



**Università Degli Studi Di Salerno**

Progetto di Ingegneria del software 2018/2019

Object Design Document

Immagine che contiene oggetto

Descrizione generata automaticamenteSommario

[1. Introduzione 2](#_Toc536194122)

[1.1 Object Design Trade-off 2](#_Toc536194123)

[1.2 Linee Guida per la Documentazione delle Interfacce 3](#_Toc536194124)

[1.3 Definizioni, acronimi e abbreviazioni 7](#_Toc536194125)

[1.4 Riferimenti 7](#_Toc536194126)

[2. Packages 7](#_Toc536194127)

# 1. Introduzione

## 1.1 Object Design Trade-off

Dopo la realizzazione dei documenti RAD e SDD abbiamo descritto in linea di massima quello che sarà il nostro sistema e quindi i nostri obiettivi, 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 la signature dei sottosistemi definiti nel System Design. Inoltre, sono specificati i trade-off e le linee guida.

**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. Ovviamente questa caratteristica aggiungerà un incremento di tempo allo sviluppo del nostro progetto.

**Prestazioni vs Costi:**

Essendo il progetto sprovvisto di budget, al fine di mantenere prestazioni elevate, per alcune funzionalità verranno utilizzati dei template open source esterni in particolare Bootstrap.

**Interfaccia vs Usabilità:**

L’interfaccia grafica è stata realizzata in modo da essere molto semplice, chiara e concisa, fa uso di form e pulsanti disposti in maniera da rendere semplice l’utilizzo del sistema da parte dell’utente finale.

**Sicurezza vs Efficienza:**

La sicurezza, come descritto nei requisiti non funzionali del RAD, rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

## 1.2 Linee Guida per la Documentazione delle Interfacce

Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice:

**Naming Convention**

● È buona norma utilizzare nomi:

1. Descrittivi

2. Pronunciabili

3. Di uso comune

4. Lunghezza medio-corta

5. Non abbreviati

6. Evitando la notazione ungherese

7. Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

**Variabili:**

● I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Quest'ultime devono essere dichiarate ad inizio blocco, solamente una per riga e devono essere tutte allineate per facilitare la leggibilità.

Esempio: elaboratoSessioneId

● È inoltre possibile, in alcuni casi, utilizzare il carattere **underscore** “**\_**”, ad esempio quando utilizziamo delle variabili costanti oppure quando vengono utilizzate delle proprietà statiche.

Esempio: CREATE\_ARGOMENTO

**Metodi:**

● I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica una azione, seguito dal nome di un oggetto.

I nomi dei metodi per l’accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Le variabili dei metodi devono essere dichiarate appena prima del loro utilizzo e devono servire per un solo scopo, per facilitare la leggibilità. Esistono però casi particolari come ad esempio nell’implementazione dei model, dove viene utilizzata l’interfaccia CRUD.

Esempio: getId(), setId()

● I commenti dei metodi devono essere raggruppati in base alla loro funzionalità, la descrizione dei metodi deve apparire prima di ogni dichiarazione di metodo, e deve descriverne lo scopo. Deve includere anche informazioni sugli argomenti, sul valore di ritorno, e se applicabile, sulle eccezioni.

**Classi e pagine :**

* I nomi delle classi e delle pagine devono cominciare con una lettera maiuscola, e anche le parole seguenti all’interno del nome devono cominciare con una lettera maiuscola. I nomi di quest’ultime devono fornire informazioni sul loro scopo.

Ogni file sorgente java contiene una singola classe e dev’essere strutturato in un determinato modo:

● Una breve introduzione alla classe

L’introduzione indica: l’autore, la versione e la data.

/\*\*

\* sommario dello scopo della classe.

\*

\* @author nome dell’autore

\* @version numero di versione della classe

\* @since data d’implementazione

\*/

● L’istruzione include che permette di importare all’interno della classe gli altri oggetti che la classe utilizza.

● La dichiarazione di classe 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.

|  |
| --- |
| ESEMPIO: class CdL {  private $matricola;  private $nome;  /\*\*  Costruttore di CdL  @param string $matricola La matricola del CdL  @param string $nome Il nome del CdL  \*/  public function \_\_construct($matricola, $nome) {  $this->matricola = $matricola;  $this->nome = $nome; }  /\*\*  \* @return string La matricola del CdL |

|  |
| --- |
| \*/  public function getMatricola() {  return $this->matricola; }  /\*\*  \* @return string Il nome del CdL  \*/  public function getNome() {  return $this->nome; }  /\*\*  \* Setta la matricola del CdL  \* @param string $matricola La matricola del CdL  \*/  public function setMatricola($matricola) {  $this->matricola = $matricola; }  /\*\*  \* Setta il nome del CdL  \* @param string $nome Il nome del CdL  \*/  public function setNome($nome) {  $this->nome = $nome; }  } |

**Pagine HTML**

Le pagine HTML sono dei tipi di documenti digitali tramite i quali sono rese disponibili all'utente finale le informazioni del World Wide Web tramite un web browser. L’insieme di pagine contenute all’interno del sito Sandwich on web dovranno essere tra loro relazionate secondo una gerarchica e una struttura ipertestuale e saranno riferibili al web server locale.

Inoltre, le pagine del sito possono essere statiche o dinamico a seconda della presenza del database, le pagine statiche dovranno utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

* Un'indentazione consiste in 4 spazi;
* Ogni tag deve avere un'indentazione maggiore o uguale del tag che lo contiene;
* Non si dovrebbero superare gli 8 livelli di indentazione;
* Qualora si superassero gli 8 livelli, è possibile azzerare il livello di indentazione, portandolo a 0 anziché incrementarlo;
* Sebbene l'uso dell'indentazione sia arbitrario, 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;
* Gli script e i fogli di stile incorporati, se non si completano in una sola riga di testo, devono partire dal livello 0 di indentazione.
* I fogli di stile e gli script incorporati nelle pagine devono rispettare le stesse linee guida che si applicano ai documenti dedicati. Le sezioni dinamiche (prodotte tramite l'oggetto out) dei documenti HTML generati da pagine JSP dovrebbero rispettare le stesse convenzioni dello HTML statico.

**Script Javascript**

Gli script che svolgono funzioni distinte dal mero rendering della pagina dovrebbero

essere collocati in file dedicati.

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

PHP.

I documenti Javascript devono essere iniziati da un commento analogo a quello

presente nei file PHP.

Le funzioni Javascript devono essere documentate in modo analogo ai metodi PHP,

scritte possibilmente un file esterno, per consentire maggiore leggibilità al codice.

Gli oggetti Javascript devono essere preceduti da un commento in stile PHPdoc, che

segue il 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) {

\*

\*

\*

}

**Fogli di stile CSS**

Tutti gli stili non inline devono essere collocati in fogli di stile separati.

Ogni regola CSS deve essere formattata come segue:

● I selettori della regola si trovano a livello 0 di indentazione, uno per riga;

● L'ultimo selettore della regola è seguito da parentesi graffa aperta ({);

● Le proprietà che costituiscono la regola sono listate una per riga e sono indentate

rispetto ai selettori, e per identificare la fine di un’istruzione si usa il punto e

virgola;

● La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una

riga;

Le proprietà e le regole poco chiare dovrebbero essere precedute da un commento

esplicativo.

Le regole possono essere divise in blocchi concettualmente legati, preceduti da

commenti in stile PHPdoc, che ne spiegano lo scopo, e seguiti da 2 righe bianche.

## 1.3 Definizioni, acronimi e abbreviazioni

**Acronimi:**

● RAD: Requirements Analysis Document

● SDD: System Design Document

● ODD: Object Design Document

● CRUD: Create Read Update Delete

**Abbreviazioni:**

● DB: DataBase

## 1.4 Riferimenti

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

● Documento SDD del progetto GamesHub

● Documento RAD del progetto GamesHub

# 2. Packages

# 

La gestione del nostro sistema è suddivisa in tre livelli ( three-tier ):

● Interface layer

● Application Logic layer

● Storage layer

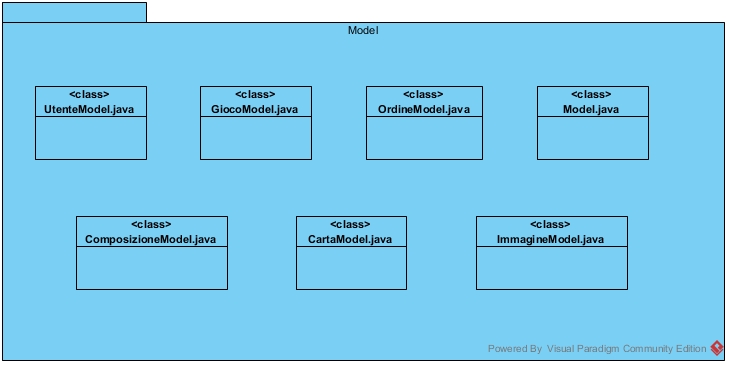
Il package GamesHub contiene sotto package che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

|  |  |  |
| --- | --- | --- |
| Interface layer | |  | | --- | | Rappresenta l’interfaccia del sistema, ed offre la possibilità all’utente di interagire con quest’ultimo, offrendo sia la possibilità di inviare, in input, che di visualizzare, in output, dati. | |
| |  | | --- | | Application Logic layer | | |  | | --- | | Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite lo Storage Layer, accede ai dati persistenti.  Si occupa di varie gestioni quali:   1. **Gestione Utente** 2. **Gestione Carrello** 3. **Gestione Ordini** 4. **Gestione Catalogo** | |
| |  | | --- | | Storage layer | | |  | | --- | | Ha il compito di memorizzare i dati sensibili del sistema, utilizzando un DBMS, inoltre riceve le varie richieste dall’ Application Logic layer inoltrandole al DBMS e restituendo i dati richiesti. | |

### 2.1.1 Package bean

|  |  |
| --- | --- |
| Classe: | Descrizione: |
| Utente.java | Descrive un utente registrato al sistema |
| Gioco.java | Descrive un gioco memorizzato nel sistema |
| Ordine.java | Descrive un ordine effettuato |
| Immagine.java | Descrive un immagine relativa ad un gioco |
| Composizione.java | Descrive la composizione di un ordine effettuato |
| Carta.java | Descrive una carta utilizzata per gli acquisti |
| Carrello.java | Descrive un carrello di un utente |

### 2.1.4 Package model



|  |  |
| --- | --- |
| Classe: | Descrizione: |
| UtenteModel.java | Il model che effettua tutte le query riguardanti le funzionalità degli account, interfacciandosi al db al quale è connesso |
| GiocoModel.java | Il model che effettua tutte le query riguardanti  le funzionalità dei giochi, interfacciandosi al db al quale è connesso |
| OrdineModel.java | Il model che effettua tutte le query riguardanti le funzionalità degli ordini, interfacciandosi al db al quale è connesso |
| ImmagineModel.java | Il model che effettua tutte le query riguardanti le funzionalità delle immagini di un gioco, interfacciandosi al db al quale è connesso |
| ComposizioneModel.java | Il model che effettua tutte le query riguardanti le funzionalità delle composizioni degli ordini, interfacciandosi al db al quale è connesso |
| CartaModel.java | Il model che effettua tutte le query riguardanti le funzionalità delle carte utilizzate per i pagamenti, interfacciandosi al db al quale è connesso |
| Model.java | Il model che effettua la connessione al database |