O.D.D.
Object Design Document

Riferimento

Versione	**1.2**
Data	28/01/2020
Destinatario	Prof. A. De Lucia
Presentato da	Alessandro Bergamo [ABe], David Capuano [DCa], Salvatore Villano [SVi], Antonio Mancuso [AMa]

Revision History

Data	Versione	Descrizione	Autori
05/12/2019	Draft 0.1	Strutturazione Documento	ABe
05/12/2019	Draft 0.2	Inserita Introduzione al Documento	ABe
05/12/2019	Draft 0.3	Inserita Strutturazione in Packages del sistema	ABe
07/12/2019	Draft 0.4	Inserimento Class Interfaces	ABe, DCa
08/12/2019	Draft 0.5	Terminazione e rifinitura Class Interfaces	ABe, DCa
10/12/2019	Draft 0.6	Rimozione Class Diagram, Class Interfaces dei Control	ABe, DCa, SVi, AMa
17/12/2019	Draft 0.7	Inserimento Class Diagram	AMa
17/12/2019	Draft 0.8	Revisione del Documento e correzioni minimali	ABe
16/01/2020	Draft 0.9	Correzzioni minimali a seguito di implementazione	ABe, DCa
18/01/2020	Draft 1.0	Inserimento System Component Diagram	ABe
24/01/2020	Draft 1.1	Correzzione e Modifica del Documento	ABe
28/01/2020	Draft 1.2	Ultimazione Class Diagram	ABe

Sommario

1. Introduzione

1.1 Object Design Trade-offs

Comprensibilità vs Costi

Si preferisce aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone attualmente non coinvolte nel progetto o le persone coinvolte che non hanno lavorato ad una parte in particolare. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorano la comprensibilità agevolando la manutenzione.

Comprare vs Costruire

Si preferisce comprare in quanto le componenti off-the-shelf che useremo sono open-source e quindi non richiedono spese aggiuntive.

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 ci interessano due framework e una libreria java per applicazioni web: Bootstrap, JQuery, Javax.Mail.

Bootstrap è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web ed è open source. Esso contiene modelli di progettazione HTML e CSS:

- Per la tipografia e l'interfaccia grafica
- Per componenti di interfaccia come moduli, bottoni, navigazione ecc.
- Per estensioni opzionali utilizzati per script in JavaScript

JQuery è un framework JavaScript open source che rende più semplice:

- Manipolare il DOM delle pagine HTML

Javax.Mail è una libreria Java che permette di inviare e-mail da codice Java.

1.3 Linea Guida per la Documentazione delle Interfacce

Per rendere più efficiente la comunicazione, dovranno essere rispettate le seguenti linee guida:

Linee guida generali:

- A tutti i metodi, le classi ed i file prodotti, deve essere allegato un commento che specifichi l'obiettivo da raggiungere e il perché delle decisioni prese.
- Prima dell'implementazione della logica di un algoritmo, esso sarà rivisto da tutti i membri del team per la correzione di eventuali errori nella logica.

Organizzazione di metodi, parametri, variabili e classi:

- La convenzione "Camel Notation" deve essere adottata da tutti i membri del team.
- Le classi avranno il nome al singolare.
- I nomi dei campi e dei parametri saranno dei sostantivi mentre, quelli dei metodi, dei verbi (tutti i nomi dovranno essere evocativi della funzione dei metodi, dei parametri, delle classi, dei file ecc. ai quali sono assegnati).
- Nel caso venga usato più volte lo stesso valore numerico all'interno del codice, è opportuno inizializzare una costante con quel valore.
- Inizializzare le variabili locali nel punto in cui sono state dichiarate a meno che il suo valore iniziale non dipenda da un calcolo che occorre eseguire prima.

Organizzazione dei file:

Ogni file deve essere correlato alla funzionalità che persegue e diviso in più file se raggiunge una dimensione difficile da leggere e comprendere.

Indentazione:

- Per l'indentazione dei file ".java" verranno seguite le regole di codice di Google, controllate con il tool CheckStyle integrato in Eclipse o in IntelliJ IDEA.
- Per l'indentazione dei file ".jsp" verrà eseguita l'indentazione automatica di Eclipse con la shortcut ctrl+shift+f o ctrl+alt+l con IntelliJ IDEA.

1.4 Definizioni, acronimi ed abbreviazioni

DBMS: Data Base Management System

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

Framework: Software di supporto allo sviluppo web

HTML: Linguaggio di mark-up per pagine web

CSS: Linguaggio usato per definire la formattazione di pagine 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, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso

Bootstrap: Framework che contiene librerie utili per lo sviluppo responsive di pagine web

JQuery: è un framework JavaScript open source che rende più semplice manipolare il DOM delle pagine HTML

1.5 Riferimenti

- RAD 2.1
- SDD 1.1
- Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, (2nd edition), Prentice-Hall, 2003
- https://getbootstrap.com/
- https://jquery.com/

2. Packages

Packages	
Nome	**Descrizione**
web	Contiene tutte i file jsp che vengono mostrati all'utente per inserire o visualizzare dati, questi file fanno richiesta alle Servlet presenti nel package com.champloo.control
com.champloo.control	Contiene tutte le classi Servlet che prendono i parametri dalle jsp e eseguono i metodi logici chiamando le classi model presenti nel package com.champloo.model
com.champloo.model	Contiene tutte le classi model che eseguono le operazioni CRUD sul database, istanziando oggetti bean presenti nel package com.champloo.bean e stabilendo una connessione al DB con gli oggetti del package com.champloo.storage
com.champloo.bean	Contiene tutte le classi bean che rappresentato i dati persistenti
com.champloo.storage	Contiene tutte le classi per stabilire una connessione con il database MySQL champloo_store_db
com.champloo.test	Contiene tutte le classi che servono per il testing white-box del Sistema
com.champloo.util	Contiene tutte le classi utili per il corretto e semplificato utilizzo del Sistema

3. Class Interfaces

Nome Classe	**UserModel**
Descrizione	Interfaccia della classe che rappresenta le funzionalità relative alla gestione del login, log-out, registrazione, aggiornamento, caricamento, eliminazione e ban di un utente. 1. registerUser(UserBean newUser) 2. retrieveUserByEmail(String user_email) 3. retrieveUserByUsername(String username) 4. retrieveAllUsers() 5. updateUser(UserBean user) 6. deleteUser(UserBean user) 7. blockUser(int user_id) 8. login(String username, String password) 9. changePassword(UserBean user, String newPassword)
Pre-condizione	1. **context** UserModel:: registerUser(newUser); **pre:** newUser!=null 2. **context** UserModel:: retrieveUserByEmail(user_email); **pre:** user_email!=null 3. **context** UserModel:: retrieveUserByUsername(username); **pre:** username!=null 4. **context** UserModel:: retrieveAllUsers(); 5. **context** UserModel:: updateUser(user); **pre:** user!=null 6. **context** UserModel:: deleteUser(user); **pre:** user!=null 7. **context** UserModel:: blockUser(user_id); **pre:** user_id!=null && user_id!=0 8. **context** UserModel:: login(username, password); **pre:** username!=null && password!=null; 9. **context** UserModel:: changePassword(user, newPassword); **pre:** user!=null && newPassword!=null;
Post-condizione	**NA**
Invarianti	**NA**

Nome Classe	**ProductModel**
Descrizione	Interfaccia della classe che rappresenta le funzionalità relative alla gestione dell'aggiunta, eliminazione, modifica e caricamento di un prodotto. **1.** addProduct(ProductBean newProduct, ProductDetailsBean newProductDetails) **2.** retrieveProductWithDetails(int id_product, int id_product_details) **3.** retrieveById(int id_product) **4.** retrieveByModel(String model_product) **5.** retrieveByCategory(String type_product) **6.** retrieveByBrand(String brand_product) **7.** retrieveByFeedbacks() **8.** retrieveByAverage() **9.** retrieveByColor(String color_product) **10.** retrieveBySize(String size_product) **11.** retrieveByStatus(String status_prod) **12.** retrieveAll() **13.** createWindow() **14.** retrieveNewProducts() **15.** deleteProduct(int id_product) **16.** updateProduct(ProductBean newProduct, ProductDetailsBean newProductDetails)
Pre-condizione	**1.** **context** ProductModel:: addProduct(newProduct, newProductDetails); **pre:** newProduct!=null && newProductDetails!=null **2.** **context** ProductModel::retrieveProductWithDetails(int id_product, int id_product_details) **pre:** id_product!=null && id_product_details!=null **3.** **context** ProductModel:: retrieveById(id_product); **pre:** id_product!=null **4.** **context** ProductModel:: retrieveByModel(model_product); **pre:** model_product!=null **5.** **context** ProductModel:: retrieveByCategory(type_product); **pre:** type_product!=null **6.** **context** ProductModel:: retrieveByBrand(brand_product); **pre:** brand_product!=null **7.** **context** ProductModel:: retrieveByFeedbacks(); **8.** **context** ProductModel:: retrieveByAverage(); **9.** **context** ProductModel:: retriveByColor(color_product); **pre:** color_product!=null **10.** **context** ProductModel:: retrieveBySize(size_product); **pre:** size_product!=null **11.** **context** ProductModel:: retrieveByStatus(status_product); **pre:** status_product!=null **12.** **context** ProductModel:: retrieveAll(); **13.** **context** ProductModel:: deleteProduct(id_product) **pre:** id_product!=null

	14. context ProductModel:: updateProduct(newProduct, newProductDetails) **pre:** newProduct!=null && newProductDetails!=null
Post-condizione	**NA**
Invarianti	**NA**

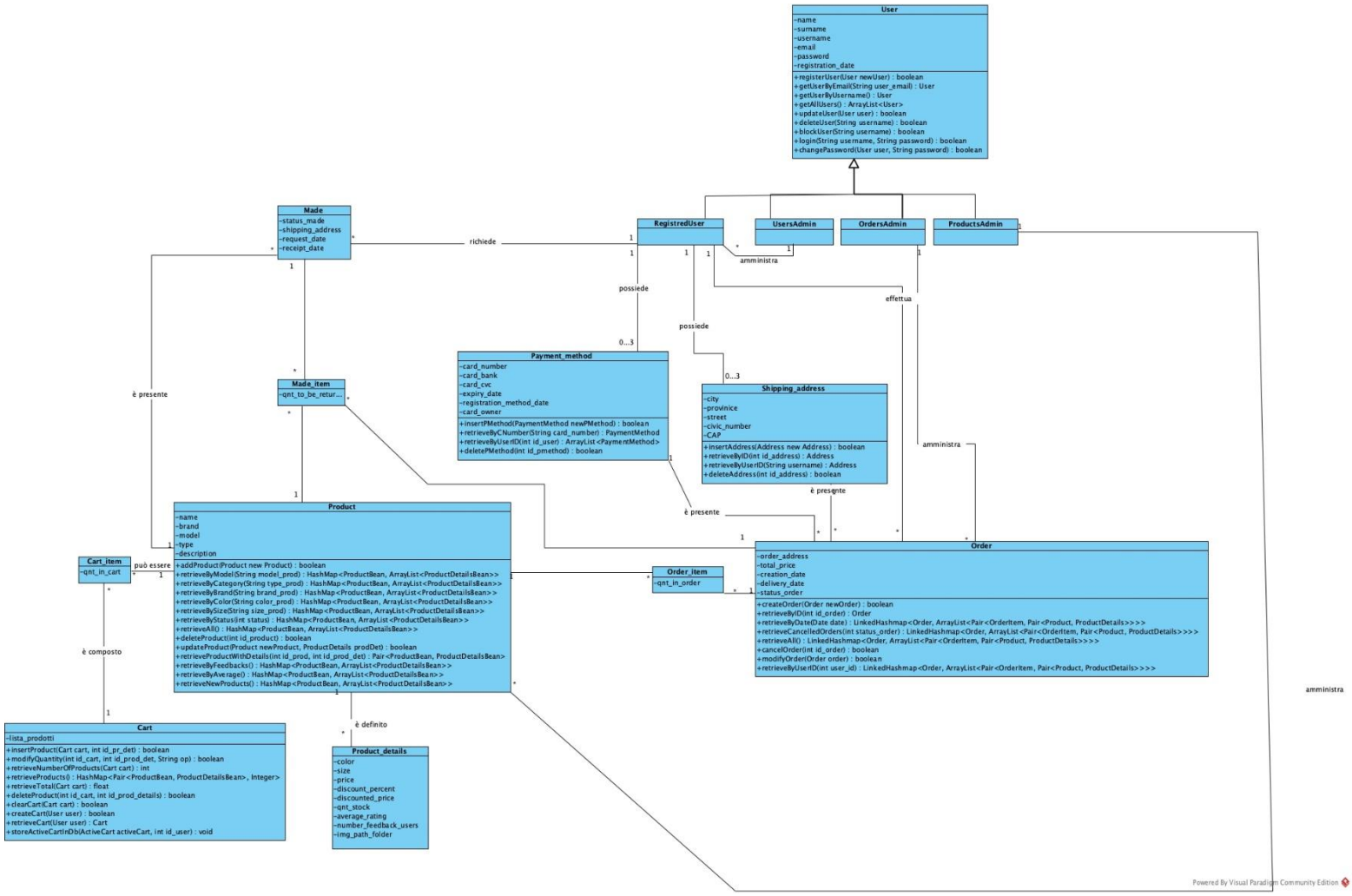
Nome Classe	**OrderModel**
Descrizione	Interfaccia della classe che rappresenta le funzionalità relative alla gestione della creazione, annulla, modifica e caricamento di un ordine. **1.** createOrder(OrderBean newOrder, HashMap<Pair<ProductBean, ProductDetailsBean>, Integer> products_in_order)retrieveByID(int id_order) **2.** modifyOrder(int order_id, int status, Date delivery_date) **3.** cancelOrder(int order_id) **4.** retrieveByID(int order_id) **5.** retrieveByUserID(int user_id) **6.** retrieveByDate(Date start_date, Date end_date) **7.** retrieveCancelledOrders(int status_order) **8.** retrieveAll() **9.** retrieveByOrder(int id_order)
Pre-condizione	**1.** **context** OrderModel:: createOrder(newOrder, products_in_order) **pre:** newOrder!=null && products_in_order!=empty **2.** **context** OrderModel:: modifyOrder(order); **pre:** order_id!=null && order_id!=0 && status!=null && status!=0 && delivery_date!=null **3.** **context** OrderModel:: cancelOrder(order_id); **pre:** order_id!=null **4.** **context** OrderModel:: retrieveByID(order_id); **pre:** order_id!=null **5.** **context** OrderModel:: retrieveByUserID(user_id); **pre:** user_id!=null **6.** **context** OrderModel:: retrieveByDate(start_date, end_date); **pre:** date!=null && end_date!=null **7.** **context** OrderModel:: retrieveCancelledOrders(status_order); **pre:** status_order!=null && status_order=5 **8.** **context** OrderModel:: retrieveAll(); **9.** **context** OrderModel:: retrieveByOrder(id_order); **pre:** id_order!=null
Post-condizione	**NA**
Invarianti	**NA**

Nome Classe	**AddressModel**
Descrizione	Interfaccia della classe che rappresenta le funzionalità relative alla gestione dell'inserimento, eliminazione e caricamento di un indirizzo. 1. insertAddress(AddressBean newAddress) 2. deleteAddress(int id_address) 3. retrieveByID(int id_address) 4. retrieveAddressByUserID(int id_user)
Pre-condizione	1. **context** AddressModel:: insertAddress(newAddress); **pre:** newAddress!=null 2. **context** AddressModel:: deleteAddress(id_address); **pre:** id_address!=null 3. **context** AddressModel:: retrieveAddressByID(id_address); **pre:** id_address!=null 4. **context** AddressModel:: retrieveAddressByUserID(id_user); **pre:** id_user!=null
Post-condizione	**NA**
Invarianti	**NA**

Nome Classe	**PaymentMethodModel**
Descrizione	Interfaccia della classe che rappresenta le funzionalità relative alla gestione dell'inserimento, eliminazione e caricamento di un metodo di pagamento. 1. insertPMethod(PaymentMethodBean newPMethod) 2. deletePMethod(int id_pmethod) 3. retrieveByCNumber(String card_number) 4. retrieveByUserID(int id_user)
Pre-condizione	1. **context** PaymentMethodModel:: insertPMethod(newPMethod); **pre:** newPMethod!=null 2. **context** PaymentMethodModel:: deletePMethod(id_pmethod); **pre:** id_pmethod!=null && id_pmethod!=0 3. **context** PaymentMethodModel:: retrieveByCNumber(card_number); **pre:** card_number!=null 4. **context** PaymentMethodModel:: retrieveByUserID(id_user); **pre:** id_user!=null && id_user!=0
Post-condizione	**NA**
Invarianti	**NA**

Nome Classe	**CartModel**
Descrizione	Interfaccia della classe che rappresenta le funzionalità relative alla gestione dell'inserimento, eliminazione e caricamento di un carrello. **1.** insertProduct(CartBean cart, int id_product_details) **2.** modifyQuantity(int id_cart, int id_product_details, String operation) **3.** retrieveNumberOfProducts(CartBean cart) **4.** retrieveProducts(CartBean cart) **5.** retrieveTotal(CartBean cart) **6.** deleteProduct(int id_cart, int id_product_details) **7.** clearCart(CartBean cart) **8.** createCart(UserBean user) **9.** retrieveCart(UserBean user) **10.** retrieveCartItem(int id_cart_item) **11.** storeActiveCartInDb(ActiveCart activeCart, int id_user)
Pre-condizione	**1. context** CartModel:: insertProduct(cart, id_product_details); **pre:** cart!=null && id_product_details!=null && id_product_details!=0 **2. context** CartModel:: modifiyQuantity(id_cart, id_product_details, operation); **pre:** id_cart!=null && id_cart!=0 && id_product_details!=null && id_product_details!=null && operation!=null **3. context** CartModel:: retrieveNumberOfProducts(cart); **pre:** cart!=null **4. context** CartModel:: retrieveProducts(cart); **pre:** cart!=null **5. context** CartModel:: retrieveTotal(cart); **pre:** cart!=null **6. context** CartModel:: deleteProduct(id_cart, id_product_details); **pre:** id_cart!=null && id_cart!=0 && id_product_details!=null && id_product_details!=0 **7. context** CartModel:: clearCart(cart); **pre:** cart!=null **8. context** CartModel:: createCart(user); **pre:** user!=null **9. context** CartModel:: retrieveCart(user); **pre:** user!=null **10. context** CartModel:: retrieveCartItem(id_cart_item) **pre:** id_cart_item!=null && id_cart_item!=0 **11. context** CartModel:: storeActiveCartInDb(activeCart, id_user) **pre:** activeCart!=null && id_user!=null && id_user!=0
Post-condizione	**NA**
Invarianti	**NA**

4. Class Diagram



5. Glossario

ODD (Object design Document): Documento che approfondisce l'analisi dei requisiti e rappresenta le decisioni relative all'implementazione.

Facade: Un oggetto che permette, attraverso un'unica interfaccia, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.

Class interface: Sezione dell'ODD che descrive le classi definite per l'implementazione, indicando i contratti (formati da precondizioni, post condizioni e invarianti).

Componenti off-the-shelf: componenti hardware e software già disponibili, open-source o meno, che possono essere usate all'interno di un progetto.

HTML (HyperText Markup Language): Linguaggio di markup per pagine web.

CSS (Cascading Style Sheets): Linguaggio usato per la formattazione di pagine HTML, XHTML E XML.

JQuery: Libreria JavaScript per la gestione di eventi e l'animazione di elementi DOM in una pagina HTML.

Bootstrap: Raccolta di software open-source per la creazione di siti e applicazioni web.

JavaScript: Linguaggio di programmazione orientato ad oggetti e ad eventi, utilizzato per la realizzazione di effetti dinamici interattivi in siti e applicazioni web.

Servlet: Oggetti in linguaggio Java utilizzati per la creazione di un'applicazione web.

Bean model: Modelli usati per rappresentare dei dati salvati permanentemente.

singleton: Design pattern che consente di creare una sola istanza di una determinata classe.

Camel Notation: La pratica, tipica del linguaggio di programmazione Java, di scrivere le parole composte senza spazi, mettendo in maiuscolo le iniziali.

DBMS (Database management system): Sistema software utilizzato per la gestione di database.

AJAX (Asynchronus JavaScript and XML): Tecnica di sviluppo software di applicazioni web basata sullo scambio in background di dati tra un web browser e un server.