OBJECT DESIGN DOCUMENT

Sommario

1. Introduzione

1.1 Trade offs

1.1.1 Compatibilità vs costi

Si preferisce aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte nel progetto o le persone coinvolte che non hanno lavorato a quella parte in particolare. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorare la comprensibilità agevola il mantenimento e anche il processo di modifica.

1.1.2 Memoria vs efficienza

Si è preferita la memoria anziché le prestazioni perché salvare risultati parziali o complessivi avrebbe causato il mantenimento di informazioni incompleti, difficili da integrare e interpretare. 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.

1.1.3 Costi vs mantenimento

Il sistema 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.

1.1.4 Interfacce vs Easy-use

L'interfaccia, grazie ad una impostazione semplice e intuitiva, permette facile utilizzo (Easyuse) delle principali funzionalità del sistema anche per gli utenti meno esperti.

1.2 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.

1.3 Linee guida per la documentazione dell'interfaccia

Per rendere il codice più estensibile e mantenibile, 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, con il nome/i dell'autore/i. Inoltre bisogna commentare, giustificare le eventuali decisioni particolari o i calcoli.

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 deve essere effettuata con un TAB e a prescindere dal linguaggio usato per la produzione del codice, ogni istruzione deve essere opportunamente indentata.

Esempio:

```
<html>
<head>
</head>
<body>
</body>
</html>
```

Deve essere sostituita da:

```
<html>
        <head>
        </head>
        <body>
        </body>
</html>
```

Parentesi

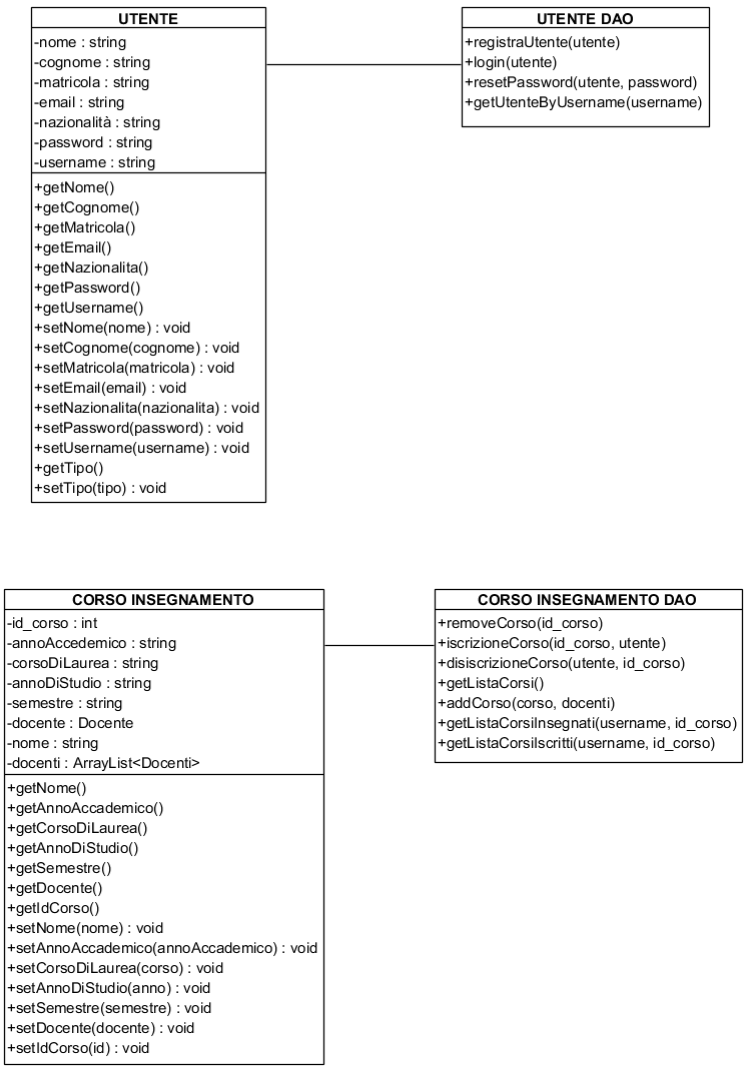
A prescindere dalle istruzioni che seguono un if o ciclo for e while, è necessario, laddove ci fosse

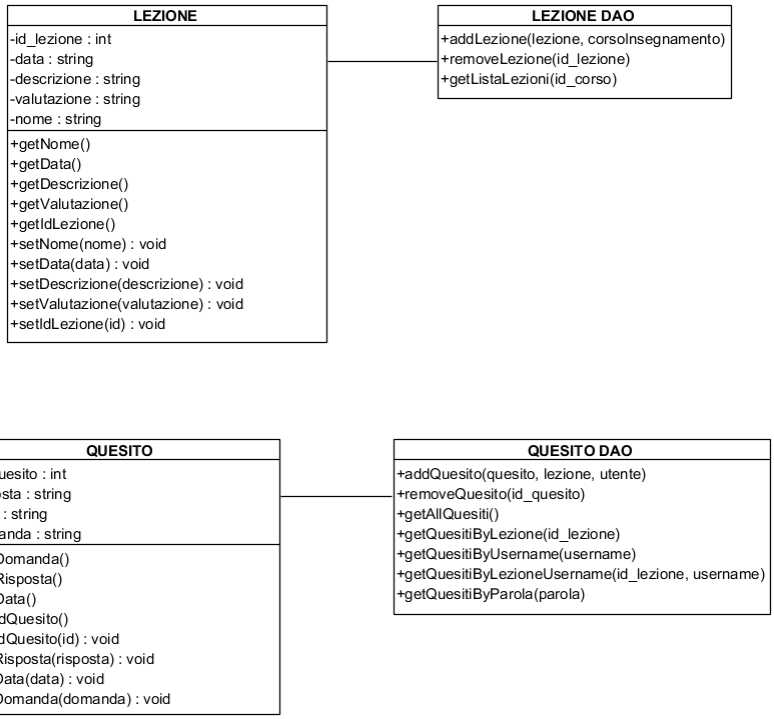
anche una sola istruzione, riportare il blocco di istruzioni tra parentesi graffe. Ogni tag di apertura deve essere necessariamente seguito dall'apposito tag di chiusura.
Una convenzione importante, per quanto riguarda l'inserimento di numeri o di valori costanti, è quella di non usare una codifica fissa (hard coding) che è fortemente sconsigliata ma di associare sempre il valore ad una variabile o semplicemente definire una macro che può essere richiamata da eventi ed essere parametrizzata. In questo modo si facilita la modifica, sostituendo solo il valore della variabile o macro, in un unico posto.

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.

Dipendenze dei sottosistemi





1.5 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.

1.6 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
- SDD_UniQuestions

2. Packages

Per Unisask è stata adottata la seguente suddivisione in pacchetti, scomponendo principalmente il sistema secondo l'architettura three-layer adottata.

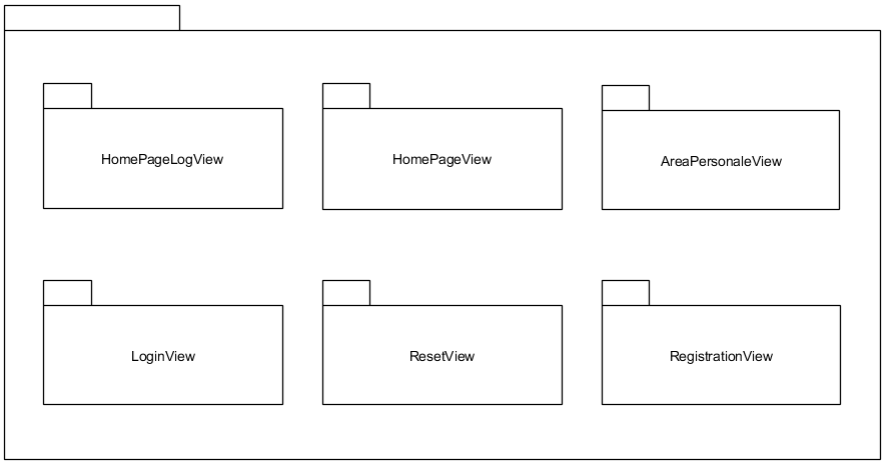
• **Interface Layer**: che contiene gli oggetti boundary, ovvero le view dei quattro sottosistemi individuati (Gestione Utente, Gestione Corsi Insegnamento, Gestione Lezioni, Gestione Domande).

• **Application Logic Layer**: contiene la logica dei quattro sottosistemi individuati (Gestione Utente, Gestione Corsi Insegnamento, Gestione Lezioni, Gestione Domande) con JavaBean e Servlet.

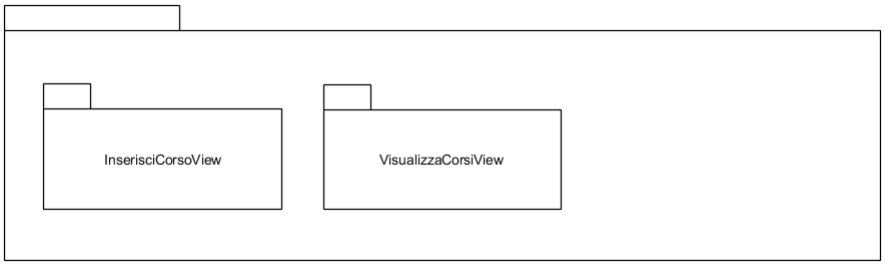
• **Data Storage Layer**: Formato da quattro classi principali che si occuperanno di interfacciarsi al database e di passare le varie query.

2.1 PK_InterfaceLayer

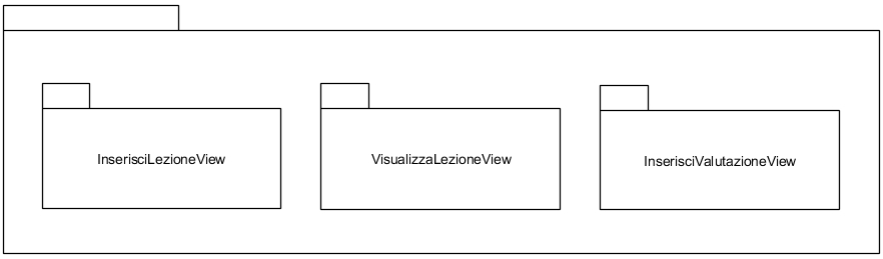
2.1.1 PK_Interface_GestioneUtente



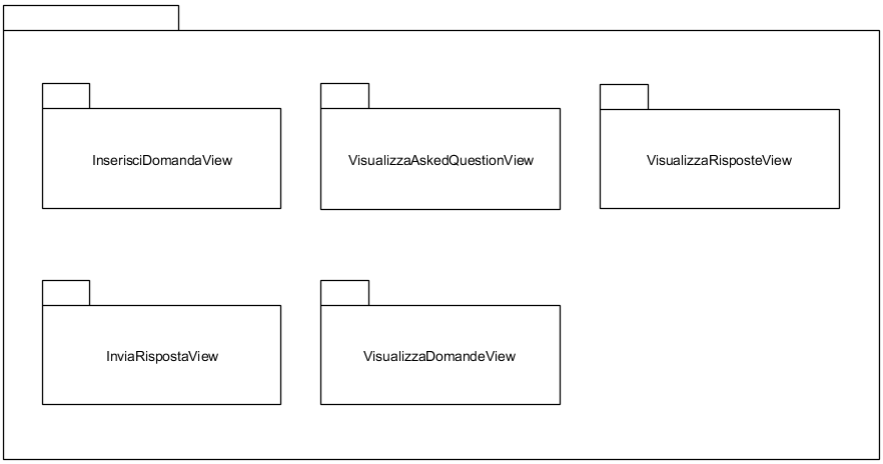
2.1.2 PK_Interface_GestioneCorsiInsegnamento



2.1.3 PK_Interface_GestioneLezioni

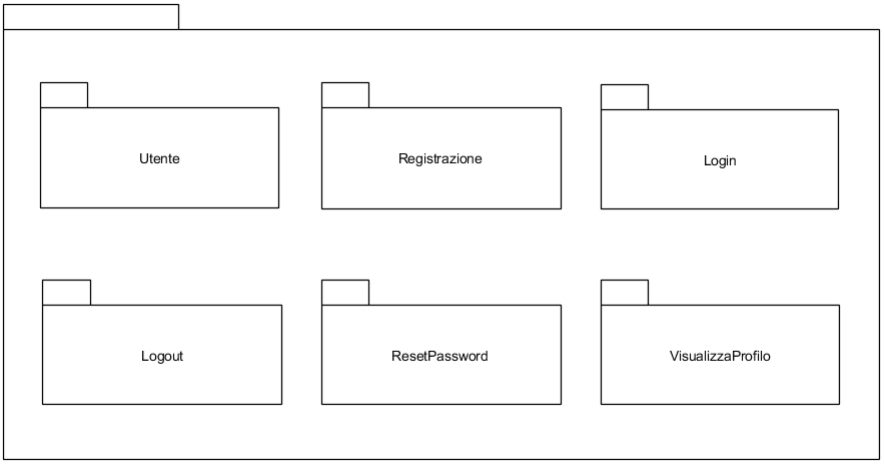


2.1.4 PK_Interface_GestioneDomande

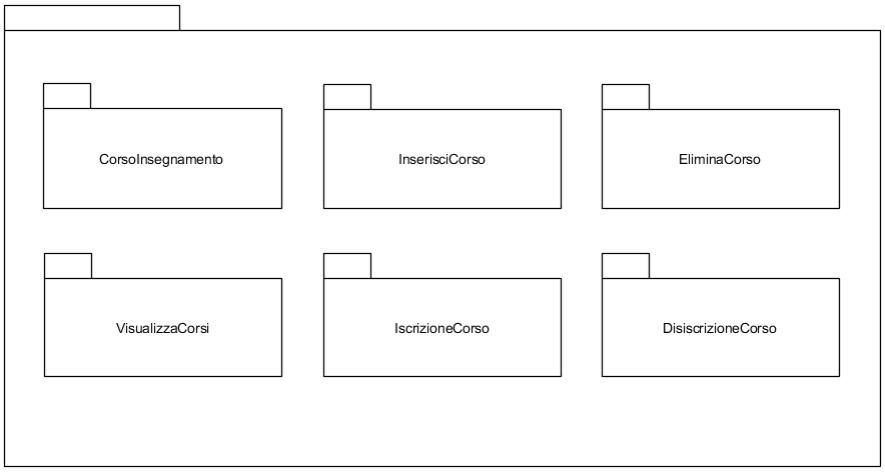


2.2 PK_ApplicationLogicLayer

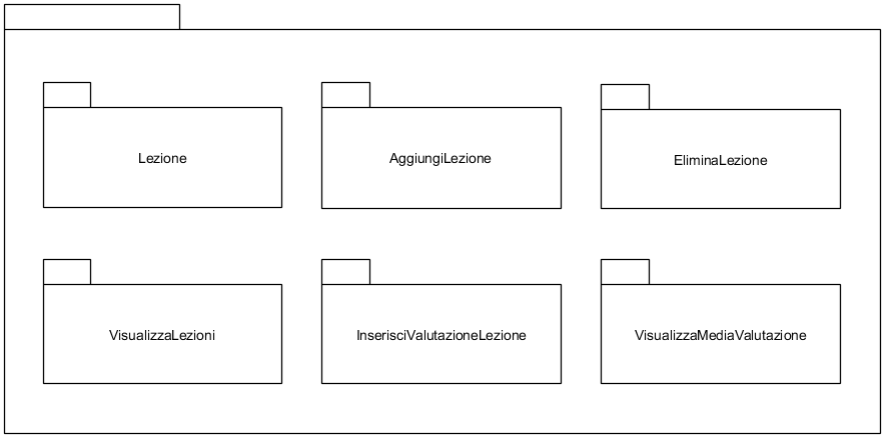
2.2.1 PK_Logic_GestioneUtente



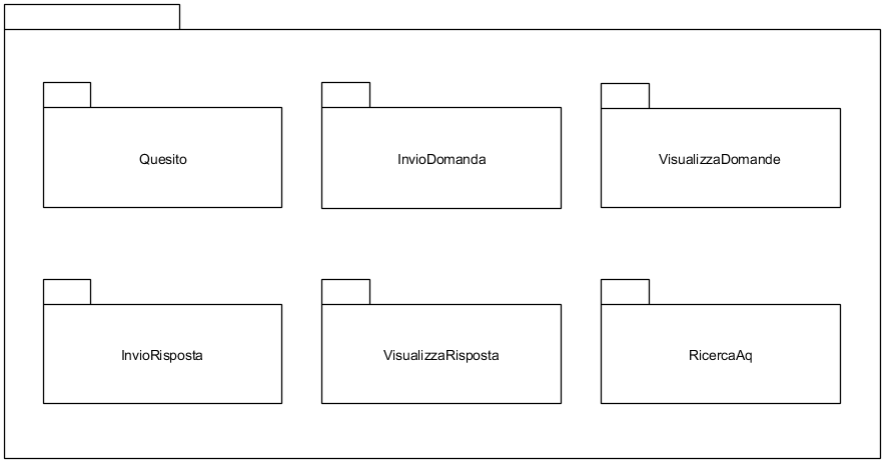
2.2.2 PK_Logic_GestioneCorsiInsegnamento



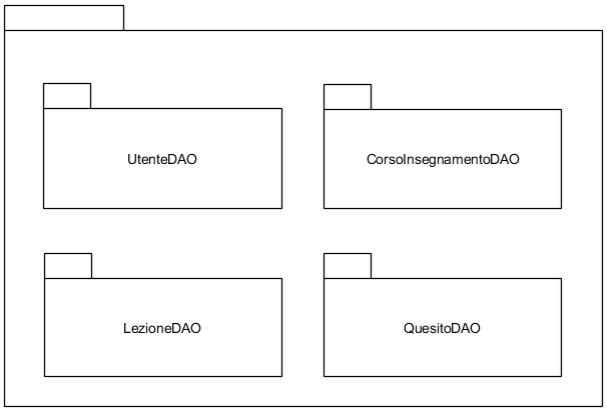
2.2.3 PK_Logic_GestioneLezioni



2.2.4 PK_Logic_GestioneDomande



2.3 PK_StorageLayer



3. Interfacce delle classi

Nome classe	Utente
Descrizione	Questa classe rappresenta l'utente registrato al sito
Attributi	- nome: String - cognome: String - matricola: String - password: String - username: String - email: String - password: String - nazionalità: String
Signature dei metodi	+ getNome() + setNome(String nome) + getCognome() + setCognome(String cognome) + getMatricola() + setMatricola(String matricola) + getPassword() + setPassword(String password) + getUsername() + setUsername(String username) + getEmail() + setEmail(String email)

	+ getPassword() + setPassword(String password) + getNazionalità() + setNazionalità(String nazionalità) + getTipo() + setTipo(String tipo)
Pre-condizioni	**Context**: Utente :: setMatricola(String matricola); **pre**: matricola non deve avere altre corrispondenze nel database. **Context**: Utente :: setUsername(String username); **pre**: username non deve avere altre corrispondenze nel database. **Context**: Utente :: setEmail(String email); **pre**: email non deve avere altre corrispondenze nel database.
Post-condizioni	
Invarianti	
Nome classe	**Corso insegnamento**
Descrizione	Questa classe rappresenta un corso d'insegnamento
Attributi	- id_corsoInsegnamento: int - nome: String - semestre: String - corsoDiLaurea: String - annoAccademico: String - annoDiStudio: String
Signature dei metodi	+ getId() + setId(int id) + getNome() + setNome(String nome) + getSemestre() + setSemestre(String semestre) + getCorsoDiLaurea() + setCorsoDiLaurea(String corsoDiLaurea) + getAnnoAccademico() + setAnnoAccademico(String annoAccademico) + getAnnoDiStudio() + setAnnoDiStudio(String annoDiStudio)
Pre-condizioni	**Context**: CorsoInsegnamento :: setId(int corsoInsegnamento); **pre**: id_corsoInsegnamento non deve avere altre corrispondenze nel database.

Post-condizioni	
Invarianti	

Nome classe	Lezione
Descrizione	Questa classe rappresenta una lezione
Attributi	- id_lezione: int - nome: String - data: String - descrizione: String - valutazione: String
Signature dei metodi	+ getId() + setId(int id) + getNome() + setNome(String nome) + getData() + setData(String data) + getDescrizione() + setDescrizione(String descrizione) + getValutazione() + setValutazione(String valutazione)
Pre-condizioni	**Context**: Lezione :: setId(int corsoInsegnamento); **pre**: id_lezione non deve avere altre corrispondenze nel database.
Post-condizioni	
Invarianti	

Nome classe	Quesito
Descrizione	Questa classe rappresenta un quesito posto ad un docente
Attributi	- id_quesito: int - domanda: String - risposta: String - data_quesito: String
Signature dei metodi	+ getId() + setId(int id) + getDomanda()

	+ setDomanda(String domanda) + getRisposta() + setRisposta(String risposta) + getDataQuesito() + setDataQuesito(String dataQuesito)
Pre-condizioni	**Context**: Quesito :: setId(int id_quesito); **pre**: id_quesito non deve avere altre corrispondenze nel database.
Post-condizioni	
Invarianti	

Nome classe	Utente DAO
Descrizione	Questa classe gestisce le interazioni della classe Utente con il database
Attributi	-connection: Connection
Signature dei metodi	+ registraUtente(Utente utente) + login(Utente utente) + resetPassword(Utente utente, String password) + getUtenteByUsername(String username)
Pre-condizioni	**Context**: Utente :: registraUtente(utente); **pre**: utente non deve avere altre corrispondenze nel database
Post-condizioni	
Invarianti	

Nome classe	Corso insegnamento DAO
Descrizione	Questa classe gestisce le interazioni della classe CorsoInsegnamento con il database
Attributi	-connection: Connection
Signature dei metodi	+ removeCorso(int id_corso) + iscrizioneCorso(int id_corso, Utente utente) + disiscrizioneCorso(int id_corso, Utente utente) + addCorso(Corso corso, ArrayList<Docente> docenti) + getListaCorsi()

	+ getListaCorsiIscritti(String username, int id_corso) + getListaCorsiInsegnati(String username, int id_corso)
Pre-condizioni	
Post-condizioni	
Invarianti	

Nome classe	Lezione DAO
Descrizione	Questa classe gestisce le interazioni della classe Lezione con il database
Attributi	-connection: Connection
Signature dei metodi	+ addLezione(Lezione lezione, CorsoInsegnamento corso) + removeLezione(int id_lezione) + getListaLezioni(int id_corso)
Pre-condizioni	
Post-condizioni	
Invarianti	

Nome classe	Quesito DAO
Descrizione	Questa classe gestisce le interazioni della classe Quesito con il database
Attributi	-connection: Connection
Signature dei metodi	+ addQuesito(Quesito quesito, Lezione lezione, Utente utente) + removeQuesito(int id_quesito) + getAllQuesiti() + getQuesitiByLezione(int id_lezione) + getQuesitiByUsername(String username) + getQuesitiByLezioneUsername(int id_lezione, String username) + getQuesitiByParola(String parola)
Pre-condizioni	

Post-condizioni	
Invarianti	