

|  |  |  |
| --- | --- | --- |
| Composizione Team | | |
| Cognome | Nome | Acronimo |
| Iannaccone | Andrea | IA |
| Rossi | Domenico | RD |
| Maione | Ciro | MC |
| Saporito | Antonio | SA |

ODD

Object

Design

Document

Tirocinio 2.5

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Cambiamenti | Autori |
| 20/12/2018 | 0.1 | Creazione del documento | [tutti] |
| 22/12/2018 | 1.0 | Modifica del documento | [tutti] |
| 28/12/2018 | 1.1 | Inserimento Class diagram eseguito con objectAID | [tutti] |

Sommario

1. [1.Introduzione 2](#_Toc505101542)

[1.1. Object design trade-offs](#_Toc505101543)

[1.2. Descrizione componenti off-the-shelf](#_Toc505101544)

[1.3. Linee guida per la documentazione dell’interfaccia](#_Toc505101545)

[1.4. Definizioni, acronimi e abbreviazioni](#_Toc505101546)

[1.5. Riferimenti](#_Toc505101547)

1. [2.Design Pattern 4](#_Toc505101548)

[2.1. Strategy pattern](#_Toc505101549)

[2.2. Façade Pattern](#_Toc505101550)

1. [3.Packages 6](#_Toc505101551)

[PK\_1 – Package Generale](#_Toc505101552)

[PK\_2 – Package Interface Generale](#_Toc505101553)

[PK\_3 – Entity](#_Toc505101554)

[PK\_4 – Gestioni](#_Toc505101555)

1. [4.Interfacce delle classi 17](#_Toc505101556)

[4.1. CD\_GU](#_Toc505101557)

[4.2. CD\_GP](#_Toc505101558)

[4.3. CD\_GSTU](#_Toc505101559)

[4.4. CD\_GSG](#_Toc505101560)

1. [Glossario 22](#_Toc505101561)

# Introduzione

## Object design trade-offs

### **Comprensibilità vs costi**

Si preferisce aggiungere costi relativi alle ore/uomo dedicate per la documentazione, al fine di rendere il codice comprensibile sia alle persone non coinvolte nel progetto che alle persone coinvolte che non hanno lavorato a quella parte in particolare. Saranno introdotti commenti nel codice, per facilitarne la comprensione e la manutenzione.

### **Prestazioni vs Costi**

Il budget a disposizione non è eccessivo, quindi il sistema sarà sviluppato utilizzando componenti open source e free*.* Non saranno garantite alte prestazioni, ma saranno comunque soddisfacenti per il normale utilizzo.

### **Costi vs Mantenimento**

Il sistema sviluppato può essere facilmente modificato ed implementato con nuove funzioni e corretto in presenza di errori, grazie all’uso di materiale open source e con l’utilizzo di javadoc.

### **Interfaccia vs Easy-use**

L’interfaccia permette un facile utilizzo (Easy-use) delle principali funzionalità del sistema anche per gli utenti meno esperti.

### **Memoria vs efficienza**

Dato l’elevato carico di utenti che possono accedere al sistema, e dato che il sistema dovrà supportare un elevato numero di query, per non penalizzare le performance del sistema, verrà utilizzato un meccanismo di caching che permette la memorizzazione dei dati con accesso più frequente. Ciò sarà implementato attraverso l’utilizzo di Cookie.

### **Sicurezza vs costi**

Dato il budget ridotto, non saranno utilizzate componenti esterne che garantiscano la massima sicurezza sui dati, ma verrà utilizzata una componente all’interno del linguaggio Java che permetta un grado di protezione soddisfacente.

### **Interfacce vs Tempo di risposta**

Il tempo di risposta tra server e interfaccia è più che sufficienti a soddisfare le richieste da parte dell’utente.

## Descrizione componenti off-the-shelf

Per il progetto software che si vuole realizzare facciamo uso di componenti off-the-shelf, che sono componenti software disponibili sul mercato per facilitare la creazione del progetto. Per il sistema che si vuole realizzare faremo uso di un framework per la realizzazione dell’interfaccia grafica e di una libreria interna a Java per la crittografia dei dati. Il framework scelto per l’interfaccia grafica è Bootstrap, un framework open source che contiene una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Questo contiene modelli di progettazione basati su HTML e CSS, per le varie componenti dell'interfaccia, come moduli, bottoni e navigazione, così come alcune estensioni opzionali di JavaScript.

## Linee guida per la documentazione dell’interfaccia

Per rendere il codice più estensibile e manutenibile, prima dell’implementazione della logica del sistema, è opportuno sottomettere le regole di implementazione, in modo che eventuali correzioni nella logica dell’applicazione possano essere apportate prima di imbattersi nella sintassi degli strumenti scelti.

**Commenti**

I commenti di implementazione sono dei mezzi per commentare il codice o per commentare una particolare implementazione. Infine dovrà essere generato il javadocs con l’utilizzo della funzione di Eclipse (Generate Javadoc).

I commenti dovrebbero essere usati per dare una panoramica del codice e per fornire informazioni aggiuntive che non sono prontamente disponibili nel codice stesso.

I metodi devono essere preceduti da un commento, o più precisamente da una documentazione che riporti l’obiettivo che si vuole raggiungere.

**Dichiarazioni**

Posizionare le dichiarazioni all’inizio del blocco del codice. Non dichiarare le variabili al loro

primo uso, può portare incomprensioni verso il programmatore e rendere la manutenibilità del codice più complessa e di difficile comprensione.

**Indentazione**

L’indentazione del codice HTML deve essere effettuata con un TAB, invece per la scrittura del codice si userà il checkstyle di Google, il quale attraverso dei plugin usati con Eclipse permetteranno di indentare correttamente il codice secondo le specifiche Google.

Es. Codice html

<html>

<head>

</head>

<body>

<div>

</div>

</body>

</html>

Deve essere sostituita da:

<html>

<head>

</head>

<body>

<div>

</div>

</body>

</html>

**Parentesi**

A prescindere dalle istruzioni che seguono un IF o ciclo FOR e WHILE, è necessario, laddove ci fosse anche una sola istruzione, riportare il blocco di istruzioni tra parentesi graffe.

**Script Javascript**

Gli script che svolgono funzioni distinte dal funzionamento di una pagina, dovrebbero essere collocati in file separati. Le funzioni e oggetti Javascript devono essere preceduti da un commento in stile Javadoc.

## Definizioni, acronimi e abbreviazioni

**HTML:** Linguaggio di mark-up per pagine web.

**CSS:** Linguaggio usato per definire la formattazione di pagine web.

**Framework:** Software di supporto allo sviluppo web.

**Javascript:** 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.

**Ecplise:** ambiente di sviluppo integrato multi-linguaggio e multi-piattaforma.

**Javadoc:** è un applicativo incluso nel Java Development Kit utilizzato per la generazione automatica della documentazione del codice sorgente scritto in linguaggio Java.

**Off-The-Shelf:** Servizi esterni di cui viene fatto utilizzo da terzi.

## Riferimenti

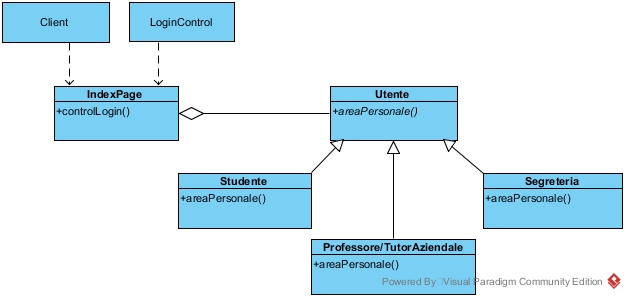
* Bernd Bruegge & Allen H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns and Java*, (2nd edition), Prentice-Hall, 2003
* Ian Sommerville, Software Engineering, Addison Wesely
* Tirocinio2.5\_SDD\_V\_1.0

# Design Pattern

## Strategy pattern

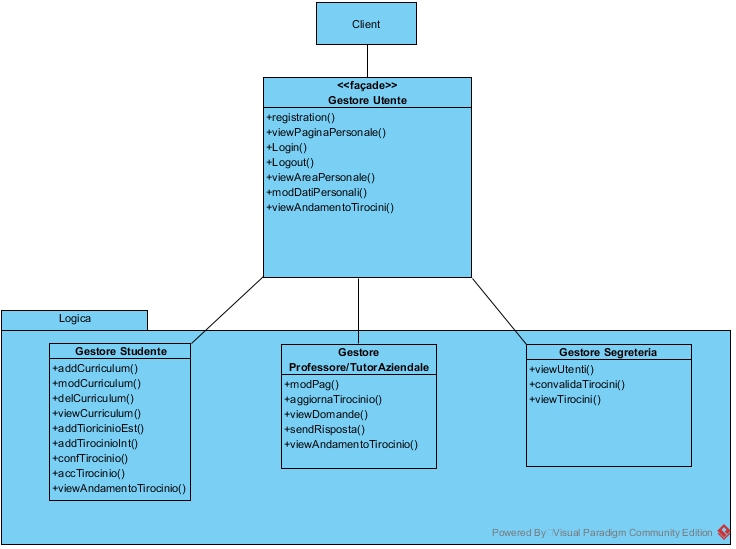
Questo pattern prevede che gli algoritmi siano intercambiabili tra loro, in base ad una specificata condizione, in modalità trasparente al client che ne fa uso. In altre parole, data una famiglia di algoritmi che implementa una certa funzionalità essi dovranno esportare sempre la medesima interfaccia, così il [client](https://it.wikipedia.org/wiki/Client) dell'algoritmo non dovrà fare nessuna assunzione su quale sia la strategia istanziata in un particolare istante.

Questo pattern verrà utilizzato per la realizzazione dell’interfaccia che permette di poter selezionare l’area personale dedicata in base all’utente loggato.



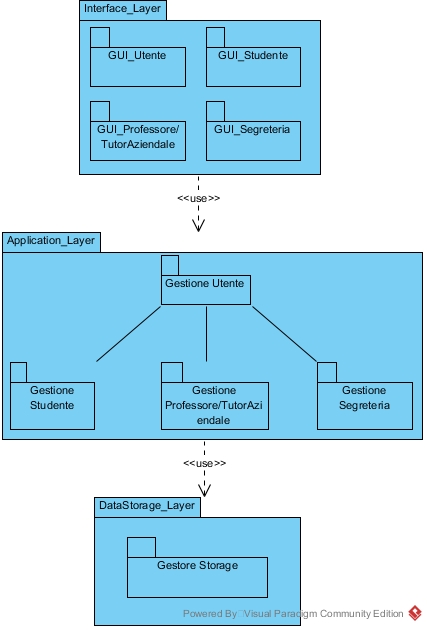
## Façade Pattern

Questo pattern fornisce un’unica interfaccia per accedere ad un insieme di oggetti che compongono un sottosistema. Il sistema farà uso del Façade Pattern per definire un’unica interfaccia a livello di logica che permette all’utente di interagire, attraverso l’interfaccia grafica, con le funzionalità del sistema vedendole come un unico sistema.



# Packages

## PK\_1 – Package Generale



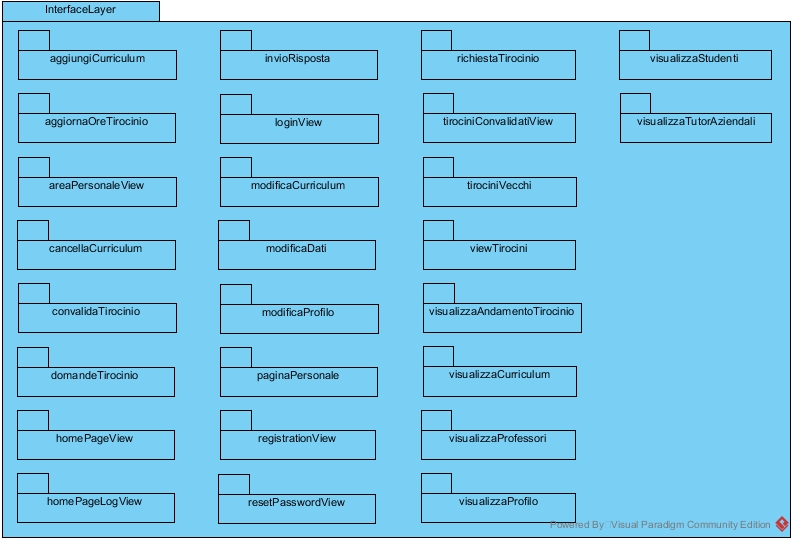
Il diagramma descrive la natura three-layer dell’applicazione mostrandone i tre package principali:

**InterfaceLayer:** GUI\_Utente, GUI\_Studente, GUI\_Professore/TutorAziendale e GUI\_Segreteria, indicano i sottosistemi che contengono tutti gli oggetti boundary.

**Application\_Layer:** contiene i quattro sottosistemi individuati (Gestione Utente, Gestione Studente, Gestione Professore/TutorAziendale e Gestione Segreteria).

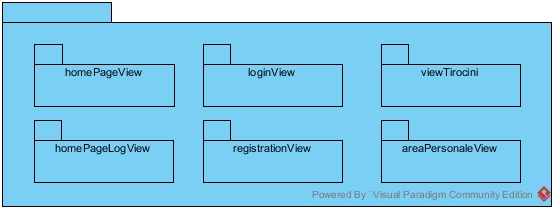
**DataSorage\_Layer:** GestoreStorage, è il sottosistema che ha il compito di effettuare operazioni verso il database.

## PK\_2 – Package Interface Generale



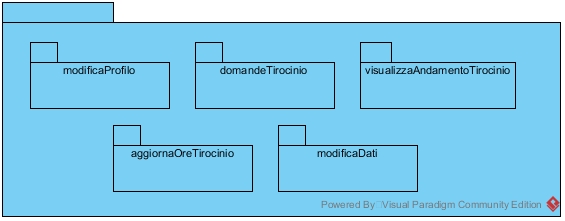
Il diagramma descrive le interfacce delle varie sezioni del sistema: **Gestione Utente, Gestione Professore/TutorAziendale, Gestione Segreteria**.

### **PK\_GU: Gestione Utenti**



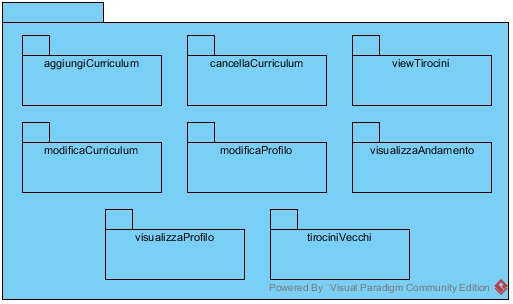
Il diagramma descrive le interfacce del sottosistema “Gestione Utente” ed in particolare le view per l’utente.

### **PK\_GP: Gestione Professore/TutorAziendale**



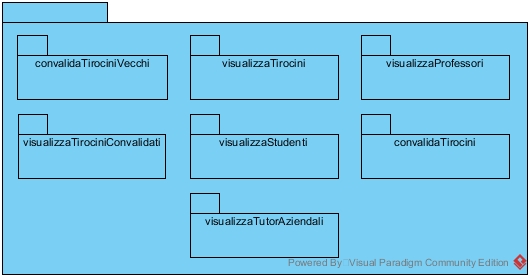
Il diagramma descrive le interfacce del sottosistema “Gestione Professore” e “Gestione TutorAziendale”.

### **PK\_GSTU: Gestione Studente**



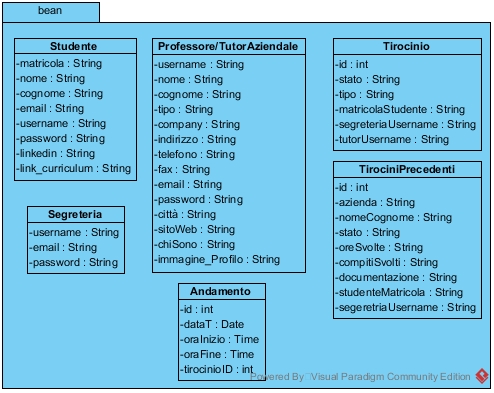
Il diagramma descrive le interfacce del sottosistema “Gestione Studente”.

### **PK\_GSG: Gestione Segreteria**



Il diagramma descrive le interfacce del sottosistema “Gestione Segreteria”.

## PK\_3 – Entity



Il package principale “bean” contiene al suo interno i package delle entity del nostro sistema. Per ogni entity sono definite le variabili e il tipo di queste.

### **PK\_3: Studente**

|  |  |
| --- | --- |
| **Nome** | Studente |
| **Descrizione** | Questa classe rappresenta l’oggetto Studente registrato alla piattaforma. |
| **Signature dei metodi** | getMatricola();  setMatricola(String matricola);  getNome();  setNome(String nome);  getCognome();  setCognome(String cognome);  getEmail();  setEmail(String email);  getUsername();  setUsername(String username);  getPassword();  setPassword(String password);  getLinkedin();  setLinkedin (String Linkedin);  getLink\_curriculum();  setLink\_curriculum (String Link\_curriculum); |
| **Precondizione** | **context:** Studente :: setMatricola(matricola) && setEmail(email) && setUsername(username) **pre:** matricola, email e username non devono avere altre corrispondenze nel database. |
| **Post-condizione** |  |
| **Invariante** |  |

### **PK\_3: Professore/TutorAziendale**

|  |  |
| --- | --- |
| **Nome** | Professore/TutorAziendale |
| **Descrizione** | Questa classe rappresenta l’oggetto professore o tutor aziendale registrato alla piattaforma. |
| **Signature dei metodi** | getID();  setID(int ID);  getStato();  setStato(String stato);  getMatricolaStudente();  setMatricolaStudente(String matricola);  getSegreteriaUser();  setSegreteriaUser(String segreteriaUser);  getTutorUsername();  setTutorUsername(String tutorUser); |
| **Precondizione** | **context:** Professore/TutorAziendale :: setID(ID) **pre:** non deve avere altre corrispondenze nel database. |
| **Post-condizione** |  |
| **Invariante** |  |

### **PK\_3: Segreteria**

|  |  |
| --- | --- |
| **Nome** | Segreteria |
| **Descrizione** | Questa classe rappresenta l’oggetto Segreteria del dipartimento di informatica. |
| **Signature dei metodi** | getUsername();  setUsername(String username);  getEmail();  setEmail(String email);  getPassword();  setPassword(String password); |
| **Precondizione** | **context:** Segreteria :: setUsername(username) && setEmail(email) **pre:** non devono avere altre corrispondenze nel database. |
| **Post-condizione** |  |
| **Invariante** |  |

### **PK\_3: Tirocinio**

|  |  |
| --- | --- |
| **Nome** | Tirocinio |
| **Descrizione** | Questa classe rappresenta l’oggetto tirocinio una volta che esso è stato richiesto. |
| **Signature dei metodi** | getID();  setID(int ID);  getStato();  setStato(String stato);  getTipo();  setTipo(String tipo);  getMatricolaStudente();  setMatricolaStudente(String matricolaStudente);  getUserSegreteria();  setUserSegreteria(String userSegreteria);  getUserTutor();  setUserTutor(String userTutor); |
| **Precondizione** | **context:** Tirocinio :: setID(id) **pre:** non deve avere altre corrispondenze nel database. |
| **Post-condizione** | **context:** Tirocinio :: setStato(stato) **post:** deve essere compreso nell’enumerazione dello stato del tirocinio. |
| **Invariante** |  |

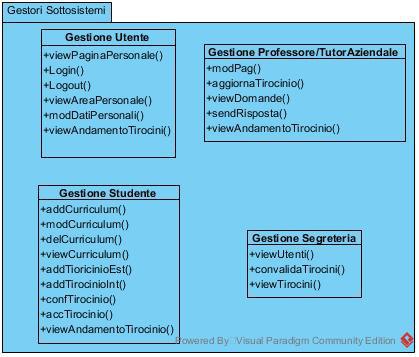
### **PK\_3: TirociniPrecedenti**

|  |  |
| --- | --- |
| **Nome** | TirociniPrecedenti |
| **Descrizione** | Questa classe rappresenta l’oggetto tirocinio che è stato effettuato nel passato una volta che esso è stato richiesto. |
| **Signature dei metodi** | getID();  setID(int ID);  getAzienda();  setAzienda(String Azienda);  getNomeCognome();  setNomeCognome(String NomeCognome);  getStato();  setStato(String stato);  getOreSvolte();  setOreSvolte(String OreSvolte);  getCompitiSvolti();  setCompitiSvolti(String CompitiSvolti);  getDocumentazione();  setDocumentazione(String Documentazione);  getMatricolaStudente();  setMatricolaStudente(String matricolaStudente);  getUserSegreteria();  setUserSegreteria(String userSegreteria); |
| **Precondizione** | **context:** TirociniPrecedenti :: setID(id) **pre:** non deve avere altre corrispondenze nel database. |
| **Post-condizione** | **context:** Tirocinio :: setStato(stato) **post:** deve essere compreso nell’enumerazione dello stato del tirocinio.  **context:** TirociniPrecedenti :: setMatricolaStudente(MatricolaStudente) **post:** deve esistere un’unica corrispondenza nel database.  **context:** TirociniPrecedenti :: setUserSegreteria (UserSegreteria) **post:** deve esistere un’unica corrispondenza nel database. |
| **Invariante** |  |

### **PK\_3: Andamento**

|  |  |
| --- | --- |
| **Nome** | Andamento |
| **Descrizione** | Questa classe rappresenta l’orario di inizio e fine per visualizzare l’andamento nel tempo del tirocinio. |
| **Signature dei metodi** | getID();  setID(int ID);  getData();  setData(Date data);  getOraInizio();  setOraInizio(Time oraInizio);  getOraFine();  setOraFine(Time oraFine);  getTirocinioID();  setTirocinioID(int IDTirocinio); |
| **Precondizione** | **context:** Andamento :: setID(ID) **pre:** non deve avere altre corrispondenze nel database. |
| **Post-condizione** | **context:** setTirocinioID(IDTirocinio) **post:** l’id deve essere collegato ad un tirocinio confermato. |
| **Invariante** | **context:** Andamento :: setData(data) **pre:** data <= data odierna  **context:** Andamento :: setOraFine(oraFine) **pre:** oraFine>oraInizio |

## PK\_4 – Gestioni



Il package “Gestioni” fa parte dell’application logic e si focalizza sulle classi che si occuperanno di implementare la logica del sistema.

### **PK\_4: Gestione Utente**

|  |  |
| --- | --- |
| **Nome** | gestioneUtente |
| **Descrizione** | Questa classe rappresenta il gestore del sottosistema “Gestione Utente”. |
| **Signature dei metodi** | viewPaginaPersonale();  Login();  Logout();  viewAreaPersonale();  modDatiPersonali();  viewAndamentoTirocini(); |
| **Precondizione** | **context:** gestioneUtente :: pre: username != username(database) |
| **Post-condizione** |  |
| **Invariante** |  |

### **PK\_4: Gestione Studente**

|  |  |
| --- | --- |
| **Nome** | gestioneStudente |
| **Descrizione** | Questa classe rappresenta il gestore del sottosistema “Gestione Studente”. |
| **Signature dei metodi** | addCurriculum();  modCurriculum();  delCurriculum();  viewCurriculum();  addTirocinioEst();  addTirocinioInt();  confTirocinio();  accTirocinio();  viewAndamentoTirocinio(); |
| **Precondizione** | **context:** gestioneStudente :: viewCurriculum() **pre:** deve essere prima caricato un curriculum.  **context:** gestioneStudente :: modCurriculum() **pre:** deve essere prima caricato un curriculum.  **context:** gestioneStudente :: delCurriculum() **pre:** deve essere prima caricato un curriculum.  **context:** gestioneStudente :: viewAndamentoTirocinio() **pre:** bisogna che nel profilo dello studente sia attivo almeno un tirocinio. |
| **Post-condizione** |  |
| **Invariante** |  |

### **PK\_4: Gestione Professore/TutorAziendale**

|  |  |
| --- | --- |
| **Nome** | gestioneProfessore/TutorAziendale |
| **Descrizione** | Questa classe rappresenta il gestore del sottosistema “Gestione Professore/TutorAziendale”. |
| **Signature dei metodi** | modPag();  aggiornaTirocinio();  viewDomanda();  sendRisposta();  viewAndamentoTirocinio(); |
| **Precondizione** |  |
| **Post-condizione** |  |
| **Invariante** |  |

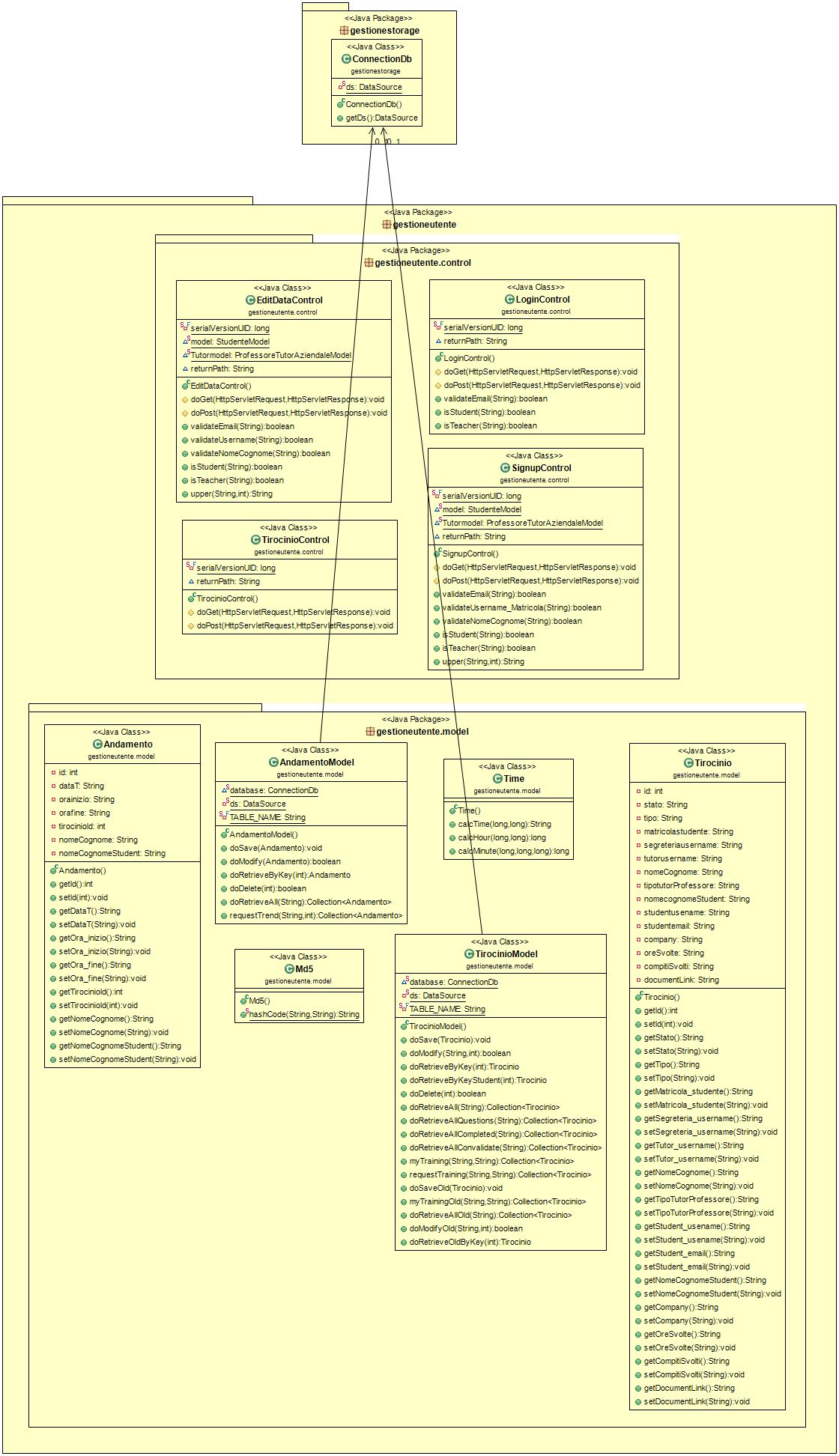
### **PK\_4: Gestione Segreteria**

|  |  |
| --- | --- |
| **Nome** | gestioneSegreteria |
| **Descrizione** | Questa classe rappresenta il gestore del sottosistema “Gestione Segreteria”. |
| **Signature dei metodi** | viewUtenti();  convalidaTirocini();  viewTirocini(); |
| **Precondizione** |  |
| **Post-condizione** |  |
| **Invariante** |  |

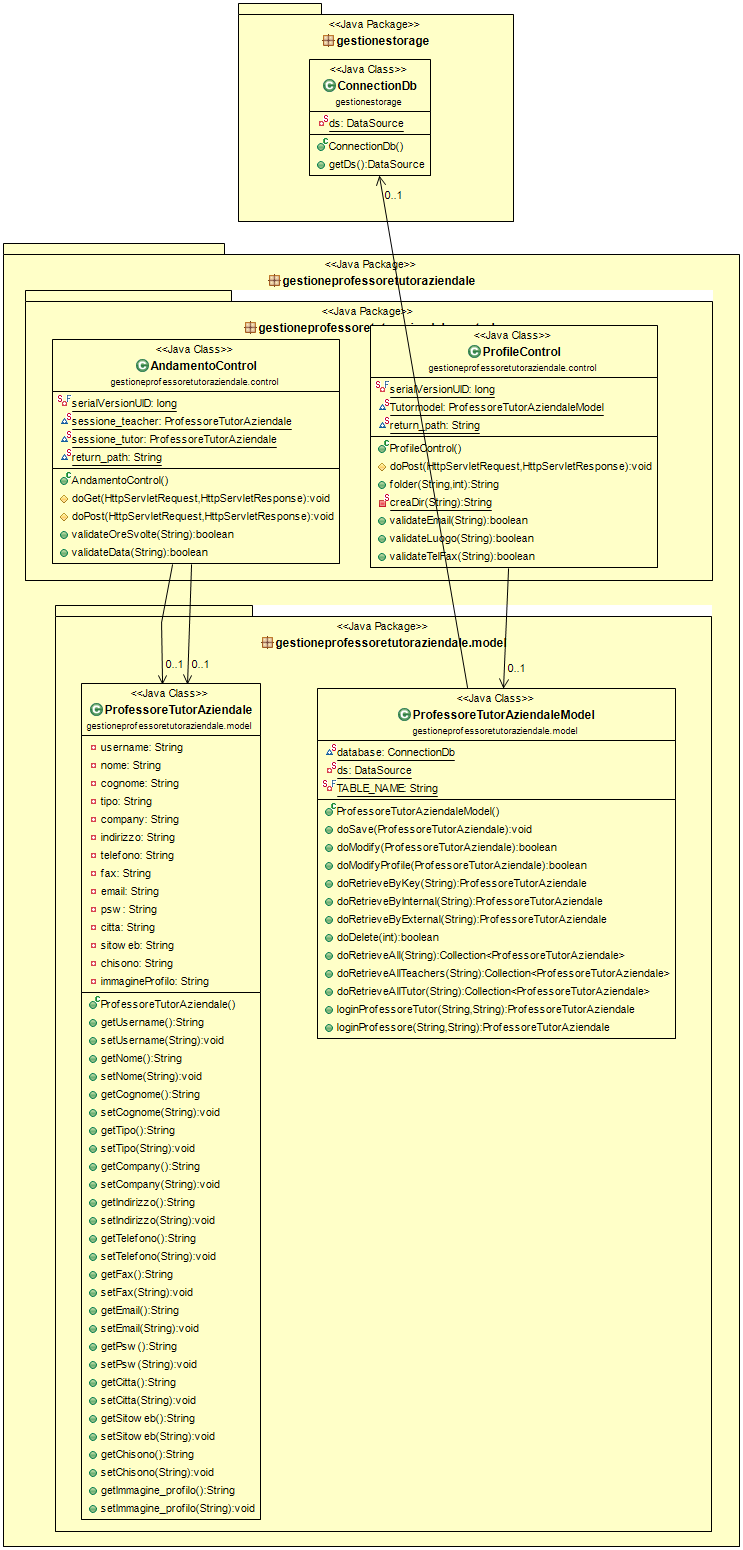
# Interfacce delle classi

I seguenti diagrammi dei packages sono stati realizzati utilizzando ObjectAID, un plugin del framework di Eclipse, dopo aver completato lo sviluppo.

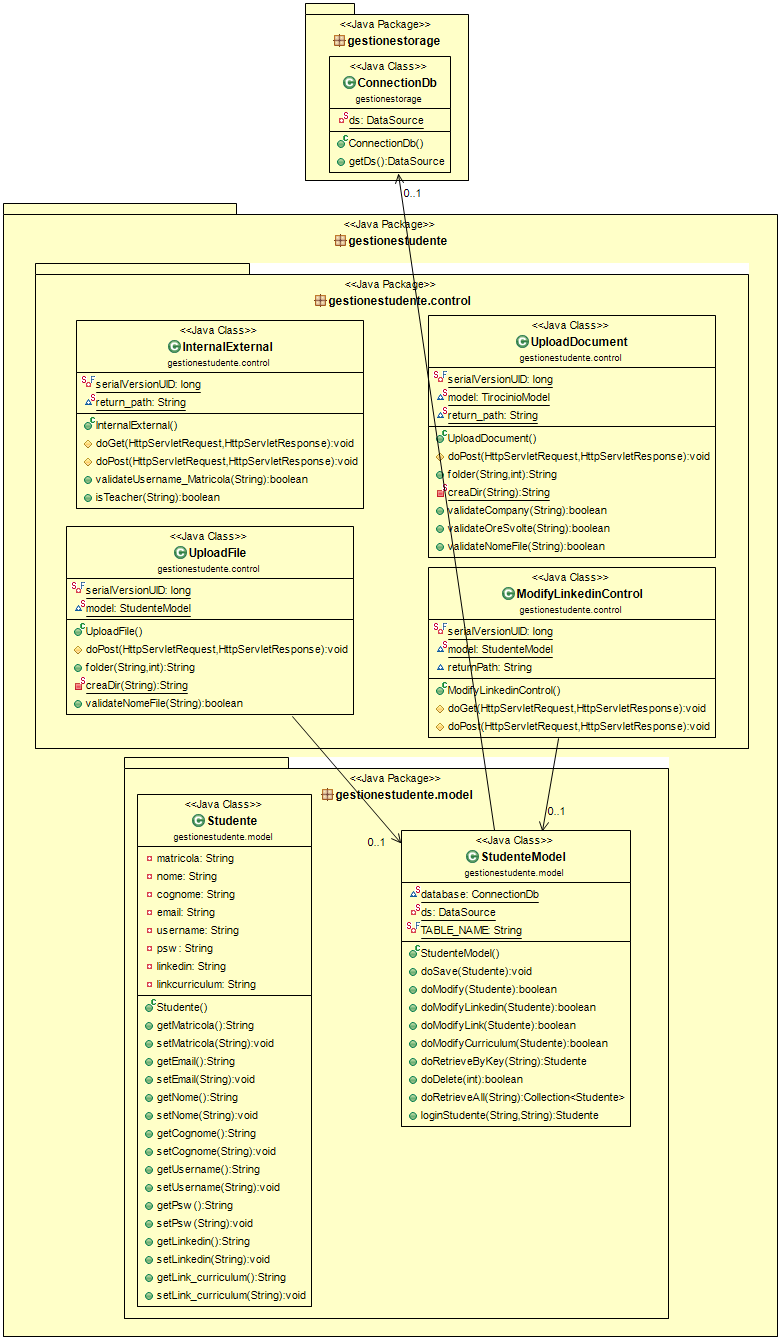
## CD\_GU



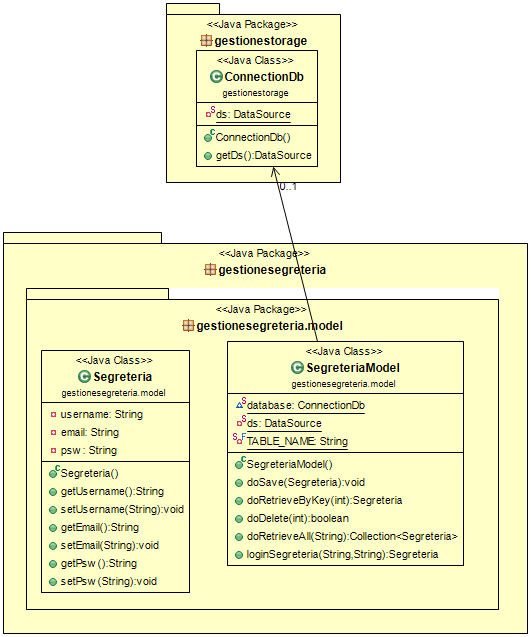
## CD\_GP



## CD\_GSTU



## CD\_GSG



# Glossario

**Package:** pacchetto**;**

**Open source:** software non protetto da copyright e liberamente modificabile dagli utenti

**ODD:** Object Design Document;

**Query:** interrogazione di un database per estrarre o aggiornare i dati che soddisfano un certo

criterio di ricerca.