Università degli Studi di Salerno

Corso di Ingegneria Del Software

Eat & Reorder

Object Design

Document

**Docente:**

Prof. Andrea De Lucia

**Studenti:**

Abate Francesco

Dello Buono Marco

De Martino Vincenzo

Gagliardi Rosario

Sommario

[1. Introduzione 3](#_Toc25831082)

[1.1. Trade-offs 5](#_Toc25831093)

1. Introduzione

Dopo aver stilato il documento di Requirements Analysis e il documento di System Design, occorre concentrarsi sugli aspetti implementativi. L’Object Design Document (ODD) ha come obiettivo quello di produrre un modello che sia in grado di unire tutte le informazioni accumulate nelle fasi precedenti. In tale documento verranno definite le interfacce delle classi, le operazioni supportate, i tipi dei dati, i parametri delle procedure, i signatures dei sottosistemi definiti nel documento di System Design, i trade-offs e le linee guida, per evitare compromessi di progettazione.

1.1. Trade-offs

**Prestazioni VS Costi**

Considerando il sistema che stiamo realizzando, possiamo dire che il non eccessivo budget a nostra disposizione ci ha consentito di realizzare il prodotto utilizzando materiali reperibili in maniera gratuita, come un server open source e un ambiente di sviluppo,partendo da zero minimizzando così i costi e rendendo il sistema più che soddisfacente.

**Perfomance VS Usabilità**

L’interfaccia, grazie all’utilizzo delle form e di un cammino semplice e intuitivo, permette un uso facilitato del sistema per tutti i tipi di utenti all’interno della nostra piattaforma. E’ stata preferita l’usabilità per rendere l’esperienza d’utilizzo il più agevole possibile rendendo l’interfaccia chiara e pulita. Il sistema deve garantire tempi di risposta brevi per ogni funzionalità.

**Interfaccia VS Tempo di risposta**

Il tempo di risposta tra server e interfaccia sono rapidi per soddisfare le esigenze di tutti i tipi di utenti collegati al sistema, negli input degli utenti verrà fatto un controllo più meticoloso dei dati andando a specificare i parametri errati di caso di insuccesso.

## **1.2. Linee guida per la Documentazione delle Interfacce**

Per l’implementazione del sistema saranno utilizzate le seguenti convenzioni.

**Naming Convention:**

È buona norma utilizzare nomi:

* Descrittivi
* Pronunciabili
* Di uso comune
* Di lunghezza medio-corta
* Non abbreviati
* Evitando la notazione ungherese
* Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

**Variabili:**

* I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. 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 iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un’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(). 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()
* Ai metodi va aggiunta una descrizione, la quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo. La descrizione del metodo deve includere anche informazioni riguardanti gli argomenti, il valore di ritorno, le eccezioni. I metodi devono essere raggruppati in base alla loro funzionalità.

**Classi e pagine:**

* I nomi delle classi e delle pagine devono iniziare con la lettera maiuscola, e anche le parole successive all’interno del nome devono iniziare con la lettera maiuscola. I nomi delle classi e delle pagine devono essere evocativi, in modo da fornire informazioni sullo scopo di quest’ultime. Ogni file sorgente .java contiene una singola classe e dev’essere strutturato in un determinato modo:
* Una breve introduzione alla classe che ne specifichi lo scopo

/\*\*

\* sommario dello scopo della classe.

\*

\*

\*

\*/

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

## **1.3. Definizioni, acronimi e abbreviazioni**

**Acronimi:**

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

**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 Eat & Reorder
* Documento RAD Eat & Reorder

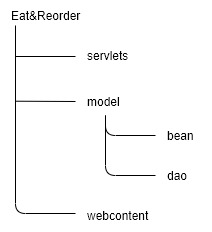
## **1.5. Design Pattern**

Nel sistema Eat&Reorder vengono utilizzati alcuni pattern, quali sono l’Adapter, Data Access Object.

\*\*\*SPIEGARE ADAPTER E DAO\*\*\*

## **2. Packages**

\*\*\*DESCRIZIONE DEI PACKAGES\*\*\*



\*\*\*Per ogni package bisogna specificare cosa c’è dentro\*\*\*

## **3. Interfacce delle classi**

Utilizzare il preset pubblicato in:

…\GitHub\Eat-Reorder\Documents\Temp - Documentazione temporanea\Eat&Reorder - Template condizioni

## **4. Class Diagram**