

ODD Object Design Document

|  |  |
| --- | --- |
| Versione | 1.0 |
| Data | 15/12/2019 |
| Destinatario | Prof. Andrea De Lucia |
| Presentato da | Riccardo Martiniello  Davide Cresci  Alessio Rizzolo  Giuseppe Caiazzo |

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| data | versione | descrizione | autori |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Sommario

[1. Introduzione 4](#_Toc27571620)

[1.1.1 Object Design Trade-off 4](#_Toc27571621)

[1.2 Linee guida per l’implementazione 5](#_Toc27571622)

[1.2.1 Naming Convention 5](#_Toc27571623)

[1.2.2 Basi di dati 7](#_Toc27571624)

[1.2.3 Pagine HTML 8](#_Toc27571625)

[1.2.4 Gestione codici Servlet 9](#_Toc27571626)

[1.2.5 Gestione JavaScript 9](#_Toc27571627)

[1.2.6 CSS 10](#_Toc27571628)

[1.3 Riferimenti 10](#_Toc27571629)

# 1. Introduzione

Il presente documento illustra l’Object Design per l’E-Commerce Dress-Store. Questo documento, usato come supporto dell’implementazione, ha lo scopo di produrre un modello capace di integrare in modo coerente e preciso tutti i servizi individuati 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, nei paragrafi successivi sono specificati i trade-off, le linee guida e i design pattern per l’implementazione.

## 1.1.1 Object Design Trade-off

Nella fase dell’Object Design sorgono diversi compromessi di progettazione ed è necessario stabilire quali punti rispettare e quali rendere opzionali.

Per quanto riguarda la realizzazione dell’E-Commerce Dress-Store, sono stati individuati i seguenti trade-off:

* **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.
* **Interfaccia vs Easy-use**: l’interfaccia dovrà presentarsi semplice ed intuitiva per garantire il successo del sistema. Infatti, l’effettivo utilizzo del sistema sarà determinato dall’utilità e le semplificazioni che saranno concretamente apportate agli utenti Potrà essere richiesta una penalizzazione in termini di interfaccia per permettere il raggiungimento delle varie funzionalità in pochi passi.
* **Efficienza vs Memoria:** dal momento che il sistema coinvolge una grande quantità di dati, si è scelto di preferire la memoria a discapito della efficienza, eliminando ridondanze dalla base di dati.
* **Sicurezza vs Prestazioni**: nel nostro sistema ogni richiesta del client viene validata attraverso l’uso di sessioni ed un controllo del livello di utenza. Inoltre, si intende utilizzare una libreria per la cifratura della password, in modo tale da poterla inserire nell’URL inviato per e-mail, permettendo quindi l’inserimento dell’utente nel database solo dopo che siano stati validati i dati

Questo trade-off resta comunque piuttosto bilanciato: le caratteristiche descritte potrebbero far aumentare il tempo di risposta del sistema, tuttavia tale aumento è completamente trascurabile.

## 1.2 Linee guida per l’implementazione

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

### 1.2.1 Naming Convention

I nomi utilizzati per la rappresentazione dei concetti principali, delle funzionalità e delle componenti generiche del sistema devono rispettare le seguenti condizioni

1. I **nomi** devono essere:

* Appartenenti alla lingua italiana, se possibile
* Di lunghezza medio-breve
* Non sostituiti da acronimi o abbreviazioni di alcun genere

1. Le **variabili** devono:

* Rispettare la Camel Notation
* Iniziare con la lettera minuscola
* Tutte le variabili dovrebbero essere private
* Tutte le variabili che non vengono mai modificate dovrebbero essere dichiarate come costanti (final).

È possibile far iniziare le variabili statiche con “\_” cosi da contraddistinguerle dal resto delle variabili presenti all’interno del codice. Questa distinzione è utile per evitare errori sull’uso improprio di tali forme di variabili.

1. Le **classi** e le **interfacce** devono:

* Rispettare la Camel Notation
* Iniziare con la lettera maiuscola
* Concludersi con il tipo di elemento che rappresentano:

Esempio: public class ProvaBean {} /\* Per una classe di tipo Bean public class ProvaClass {} /\* Per una classe generica \*/

public interface ProvaInterface /\* Per un’interfaccia \*/

1. Le **variabili costanti** devono:

* Utilizzare solamente caratteri maiuscoli
* Separare i vari nomi che la compongono da “\_”
* Evitare di iniziare con “\_”
* Essere di lunghezza medio-breve

1. I **pacchetti** devono:

* Contenere solamente caratteri minuscoli
* Contenere solamente caratteri a-z
* Di semantica affine con gli elementi di cui è composto

1. I **metodi** devono:

* Iniziare con lettere minuscole
* Evitare di iniziare con GET o SET se non si trattano di metodi setting o getting della classe corrispondente.
* Rispettare la Camel Notation

1. Le pagine **JSP** devono, quando eseguite, produrre, in ogni circostanza, un documento conforme allo standard HTML versione 5. Le parti Java delle pagine devono aderire alle convenzioni per la codifica in Java, con le seguenti puntualizzazioni:

* Il tag di apertura (<%) è seguito immediatamente dalla fine della riga
* Il tag di chiusura (%>) si trova all'inizio di una riga
* Contengono solamente caratteri minuscoli
* Sono di lunghezza medio-breve
* Hanno nomi composte da una singola parola in inglese o in italiano

### 1.2.2 Basi di dati

Le tabelle della base di dati dovrebbero rispettare la terza forma normale di Codd (3NF). Ove ciò non si verifichi, tale fatto deve essere riportato e motivato nella documentazione della base di dati. Le scelte di gestione nel trattamento dell'integrità referenziale devono essere riportate e motivate nella documentazione della base di dati.

### 1.2.3 Pagine HTML

Le pagine HTML, statiche e dinamiche, devono essere totalmente aderenti allo standard HTML versione 5. Inoltre, il codice HTML statico deve utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

* Un'indentazione consiste in una tabulazione
* Ogni tag deve avere un'indentazione maggiore del tag che lo contiene
* Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura
* I tag di commento devono seguire le stesse regole che si applicano ai tag normali

<html>

<head>

</head>

<body>

<div>

Contenuto DIV

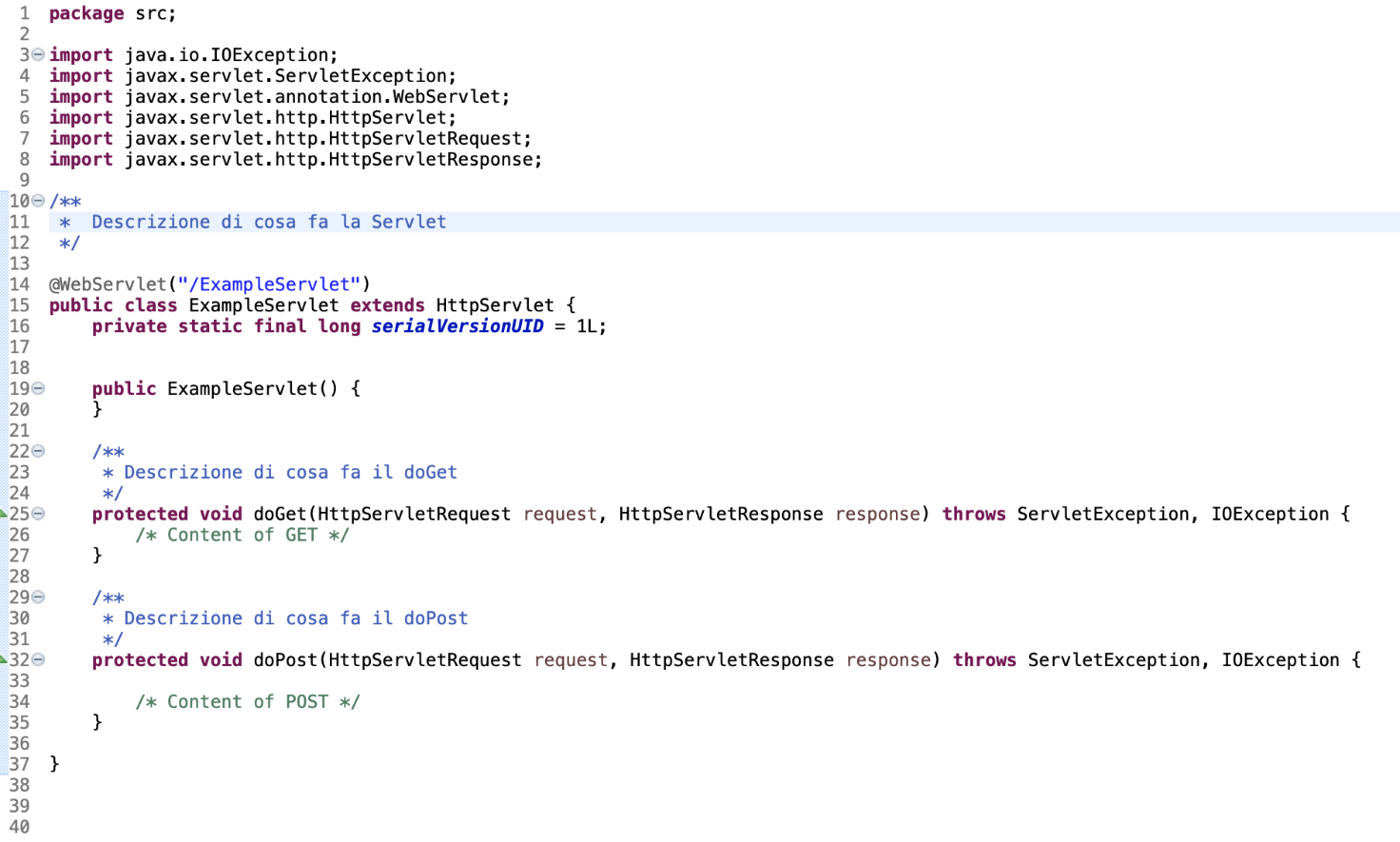
</div>

</body>

</html>

## 1.2.4 Gestione codici Servlet

Le Servlet devono seguire il seguente formato:



## 1.2.5 Gestione JavaScript

Gli script JavaScript devono essere collocati in file dedicati.

Il codice JavaScript deve seguire le stesse convenzioni adottate per il linguaggio Java.

Gli scipt devono attenersi al seguente formato:

/\*\*

\* Descrizione breve

\* Eventuale ulteriore descrizione

\* Specifica degli argomenti del costruttore (@param)

\*

\* Metodo nomeMetodo1

\* Descrizione breve

\* Eventuale ulteriore descrizione

\* Specifica degli argomenti (@param)

\* Specifica dei risultati (@return)

\*

\* Metodo nomeMetodo2

\* Descrizione breve

\* Eventuale ulteriore descrizione

\* Specifica degli argomenti (@param)

\* Specifica dei risultati (@return)

\*

\* ...

\*/

function ClasseX(a, b, c) {

# 1.2.6 CSS

Tutti gli stili devono essere collocati in file separati.

Ogni regola CSS deve essere formattata come segue:

1. L’ultimo selettore della regola è seguito da parentesi graffa aperta “{“;
2. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
3. La regola termina con la parentesi graffa chiusa “}”

# 

# 1.3 Riferimenti

Il presente documento, fa riferimento ai documenti precedentemente rilasciati:

* RAD;
* SDD.

Inoltre, si fa riferimento anche al libro di testo. “B.Bruegge, A.H. Dutoit, Object Oriented Software Engineering – Using UML, Patterns and Java, Prentice Hall.”