
# O.D.D.

## Object Design Document

"Maggico Car & Motorbike Parts"

Raffaele Coscione 0512106006

Vincenzo Tortora 0512102104

Francesco Carotenuto 0512104798

Giovanni Renzulli 0512105730

# Top Manager

| Nome |
| --- |
| Andrea De Lucia |

# Partecipanti

| Nome | Matricola |
| --- | --- |
| Raffaele Coscione | 0512106006 |
| Vincenzo Tortora | 0512102104 |
| Francesco Carotenuto | 0512104798 |
| Giovanni Renzulli | 0512105730 |

# History

| Data | Versione | Cambiamenti | Autore |
| --- | --- | --- | --- |
| 20/01/2020 | 1.0 | Stesura delle componenti ed aggiunta introduzione | Francesco Carotenuto |
| 21/01/2020 | 1.1 | Aggiunta package Manager | Raffaele Coscione |
| 31/01/2020 | 1.2 | Aggiunta Interfacce delle classi | Vincenzo Tortora |
| 13/02/2020 | 1.3 | Completamento e revisione ODD | Francesco Carotenuto |

# Sommario


1. INTRODUZIONE ..... 4

1.1 Object Design Trade-Offs ..... 4

1.2 Linee guida per l'implementazione ..... 5

1.3 Definizione, acronimi e abbreviazioni ..... 5

2. Design Pattern ..... 7

2.1 Design Pattern Globali ..... 6

3. Package Components ..... 8

3.1 Package Bean ..... 8

3.2 Package View ..... 9

3.3 Package Control ..... 11

3.4 Package Autenticazione ..... 12

3.5 Package Manager ..... 13

4.Class Interfaces ..... 15

4.1 Prodotti Manager ..... 14

4.2 Carrello Manager ..... 15

4.3 Admin Manager ..... 16

4.4 Cliente Manager ..... 17

4.5 Utente Manager ..... 18

# 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 gli obbiettivi da seguire, 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 le signature dei sottosistemi definiti nel System Design. Inoltre, sono specificati i trade-off e le linee guida.

**Prestazioni vs Costi**

Non avendo finanziamenti esterni, si utilizzeranno delle tecnologie open-source in grado di gestire il sistema in maniera gratuita. Nello specifico, verrà utilizzato un database relazionale come repository centrale per i dati gestiti dal sistema e un web server monolitico per la gestione dell'interazione con gli utenti.

**Sicurezza vs Efficienza**

Il sistema si baserà prevalentemente sulla gestione della sicurezza per evitare accessi non autorizzati cosi da proteggere informazioni personali quali e-mail, password e carte di credito.

**Comprensibilità vs Tempo**

Il codice deve essere il più chiaro possibile, ogni componente deve essere accompagnato da un commento in grado di descrivere quali operazioni si stanno implementando. Questa forma di comprensibilità del codice porterà dei rallentamenti in fase implementativa e di testing andando a creare, però, vantaggi sulla comprensione globale del sistema e delle sue componenti.

**Interfaccia vs Usabilità**

L'interfaccia verrà gestita in modo tale da poter essere il più semplice ed intuitiva possibile, attraverso l'uso di form e bottoni di facile comprensione per l'utente finale.

## 1.2 Linee guida per la documentazione delle interfacce

Per lo scrittura del codice si seguiranno le seguenti linee guida: Naming Convention Utilizzeremo le seguenti convenzioni per i nomi:

- Descrittivi
- Pronunciabili
- Di uso comune
- Di lunghezza media
- Non abbreviati
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili

• I nomi delle variabili devono sempre iniziare con una lettera minuscola e le parole successive con una maiuscola. Le dichiarazioni delle variabili devono essere fatte all'inizio del blocco di codice, le variabili dello stesso tipo sono dichiarate sulla stessa riga.

• In alcuni casi viene utilizzato il carattere underscore "_", per le variabili costanti oppure quando vengono utilizzate delle proprietà statiche.

Metodi

• I nomi dei metodi devono iniziare con una lettera minuscola, e le parole successive con lettera maiuscola. Solitamente il nome di un metodo consiste in un verbo che identifica l'azione da svolgere, seguite dal nome di un oggetto.

• I nomi dei metodi per l'accesso alle variabili devono essere del tipo "getNomeVariabile()", mentre i metodi per la modifica delle variabili devono essere del tipo "setNomeVariabile()".

• Se viene utilizzata una variabile all'interno di un metodo, questa deve essere sempre dichiarata prima del suo utilizzo e deve essere utilizzata per un solo scopo, per facilitare la leggibilità del codice.

• Ad ogni metodo viene aggiunta una descrizione per specificare la loro funzione, come i valori riguardanti gli argomenti, i valori di ritorno e le eccezioni. Questa descrizione deve essere posizionata prima della dichiarazione del metodo.

Classi e pagine

• I nomi delle classi devono iniziare con una lettera maiuscola e anche le parole successive, mentre i nomi delle pagine possono iniziare sia con minuscole che con maiuscole

## 1.3 Definizioni, acronimi e abbreviazioni

**Acronimi:**

• SDD: System Design Document • ODD: Object Design Document • RAD: Requirements Analysis Document

**Abbreviazioni:**

• DB: Database • DBMS: Database Management System

**Definizioni:**

• Servlet: Classi ed oggetti Java per la gestione di operazioni su un Web Server

## 1.4 Riferimenti

Il contesto è ripreso dal RAD e dall' SDD del progetto Maggico Car & Motorbike Parts.

È stato anche usato come riferimento il libro: Object-Oriented Software Engineering: Using UML, Patterns, and Java, 3rd Edition Prentice Hall, Upper Saddle River, NJ, September 25, 2009.

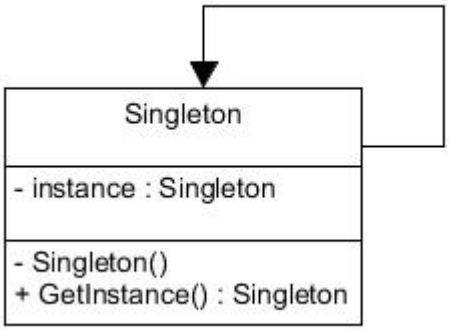
inoltre, sono stati usati dei materiali di supporto visionabili al link: https://wwwbruegge.in.tum.de/lehrstuhl_1/component/content/article/217-OOSE.

Infine è stato usato anche il libro : Design Patterns: Elements of Reusable Object-Oriented Software, Gang of four, 1994.

# 2. Design Pattern

## 2.1 Design Pattern Globali

Utilizzeremo il Singleton pattern per le classi di tipo Manager.

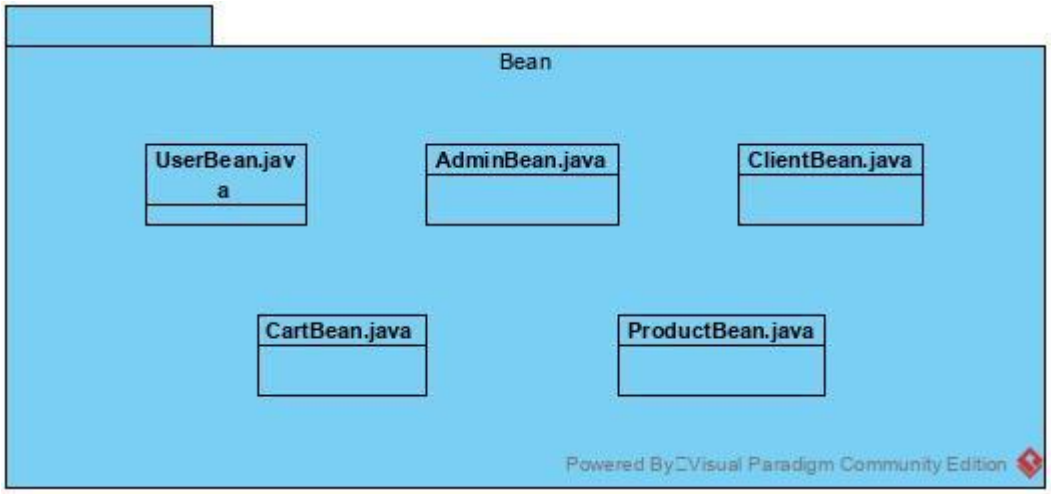



Il singleton è un design pattern che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.
L'implementazione più semplice di questo pattern prevede che la classe singleton abbia un unico costruttore privato, in modo da impedire l'istanziazione diretta della classe. La classe fornisce in un metodo getter statico che restituisce l'istanza della classe (sempre la stessa), creandola preventivamente o alla prima chiamata del metodo, e memorizzandone il riferimento in un attributo privato anch'esso statico.

# 3. Package Components

### 3.1 Package Bean

Il package bean include tutte le classi JavaBean. Di seguito, ecco riportato il package che raggruppa tutte le classi del sistema che hanno tale caratteristica:

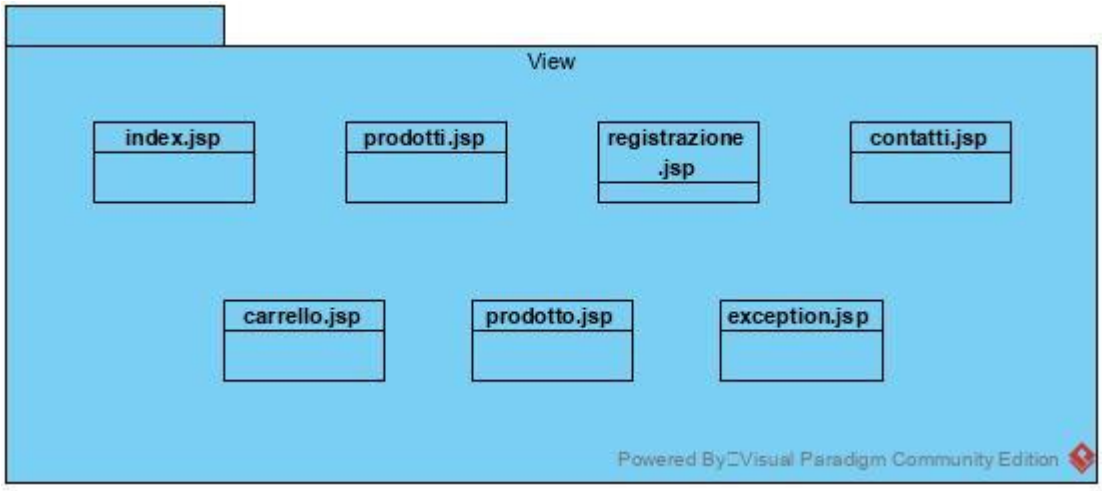



| Nome | Descrizione |
|---|---|
| UserBean.java | Descrive un account generico |
| AdminBean.java | Descrive un account di tipo amministratore |
| ClientBean.java | Descrive un account di tipo cliente |

| | |
|---|---|
| CartBean.java | Descrive un carrello personale di un singolo cliente |
| ProductBean.java | Descrive un prodotto disponibile |

### 3.2 Package View

Il package view include tutte le pagine JSP, ossia quelle pagine che gestiscono la visualizzazione dei contenuti da parte di un utente del sistema. Di seguito, ecco riportato il package che raggruppa tutte le pagine del sistema che hanno tale caratteristica:

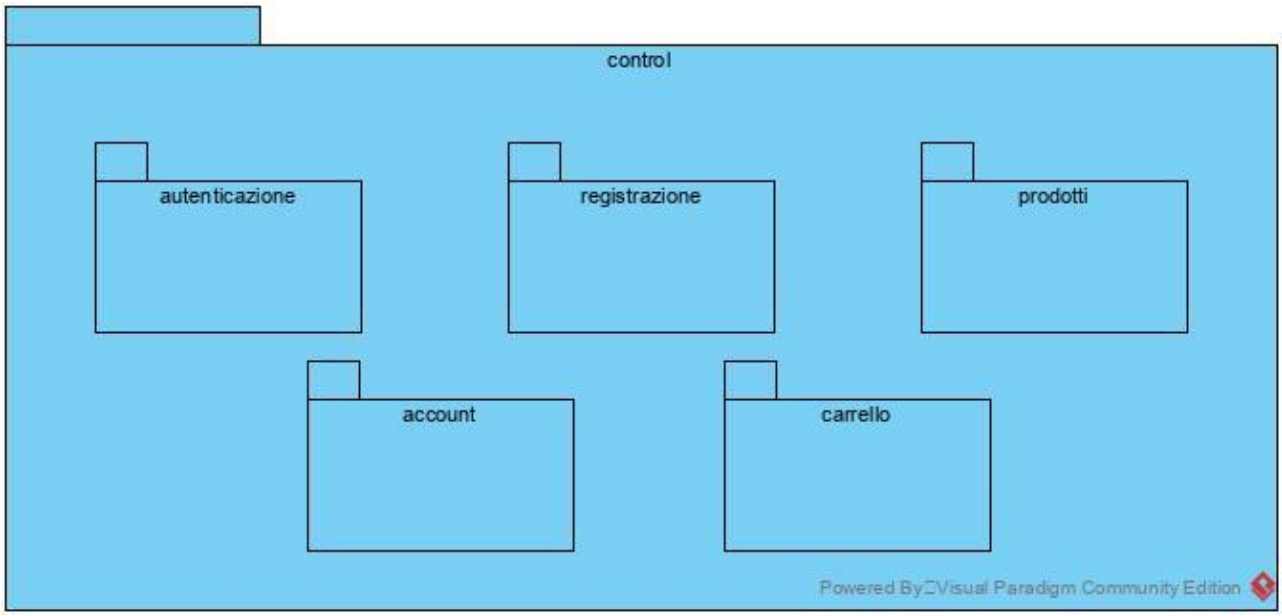



| Nome | Descrizione |
|---|---|
| index.jsp | Pagina di presentazione della piattaforma, presenta il form di login e le varie opzioni di navigazione |
| prodotti.jsp | Pagina per la visualizzazione del catalogo. Include una barra di ricerca per cercare un prodotto all'interno del catalogo. |
| registrazione.jsp | Pagina per effettuare la registrazione di un nuovo cliente. |
| contatti.jsp | Pagina avente le informazioni necessari affinché si possa contattare un responsabile dell'azienda |
| carrello.jsp | Pagina contenente tutti i prodotti messi nel carrello dal cliente. |
| exception.jsp | Pagina per la visualizzazione di un errore verificatosi durante la navigazione. |
| prodotto.jsp | Pagina per la visualizzazione di un singolo prodotto. Nel caso dell'amministratore, questa pagina darà |

| | |
|---|---|
| | anche strumenti per l'aggiornamento dello stesso. |

## 3.3 Package Control

Il package control include tutte le classi servlet che rappresentano la logica applicativa della piattaforma. Il sistema vede la separazione di tali classi in sotto-package differenziati dai vari sottosistemi della piattaforma proposta. Di seguito, illustriamo la suddivisione dei sotto-package con il seguente diagramma:

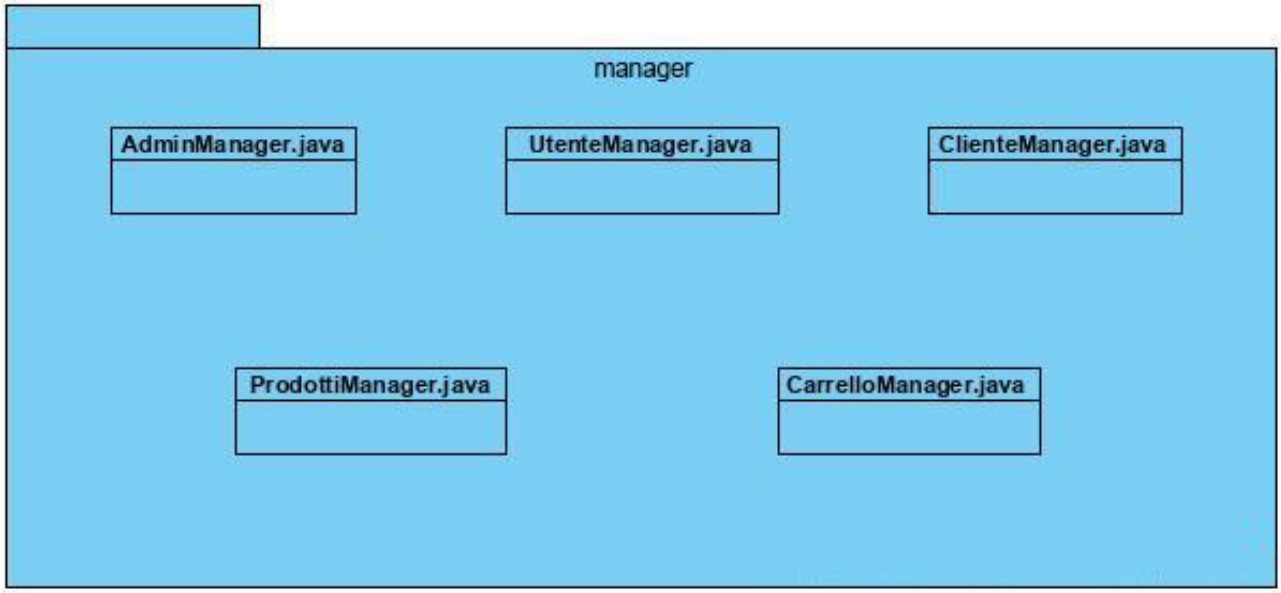


### 3.4 Package autenticazione

Il package autenticazione include tutte quelle classi servlet adibite a svolgere una funzionalità del sottosistema di gestione delle autenticazioni.


| Nome | Descrizione |
|---|---|
| LoginServlet.java | Controller che permette di completare un'operazione di login. |
| LogoutServlet.java | Controller che permette di effettuare il log out. |

### 3.5 Package Manager




| Nome | Descrizione |
|---|---|
| AdminManager.java | Classe di gestione dei dati di un amministratore |
| UtenteManager.java | Classe di gestione dei dati di un utente generico |
| ClienteManager.java | Classe di gestione dei dati di un cliente |
| ProdottiManager.java | Classe di gestione dei dati dei prodotti |
| CarrelloManager.java | Classe di gestione dei dati del carrello |

# 4.Class Interfaces

### 4.1 Prodotti Manager

| Nome Classe | ProductManager |
|---|---|
| **Descrizione** | Classe che gestisce alcune funzionalità inerenti ai dati dei prodotti |
| **Pre-codizioni** | **context** ProductManager:: doRetriveProdByName(name):<br>**pre: name! = null**<br><br>**context** ProductManager:: doRetrieveAll()<br>**pre: prodotto! = null**<br><br>**context** productManager:: doSave(name, prezzo, descrizione, imgLink)<br>**pre: name! = null && prezzo!= null && descrizione!= null && imgLink!= null**<br><br>**context** productManager:: doUpdate(name, prezzo, descrizione, imgLink)<br>**pre: name! = null && prezzo!= null && descrizione!= null && imgLink!= null**<br><br>**context** productManager:: doUpdate(product)<br>**pre: product! = null**<br><br>**context** productManager:: checkProduct(product)<br>**pre: product! = null** |
| **Post-condizioni** | **context** productManager:: doSave(name, prezzo, descrizione, imgLink)<br>**post:** doRetriveProdByName(name)== product |
| **Invarianti** | |

### 4.2 Carrello Manager

| Nome Classe | CarrelloManager |
|---|---|
| **Descrizione** | Classe che gestisce alcune funzionalità inerente al carrello |
| **Pre-codizioni** | **Context** CarrelloManager:: doSave(user, id_Product, quantita)<br>**Pre:** user! = null && id_Product! = null && quantita!= null<br><br>Context CarrelloManager:: doDelete(client)<br>Pre: client!= null<br><br>**Context** CarrelloManager:: doUpdate( id, quantita, client)<br>**Pre**: id!= null && quantita!= null && client!= null<br><br>**Context** CarrelloManager:: checkOut(name_cliente, name_product)<br>Pre: name_cliente!= null && name_product!= null<br><br>**Context** CarrelloManager checkIban(client.getIban())<br>**Pre** iban!= null<br><br>**Context** CarrelloManager:: doRetrieveAll()<br>**Pre**: product!= null |
| **Post-condizioni** | **Context** CarrelloManager checkOut(carrello, name_client)<br>**Pre:** carrello!= null && name_client!= null |
| **Invarianti** | |

### 4.3 Admin Manager

| Nome Classe | AdminManager |
|---|---|
| **Descrizione** | Classe che gestisce alcune funzionalità inerente all'Admin Manager |
| **Pre-codizioni** | **context** AdminManager:: doRetrieveByKey(key)<br>**pre**: key! =null<br><br>**context** AdminManager::checkUser(user)<br>**pre**: user!=null<br><br>context AdminManager::login(mail,password)<br>**pre**: mail!=null && password!=null<br><br>**context** AdminManager::checkAccount(account)<br>**pre**: account! =null<br><br>**context** AdminManager:: isWellFormatted(account):<br>**pre**: account! =null<br><br>**context:** AdminManager :: isAdmin(account)<br>**pre**: account != null |
| **Post-condizioni** | **context:** AdminManager :: isAdmin(account)<br>**post**: doRetrievrByKey(key).isAdmin()==true |
| **Invarianti** | |

### 4.4 Cliente Manager

| Nome Classe | ClienteManager |
|---|---|
| **Descrizione** | Classe che gestisce alcune funzionalità inerente all'Admin Manager |
| **Pre-codizioni** | **context** ClienteManager:: doRetrieveByKey(key)<br>**pre**: key! =null<br><br>**context** ClienteManager::checkUser(user)<br>**pre**: user!=null<br><br>context ClienteManager::login(mail,password)<br>**pre**: mail!=null && password!=null<br><br>**context** ClienteManager::checkAccount(account)<br>**pre**: account! =null |
| **Post-condizioni** | **context**: ClienteManager:: isAdmin(account) |
| **Invarianti** | |

### 4.5 Utente Manager

| Nome Classe | UtenteManager |
|---|---|
| **Descrizione** | Classe che gestisce alcune funzionalità inerente all'Admin Manager |
| **Pre-codizioni** | **context** UtenteManager:: doRetrieveByKey(key)<br>**pre**: key! =null<br><br>**context** UtenteManager :: modificaPassword(mail,password)<br>**pre**: mail!=null && password!=null && checkMail(mail)<br><br>context UtenteManager:: createAdmin(id, password, email)<br>**pre**: id!= null && password!= null && email!= null<br><br>**context:** UtenteManager :: isAdmin(account)<br>**pre**: account != null |
| **Post-condizioni** | **context:** UtenteManager:: isAdmin(account)<br>**post**: doRetrievrByKey(key).isAdmin()==false<br><br>**context** UtenteManager::modificaPassword(mail,password):<br>post: doRetrieveByKey(mail).getPassword() == password |
| **Invarianti** | |