Università degli Studi di Salerno
Corso di Ingegneria del Software

MotorSpace
Object Design Document (ODD)

MOTORSPACE

Partecipanti:

Nome	Matricola
Marco Basile	**0512104898**
Federico Citarella	**0512105900**
Raffaele Chiarolanza	**0512105060**
Emanuele D'Auria	**0512103680**

1.Introduzione

1.1 Object design trade-offs

Durante l'Object Design ci si è trovati di fronte ad alcuni bivi. Di seguito sono stati riportati i necessari trade-off:

● **Leggibilità vs tempo:** Durante la stesura del codice si rispetterà lo standard proposto da google per la programmazione tramite linguaggio java e saranno aggiunti commenti con eventuali chiarimenti,ciò permetterà una migliore leggibilità ed agevolerà la manutenzione ed eventuali modifiche al codice anche da parte di programmatori che non hanno lavorato sul progetto dall'inizio. Tuttavia ciò porterà ad un allungamento dei tempi di scrittura del codice stesso e quindi di implementazione e sviluppo del sistema.

● **Affidabilità vs Tempo di risposta:** Durante lo sviluppo del sistema si è preferito puntare maggiormente l'attenzione sull'affidabilità in modo tale da garantire un controllo accurato sui dati di input e minimizzare gli errori.

● **Usabilità vs Funzionalità:** Il sistema sarà facilmente fruibile dall'utente a discapito delle funzionalità offerte da quest' ultimo.

1.2 Componenti off-the-shelf

Per l'implementazione del sistema utilizzeremo componenti self ossia componenti software già disponibili e utilizzati per facilitare la creazione del software. Il framework che utilizzeremo per la realizzazione delle interfacce grafiche delle pagine web è Bootstrap. Per permettere all'interfaccia di rispondere alle azioni dell'utente e quindi velocizzare il sistema aggiungendo logica applicativa lato client si utilizzeranno JQuery.

1.3 Documentazione dell'interfacce

Nell'implementazione del sistema i programmatori dovranno attenersi alle linee guida definite di seguito.

1.3.1 Classi e interfacce Java

Nella scrittura di codice per le classi Java ci si atterrà allo standard di Google relativo alla programmazione in Java consultabile al seguente link: http://google.github.io/styleguide/javaguide.html#s4.6-whitespace

In questo standard ogni metodo ed ogni file possono non essere preceduti da un commento, inoltre, potranno esserci commenti in merito a particolari decisioni o calcoli. La convenzione utilizzata per i nomi delle variabili è la seguente: camelCase. Essa consiste nello scrivere parole composte in modo che ogni parola al centro della frase inizia con una lettera maiuscola. Quando si codificano classi e interfacce Java, si dovrebbero rispettare le seguenti regole di formattazione:

1. Non si devono inserire spazi tra il nome del metodo e la parentesi tonda "(" che apre la lista dei parametri.

2. La parentesi graffa aperta "{" si deve trovare sulla stessa linea dell'istruzione di dichiarazione.

3. La parentesi graffa chiusa "}" si troverà su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell'interfaccia. I nomi delle classi devono essere sostantivi con l'iniziale maiuscola. I nomi delle classi dovrebbero essere semplici, descrittivi e inerenti al dominio applicativo. Non devono essere usati underscore per legare nomi. I nomi dei metodi iniziano con una lettera minuscola, non sono consentiti caratteri speciali e dovranno essere semplici, descrittivi e inerenti al dominio applicativo. Sia i nomi delle classi che i nomi dei metodi dovranno seguire la notazione camelCase descritta sopra.

1.3.2 Pagine lato Server (JSP)

Le pagine JSP quando eseguite dovranno produrre un documento conforme allo standard HTML 5. Il codice Java presente nelle JSP deve aderire alle convenzioni descritte nel punto 1.3.1, con le seguenti specifiche:

1. Il tag di apertura (<%) è seguito immediatamente dalla fine della riga;

2. Il tag di chiusura (%>) si trova all'inizio di una riga;

3. È possibile evitare le due regole precedenti, se il corpo del codice Java consiste in una singola istruzione (<%= %>).

1.3.3 Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML 5 e il codice deve essere indentato, per facilitare la lettura, secondo le seguenti regole:

1. Un'indentazione consiste in una tabulazione;

2. Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;

3. Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;

4. I tag di commento devono seguire le stesse regole che si applicano ai tag normali.

1.3.4 Script Javascript

Gli Script in Javascript devono rispettare le seguenti convenzioni:

1. Gli script che svolgono funzioni distinte dal rendering della pagina dovrebbero essere collocati in file dedicati.

2. Il codice Javascript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.

3. Le funzioni Javascript devono essere documentate in modo analogo ai metodi Java.

1.3.5 Fogli di stile CSS

I fogli di stile (CSS) devono seguire la seguente convenzione:

Tutti gli stili non inline devono essere collocati in fogli di stile separati. Inoltre ogni regola CSS deve essere formattata come segue:

1. I selettori della regola si devono trovare nella stessa 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;

1.3.6 Database SQL

I nomi delle tabelle devono seguire le seguenti regole:

1. Ogni nome di tabella deve seguire la notazione CamelCase;

2. Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

I nomi dei campi delle tabelle devono seguire le seguenti regole:

1.1 Il nome deve essere tutto minuscolo.

1.2 Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

1.4 Design Pattern

1.4.1 MVC

Il design pattern MVC consente la suddivisione del sistema in tre blocchi principali: Model, View e Controller. Il Model modella i dati del dominio applicativo e fornisce i metodi di accesso ai dati persistenti, il View si occupa della presentazione dei dati all'utente e di ricevere da quest'ultimo gli input, infine il Controller riceve i comandi dell'utente attraverso il View e modifica lo stato di quest'ultimo e del Model.

1.4.2 DAO (Data Access Object) Pattern

Il DAO pattern è utilizzato per il mantenimento di una rigida separazione tra le componenti del Model e il Controller in un'applicazione basata sul paradigma MVC.

Il pattern sarà composta da:

•**Data Access Object Interfaccia**: interfaccia che definisce le operazioni che saranno performante sugli oggetti del model.

•**Data Access Object classe concreta:** classe che implementa l'interfaccia descritta sopra.

•**Model Object**: POJO contenenti i metodi get/set per salvare i dati ritirati attraverso le classi DAO.

1.5 Definizioni, Acronimi e abbreviazioni JSP:

JSP: acronimo di JavaServer Pages (talvolta detto Java Scripting Preprocessor), è una tecnologia di programmazione web in java per lo sviluppo della logica di presentazione (tipicamente secondo il pattern MVC) di applicazioni web.

MVC: acronimo di Model-view-controller, è un pattern architetturale molto diffuso nello sviluppo di sistemi software.

Off-The-Shelf: Servizi esterni al sistema di cui viene fatto utilizzo. Bootstrap: è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il web.

HTML: Linguaggio di markup utilizzato per lo sviluppo di pagine Web.

CSS: acronimo di Cascading Style Sheets è un linguaggio usato per definire la formattazione delle pagine Web.

JavaScript: JavaScript è un linguaggio di scripting utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi.

JQuery: JQuery è una libreria JavaScript per applicazioni web.

AJAX: AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive.

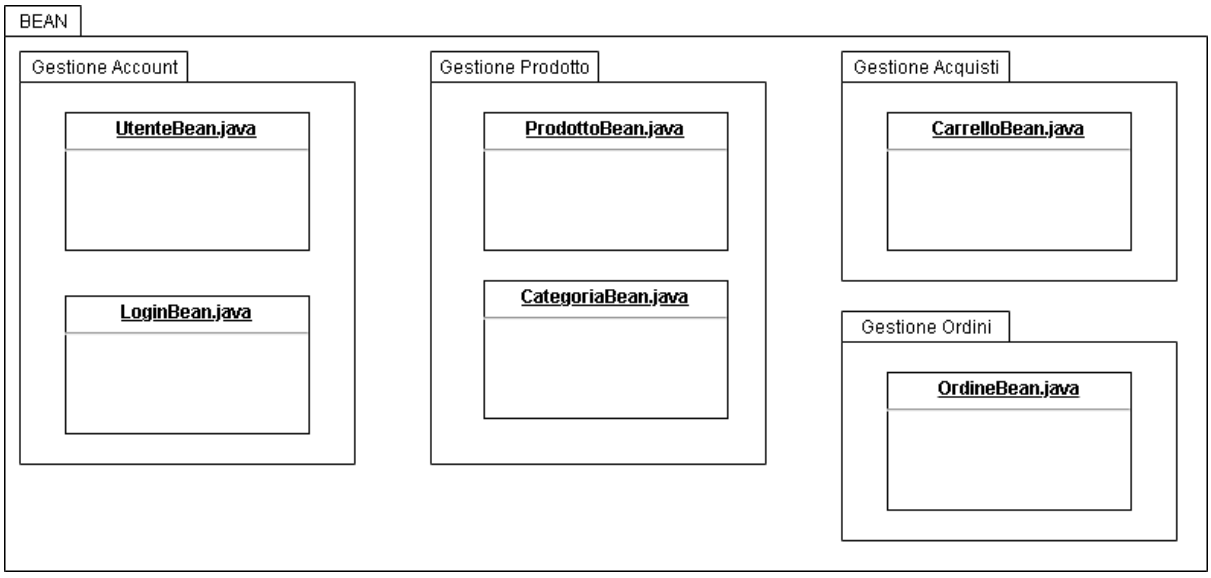
camelCase: È una tecnica di naming delle variabili adottata dallo standard Google Java. Essa consiste nello scrivere più parole insieme delimitando l'inizio di una nuova parola con una lettera maiuscola.

Servlet: i servlet sono oggetti scritti in linguaggio Java che operano all'interno di un server web.

Tomcat: Apache Tomcat è un web server open source. Implementa le specifiche JavaServer Pages (JSP) e servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java

2. Packages

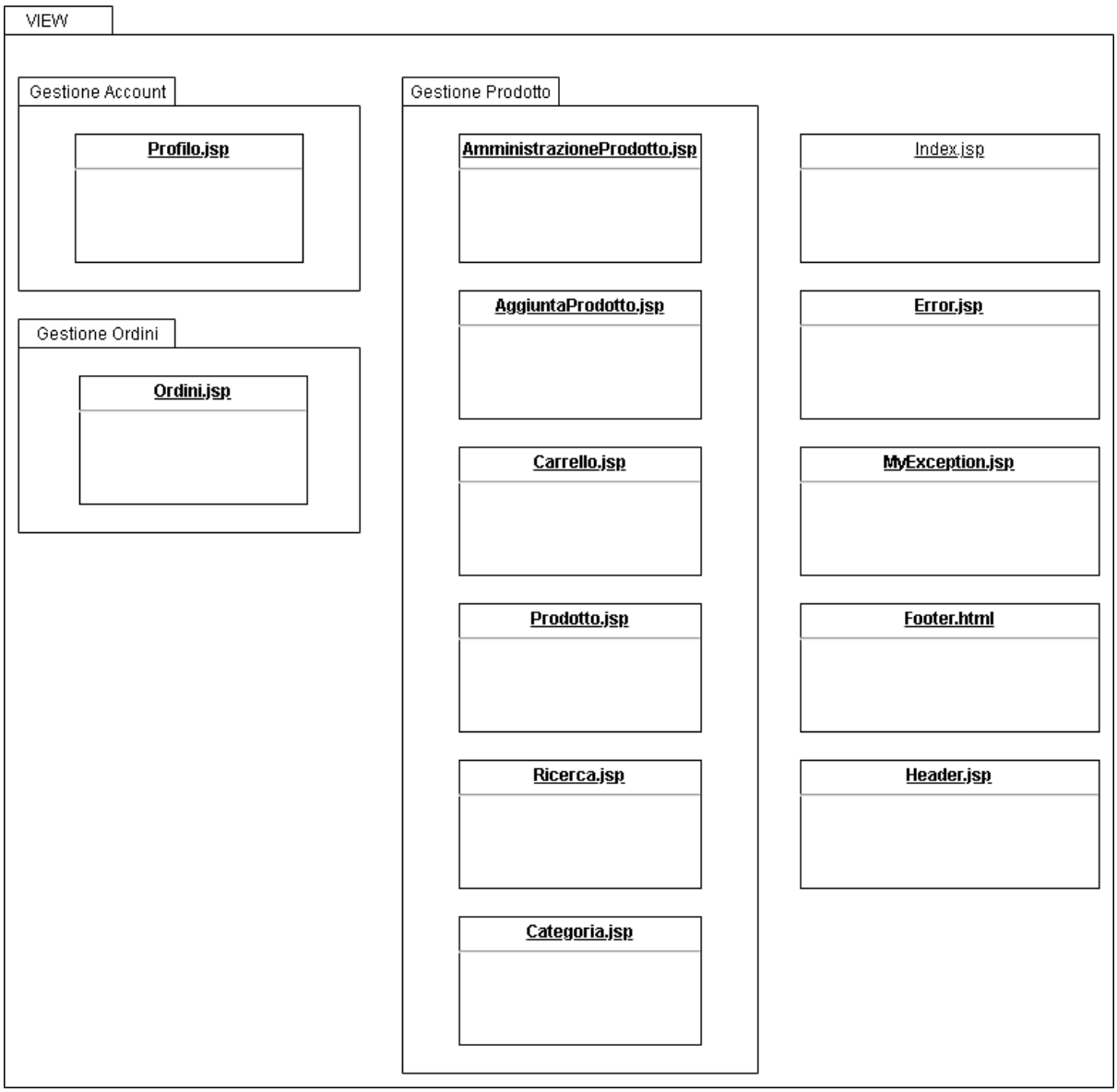
2.1 BeanPackage



Classe	Descrizione
UtenteBean.java	Classe che descrive un utente registrato
LoginBean.java	
ProdottoBean.java	Classe che descrive un prodotto

CategoriaBean.java	Classe che descrive una categoria
CarrelloBean.java	Classe che descrive il carrello
OrdineBean.java	Classe che descrive un ordine

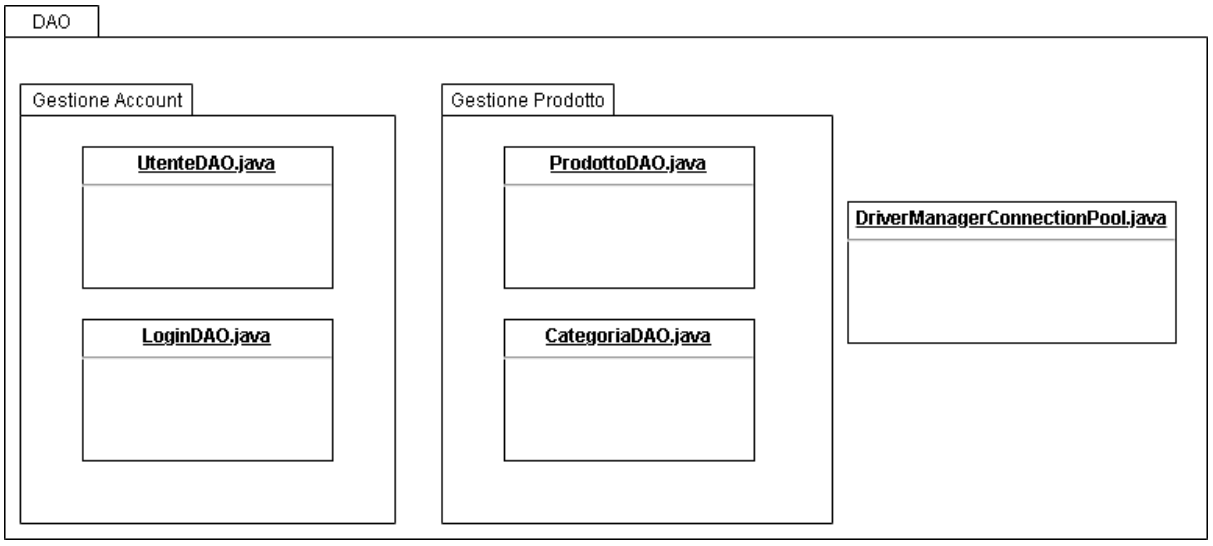
2.2 View



Classe	Descrizione
Profilo.jsp	Pagina che visualizza il profilo dell'utente
Ordini.jsp	Pagina che visualizza la lista degli ordini

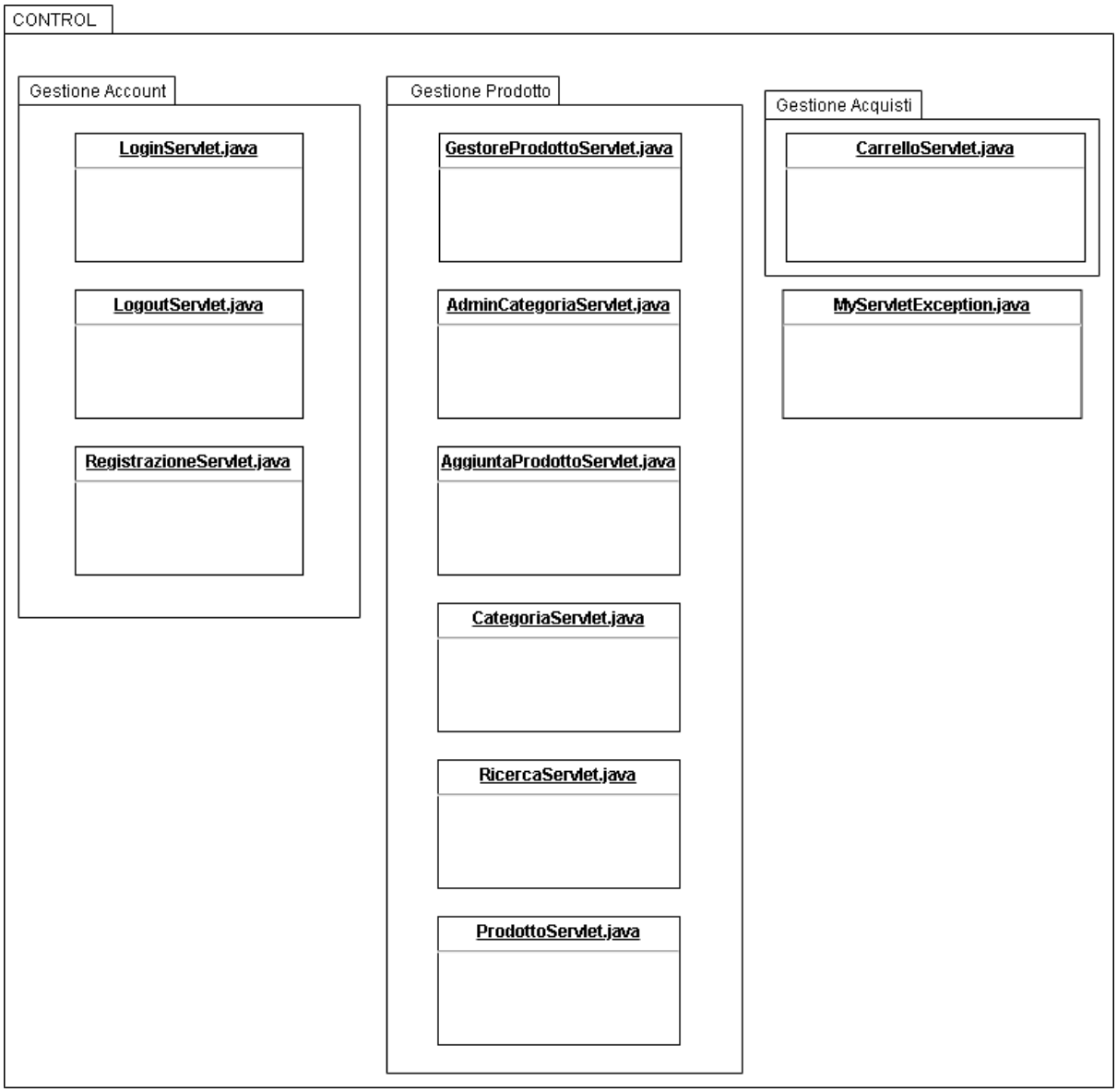
AmministrazioneProdotto.jsp	Pagina per modificare od oscurare un prodotto
AggiuntaProdotto.jsp	Pagina utile all'aggiunta prodotto
Carrello.jsp	Pagina contenente i prodotti del carrello
Prodotto.jsp	Pagina per visualizzare il prodotto
Ricerca.jsp	Pagina utile alla ricerca di un prodotto
Categoria.jsp	Pagina ()
Index.jsp	Pagina della homepage
Error.jsp	Pagina di errore
MyException.jsp	Pagina di eccezione
Header.jsp	Pagina contenente l'header
Footer.html	Pagina contenente il footer

2.3 Model



Classe	Descrizione
UtenteDAO.java	Classe contenente metodi utili a manipolare le informazioni del database relative all'entità Utente
LoginDAO.java	
ProdottoDAO.java	Classe contenente metodi utili a manipolare le informazioni del database relative all'entità Prodotto
CategoriaDAO.java	Classe contenente metodi utili a manipolare le informazioni del database relative all'entità Categoria
DriverManagerConnectionPool.java	Permette la connessione col database

2.4 Controller



Classe	Descrizione
LoginServlet.java	Classe che effettua il login di un utente nel sistema
LogoutServlet.java	Classe che effettua il logout di un utente dal sistema
RegistrazioneServlet.java	Classe che permette la registrazione di un utente nel sistema

GestoreProdottoServlet.java	Classe che permette la gestione del prodotto
AdminCategoriaServlet.java	Classe che permette di gestire le funzionalità di admin
AggiuntaProdottoServlet.java	Classe che permette l'aggiunta di un prodotto
RicercaServlet.java	Classe che permette la ricerca all'interno del catalogo
CategoriaServlet.java	Classe che permette di costituire il catalogo del sistema
ProdottoServlet.java	Classe che gestisce i prodotti
CarrelloServlet.java	Classe che permette la gestione del carrello utente
MyServletException.java	Classe generica per gestire le eccezioni

3 Class Interfaces

3.1 UtenteDAO

<table>
<tr><th colspan="3">UtenteDAO.java</th></tr>
<tr><th>Metodo</th><th>Contesto</th><th>Condizioni</th></tr>
<tr><td>doRetrieveById</td><td>UtenteDAO: doRetrieveByUsername (username)</td><td><table><tr><td>Pre</td><td>utente!=null
&&
self.utente-
>exists(u|u.username=username)</td></tr><tr><td>Post</td><td>result(Utente(u))</td></tr></table></td></tr>
<tr><td>doDelete</td><td>UtenteDAO: doDelete(username)</td><td><table><tr><td>Pre</td><td>utente!=null
&&</td></tr></table></td></tr>
</table>

<table>
<tr><td></td><td></td><td><table>
<tr><td></td><td>self.utente->exists(u|u.username=username)</td></tr>
<tr><td>Post</td><td>utente!=null
&&
!self.utente->exists(u|u.username.equals(username))</td></tr>
</table></td></tr>
<tr><td>doUpdate</td><td>UtenteDAO: doUpdate(Utente)</td><td><table>
<tr><td>Pre</td><td>Nome!=null

&&cognome!=null

&&username!=null

&&email!=null

&&dataDiNascita!=null

&&password!=null

&&ripetiPassword!=null

&&email.contains(“.”)

&&email.contains(“@”)

&&nome.length=(min3,max16)

&&cognome.length=(min3,max16)

&&password contiene un numero

&&password contiene un carattere minuscolo

&&password contiene un carattere maiuscolo

&&password contiene un carattere speciale

&&ripetiPassword=password

&&utente→forAll
(g |g.username
!=utente.username)

&&utente→forAll
(g |g.email !=utente.email)</td></tr>
</table></td></tr>
</table>

		Post: self.utenti->exists(g |g.username != utente.username) && self.utenti->exists(g |g.email != utente.email) && u.nome.equals(nome) && u.cognome.equals(cognome) && u.email.equals(email) && u.password.equals(password)
doSave	**UtenteDAO: doSave(Utente)**	Pre: Nome!=null &&cognome!=null &&username!=null &&email!=null &&dataDiNascita!=null &&password!=null &&ripetiPassword!=null &&email.contains(“.”) &&email.contains(“@”) &&nome.length=(min3,max16) &&cognome.length=(min3,max16) &&password contiene un numero &&password contiene un carattere minuscolo &&password contiene un carattere maiuscolo &&password contiene un carattere speciale &&ripetiPassword=password &&utente→forAll (g |g.username

		!=utente.username) &&utente→forAll (g |g.email !=utente.email) Post utente→ exists(utente)

3.2 CategoriaDAO

CategoriaDAO.java		
Metodo	**Contesto**	**Condizioni**
doRetrieveById	**CategoriaDAO: doRetrieveById(idCategoria)**	Pre: categoria!=null && self.categoria->exists(c|c.idCategoria=idCategoria) Post: result(Categoria(c))
doDelete	**CategoriaDAO: doDelete(idCategoria)**	Pre: categoria!=null && self.categoria->exists(c|c.idCategoria=idCategoria) Post: categoria!=null && !self.categoria->exists(c|c.idCategoria.equals(idCategoria))
doUpdate	**CategoriaDAO: doUpdate(Categoria)**	Pre: Nome!=null && Descrizione!=null Post: self.categoria->exists(g |g.idCategoria!= categoria.idCategoria) &&

		u.nome.equals(nome) && u.descrizione.equals (descrizione)
doSave	**CategoriaDAO: doSave(Categoria)**	Pre: Nome!=null && Descrizione!=null && idCategoria!=null Post: categoria→ exists(categoria)

3.3 ProdottoDAO

ProdottoDAO.java		
Metodo	**Contesto**	**Condizioni**
doRetrieveById	**ProdottoDAO: doRetrieveByCodice(codice Prodotto)**	Pre: prodotto!=null && self.prodotto->exists(c|c.codiceProdotto=idProdotto) Post: prodotto!=null && !self.prodotto->exists (c|c.codiceProdotto.equals (codiceProdotto))
doRetrieveBy Nome	**ProdottoDAO: doRetrieveBy**	

<table>
<tr><td>Or Descrizione</td><td>NomeOrDescrizione (str, offset, limit)</td><td><table>
<tr><td>Pre</td><td>prodotto!=null
&&
str!=null
&&
offset>=0
&&
limit>0
self.prodotto-
>exists(c|c.str=str)</td></tr>
<tr><td>Post</td><td>result(Prodotto(c))</td></tr>
</table></td></tr>
<tr><td>doRetrieveBy Categoria</td><td>ProdottoDAO: doRetrieveByCategoria (categoria)</td><td><table>
<tr><td>Pre</td><td>prodotto!=null
&&
categoria!=null
&&
self.prodotto-
>exists(c|c.categoria=categoria)</td></tr>
<tr><td>Post</td><td>result(Prodotto(c))</td></tr>
</table></td></tr>
<tr><td>doDelete</td><td>ProdottoDAO: doDelete(codiceProdotto)</td><td><table>
<tr><td>Pre</td><td>prodotto!=null
&&
self.prodotto-
>exists(c|c.codiceProdotto=codiceProdotto)</td></tr>
<tr><td>Post</td><td>prodotto!=null
&&
!self.prodotto-
>exists(c|c.codiceProdotto.equals(codiceProdotto))</td></tr>
</table></td></tr>
<tr><td>doUpdate</td><td>ProdottoDAO: doUpdate(Prodotto)</td><td><table>
<tr><td>Pre</td><td>nomeProdotto!=null
&&
categoria!=null
&&
descrizione!=null
&&
immagine!=null
&&
marca!=null
&&
prezzo>0.0</td></tr>
<tr><td>Post</td><td>self.prodotto->exists(g
|g.codiceprodotto !=</td></tr>
</table></td></tr>
</table>

		prodotto.codiceProdotto) && u.nomeProdotto.equals(nomeProdotto) && u.categoria.equals(categoria) && u.descrizione.equals(descrizione) && u.immagine.equals(immagine) u.marca.equals(marca) u.prezzo.equals(prezzo)
doSave	**ProdottoDAO: doSave(Prodotto)**	Pre: nomeProdotto!=null && categoria!=null && descrizione!=null && immagine!=null && codice!=null && marca!=null && prezzo>0.0 && quantità>0 Post: prodotto→ exists(prodotto)