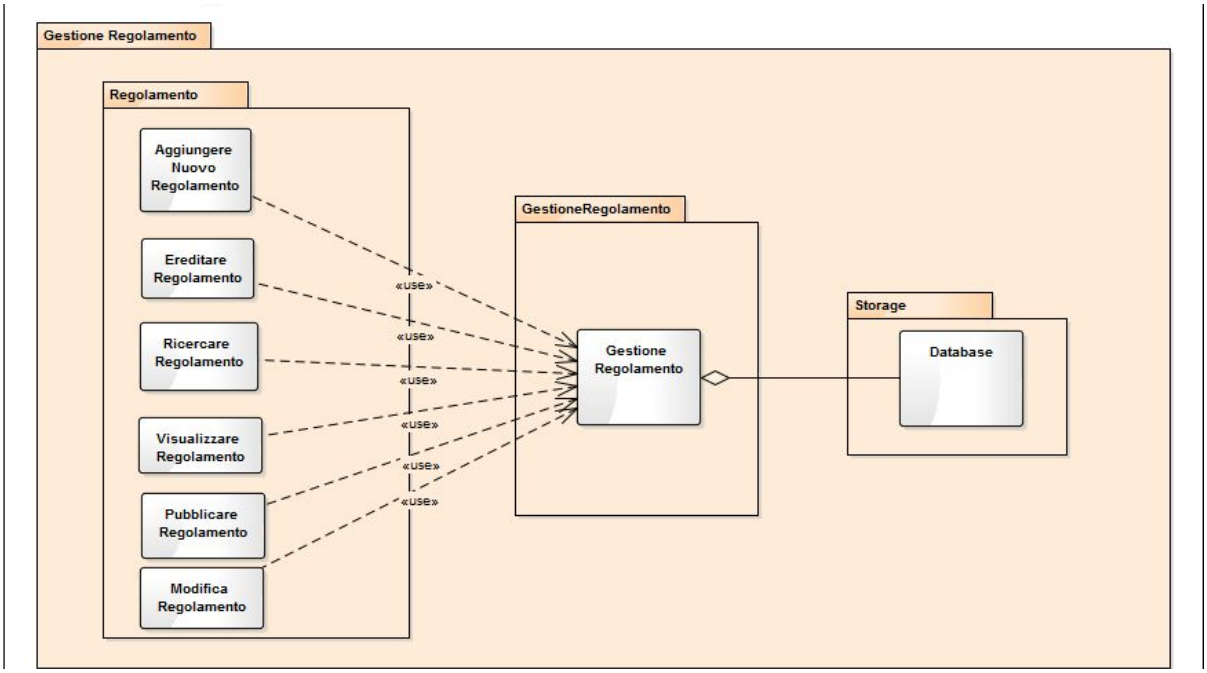
Il package Insegnamento si trova nel **presentation Layer** del server dell'applicazione Pr.D e include le operazioni che si possono effettuare sugli insegnamenti presente nel sistema
Il package GestioneInsegnamento si trova nel **application Layer** del server dell'applicazione Pr.D e include le operazioni sulla gestione degli insegnamenti

5.4 Gestione Ordinamento



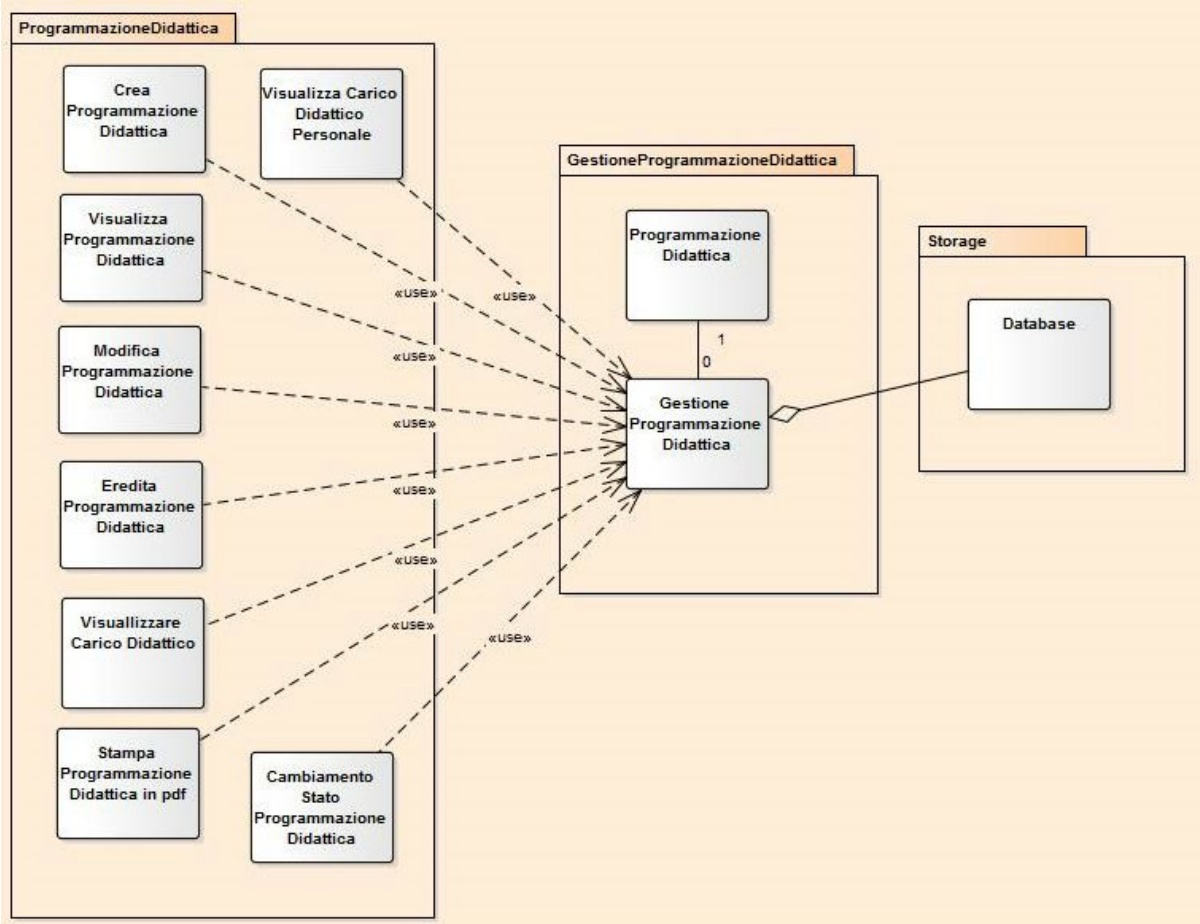
Il package Ordinamento si trova nel **presentation Layer** del server dell'applicazione Pr.D e include le operazioni che si possono effettuare sull'ordinamento presente nel sistema
Il package GestioneOrdinamento si trova nel **application Layer** del server dell'applicazione Pr.D e include le operazioni sulla gestione dell'ordinamento.

5.5 Gestione Regolamento



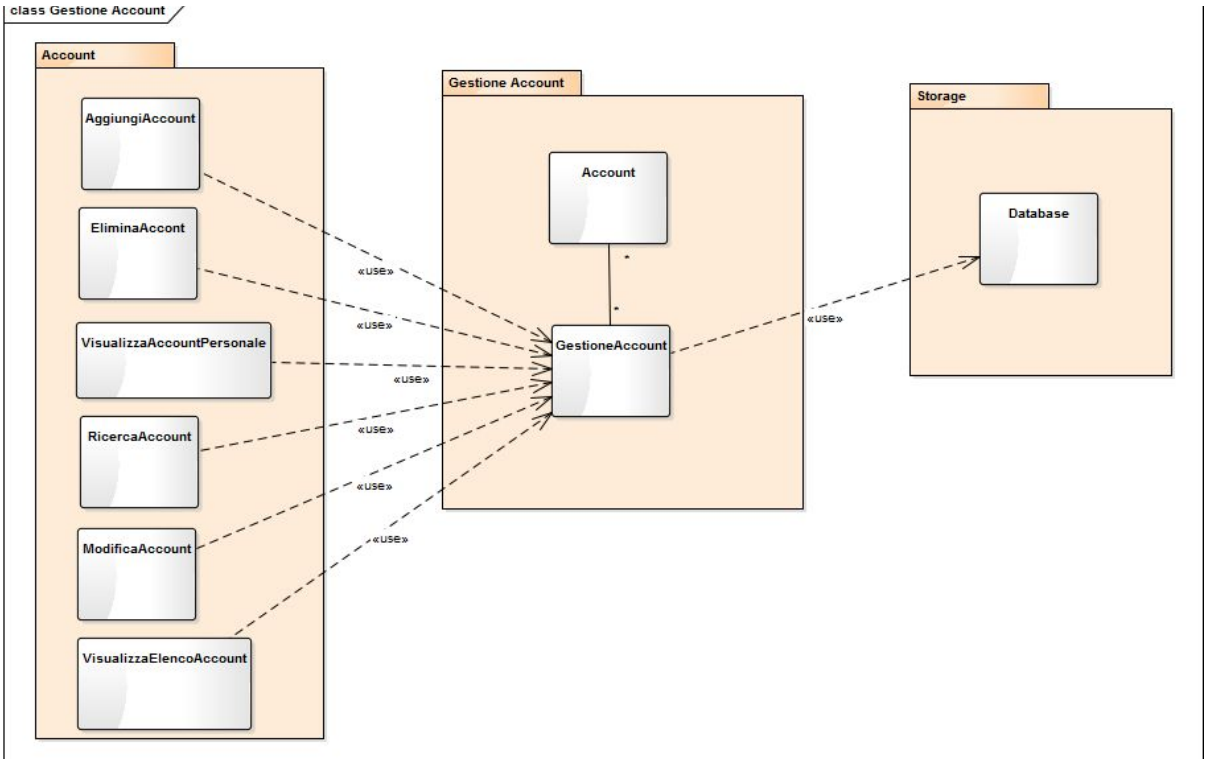
Il package Regolamento si trova nel **presentation Layer** del server dell'applicazione Pr.D e include le operazioni che si possono effettuare sul regolamento presente nel sistema
Il package GestioneRegolamento si trova nel **application Layer** del server dell'applicazione Pr.D e include le operazioni sulla gestione del regolamento

5.6 Gestione Programmazione Didattica



Il package ProgrammazioneDidattica si trova nel **presentation Layer** del server dell'applicazione Pr.D e include le operazioni che si possono effettuare sulla programmazione didattica presente nel sistema
Il package GestioneProgrammazioneDidattica si trova nel **application Layer** del server dell'applicazione Pr.D e include le operazioni sulla gestione della programmazione didattica

5.7 Gestione Account

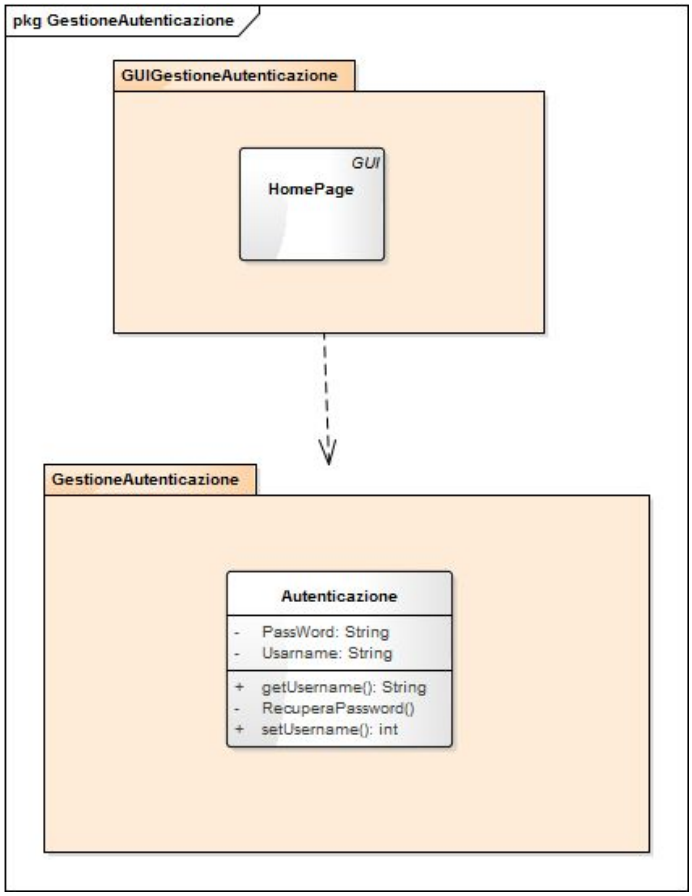


Il package Account si trova nel **presentation Layer** del server dell'applicazione Pr.D e include le operazioni che si possono effettuare sull'account presente nel sistema
Il package GestioneAccount si trova nel **application Layer** del server dell'applicazione Pr.D e include le operazioni sulla gestione dell'account.

6. Class Interfaces

6.1 Gestione Autenticazione

6.1.1 Utente Registrato



Di seguito viene riportata la descrizione delle classi:

GuiGestioneAutenticazione:

- Home Page

Gestione Autenticazione:

- Autenticazione

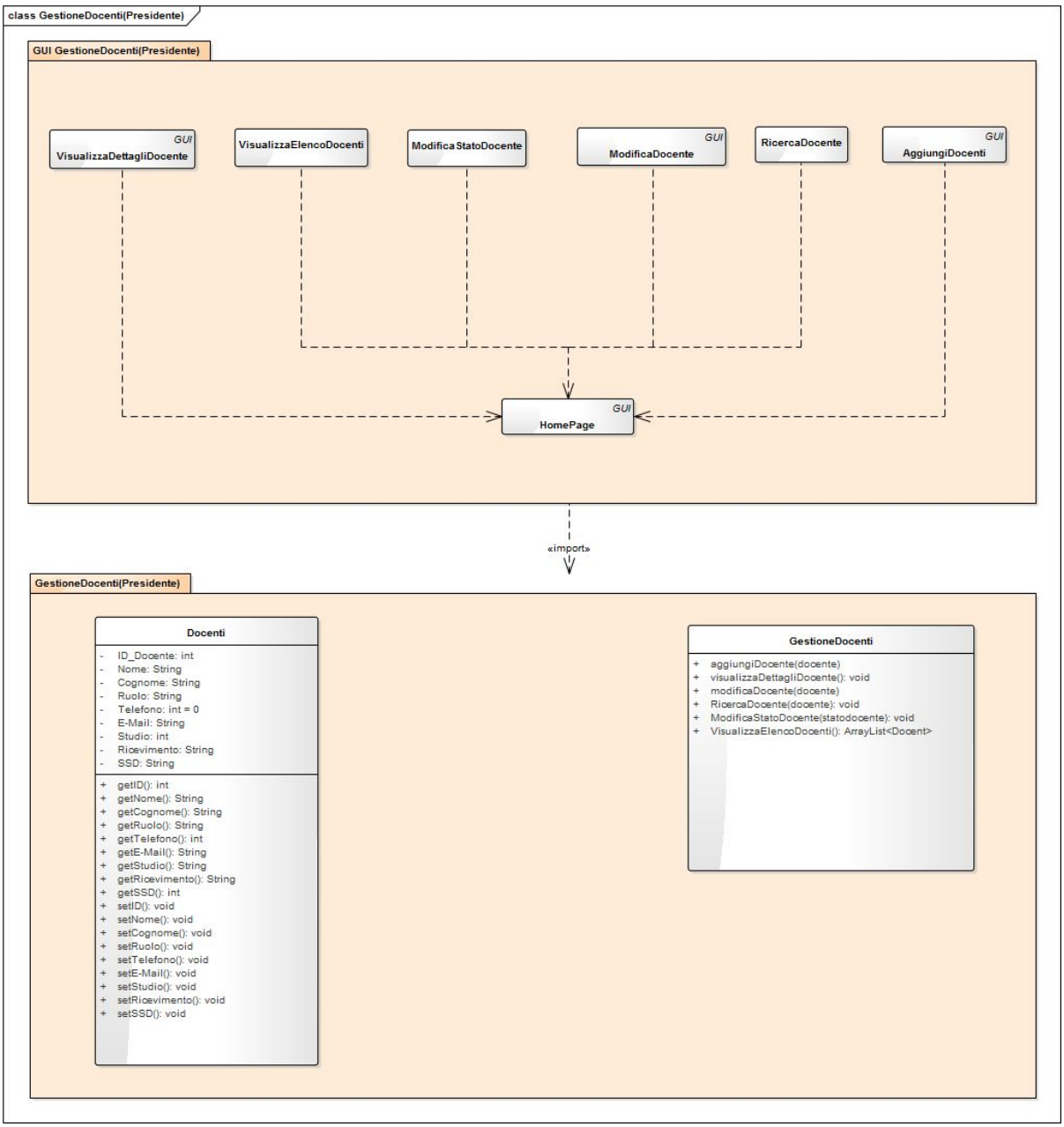
Di seguito viene fornita una panoramica dei contratti legati a ciascuna classe:
Nome Classe: Autenticazione
Invariante: Il login deve essere univoco

Precondizioni:
+RecuperaPassword(): account esistente
+Logout(): L'utente deve aver effettuato il login
+Login(): L'utente deve avere un account associato al sito

Postcondizioni:
+Login(): L'utente è loggato
+Logout(): L'utente è uscito dal sistema
+RecuperaPassword(): L'utente viene munito nuovamente della password

6.2 Gestione Docenti

6.2.1 Presidente



Di seguito viene riportata la descrizione delle classi:

GUI Gestione Docente (Presidente):

- AggiungiDocente
- ModificaDocente
- ModificaStatoDocente
- RicercaDocente
- VisualizzaDettagliDocente
- VisualizzaElencoDocente

Gestione Docente (Presidente):

- Docente
- Gestione Docente

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

NomeClasse: Docente
Invariante: La matricola del docente deve essere univoca.

Docente
-matricola: String -nome: String -cognome: String -email: String -telefono: String -ricevimento: String -ruolo: String -settoreScientificoDisciplinare: String -stato: String -studio: String
+**getMatricola()**: String +**getNome()**: String +**getCognome()**: String +**getEmail()**: String +**getNumeroDiTelefono()**: String +**getRicevimento()**: String +**getRuolo()**: String +**getStato()**: String +**getStudio()**: String +**getSettoreScientificoDisciplinare()**: String +**setMatricola(matricola)**: String +**setNome(nome)**: String +**setCognome(cognome)**: String +**setEmail(email)**: String +**setNumeroDiTelefono(telefono)**: String +**setRicevimento(ricevimento)**: String +**setRuolo(ruolo)**: String +**setStato(stato)**: String +**setStudio(studio)**: String +**setSettoreScientificoDisciplinare(settoreScientificoDisciplinare)**: String

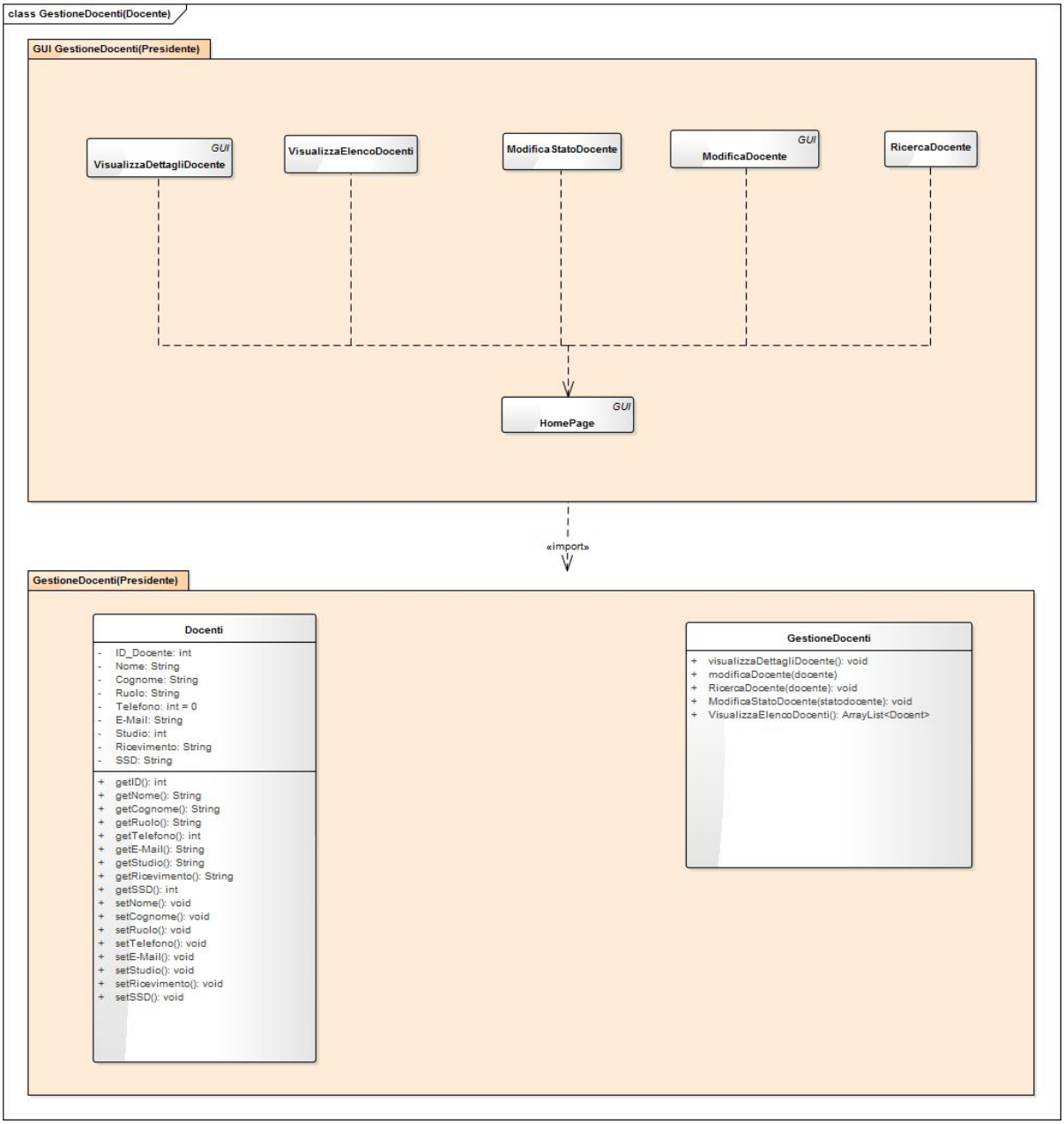
NomeClasse: Gestione Docente
Invariante: -

GestioneDocente
-database
+ricercaDocente(tutti:int, matricola:string): List **+aggiungiDocente(docente:Docente): boolean** **+aggiornaDocente(docente:Docente): boolean** **+modificaStatoDocente(docente:Docente, nuovoStato:int): boolean**

Precondizioni:
+ **ricercaDocente(): ArrayList <Docente>:** i docenti devono essere presenti nel database.
+ **aggiungiDocente(Docente):** il Presidente deve aver effettuato l'accesso.
+ **aggiornaDocente(Docente):** il Presidente deve aver effettuato l'accesso.
+ **modificaStatoDocente(Docente, nuovoStato):** il Presidente deve aver effettuato l'accesso.

Postcondizioni:
+ **aggiungiDocente(Docente):** nessun campo deve essere nullo.
+ **aggiornaDocente(Docente):** nessun campo deve essere nullo.
+ **modificaStatoDocente(Docente, nuovoStato):** il campo non deve essere nullo.
+ **ricercaDocente(): ArrayList <Docente>:** il sistema visualizza il docente ricercato o tutti i docenti.

6.2.2 Docente



Di seguito viene riportata la descrizione delle classi:

GUI Gestione Docente (Docente):

- ModificaDocente
- ModificaStatoDocente
- RicercaDocente
- VisualizzaDettagliDocente
- VisualizzaElencoDocente

Gestione Docente (Docente):

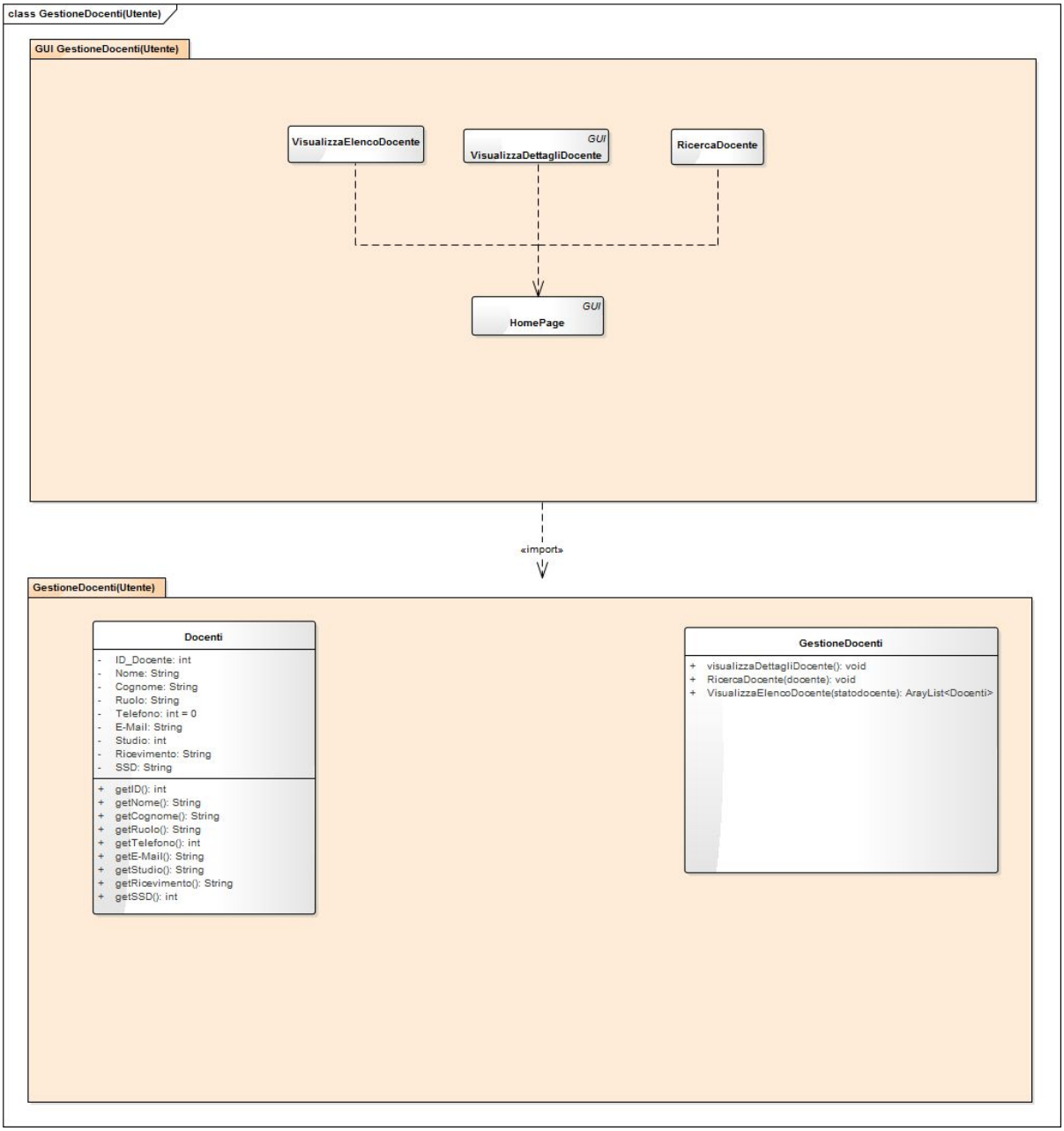
- Docente
- Gestione Docente

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

NomeClasse: Docente
Invariante: La matricola del docente deve essere univoca.
Precondizioni: -
Postcondizioni: -

NomeClasse: Gestione Docente
Invariante: -
Precondizioni:
+ **visualizzaDettagliDocente(): ArrayList <Docente>:** i professori devono essere presenti nel database.
+ **visualizzaElencoDocenti(): ArrayList <Docente>:** i professori devono essere presenti nel database..
+ **modificaDocente(Docente):** il Presidente deve aver effettuato l'accesso.
+ **modificaStatoDocente(Docente):** il Presidente deve aver effettuato l'accesso.
Postcondizioni:
+ **modificaDocente(Docente):** nessun campo deve essere nullo.
+ **modificaStatoDocente(Docente):** il campo non deve essere nullo.
+ **visualizzaDettagliDocente(): ArrayList <Docente>:** il sistema visualizza il professore ricercato.
+ **visualizzaElencoDocenti(): ArrayList <Docente>:** il sistema visualizza tutti i docenti presenti all'interno del sistema.

6.2.3 Utente



GUI Gestione Docente (Utente):

- RicercaDocente
- VisualizzaDettagliDocente
- VisualizzaElencoDocente

Gestione Docente (Presidente):

- Docente
- Gestione Docente

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

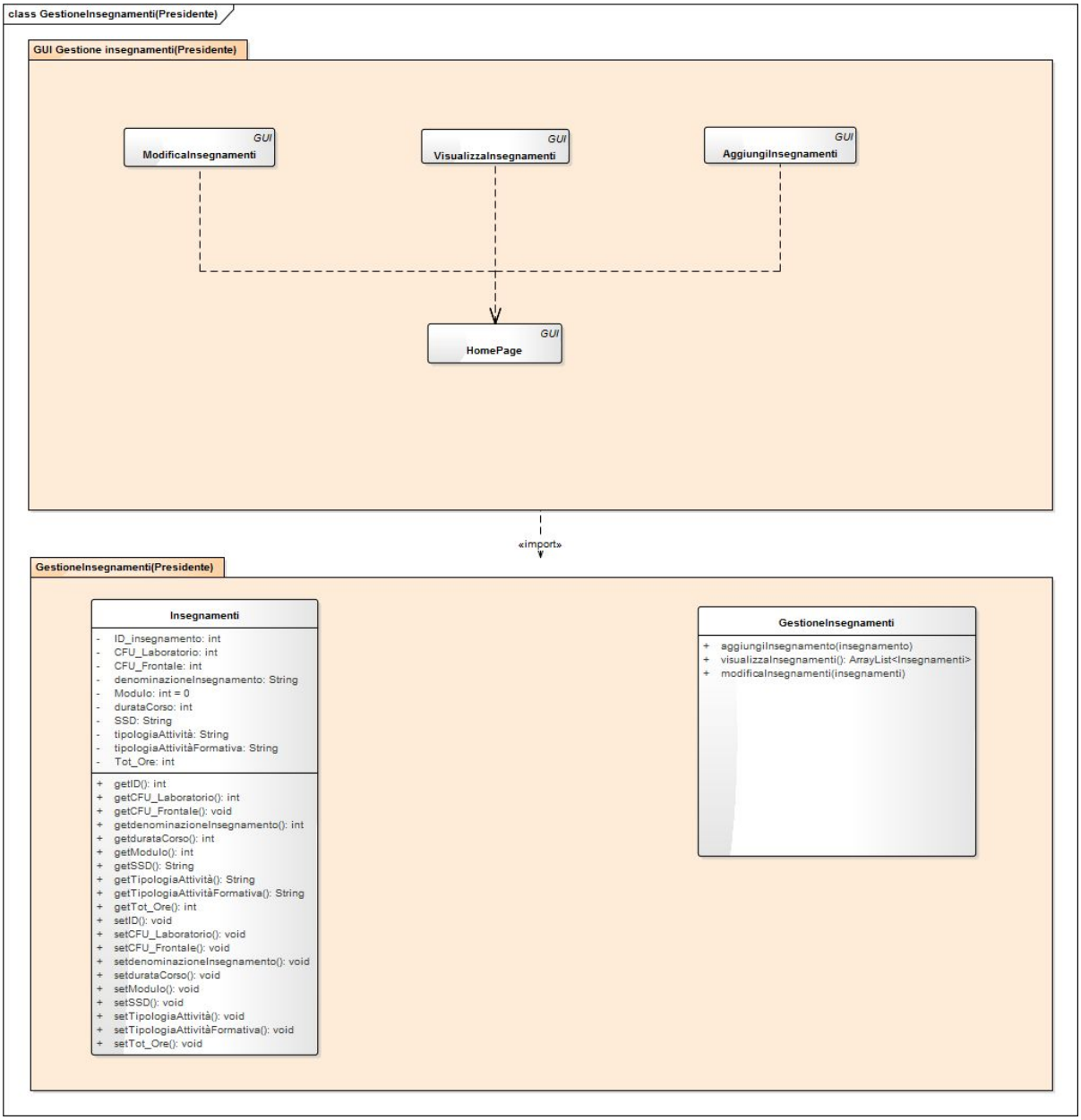
NomeClasse: Docente
Invariante: La matricola del docente deve essere univoca.
Precondizioni: -
Postcondizioni: -

GestioneDocente
-database
+ricercaDocente(tutti:int, matricola:string): List **+aggiungiDocente(docente:Docente): boolean** **+salvaModifiche(insegnamento:Insegnamento): int** **+aggiornaDocente(docente:Docente): boolean** **-modificaStatoDocente(docente:Docente, nuovoStato:int): boolean**

NomeClasse: Gestione Docente
Invariante: -
Precondizioni:
+ **visualizzaDettagliDocente(): ArrayList <Docente>**: i professori devono essere presenti nel database.
+ **visualizzaElencoDocenti(): ArrayList <Docente>:** i professori devono essere presenti nel database.
Postcondizioni:
+ **visualizzaDettagliDocente(): ArrayList <Docente>:** il sistema visualizza il professore ricercato.
+ **visualizzaElencoDocenti(): ArrayList <Docente>:** il sistema visualizza tutti i docenti presenti all'interno del sistema.

6.3 Gestione Insegnamenti

6.3.1 Presidente



GUIGestione Insegnamenti(Presidente)

- HomePage
- AggiungiInsegnamento
- ModificaInsegnamento
- VisualizzaInsegnamento

Gestione Insegnamenti(Presidente)

- Insegnamenti
- Gestioneinsegnamenti

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

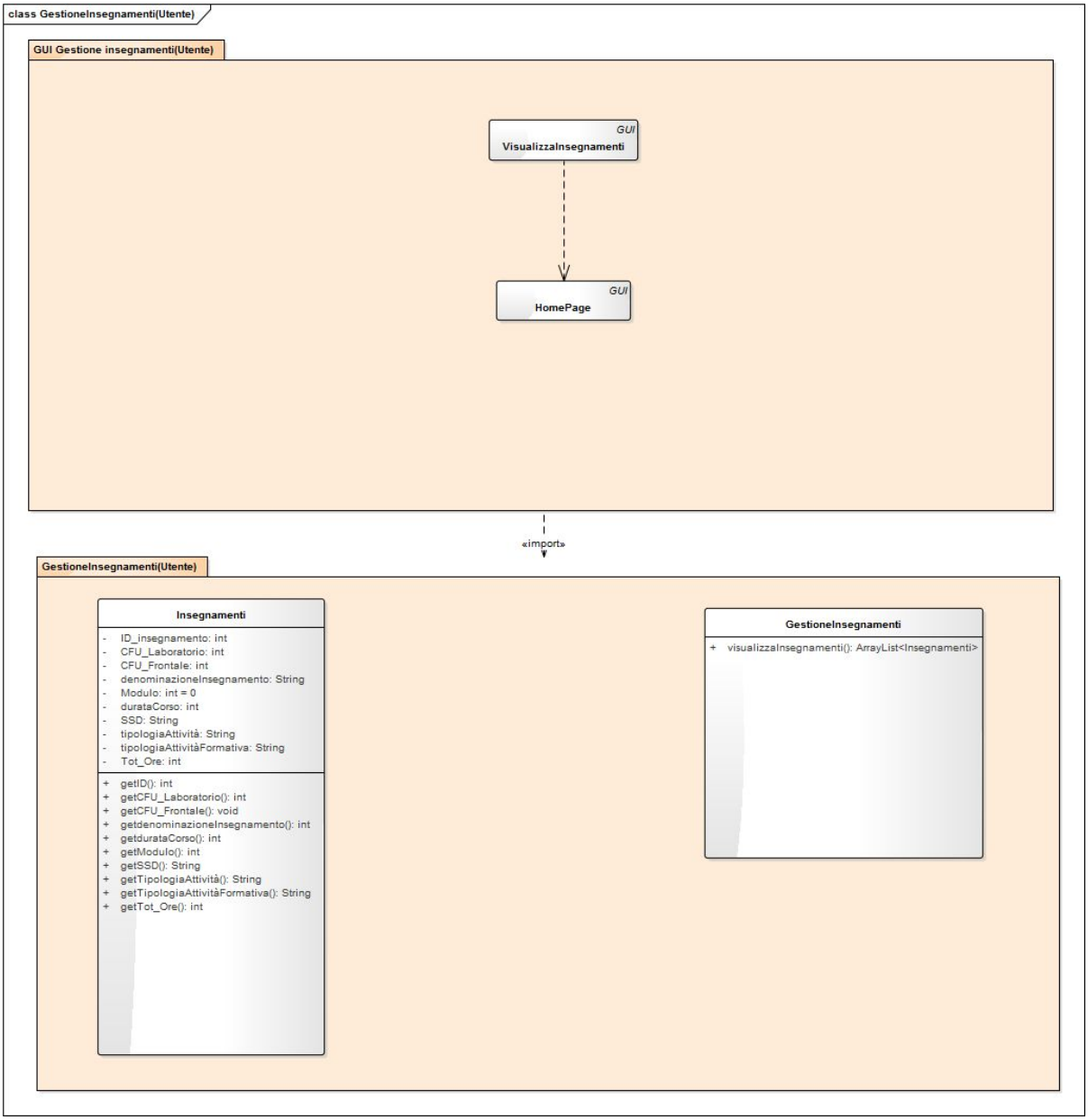
NomeClasse: Insegnamenti
Invariante: L'ID relativo all'insegnamento deve essere univoco
Precondizioni: -
Postcondizioni: -

GestioneInsegnamenti
-database
+visualizzaInsegnamenti(): List **+aggiungiInsegnamento(insegnamento:Insegnamento): int** **+salvaModifiche(insegnamento:Insegnamento): int** **+getSSD(): List** **-isExist(clausola:String): boolean**

Precondizioni:
+ **visualizzaInsegnamenti()**: Gli insegnamenti devono essere presenti nel database
+ **aggiungiInsegnamento(insegnamento:Insegnamento):** insegnamento deve essere diverso da NULL e non deve essere già presente nel database
+ **salvaModifiche(insegnamento:Insegnamento)**: salva le modifiche sull'insegnamento se e solo se non è uguale a un insegnamento presente nel database
+ **getSSD()**: i SSD devono essere presenti nel database
- isExist(clausola:String) : il valore di clausola deve essere una stringa non vuota

Postcondizioni:
+ **visualizzaInsegnamenti():** restituisce gli insegnamenti presenti nel database
+ **aggiungiInsegnamento(insegnamento:Insegnamento):** restituisce 0 se l'aggiunta avviene con successo
+ **salvaModifiche(insegnamento:Insegnamento)**: restituisce 0 se l'aggiunta avviene con successo
+ **getSSD()**: restituisce i SSD presenti nel database
- isExist(clausola:String) : restituisce true se esiste un insegnamento

6.3.2 Utente



GUIGestione Insegnamenti(Utente)

- HomePage
- VisualizzaInsegnamento

Gestione Insegnamenti(Utente)

- Insegnamenti
- Gestioneinsegnamenti

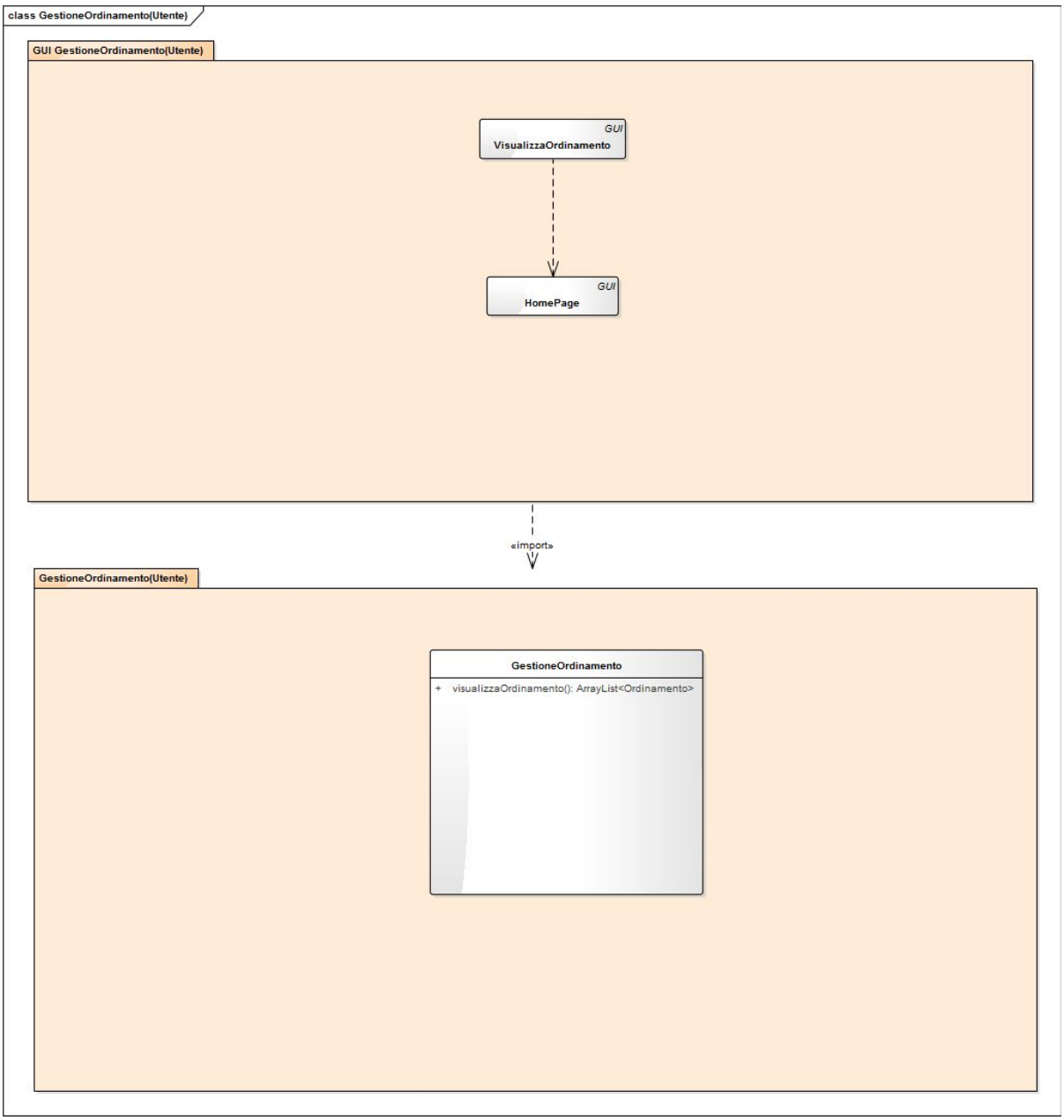
Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

NomeClasse: Insegnamenti
Invariante: L'ID relativo all'insegnamento deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeClasse: Gestioneinsegnamenti
Invariante: -
Precondizioni: +visualizzaInsegnamenti(): ArrayList <insegnamento>: gli insegnamenti devono essere nel database.
Postcondizioni: +visualizzaInsegnamenti():Il sistema visualizza l'elenco con tutti gli insegnamenti.

6.4 Gestione Ordinamento

6.4.1 Utente



GUIGestioneOrdinamento(Utente)

- HomePage
- VisualizzaOrdinamento

GestioneOrdinamento(Utente)

- Ordinamento
- GestioneOrdinamento

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

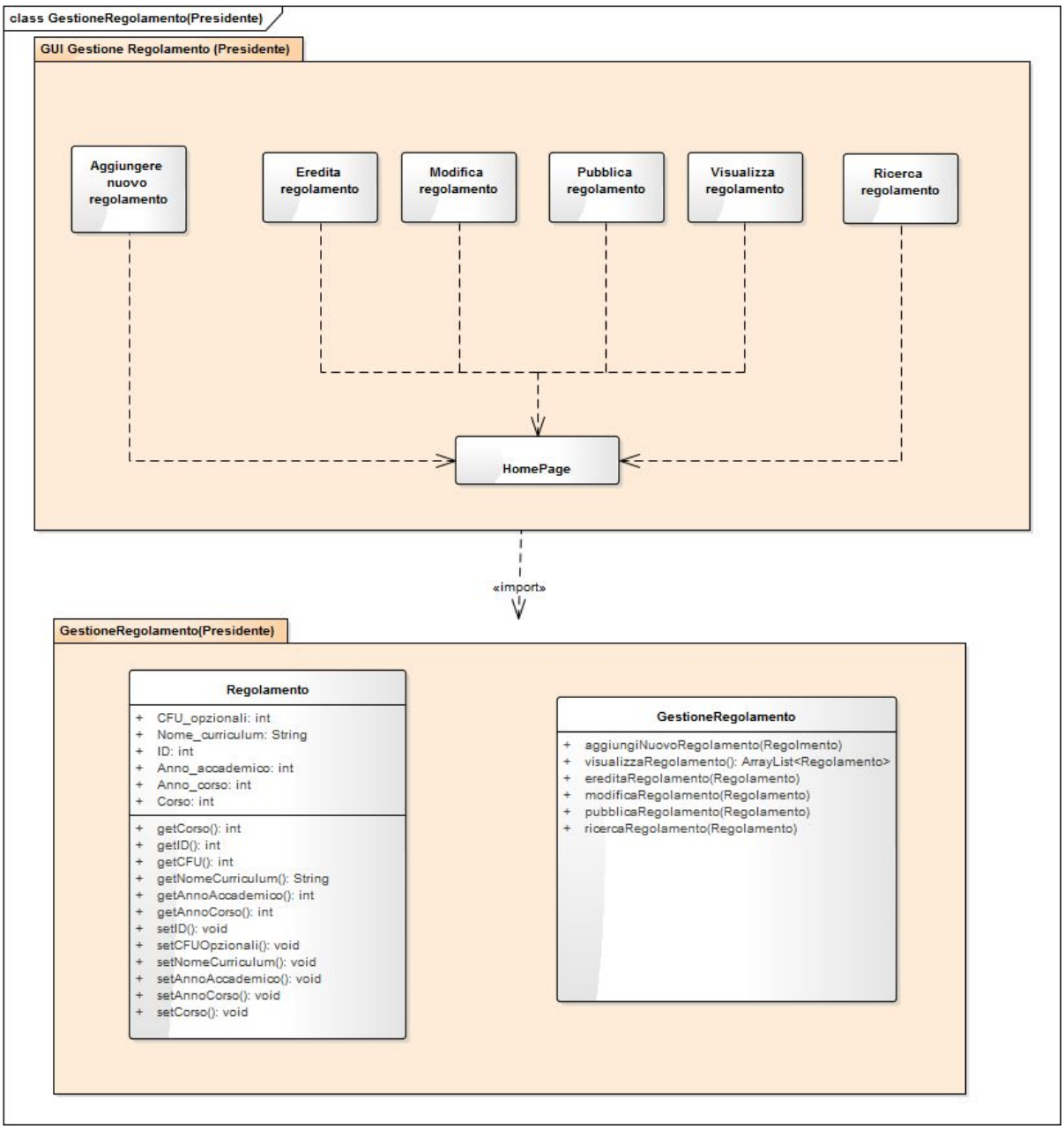
NomeClasse: Ordinamento
Invariante: L'ID relativo all'ordinamento deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeClassea: GestioneOrdinamento
Invariante: -
Precondizioni: +visualizzaOrdinamento():String, gli insegnamenti devono essere nel database.
Postcondizioni: +visualizzaOrdinamento(): Il sistema visualizza l'elenco con tutti gli ordinamenti.

GestioneOrdinamento
-database
+visualizzaOrdinamento():(String anno, String corso): String

6.5 Gestione Regolamento

6.5.1 Presidente



GUIGestioneRegolamento(Presidente)

- HomePage
- AggiungiRegolamento
- EreditaRegolamento
- ModificaRegolamento
- PubblicaRegolamento
- VisualizzaRegolamento
- RicercaRegolamento

GestioneRegolamento(Presidente)

- Regolamento
- GestioneRegolamento

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

NomeClasse: Regolamento
Invariante: L'ID relativo al regolamento deve essere univoco
Precondizioni: -
Postcondizioni: -

GestioneRegolamento
-database
+getAnniAccademici(): List **+getCurriculum(corso:String, annoAccademico:String, stato:String): String** **+getRangeCfuAttivita(annoAccademico:String, attivita_formativa): int** **+getSSD(annoAccademico:String, tipologia:String, ambito:String, corso:String): List** **+pubblicaRegolamento(id:String): int** **+getRegolamentoCompleto(corso:String, curriculum:String, annoAccademico:String, annoCorso:int, stato:int): List** **getRegolamento(corso:String, curriculum:String, annoAccademico:String, annoCorso:int, stato:int): List** **+eliminaInsegnamento(id:int, nomeSessione:String, curr:String):void** **+controllaRegolamento(curr:String, corso:String, tipologiaAttivita:String, anno:String, ambito:String, tipologiaAttivitaInsegnamento:String):int** **+controllaRegolamentoPerAmbito(curr:String, anno:String, corso:String):String** **+getCFUOpzionali(n:int, curr:String):int** **+salvaRegolamento(curr:String, anno:String, corso:String, stato:String):Void**

NomeClasse: GestioneRegolamento
Invariante: -

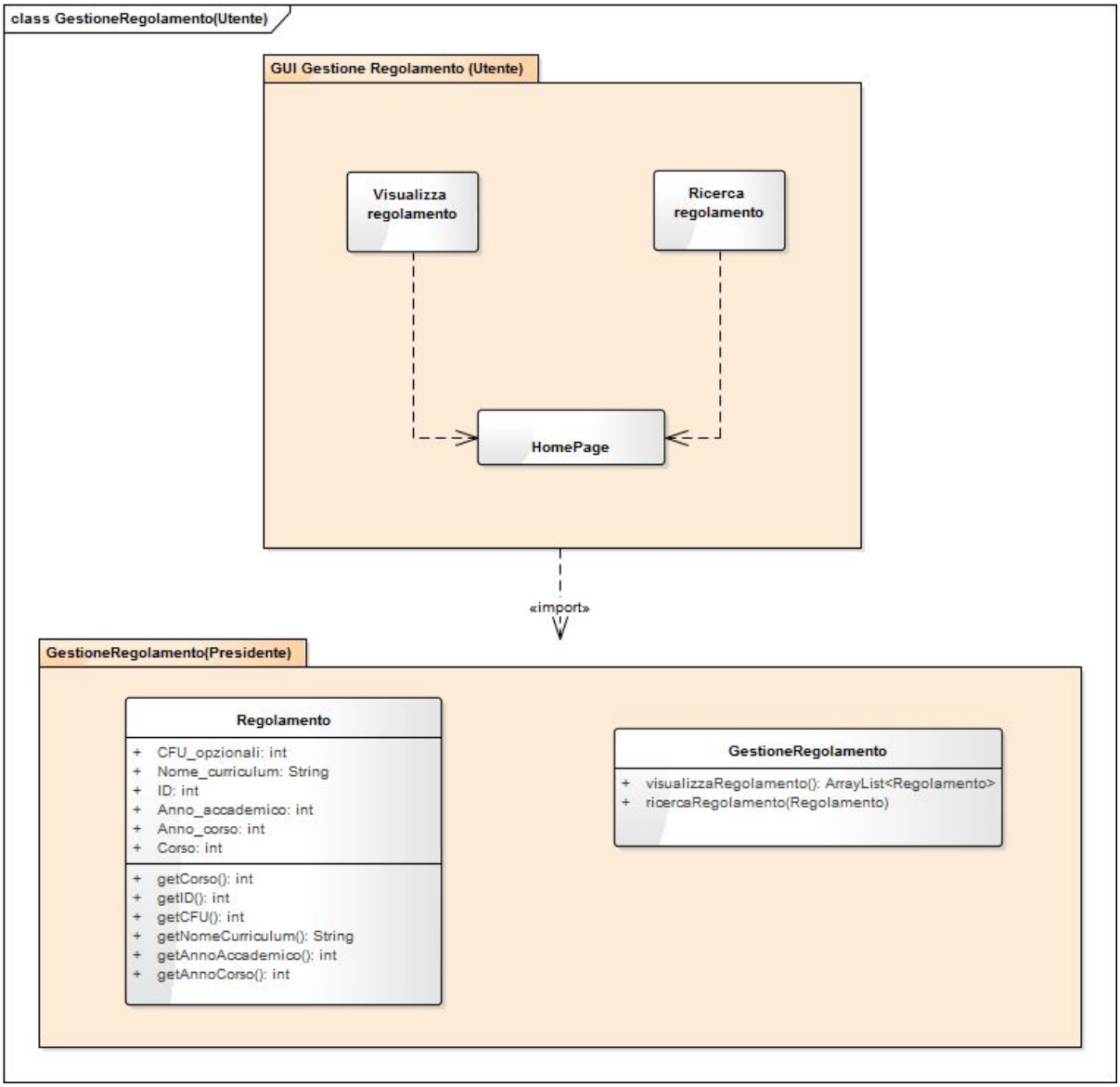
Precondizioni:

+getAnniAccademici():Anni accademici presenti nel database
+getCurriculum(corso:String, annoAccademico:String, stato:String): Curriculum presenti nel database
+getRangeCfuAttivita(annoAccademico:String, attivita_formativa): Attività presenti nel database
+getSSD(annoAccademico:String, tipologia:String, ambito:String, corso:String): SSD presenti nel database
+pubblicaRegolamento(id:String): tutti i dati compilati
+getRegolamentoCompleto(corso:String, curriculum:String, annoAccademico:String, annoCorso:int, stato:int): Regolamenti presenti nel database
+getRegolamento(corso:String, curriculum:String, annoAccademico:String, annoCorso:int, stato:int): Regolamenti presenti nel database
+eliminaInsegnamento(id:int, nomeSessione:String, curr:String): Tutti i campi compilati presenti nel database
+controllaRegolamento(curr:String, corso:String, tipologiaAttivita:String, anno:String, ambito:String, tipologiaAttivitaInsegnamento:String): Tutti i campi compilati
+controllaRegolamentoPerAmbito(curr:String, anno:String, corso:String): Tutti i campi compilati
+getCFUOpzionali(n:int, curr:String): Insegnamenti presenti nel database
+salvaRegolamento(curr:String, anno:String, corso:String, stato:String): Tutti i campi compilati

Postcondizioni:

+getAnniAccademici(): Il sistema visualizza gli anni accademici
+getCurriculum(corso:String, annoAccademico:String, stato:String): Il sistema restituisce i curriculum
+getRangeCfuAttivita(annoAccademico:String, attivita_formativa): Il sistema restituisce i range di cfu delle attività
+getSSD(annoAccademico:String, tipologia:String, ambito:String, corso:String): Il sistema restituisce gli ssd
+pubblicaRegolamento(id:String): Il sistema salva il regolamento come pubblicato
+getRegolamentoCompleto(corso:String, curriculum:String, annoAccademico:String, annoCorso:int, stato:int): Il sistema restituisce il regolamento completo di un dato anno e curriculum
+getRegolamento(corso:String, curriculum:String, annoAccademico:String, annoCorso:int, stato:String): il sistema restituisce il regolamento di un dato anno e curriculum
+eliminaInsegnamento(id:int, nomeSessione:String, curr:String): Viene eliminato un insegnamento dal regolamento
+controllaRegolamento(curr:String, corso:String, tipologiaAttivita:String, anno:String, ambito:String, tipologiaAttivitaInsegnamento:String): nessun vincolo violato
+controllaRegolamentoPerAmbito(curr:String, anno:String, corso:String): nessun vincolo violato
+getCFUOpzionali(n:int, curr:String): numero di CFU opzionali
+salvaRegolamento(curr:String, anno:String, corso:String, stato:String): restituisce 0 se il regolamento viene salvato con successo

6.5.2 Utente



GUIGestioneRegolamento(Utente)

- HomePage
- VisualizzaRegolamento
- RicercaRegolamento

GestioneRegolamento(Presidente)

- Regolamento
- GestioneRegolamento

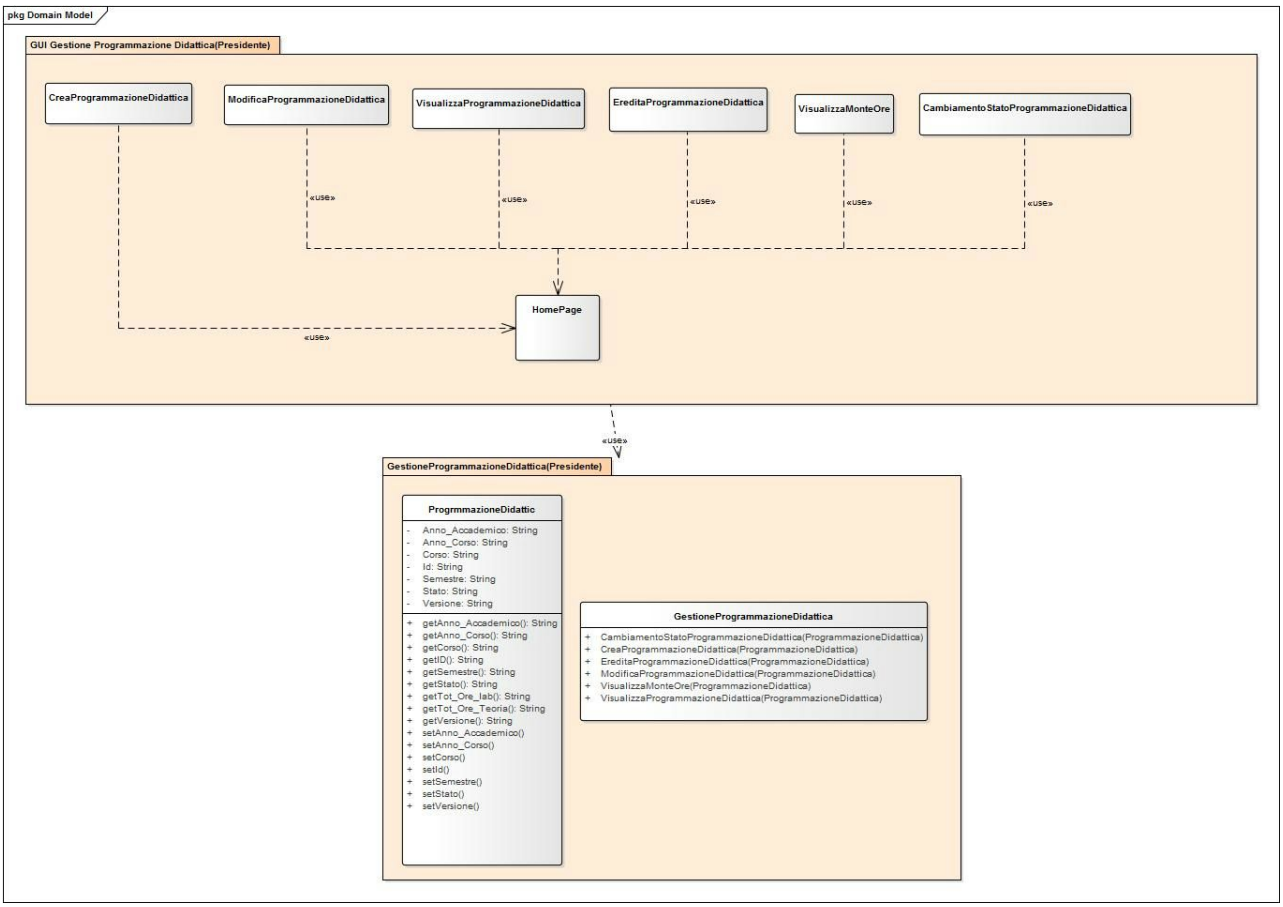
Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

NomeClasse: Regolamento
Invariante: L'ID relativo al regolamento deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeInterfaccia: GestioneRegolamento
Invariante: -
Precondizioni:
+visualizzaRegolamento(): ArrayList <regolamento>: il regolamento deve essere nel database.
Postcondizioni:
+visualizzaRegolamento():Il sistema visualizza l'elenco con tutti i regolamenti.

6.6 Gestione Programmazione Didattica

6.6.1 Presidente



GUIGestione Programmazione Didattica(Presidente)

- HomePage
- Crea Programmazione Didattica
- Modifica Programmazione Didattica
- VisualizzaProgrmmazione Didattica
- Eredita Programmazione Didattica
- Visualizza Monte Ore di un Docente
- Cambiamento Stato Programmazione Didattica

Gestione Programmazione Dinamica

- Programmazione Didattica
- GestioneProgrammazioneDidattica

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

NomeClasse:Programmazione Didattica
Invariante: L'Id deve essere univoco.
Precondizioni: -
Postcondizioni: -

GestioneProgrammazioneDidattica
-database
+creaProgDid(progD:progD, associa:Associa): null **+getCurriculum(corso:String, annoAccademico:String): String** **+getCurriculumProgDid(annoAcc:String, corso:String, semestre:int): String** **+getDocenti(): List** **+getInsegnamenti(annoCorso:String, corso:String, curriculum:String, annoAccademico:String): List** **+getMonteOre(nome:String, cognome:String): String** **+getCaricoDidattico(username:String): String** **+salvaStato(): null** **+cambiaStato(nuovoStato:String, id:String): null** **+getProgrammazioniDidattiche(): List** **+getAccademico(corso:String): List** **+getTotOreProf(matricola:String,ruolo:String): int** **+getProgDidVis(corso:String, anno_accademico:String, anno_corso:String, utente:String,semestre:String):String**

NomeClasse: Gestione Programmazione Didattica
Invariante: -
Precondizioni:

+creaProgDid(progD:progD, associa:Associa): null L'associazione da effettuare non dev'essere già presente nel database
+getCurriculum(corso:String, annoAccademico:String): String Il curriculum deve essere presente nel database
+getCurriculumProgDid(annoAcc:String, corso:String, semestre:int): String Il curriculum deve essere presente nel database.
+getDocenti(): List Deve essere presente nel database almeno un docente.

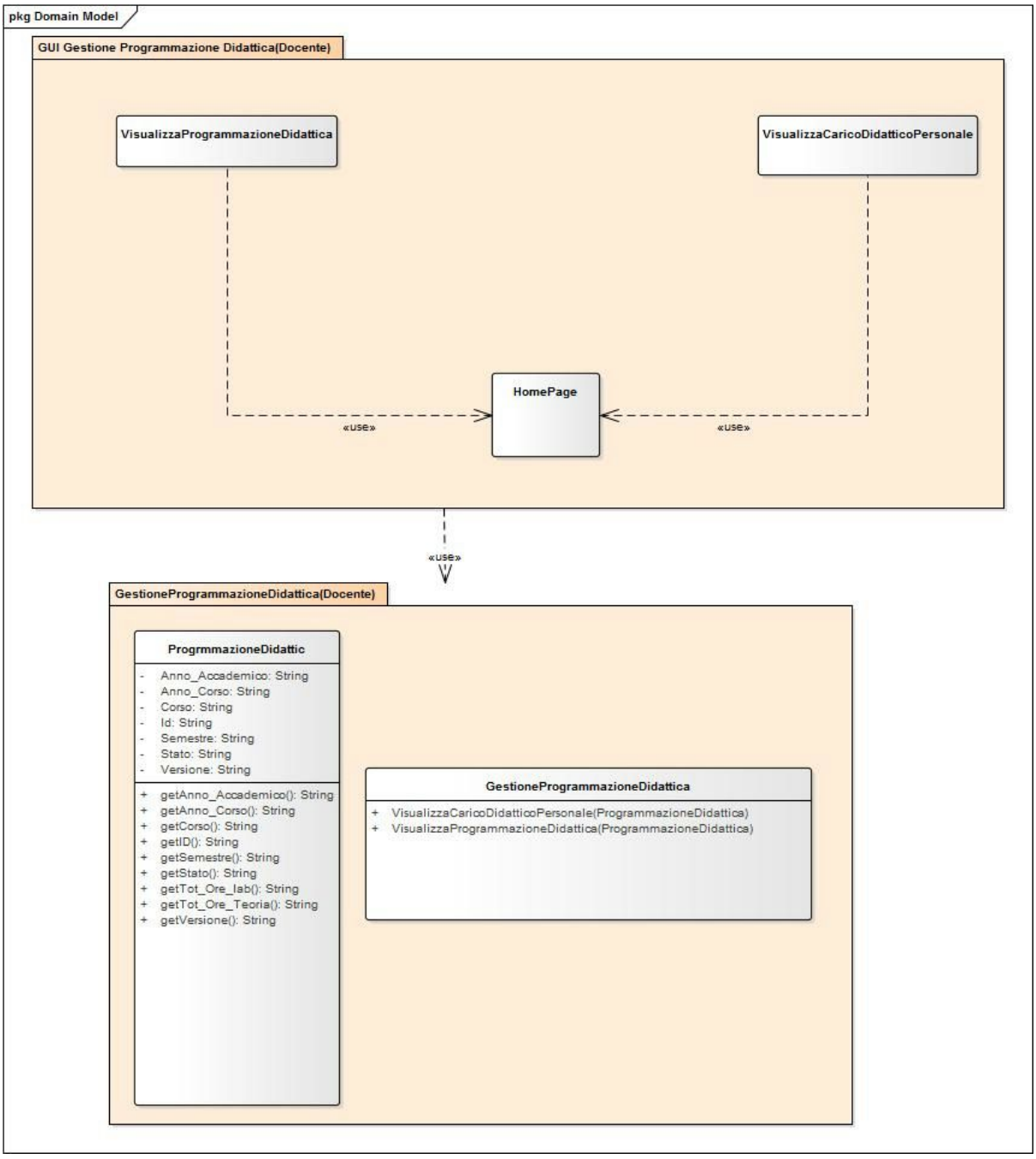
+getInsegnamenti(annoCorso:String, corso:String, curriculum:String, annoAccademico:String): List Deve essere presente nel database almeno un insegnamento
+**getMonteOre(nome:String, cognome:String): String** Il docente ricercato deve essere presente nel database
+getCaricoDidattico(username:String): String: username deve essere un valore valido.
+salvaStato(): null: la sessione deve contenere un nuovo stato valido per la programmazione didattica.
+cambiaStato(nuovoStato:String, id:String): null: nuovoStato deve essere un valore valido e id deve corrispondere ad una programmazione didattica presente nel database.
+getProgrammazioniDidattiche(): List: devono essere presenti delle programmazioni didattiche all'interno del database.
+getAccademico(corso:String): List: corso deve essere presente nel database la programmazione didattica per cui si vuole l'anno accademico
+getTotOreProf(matricola:String,ruolo:String): int: matricola deve corrispondere alla matricola di un docente presente all'interno del database, ruolo deve corrispondere ad un ruolo valido.
+getProgDidVis(corso:String, anno_accademico:String, anno_corso:String, utente:String,semestre:String):String corso, anno_accademico, anno_corso, utente, semestre devono essere dei valori validi.

Postcondizioni:

+creaProgDid(progD:progD, associa:Associa): null inserisce nel database l'associazione
+getCurriculum(corso:String, annoAccademico:String): String restituisce il curriculum
+getCurriculumProgDid(annoAcc:String, corso:String, semestre:int): String restituisce il curriculum
+getDocenti(): List Restituisce la lista dei docenti
+getInsegnamenti(annoCorso:String, corso:String, curriculum:String, annoAccademico:String): List Restituisce la lista degli insegnamenti
+**getMonteOre(nome:String, cognome:String): String** restituisce il monte ore assegnato al docente ricercato
+getCaricoDidattico(username:String): String: restituisce il carico didattico di un docente
+salvaStato(): null: salva il nuovo stato della programmazione didattica che viene preso da una variabile di sessione.
+cambiaStato(nuovoStato:String, id:String): null: memorizza all'interno di una variabile di sessione il nuovo stato da associare ad una programmazione didattica.
+getProgrammazioniDidattiche(): List: questo metodo restituisce le programmazioni didattiche presenti nel database.
+getAccademico(corso:String): List: questo metodo restituisce l'anno accademico di una programmazione didattica in base al corso selezionato.
+getTotOreProf(matricola:String,ruolo:String): int: questa funzione restituisce il totale ore di un dato docente.

+getProgDidVis(corso:String, anno_accademico:String, anno_corso:String, utente:String,semestre:String):String: questa funzione restituisce una programmazione didattica che viene ricercata.

6.6.2 Docente



Di seguito viene riportata la descrizione delle classi:

GUI Gestione Programmazione Didattica (Docente):

- Visualizza Programmazione Didattica
- Visualizza Carico Didattico Personale

Gestione Programmazione Didattica (Docente):

- Programmazione Didattica
- Gestione Programmazione Didattica

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

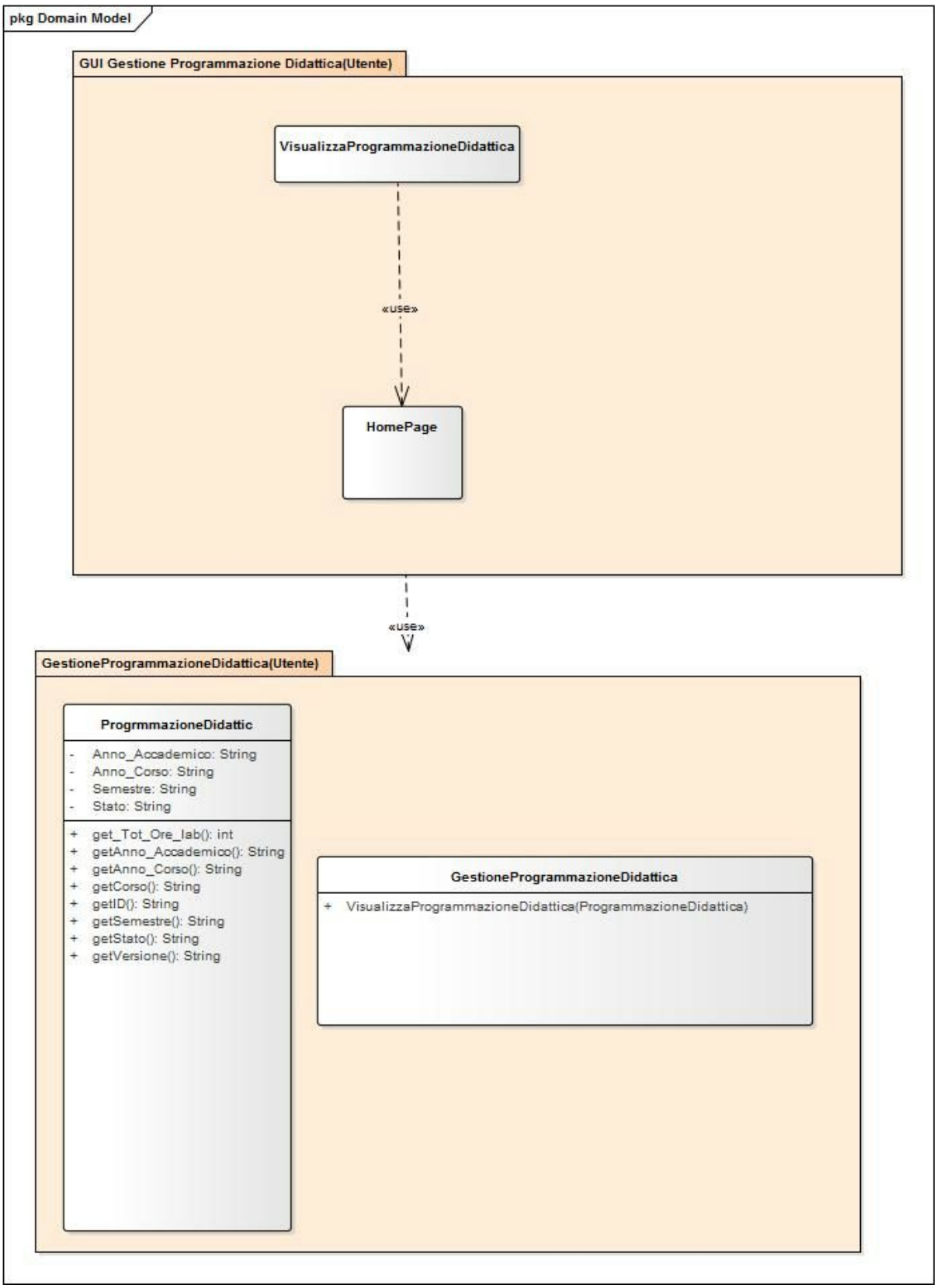
NomeClasse: Programmazione Didattica
Invariante: L' Id della programmazione deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeInterfaccia: Gestione Programmazione Didattica
Invariante: -

Precondizioni:
+ **visualizzaProgrammazioneDidattica(Docente): ArrayList <programmazioneDidattica>:** la programmazione didattica deve essere presente nel database
+ **visualizzaCaricoDidatticoPersonale(Docente):** il Presidente deve aver effettuato l'accesso.

Postcondizioni:
+ **visualizzaProgrammazioneDidattica(Docente): ArrayList <programmazioneDidattica>:** il sistema visualizza la programmazione didattica
+ **visualizzaCaricoDidatticoPersonale(Docente):** il campo non deve essere nullo

6.6.3 Utente


pkg Domain Model
GUI Gestione Programmazione Didattica(Utente)
VisualizzaProgrammazioneDidattica
«use»
HomePage
«use»
GestioneProgrammazioneDidattica(Utente)
ProgrmmazioneDidattic
- Anno_Accademico: String
- Anno_Corso: String
- Semestre: String
- Stato: String
+ get_Tot_Ore_lab(): int
+ getAnno_Accademico(): String
+ getAnno_Corso(): String
+ getCorso(): String
+ getID(): String
+ getSemestre(): String
+ getStato(): String
+ getVersione(): String
GestioneProgrammazioneDidattica
+ VisualizzaProgrammazioneDidattica(ProgrammazioneDidattica)

Di seguito viene riportata la descrizione delle classi:

GUI Gestione Programmazione Didattica (Utente):

- Visualizza Programmazione Didattica

Gestione Programmazione Didattica (Utente):

- Programmazione Didattica
- Gestione Programmazione Didattica

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

NomeClasse: Programmazione Didattica
Invariante: L' Id della programmazione deve essere univoco
Precondizioni: -
Postcondizioni: -

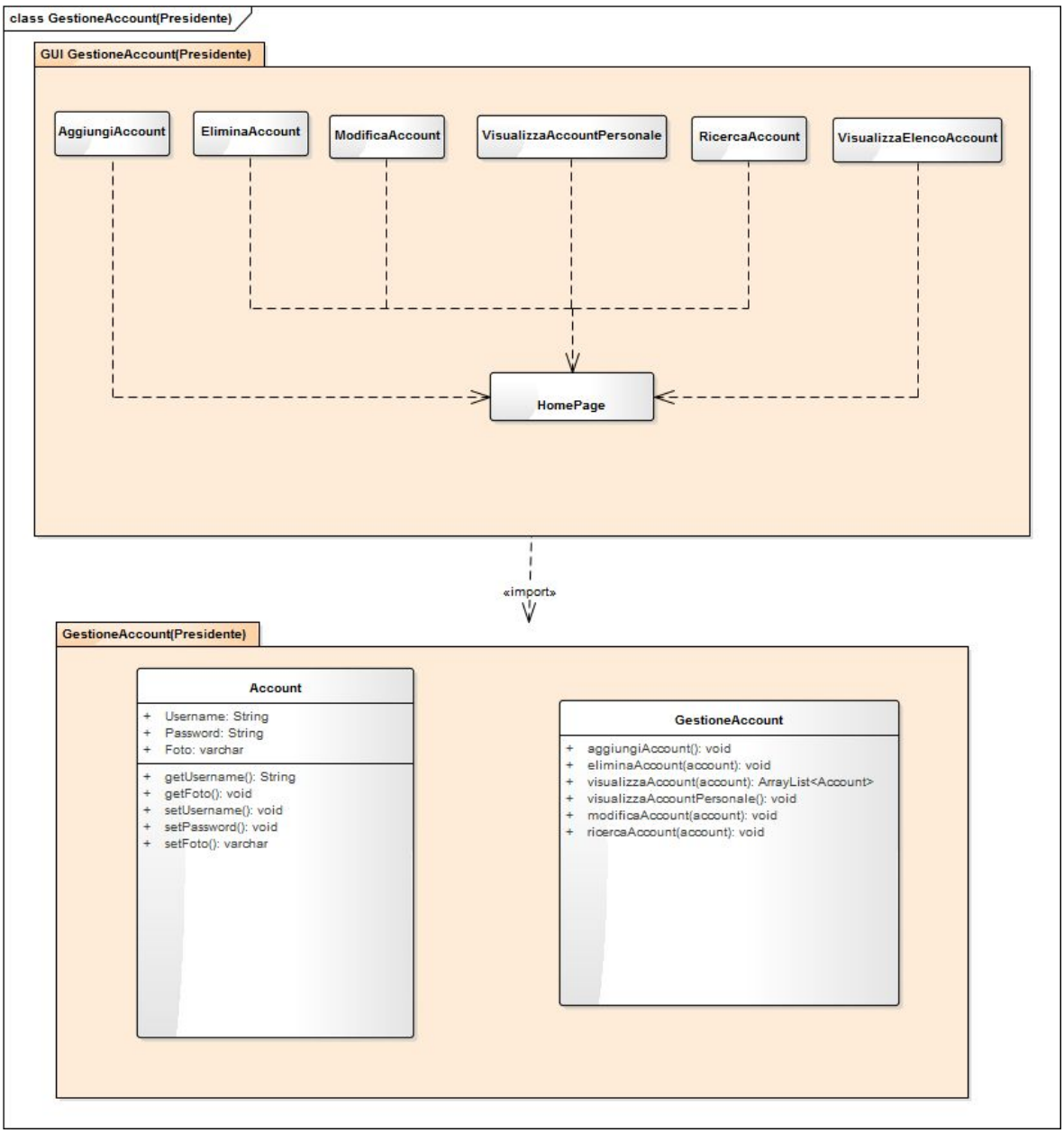
NomeInterfaccia: Gestione Programmazione Didattica
Invariante: -

Precondizioni:
+ visualizzaProgrammazioneDidattica(Docente): ArrayList <programmazioneDidattica>: la programmazione didattica deve essere presente nel database

Postcondizioni:
+ visualizzaProgrammazioneDidattica(): ArrayList <programmazioneDidattica>: il sistema visualizza la programmazione didattica

6.7 Gestione Account

6.7.1 Presidente



GUIGestioneAccount(Presidente)

- HomePage
- AggiungiAccount
- EliminaAccount
- ModificaAccount
- VisualizzaAccountPersonale
- RicercaAccount
- VisualizzaElencoAccount

GestioneAccount

- Account
- GestioneAccount

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

NomeClasse:Account
Invariante: L'username deve essere univoco
Precondizioni: -
Postcondizioni: -

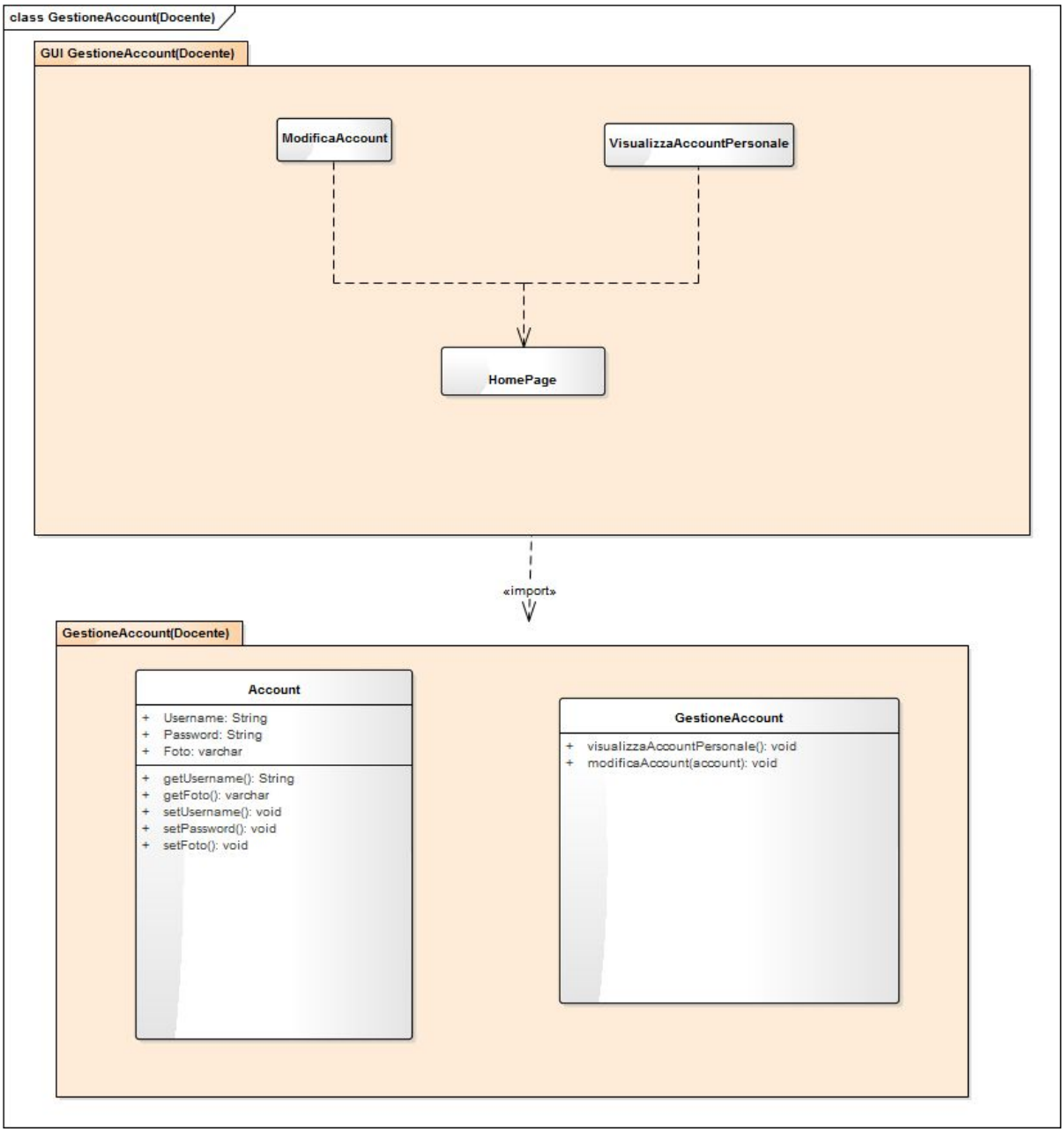
NomeInterfaccia: GestioneAccount
Invariante: -

Precondizioni:
+ **visualizzaElencoAccount(): ArrayList <Account>:** Gli account devono essere presente nel sistema.
+ **visualizzaAccountPersonale():**l'account deve essere presente nel sistema.
+ **aggiungiAccount(account):** il Presidente deve aver effettuato l'accesso.
+ **modificaAccount(account):** il Presidente deve aver effettuato l'accesso.
+ **eliminaAccount(account):** il Presidente deve aver effettuato l'accesso.

Postcondizioni:
+ **aggiungiAccount(account):** nessun campo deve essere nullo.
+ **modificaAccount(account):** nessun campo deve essere nullo.
+ **eliminaAccount(account):** nessun campo deve essere nullo.
+ **visualizzaElencoAccount: ArrayList <Account>:** il sistema visualizza l'elenco con tutti gli account.
+ **visualizzaAccountPersonale():** il sistema visualizza l'account.

GestioneAccount
-database
+aggiungiAccount(String matricola, String username, String password): **+modificaAccount(String username, String password):** **+eliminaAccount(String matricola, String username, String password):** **+aggiungiAccount(String matricola, String username, String password):** **+visualizzaElencoAccount(): ArrayList<String>** **+visualizzaAccountPersonale(String username, String password):**

6.7.2 Docente



GUIGestioneAccount(Docente)

- HomePage
- ModificaAccount
- VisualizzaAccountPersonale

GestioneAccount

- Account
- GestioneAccount

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

NomeClasse:Account
Invariante: L'username deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeInterfaccia: GestioneAccount
Invariante: -

GestioneAccount
-database
+modificaAccount(String username, String password): **+eliminaAccount(String matricola, String username, String password):** **+visualizzaAccountPersonale(String username, String password):**

Precondizioni:
+ **visualizzaAccountPersonale():**l'account deve essere presente nel sistema.
+ **modificaAccount(account):** il Docente deve aver effettuato l'accesso.

Postcondizioni:
+ **modificaAccount():** nessun campo deve essere nullo.
+ **visualizzaAccountPersonale():** il sistema visualizza l'account.

7.Glossario

Segue un elenco di termini, abbreviazioni e acronimi utilizzati all'interno di questo documento.

- **Pr. D.**: nome del sistema software sviluppato.

- **GUI**: Graphical User Interface.