Università degli Studi di Salerno

Corso di Ingegneria del Software

Object Design Document

ZyphyK Sport

Docente:

Prof. Andrea De Lucia

Tutor:

Gilberto Recupito

Partecipanti:

Consiglio Luigi

D'Antuono Francesco Paolo

Vitale Ciro

Partecipanti al progetto:

nome	matricola	e-mail
Consiglio Luigi (CL)	0512110485	l.consiglio1@studenti.unisa.it
D'Antuono Francesco Paolo (DFP)	0512109798	f.dantuono10@studenti.unisa.it
Vitale Ciro (VC)	0512110719	c.vitale55@studenti.unisa.it

Revision History

data	versione	descrizione	autore
19/12/2022	1.00	inizio stesura introduction	CL
21/12/2022	1.01	stesura object design trade-offs	DFP,VC
21/12/2022	1.02	revisione object design trade-offs	Tutto il team
22/12/2022	1.03	stesura interface documentation: naming convention, variabili	VC
26/12/2022	1.04	stesura interface documentation: metodi, classi e pagine	VC
26/12/2022	1.05	revisione e modifica interface documentation	DFP,CL
26/12/2022	1.06	stesura design pattern	CL,VC
27/12/2022	1.07	revisione interface documentation: metodi	DFP
27/12/2022	1.08	revisione design pattern	Tutto il team
28/12/2022	1.09	stesura packages	DFP
28/12/2022	1.10	revisione packages	VC
28/12/2022	1.11	inizio stesura class interfaces: GestioneUtente	DFP,CL
29/12/2022	1.12	class interfaces: GestioneVendite	VC,DFP
29/12/2022	1.13	class interfaces: GestioneCatalogo	DFP

02/01/2023	1.14	class interfaces: Utils	CL
04/01/2023	1.15	revisione class interfaces	VC, CL
07/01/2023	2.00	revisione del documento	Tutto il team

ODD

Introduction

L'Object Design Document (ODD) è un documento che viene utilizzato durante il processo di sviluppo di un software per definire il modello di progettazione che verrà utilizzato per implementare il sistema. Si basa sui documenti di RAD e SDD, che contengono le informazioni raccolte durante la fase di analisi dei requisiti e di progettazione del sistema. L'ODD descrive in dettaglio le interfacce delle classi, le operazioni supportate, i tipi di dati, i parametri delle procedure, i signature dei sottosistemi, i trade-off e le linee guida per evitare compromessi di progettazione durante lo sviluppo del software. In altre parole, l'ODD è una sorta di "piano di costruzione" per il software, che viene utilizzato come guida durante il processo di sviluppo.

Object design trade-offs

Performance vs Supportability

Migliorare la performance del software potrebbe richiedere l'eliminazione di alcune funzionalità di supporto, come la generazione di report o la gestione degli errori. Al contempo, mantenere un elevato livello di supportability potrebbe richiedere una leggera riduzione delle prestazioni.

La scelta è di garantire maggiore performance a discapito di alcune funzionalità di supporto.

Dependability vs Usability

Un software altamente affidabile potrebbe avere un'interfaccia utente meno intuitiva o meno attraente, mentre un software facile da usare potrebbe avere un livello di affidabilità leggermente inferiore.

La scelta è di garantire maggiore affidabilità a discapito dell’usabilità.

Interface documentation guidelines

Gli sviluppatori dovranno seguire determinate linee guida per la stesura del codice:

Naming Convention

È buona norma utilizzare nomi:

- descrittivi;
- pronunciabili;
- di uso comune;
- di lunghezza medio-corta;
- utilizzando solo caratteri consentiti (a-z, A-Z, 0-9).

Variabili

- i nomi delle variabili devono iniziare con la lettera minuscola e rispettare la notazione a cammello. La dichiarazione delle variabili deve essere effettuata ad inizio blocco; in ogni riga vi deve essere una sola dichiarazione di variabile e va effettuato l'allineamento per migliorare la leggibilità;
- in determinati casi, è possibile utilizzare il carattere underscore "_", ad esempio quando si fa uso di variabili costanti oppure quando si fa uso di proprietà statiche.

Metodi

- i nomi dei metodi devono seguire una convenzione di scrittura: iniziare con la lettera minuscola e rispettare la notazione a cammello. Di solito il nome del metodo è costituito da un verbo che identifica un'azione, seguito dal nome di un oggetto. Inoltre, i metodi che servono per accedere o modificare le variabili di una classe dovrebbero essere denominati utilizzando il prefisso "get" o "set", seguito dal nome della variabile in questione. Se viene dichiarata una variabile all'interno di un metodo quest'ultima deve essere dichiarata appena prima del suo utilizzo e deve servire per un solo scopo, per facilitare la leggibilità.
  Esempio: getId(), setId();
- è importante che ogni metodo sia preceduto da una descrizione che ne spieghi lo scopo, i parametri che accetta e il valore di ritorno. Inoltre, i metodi dovrebbero essere organizzati in base alla loro funzionalità all'interno della classe.

Classi e pagine

- i nomi delle classi e delle pagine devono iniziare con la lettera maiuscola e devono rispettare la notazione a cammello. I nomi delle classi devono rappresentare lo scopo di quest'ultime.
  Ogni file sorgente *.java* contiene una singola classe e dev'essere strutturata nel seguente modo;
- la dichiarazione di una classe è 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;
  6. commento e dichiarazione metodi e variabili.

Definitions, acronyms, and abbreviations

Di seguito l'elenco delle definizioni che, per semplicità, sostituiremo con acronimi:

Acronimo	Definizione
RAD	Requirements Analysis Document
SDD	System Design Document
DAO	Data Access Object
DB	Database
OCL	Object Constraint Language

References

Di seguito una lista di riferimenti agli altri documenti del progetto a cui si fa riferimento:

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

Design Pattern

DAO

Il Design Pattern DAO (Data Access Object) è un modello di progettazione utilizzato per separare l'accesso ai dati da un'applicazione o da un sistema. Consiste in un oggetto che si occupa di gestire l'accesso ai dati presenti in un database, in un file o in qualsiasi altra fonte di dati.

L'obiettivo principale del pattern DAO è quello di separare l'accesso ai dati dall'applicazione stessa, in modo da poter modificare o sostituire facilmente la fonte di dati senza dover modificare il codice dell'applicazione. Ciò rende più facile il mantenimento del codice e consente di fare più facilmente il deploy dell'applicazione in ambienti diversi.

Il pattern DAO prevede la creazione di una classe DAO per ogni tipo di oggetto che si desidera gestire. Ogni classe DAO si occupa di gestire l'accesso ai dati per quel tipo di oggetto e fornisce metodi per eseguire operazioni come il recupero, l'inserimento, l'aggiornamento e la cancellazione dei dati.

Packages

La suddivisione dei packages è stata fatta rispettando la divisione in sottosistemi definita nell'SDD e inoltre sono stati divisi ulteriormente per il tipo di classe che rappresentano

GestioneUtente Packages

Nel package *GestioneUtente* sono presenti le classi che permettono di gestire l'autenticazione dei clienti e dei gestori e la registrazione dei clienti. Inoltre, si occupa anche delle funzionalità di logout.

Nel package sono presenti le seguenti classi:

- Bean:
  - ClientiBean
  - GestoriCatalogoBean
  - GestoriOrdiniBean
- DAO:
  - ClientiDAO
  - GestoriCatalogoDAO
  - GestoriOrdiniDAO
- Interface:
  - ClientiInterf
  - GestoriCatalogoInterf
  - GestoriOrdiniInterf
- Servlet:
  - LoginServlet
  - LogoutServlet
  - SignUpServlet

GestioneVendite Packages

Nel package *GestioneVendite* sono presenti le classi che permettono la gestione degli ordini da parte del gestore degli ordini: modifica dello stato degli ordini; invece, per quanto riguarda il cliente, gestione del carrello: inserimento, modifica quantità e rimozione dei prodotti da esso, si occupa del processo di checkout e della creazione dell'ordine.

Nel package sono presenti le seguenti classi:

- Bean:
  - CartsBean
  - CartsContainsProdsBean
  - OrdersBean
  - OrdersContainsProdsBean

- DAO:
    - CartsDAO
    - CartsContainsProdsDAO
    - OrdersDAO
    - OrdersContainsProdsDAO
- Interface:
    - CartsInterf
    - CartsContainsProdsInterf
    - OrdersInterf
    - OrdersContainsProdsInterf
- Servlet:
    - AddToCartServlet
    - CheckOutServlet
    - GetProdsFromCartServlet
    - ModQuantityInCartServlet
    - OrderManageServlet
    - RemFromCartServlet

GestioneCatalogo Packages

Nel package GestioneCatalogo sono presenti le classi per la gestione del catalogo, da parte del gestore del catalogo, quindi gestione dei prodotti: inserimento, modifica ed eliminazione; invece, per quanto riguarda il cliente e l'utente non loggato, questo sottosistema gestisce la visualizzazione del catalogo e dei prodotti, con la possibilità di ricerca per filtri oppure mediante testo

Nel package sono presenti le seguenti classi:

- Bean:
    - ManagesProdsBean
    - ProductsBean
    - SizesBean
- DAO:
    - ManagesProdsDAO
    - ProductsDAO
    - SizesDAO
- Interface:
    - ManagesProdsInterf
    - ProductsInterf
    - SizesInterf
- Servlet:
    - AddInCatalogServlet
    - ModInCatalogServlet
    - RemFromCatalogServlet

- ImageServlet

Utils Package

Questo package contiene tutte le altre classi di servizio che ci servono per la connessione al DB e altri set-up non inerenti ad oggetti di sistema.

Nel package sono presenti le seguenti classi:

- MainContext
- PopolamentoDBServlet

Class interfaces

Di seguito riportiamo le interfacce delle classi implementate con le pre e post-condizioni definite in OCL.

GestioneUtente

ClientiDAO

Classe			
Nome		Descrizione	
ClientiDAO		Classe che implementa l'interfaccia ClientiInterf per la gestione dei dati dei Clienti	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::ClientiDAO public synchronized void doSave(String username, int cartId, String name, String surname, String email, String pass_word, LocalDate birthDate)	permette il salvataggio di un nuovo Cliente nel database	pre: username != null and cartId > 0 and name != null and surname != null and email != null and pass_word != null and birthDate != null	-
Context::ClientiDAO public synchronized void doUpdate(String username, String name, String surname, String email, String pass_word, LocalDate birthDate)	permette l'aggiornamento dei dati di un Cliente già esistente nel database	pre: username != null and name != null and surname != null and email != null and pass_word != null and birthDate != null	-

Context::ClientiDAO public synchronized void doDelete(String username)	permette l'eliminazione di un Cliente dal database con username fornito	pre: username != null	-
Context::ClientiDAO public synchronized ClientiBean doRetrieveByKey(String username)	permette la restituzione dei dati di un Cliente presente nel database con username fornito	pre: username != null	-
Context:: ClientiDAO public synchronized Set<ClientiBean>doRetrieveAll(String order)	permette la restituzione dei dati di tutti i Clienti presenti nel database	-	-

GestoriCatologoDAO

Classe			
Nome		Descrizione	
GestoriCatalogoDAO		Classe che implementa l'interfaccia GestoriCatalogoInterf per la gestione dei dati dei Gestori Catalogo	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::GestoriCatalogoDAO public synchronized void doSave(String username, String name, String surname, String email, String pass_word, int ral)	permette salvataggio di un nuovo Gestore Catalogo nel database	pre: username != null and name != null and surname != null and email != null and pass_word != null and ral >= 0	-

Context::GestoriCatalogoDAO public synchronized void doUpdate(String username, String name, String surname, String email, String pass_word, int ral)	permette l'aggiornamento dei dati di un Gestore Catalogo già esistente nel database	pre: username != null and name != null and surname != null and email != null and pass_word != null and ral >= 0	-
Context::GestoriCatalogoDAO public synchronized void doDelete(String username)	permette l'eliminazione di un Gestore Catalogo dal database con username fornito	pre: username != null	-
Context::GestoriCatalogoDAO public synchronized GestoriCatalogoBean doRetrieveByKey(String username)	permette la restituzione dei dati di un Gestore Catalogo presente nel database con username fornito	pre: username != null	-
Context::GestoriCatalogoDAO public synchronized Set<GestoriCatalogoBean> doRetrieveAll(String order)	permette la restituzione dei dati di tutti i Gestori Catalogo presenti nel database	-	-

GestoriOrdiniDAO

Classe	
Nome	Descrizione
GestoriOrdiniDAO	Classe che implementa l'interfaccia GestoriOrdiniInterf per la gestione dei dati dei Gestori Ordini

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::GestoriOrdiniDAO public synchronized void doSave(String username, String name, String surname, String email, String pass_word, int ral)	permette il salvataggio di un nuovo Gestore Ordini nel database	pre: username != null and name != null and surname != null and email != null and pass_word != null and ral >= 0	-
Context::GestoriOrdiniDAO public synchronized void doUpdate(String username, String name, String surname, String email, String pass_word, int ral)	permette l'aggiornamento dei dati di un Gestore Ordini già esistente nel database	pre: username != null and name != null and surname != null and email != null and pass_word != null and ral >= 0	-
Context::GestoriOrdiniDAO public synchronized void doDelete(String username)	permette l'eliminazione di un Gestore Ordini dal database con username fornito	pre: username != null	-
Context::GestoriOrdiniDAO public synchronized GestoriOrdiniBean doRetrieveByKey(String username)	permette la restituzione dei dati di un Gestore Ordini presente nel database con username fornito	pre: username != null	-
Context::GestoriOrdiniDAO public	permette la restituzione dei dati di	-	-

synchronized Set<GestoriOrdiniBean> doRetrieveAll(String order)	tutti i Gestori Ordini presenti nel database		

LoginServlet

Classe	
Nome	Descrizione
LoginServlet	Classe che permette l'accesso di un utente già registrato

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::LoginServlet public void doPost(HttpServletRequest request, HttpServletResponse response)	permette l'autenticazione di un utente già registrato mediante username e password passati nella request	pre: request != null and response != null and request.getParameter("username") != null and request.getParameter("password") != null and getServletContext().getAttribute("DataSource") != null	-
Context::LoginServlet private String checkLogin(String username, String password)	permette il controllo dei parametri username e password confrontandoli con il contenuto del database	pre: username != null and password != null	-

LogoutServlet

Classe	
Nome	Descrizione
LogoutServlet	Classe che permette il logout dell'utente attualmente loggato

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::LogoutServlet	permette il logout	request.getSession().getAttr	request.getSession().getAttr

public void doGet(HttpServletRequest request, HttpServletResponse response)	dell'utente attualmente loggato e, quindi, l'invalidazione della sessione attuale	ibute("roles") != null	ibute("roles") == null

SignUpServlet

Classe			
Nome		Descrizione	
SignUpServlet		Classe che permette la registrazione di un nuovo Cliente	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::SignUpServlet public void doPost(HttpServletRequest request, HttpServletResponse response)	permette la registrazione di un nuovo Cliente e, di conseguenza, l'autenticazione	pre: request != null and response != null and request.getParameter("nome") != null and request.getParameter("cognome") != null and request.getParameter("username") != null and request.getParameter("data") != null and request.getParameter("email") != null and request.getParameter("password") != null and request.getParameter("conferma") != null and getServletContext().getAttri	request.getSession().getAttribute("roles") == "cliente"

		bute("DataSource") != null and request.getSession().getAttribute("roles") == null	

GestioneVendite

CartsDAO

Classe	
Nome	Descrizione
CartsDAO	Classe che implementa l'interfaccia CartsInterf per la gestione dei dati del carrello

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::CartsDAO public synchronized void doSave(int amount)	permette il salvataggio di un nuovo carrello nel database	pre: amount!=null and amount > 0	-
Context::CartsDAO public synchronized void doUpdate(int id, int amount)	permette l'aggiornamento dei dati di un carrello già esistente nel database	pre: id!=null and id > 0 amount!=null and amount > 0	-
Context::CartsDAO public synchronized void doDelete(int id)	permette l'eliminazione di un carrello dal database con username fornito	pre: id!=null and id > 0	-
Context::CartsDAO public synchronized ClientiBean doRetrieveByKey(int id)	permette la restituzione dei dati di un carrello presente nel database con id fornito	pre: id!=null and id > 0	-
Context:: CartsDAO public synchronized Set<ClientiBean>doRetrieveAl	permette la restituzione dei dati di tutti i carrelli presenti nel database	-	-

l(String order)			

CartsContainsProdsDAO

Classe	
Nome	Descrizione
CartsContainsProdsDAO	Classe che implementa l'interfaccia CartsContainsProdsInterf per la gestione dei dati dei prodotti contenuti nel carrello corrispondente ad ogni Cliente

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::CartsContainsProdsDAO public synchronized void doSave(int cartId, String productId, int quantity, int size)	permette il salvataggio della relazione tra carrello e i prodotti contenuti al suo interno nel database	pre: cartId != null and cartId > 0 and productId != null and quantity != null and quantity > 0 and size != null	-
Context::CartsContainsProdsDAO public synchronized void doUpdate(int cartId, String productId, int quantity)	permette l'aggiornamento dei dati dei prodotti che sono presenti all'interno di un carrello già esistente nel database	pre: cartId != null and cartId > 0 and productId != null and quantity != null and quantity > 0	-
Context::CartsContainsProdsDAO public synchronized void doDelete(int cartId, String productId, int size)	permette l'eliminazione di un prodotto all' interno di un carrello dal database con cartId, productId e size forniti	pre: cartId != null and cartId > 0 and productId != null and quantity != null and size != null	-
Context::CartsContainsProdsDAO public	permette la restituzione dei dati dei	pre: cartId != null and cartId > 0	-

synchronized CartsContainsProdsBean doRetrieveByKey(int cartId, String productId, int size)	prodotti che si trovano in un carrello presente nel database con cartId, productId e size forniti	productId != null and quantity != null and size != null	
Context:: CartsContainsProdsDAO public synchronized Set<ClientiBean>doRetrieveAll(String order)	permette la restituzione dei dati di tutti i prodotti presenti nei carrelli che si trovano nel database	-	-
Context:: CartsContainsProdsDAO public synchronized Set<CartsContainsProdsBean> doRetrieveAllByCartId(int cartId, String order)	permette la restituzione dei dati di tutti i prodotti presenti in un singolo carrello	pre: cartId != null cartId > 0	-

OrdersDAO

Classe			
Nome		Descrizione	
OrdersDAO		Classe che implementa l'interfaccia OrdersInterf per la gestione dei dati degli ordini che sono stati creati ad ogni acquisto effettuato da un Cliente	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::OrdersDAO public synchronized int doSave(String clienteUsernam, String gestOrdUsernam	permette il salvataggio dell'ordine nel database	pre: clienteUsername!=null and dateTime != null and shippingAddress != null and paymentMethod	-

e,LocalDateTime dateTime, String shippingAddress, String paymentMethod, int amount, boolean sent)		!= null and amount != null and amount > 0 and sent != null and sent == false	
Context::OrdersDAO public synchronized void doUpdateSent(int id, String gestOrdUsername)	permette l'aggiornamento dei dati dell'ordine già esistente nel database	pre: id != null and id > 0 and gestOrdUsername!= null and this.doRetrieveByKey(id).getSent() == false	post: this.doRetrieveByKey(id).getSent() == true
Context::OrdersDAO public synchronized void doDelete(int id)	permette l'eliminazione di un ordine dal database con id fornito	pre: id!= null and id > 0	-
Context::OrdersDAO public synchronized OrdersBean doRetrieveByKey(int id)	permette la restituzione dell' ordine presente nel database con id fornito	pre: id!= null and id > 0	-
Context:: OrdersDAO public synchronized Set<OrdersBean> doRetrieveAll(String order)	permette la restituzione dei dati di tutti gli ordini che si trovano nel database	-	-
Context:: OrdersDAO public synchronized Set<OrdersBean> doRetrieveAllCliente(String username, String order)	permette la restituzione dei dati di tutti gli ordini che riguardano un solo cliente	pre: username !=	-

OrdersContainsProdsDAO

Classe	
Nome	Descrizione
OrdersContainsProdsDAO	Classe che implementa l'interfaccia OrdersContainsProdsInterf per la gestione dei dati dei prodotti contenuti negli ordini effettuati dai Clienti

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::OrdersContainsProdsDAO public synchronized void doSave(int orderId, String productId, int quantity, int size)	permette il salvataggio della relazione tra ordine e i prodotti contenuti al suo interno nel database	pre: orderId != null and orderId > 0 and productId != null and quantity != null and quantity > 0 and size != null	-
Context::OrdersContainsProdsDAO public synchronized void doDelete(int orderId, String productId)	permette l'eliminazione di un prodotto all' interno di un ordine dal database con orderId e productId forniti	pre: orderId != null and orderId > 0 and productId != null	-
Context::OrdersContainsProdsDAO public synchronized Set<OrdersContainsProdsBean> doRetrieveAll(String order)	permette la restituzione dei dati di tutti i prodotti presenti negli ordini che si trovano nel database	-	-
Context::OrdersContainsProdsDAO public synchronized Set<OrdersContainsProdsBean> doRetrieveAllP	permette la restituzione dei prodotti contenuti in un ordine presente nel database	pre: orderId != null and orderId > 0	-

rods(int orderId, String order)	con orederId fornito		

AddToCartServlet

Classe			
Nome		Descrizione	
AddToCartServlet		Classe che permette l'aggiunta di un prodotto al carrello	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::AddToCartServlet public void doGet(HttpServletRequest request, HttpServletResponse response)	permette l'aggiunta di un prodotto al carrello attraverso il passaggio nella richiesta di parametri	pre: request != null and response != null and request.getSession().getAttribute("roles") != null and getServletContext().getAttribute("DataSource") != null and request.getSession().getAttribute("carrello") != null and request.getParameter("id") != null and request.getParameter("size") != null	-

CheckOutServlet

Classe	
Nome	Descrizione
CheckOutServlet	Classe che permette l'acquisto dei prodotti presenti nel carrello con la creazione dell'ordine
Metodi	

Nome	Descrizione	Pre-Condition	Post-Condition
Context::CheckOutServlet public void doPost(HttpServletRequest request, HttpServletResponse response)	permette l'acquisto dei prodotti presenti nel carrello e il conseguente salvataggio nell'ordine	pre: request != null and response != null and request.getSession().getAttribute("utente") != null and getServletContext().getAttribute("DataSource") != null and request.getSession().getAttribute("carrello") != null	-

GetProdsFromCartsServlet

Classe	
Nome	Descrizione
GetProdsFromCartsServlet	Classe che permette di prelevare tutti i prodotti presenti nel carrello

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::GetProdsFromCartsServlet public void doGet(HttpServletRequest request, HttpServletResponse response)	permette di prelevare tutti i prodotti presenti nel carrello e riproporli alla pagina del carrello	pre: request != null and response != null and request.getSession().getAttribute("roles") != null and getServletContext().getAttribute("DataSource") != null and request.getSession().getAttribute("carrello") != null	-

ModQuantityInCartServlet

Classe	
Nome	Descrizione
ModQuantityInCartServlet	Classe che permette di modificare la quantità dei prodotti nel carrello aggiornando il totale

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::ModQuantityInCartServlet public void doGet(HttpServletRequest request, HttpServletResponse response)	permette di modificare la quantità dei prodotti nel carrello inviata dalla richiesta	pre: request != null and response != null and request.getSession().getAttribute("roles") != null and getServletContext().getAttribute("DataSource") != null and request.getSession().getAttribute("carrello") != null and request.getParameter("offset") != null and request.getParameter("size") != null	-

OrderManageServlet

Classe	
Nome	Descrizione
OrderManageServlet	Classe che permette di modificare lo stato di un ordine da non spedito a spedito

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition

Context::OrderManageServlet public void doGet(HttpServletRequest request, HttpServletResponse response)	permette di spedire un ordine	pre: request != null and response != null and request.getSession().getAttribute("utente") != null and getServletContext().getAttribute("DataSource") != null and request.getParameter("id") != null	-

RemFromCartServlet

Classe			
Nome		Descrizione	
RemFromCartServlet		Classe che permette di rimuovere il prodotto dal carrello aggiornando il totale	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::RemFromCartServlet public void doGet(HttpServletRequest request, HttpServletResponse response)	permette di rimuovere un prodotto dal carrello	pre: request != null and response != null and request.getSession().getAttribute("roles") != null and getServletContext().getAttribute("DataSource") != null and request.getSession().getAttribute("carrello") != null and request.getParameter("id") != null and request.getPar	-

		ameter("size") != null	

GestioneCatalogo

ManagesProdsDAO

Classe			
Nome		Descrizione	
ManagesProdsDAO		Classe che implementa l'interfaccia ManagesProdsInterf per la gestione delle operazioni (per tenerne traccia) di inserimento, cancellazione e modifica sul prodotto	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::ManagesProdsDAO public synchronized void doSave(String gestCatUsername, String productId, int tipologia)	permette il salvataggio della relazione di gestione prodotto nel database	pre: gestCatUsername!= null and productId > 0 and tipologia >= 0	-
Context::ManagesProdsDAO public synchronized void doDelete(int id)	permette l'eliminazione della relazione di gestione prodotto nel database di un id fornito	pre: id > 0	-
Context::ManagesProdsDAO public synchronized ManagesProdsBean doRetrieveByKey(int id)	permette la restituzione dei dati della relazione di gestione prodotto nel database con un id fornito	pre: id > 0	-
Context::ManagesProdsDAO	permette la restituzione	-	-

public synchronized Set<ManagesProdsBean> doRetrieveAll(String order)	dei dati di tutte le relazioni di gestione prodotto presenti nel database		

ProductsDAO

Classe			
Nome		Descrizione	
ProductsDAO		Classe che implementa l'interfaccia ProductsInterf per la gestione dei dati	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::ProductsDAO public synchronized void doSave(String id, String name, String sport, String brand, int price)	permette il salvataggio del prodotto nel database	pre: id >0 and name!= null and sport!= null and brand!= null and price>=0	-
Context::ProductsDAO public synchronized void doUpdate(String id, String name, String sport, String brand, int price)	permette l'aggiornamento del prodotto nel database	pre: id >0 and name!= null and sport!= null and brand!= null and price>=0	-
Context::ProductsDAO public synchronized void doDelete(String id)	permette l'eliminazione del prodotto nel database di un id fornito	pre: id >0	-

Context::ProductsDAO public synchronized ProductsBean doRetrieveByKey(String id)	permette la restituzione dei dati del prodotto nel database di un id fornito	pre: id >0	-
Context::ProductsDAO public synchronized Set<SizesBean> doRetrieveAll(String order)	permette la restituzione dei dati di tutti i prodotti presenti nel database	-	-
Context::ProductsDAO public synchronized Set<ProductsBean> doRetrieveAllExists(String order)	permette la restituzione dei dati di tutti i prodotti presenti nel database che non sono stati eliminati	-	-

SizesDAO

Classe			
Nome		Descrizione	
SizesDAO		Classe che implementa l'interfaccia SizesInterf per la gestione delle taglie dei prodotti	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::SizesDAO public synchronized int doSave(String productId, int valued)	permette il salvataggio della taglia nel database	pre: productId >0 and valued >0	-
Context::SizesDAO public synchronized void	permette l'eliminazione della taglia nel database di un id	pre: pre: productId >0 and valued >0	-

doDelete(String productId, int valued)	fornito e un valore fornito		
Context::SizesDAO public synchronized Set<SizesBean> doRetrieveByProductId(String productId, String order)	permette la restituzione dei dati della taglia nel database di un id fornito	pre: id >0	-
Context::SizesDAO public synchronized Set<ProductsBean> doRetrieveAll(String order)	permette la restituzione dei dati di tutte le taglie presenti nel database	-	-

AddInCatalogServlet

Classe			
Nome		Descrizione	
AddInCatalogServlet		Classe che permette l'aggiunta di un prodotto nel catalogo	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::AddInCatalogServlet public void doPost(HttpServletRequest request, HttpServletResponse response)	permette l'aggiunta di un prodotto nel catalogo inserendo i suoi attributi che sono: nome, codice scarpa, sport, brand e prezzo	pre: request != null and response != null and request.getParameter("nomeProd") != null and request.getParameter("productId") != null and request.getParameter("sport") != null and request.getParameter("brand") != null and request.getPar	-

		ameter(“price”) != null and request.getParameter(“inputImage”) != null and getServletContext().getAttribute("DataSource") != null and getServletContext().getRealPath("/"+"img"+"/prod"+File.separator+fileName) != null	

ImageServlet

Nome		Descrizione	
ImageServlet		Classe che permette l’aggiunta di un immagine per un prodotto	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::ImageServlet public void doGet(HttpServletRequest request, HttpServletResponse response)	permette l’aggiunta di un’immagine relativa ad un prodotto	pre: request != null and response != null and request.getParameter(“immagine”) getServletContext().getRealPath("/"+"img"+"/prod"+File.separator+nomeImg) != null	-

ModInCatalogServlet

Nome		Descrizione	
ModInCatalogServlet		Classe che permette la modifica di un prodotto nel catalogo	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition

Context::ModInCatalogServlet public void doPost(HttpServletRequest request, HttpServletResponse response)	permette la modifica di un prodotto nel catalogo	pre: request != null and response != null and request.getParameter(“nomeProd”) != null and request.getParameter(“productId”) != null and request.getParameter(“sport”) != null and request.getParameter(“brand”) != null and request.getParameter(“price”) != null	-

RemFromCatalogServlet

Nome	Descrizione
RemFromCatalogServlet	Classe che permette la modifica di un prodotto nel catalogo

Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::RemCatalogServlet public void doGet(HttpServletRequest request, HttpServletResponse response)	permette la rimozione di un prodotto nel catalogo	pre: request != null and response != null and request.getParameter(“productId”) != null and request.getSession().getAttribute("utente")	-

Utils

MainContext

Nome		Descrizione	
MainContext		Classe che permette di inizializzare ed eliminare il contesto della web application	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::MainContext public void contextInitialized(ServletContextEvent event)	permette l'inizializzazione del contesto con la connessione al database	pre: event != null	post: event.getServletContext().setAttribute("DataSource") != null
Context::MainContext public void public void contextDestroyed(ServletContextEvent event)	permette l'eliminazione del contesto	-	event == null

PopolamentoDBServlet

Nome		Descrizione	
PopolamentoDBServlet		Classe che permette il popolamento del database	
Metodi			
Nome	Descrizione	Pre-Condition	Post-Condition
Context::PopolamentoDBServlet protected void doGet(HttpServletRequest request, HttpServletResponse response)	permette il popolamento del database	pre: request != null and response != null and getServletContext().getAttribute("DataSource") != null	-

Class Diagram