UNIVERSITÀ DEGLI STUDI
DI SALERNO

Ingegneria del Software

A.A. 2017/2018

Object Design Document

Top Manager:

Professore
Andrea De Lucia

Partecipanti:

Nome	Matricola
Vittorio Ventura	05121 05766

Indice

1.Introduzione

1.1 Object Design Trade-offs

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea di massima quello che sarà il nostro sistema e quindi i nostri obiettivi, tralasciando gli aspetti implementativi. Il seguente documento ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate 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 sono specificati i trade-off e le linee guida.

Comprensibilità vs Tempo:

Il codice deve essere quanto più comprensibile possibile per facilitare la fase di testing ed eventuali future modifiche. Il codice sarà quindi accompagnato da commenti che ne semplifichino la comprensione. Ovviamente questa caratteristica aggiungerà un incremento di tempo allo sviluppo del nostro progetto.

Prestazioni vs Costi:

Il progetto è portato avanti senza alcuna sovvenzione economica, di conseguenza tutti i componenti che formeranno il sistema sono scritti da 0, si farà il possibile per evitare di intaccare il meno possibile le prestazioni del sistema per abbassare il più possibile i costi dell'intero sistema

Interfaccia vs Usabilità:

L'interfaccia grafica è stata realizzata in modo da essere molto semplice, chiara e concisa. Fa uso di form e pulsanti disposti in maniera da rendere semplice l'utilizzo del sistema da parte dell'utente finale. Fin dal primo momento la comodità di utilizzo è stata messa in primo piano durante il design, scelta che si rispecchia fin dai primi mock-up.

Sicurezza vs Efficienza:

Il poco tempo a disposizione non ci permette di rendere il sistema resistente a possibili attacchi esterni. Metteremo comunque il minimo di sicurezza possibile nel progetto, assicurandoci che il login/registrazione e tutte le altre interazioni con il database siano sicure e quanto più efficienti possibili.

1.2 Linee Guida per la Documentazione delle Interfacce

Lo sviluppatore dovrà seguire alcune linee guida per la scrittura del codice:

Naming Convention

- **È** buona norma utilizzare nomi:

    1. Descrittivi
    2. Pronunciabili
    3. Di uso comune
    4. Lunghezza medio-corta
    5. Non abbreviati
    6. Evitando la notazione ungherese
    7. Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili

- I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la maiuscola. Quest'ultime devono essere dichiarate ad inizio blocco, solamente una per riga e devono essere tutte allineate per facilitare la leggibilità.

- E' inoltre possibile, in alcuni casi, utilizzare il carattere underscore "_", ad esempio quando utilizziamo delle variabili costanti oppure quando vengono utilizzate delle proprietà statiche.

Metodi:

- I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica un'azione, seguito dal nome di un oggetto. I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Le variabili dei metodi devono essere dichiarate appena prima del loro utilizzo e devono servire per un solo scopo, per facilitarne la leggibilità. Esistono però casi particolari come ad esempio nell'implementazione dei model, dove viene utilizzata l'interfaccia CRUD.

Esempio: getId(), setId()

Classi e pagine:

- I nomi delle classi devono cominciare con una lettera maiuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di queste ultime devono fornire informazioni sul loro scopo.
- I nomi delle pagine devono cominciare con una lettera minuscola. Tutte le parole al loro interno sono scritte in minuscolo e separate dal carattere '-'.
- I nomi delle servlet sono analoghi a quelli delle classi, con l'aggiunta della parola "Servlet" come ultima parola.

Ogni file sorgente java contiene una singola classe e dev'essere strutturato in un determinato modo:

- L'istruzione include che permette di importare all'interno della classe gli altri oggetti che la classe utilizza.

- La dichiarazione di classe caratterizzata da:

    1. Dichiarazione della classe pubblica
    2. Dichiarazioni di costanti
    3. Dichiarazioni di variabili di classe
    4. Dichiarazioni di variabili d'istanza
    5. Costruttore

1.3 Definizioni, acronimi e abbreviazioni

Acronimi:

- RAD: Requirements Analysis Document
- SDD: System Design Document
- ODD: Object Design Document
- CRUD: Create Read Update Delete

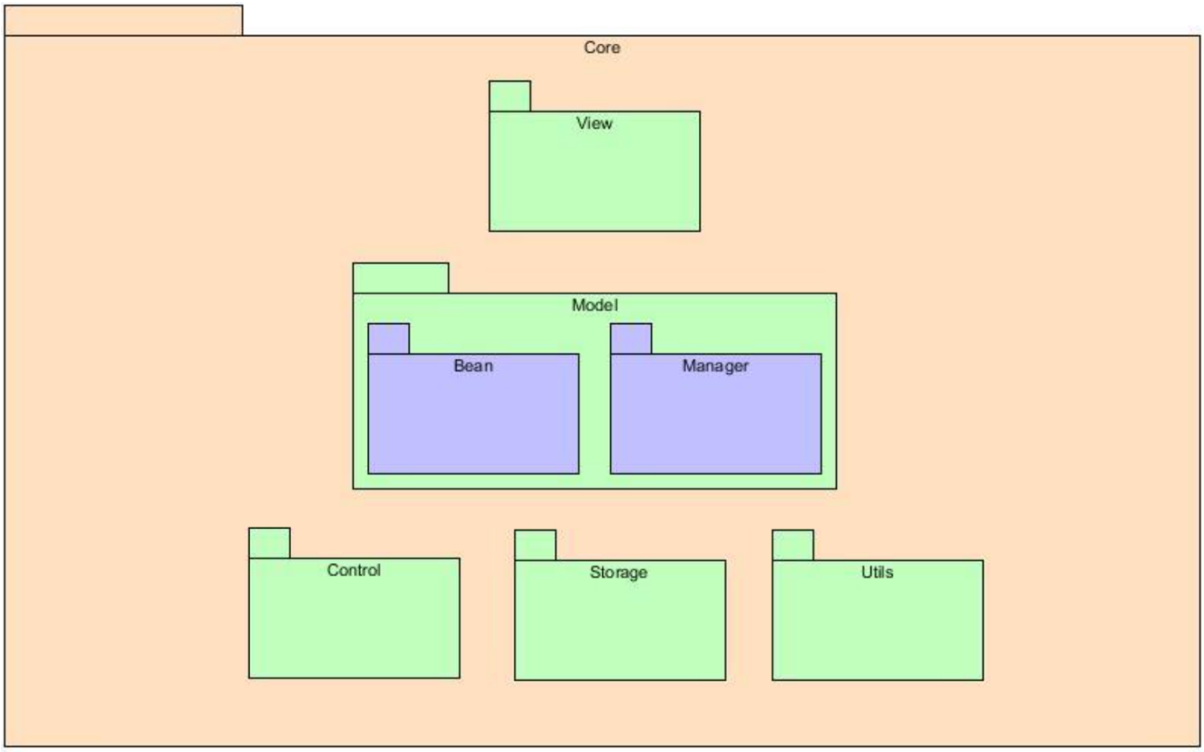
Abbreviazioni:

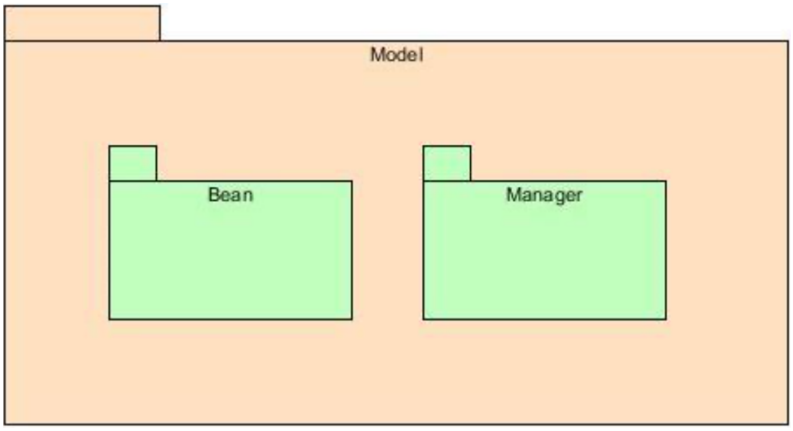
- DB: DataBase

1.4 Riferimenti

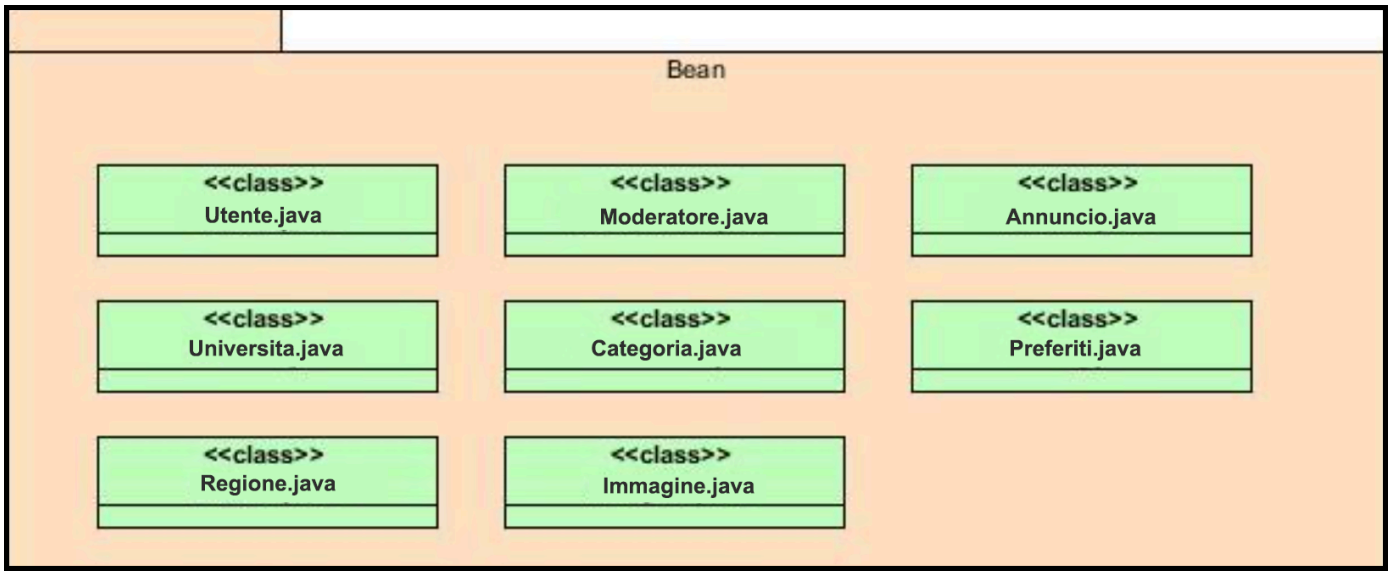
- B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java,Prentice Hall, 3rd edition, 2009
- Documento SDD del progetto UniAds
- Documento RAD del progetto UniAds

2. Packages


Core
View
Model
Bean
Manager
Control
Storage
Utils


Model
Bean
Manager

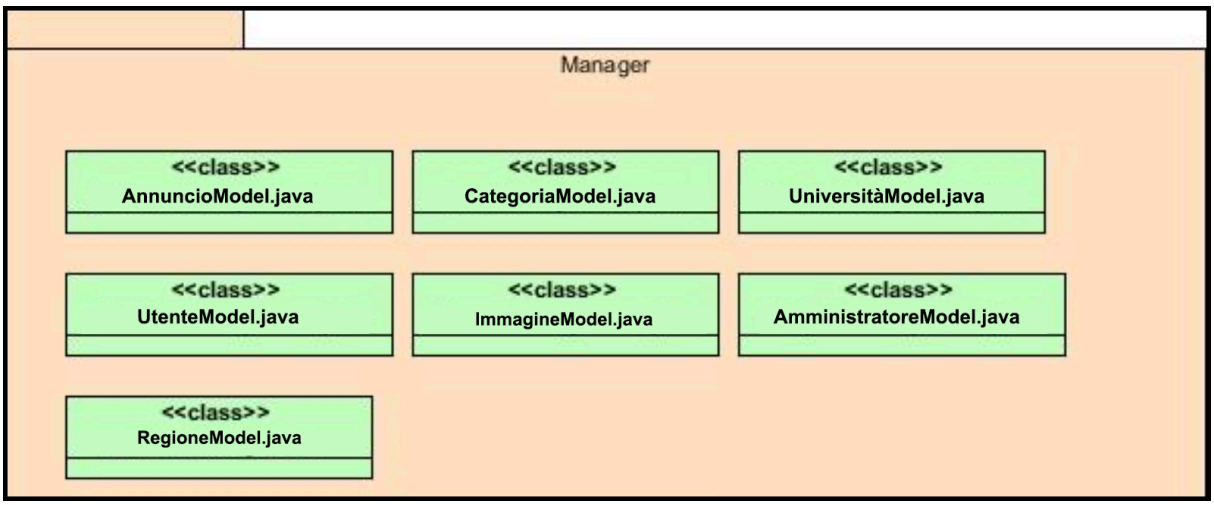
2.1.1 Packages Entities



Classe:	**Descrizione:**
Admin.java	Descrive un amministratore del sistema.
Utente.java	Descrive una persona che fa uso del sistema.
Annuncio.java	Descrive un annuncio all'interno del sistema.
Universita.java	Descrive un'università all'interno del sistema.
Categoria.java	Descrive una categoria all'interno del sistema.

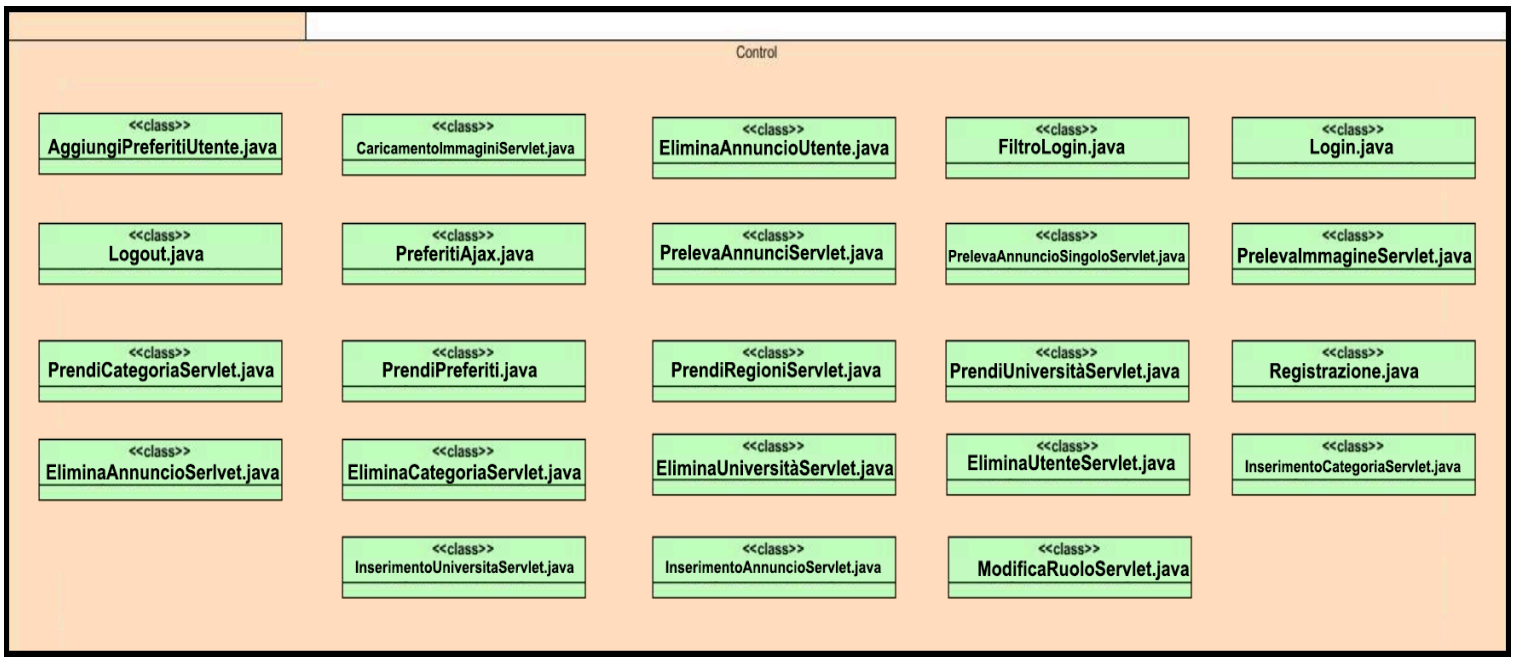
Preferiti.java	Descrive i preferiti all’interno del sistema.
Regioni.java	Descrive la regione in cui è localizzata un’ università.
Immagine.java	Descrive un immagine all’interno del sistema.

2.1.2 Packages Manager



Classe:	**Descrizione:**
AnnuncioModel.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce gli annunci
CategoriaModel.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce le categorie
UniversitàModel.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce le università
UtenteModel.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce le registrazioni di utenti.
ImmagineModel.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce le immagini
AmministratoreModel.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce gli amministratori
RegioneModel.java	Si tratta di una classe che funge da interfaccia del sottosistema che gestisce le regioni.

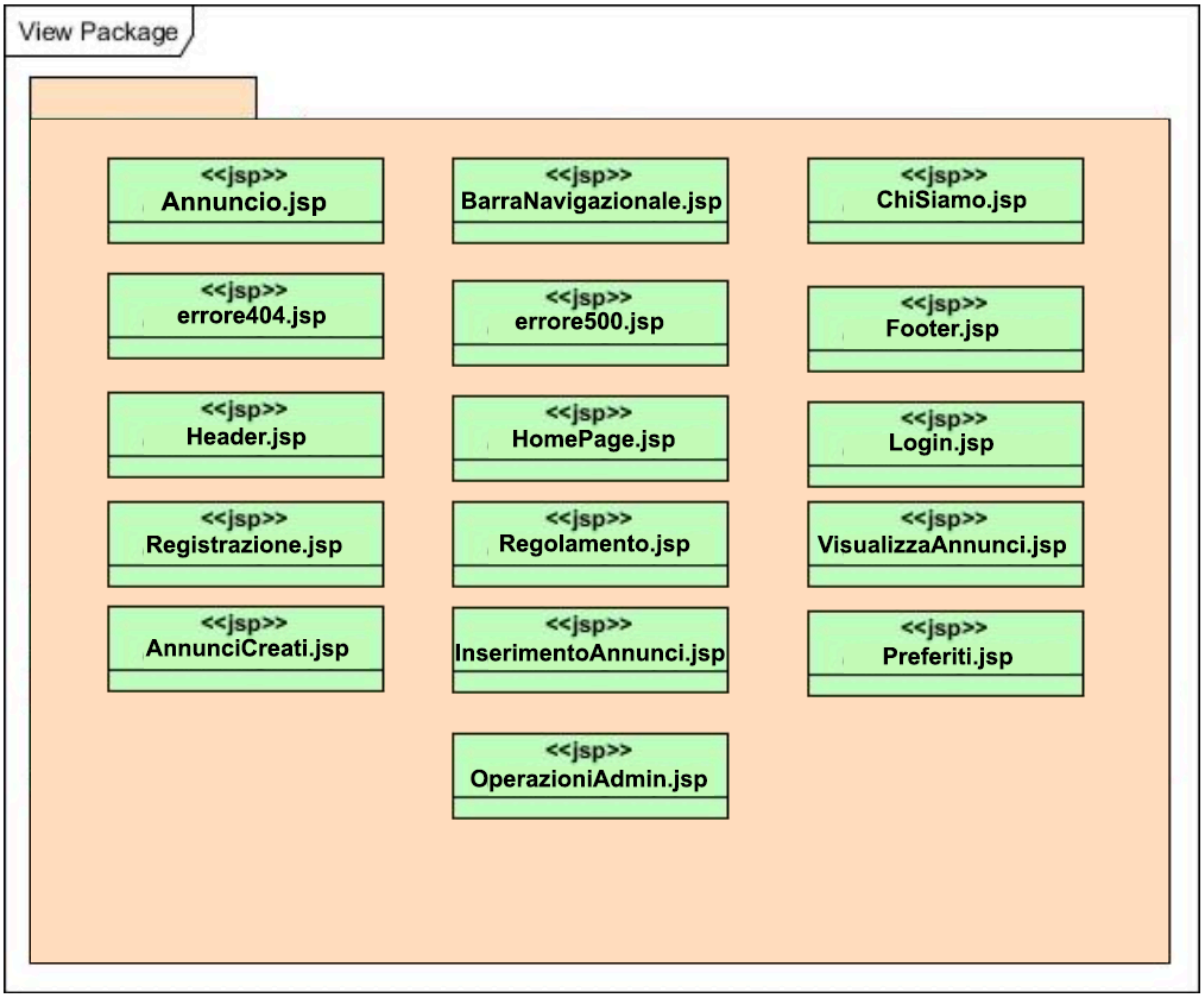
2.1.3 Packages Control



Classe:	**Descrizione:**
Login.java	Gestisce il login degli utenti.
Logout.java	Gestisce il logout degli utenti.
AggiungiPreferitiUtente.java	Gestisce l'aggiunta ai preferiti.
CaricamentoImmaginiServlet.java	Gestisce il caricamento di immagini.
EliminaAnnuncioServlet.java	Gestisce l'eliminazione di un annuncio.
PreferitiAjax.java	Gestisce i preferiti.
PrelevaAnnunciServlet.java	Gestisce la prelevazione di annunci per ricerca.
PrelevaAnnuncioSingoloServlet.java	Gestisce la prelevazione del singolo annuncio.
PrelevaImmagineServlet.java	Gestisce la prelevazione dell'immagine riferita all'annuncio corrispondente.
PrendiCategoriaServlet.java	Gestisce la prelevazione delle categorie.
PrendiPreferiti.java	Gestisce la prelevazione dei preferiti dell'utente.
PrelevaRegioniServlet.java	Gestisce la prelevazione delle regioni.

PrelevaUniversitaServlet.java	Gestisce la prelevazione delle università per la ricerca
Registrazione.java	Gestisce la registrazione degli utenti.
EliminaAnnuncioServlet.java	Gestisce l'eliminazione degli annunci
EliminaCategoriaServlet.java	Gestisce l'eliminazione delle categorie da parte del gestore della piattaforma
EliminaUniversitàServlet.java	Gestisce l'eliminazione delle università da parte del gestore della piattaforma
EliminaUtenteServlet.java	Gestisce l'eliminazione di utenti da parte del gestore della piattaforma
InserimentoCategoriaServlet.java	Gestisce l'aggiunta delle categorie da parte del gestore della piattaforma
InserimentoUniversitaServlet.java	Gestisce l'aggiunta delle università da parte del gestore della piattaforma
InserimentoAnnuncioServlet.java	Gestisce l'aggiunta di un annuncio
ModificaRuoloServlet.java	Gestisce la modifica dei ruoli degli utenti da parte del gestore della piattaforma

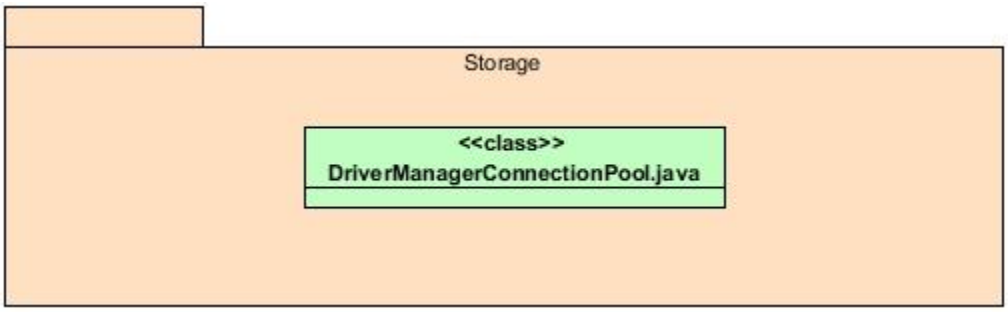
2.1.4 Packages View



Classe:	**Descrizione:**
Login.jsp	Visualizza la pagina che permette di effettuare il login.
HomePage.jsp	Visualizza la pagina iniziale del sistema.
Annuncio.jsp	Visualizza la pagina del singolo annuncio aperto nella ricerca effettuata.
BarraNavigazionale.jsp	Permette la visualizzazione della barra navigazionale con gli strumenti della ricerca.
ChiSiamo.jsp	Visualizza la pagina dove è descritta la piattaforma UniAds, la sua storia e il suo scopo.
Footer.jsp	Permette la visualizzazione dell'elemento footer nella pagina della piattaforma.
Header.jsp	Permette la visualizzazione dell'elemento footer nella pagina della piattaforma.
Regolamento.jsp	Visualizza la pagina dove è descritto il regolamento della piattaforma UniAds.
VisualizzaAnnunci.jsp	Visualizza la pagina dove vengono mostrati i risultati della ricerca.
Registrazione.jsp	Visualizza la pagina che permette di registrare una persona al sito.
AnnunciCreati.jsp	Visualizza la pagina dove sono presenti gli annunci creati dall'utente.
InserimentoAnnunci.jsp	Visualizza la pagina con il form per la creazione di un annuncio.
Preferiti.jsp	Visualizza la pagina dove sono presenti gli annunci preferiti dell'utente.
OperazioniAdmin.jsp	Visualizza la pagina dove sono presenti tutti le operazioni di gestione della piattaforma.

2.1.5 Storage Package

DriverManagerConnectionPool.java Si occupa di gestire il pool delle connessioni al database.



Classe:	**Descrizione:**
DriverManagerConnectionPool.java	Si occupa di gestire il pool delle connessioni al database

3.Interfaccia delle classi

Ogni metodo e ogni classe sarò opportunamente descritta nella documentazione “javadoc” allegata.

4.Design Pattern

Model-view-controller MVC, è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business. Il componente centrale del MVC, il modello, cattura il comportamento dell'applicazione in termini di dominio del problema, indipendentemente dall'interfaccia utente. Il modello gestisce direttamente i dati, la logica e le regole dell'applicazione. Una vista può essere una qualsiasi rappresentazione in output di informazioni, come un grafico o un diagramma. Sono possibili viste multiple delle stesse informazioni, come ad esempio un grafico a barre per la gestione e la vista tabellare per l'amministrazione. La terza parte, il controller, accetta l'input e lo converte in comandi per il modello e/o la vista.