UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

INGEGNERIA DEL SOFTWARE 2017-2018

Object Design Document versione 2.0

Easybook

Data: 14/01/18

Sommario

1. Introduzione

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.

2. Compromessi dell'object design

- **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.
- **Prestazioni vs Costi**: Essendo il progetto sprovvisto di budget, al fine di mantenere prestazioni elevate, per alcune funzionalità verranno utilizzate delle librerie esterne open source.
- **Interfaccia vs Usabilità**: L'interfaccia grafica fa uso di form e bottoni disposti in maniera da rendere semplice l'utilizzo del sistema da parte dell'utente finale.
- **Sicurezza vs Efficienza**: L'autenticazione di un utente nel sistema avviene fornendo numero di tessera e password. Ogni utente può accedere solo alle funzionalità a cui il suo tipo di account è autorizzato.

3. Linee guida per l'implementazione

3.1 Nomi di file

I file java utilizzano i seguenti suffissi:

- .class: per il bytecode
- .java: per i sorgenti

3.2 Organizzazione dei file

Un file consiste di sezioni che dovrebbero essere separate da linee bianche e un commento opzionale che identifica ogni sezione. File più lunghi di 2000 linee sono ingombranti e devono essere evitati.

3.2.1 File sorgenti

Ogni file sorgente deve contenere una sola classe o interfaccia pubblica (come da convezione Java). All'interno dello stesso file potranno, tuttavia, essere inserite classi private associate alla classe pubblica. La classe o interfaccia pubblica deve essere la prima del file.

I file sorgenti hanno la seguente struttura:

- **Commento iniziale:** Un commento che specifichi il nome della classe e il copyright.
- **Package & Import:** Le prime linee di codice specifica in quale package il file è contenuto. Le righe successive possono contenere direttive di import per l'utilizzo di classi non contenute del package corrente.
- **Descrizione della classe:** La dichiarazione della classe è preceduta da un commento che descrive più approfonditamente le funzionalità della classe, le metodologie utilizzate, l'autore della classe e la versione. Il commento fa utilizzo delle annotazioni javadoc.
- **Dichiarazione della classe (o dell'interfaccia):** Gli elementi della classe devono essere dichiarati come segue:
    1. La prima istruzione deve essere la dichiarazione della classe (o interfaccia) pubblica.
    2. Le variabili vengono dichiarate all'inizio della classe e seguono il seguente ordine: prima le costanti (final static), poi le variabili di classe (static), in seguito le variabili di istanza (in ordine: public, protected, private).
    3. I costruttori vengono dichiarati prima dei metodi.
    4. Infine vengono dichiarati i metodi. Essi sono raggruppati in base alle funzionalità e non in base alla visibilità.

3.3 Indentazione

Come unità di indentazione viene utilizzata una tabulazione.

3.3.1 Lunghezza delle linee

La lunghezza massima di una riga è di 80 caratteri. Per la documentazione e i commenti, la lunghezza massima è di 70 caratteri.

3.3.2 Spostamento di linee

Quando un'espressione supera la lunghezza della linea, occorre spezzarla secondo i seguenti principi generali:

- Interrompere la linea dopo una virgola;
- Interrompere la linea prima di un operatore;
- Allineare la nuova linea con l'inizio dell'espressione nella linea precedente.

3.4 Commenti

All'interno del codice sorgente i commenti verranno utilizzati per due scopi: commenti di documentazione e commenti di implementazione. I commenti di implementazione verranno utilizzati per commentare una specifica metodologia di implementazione. I commenti di documentazione descriveranno il funzionamento di una classe, di un metodo o di una variabile.

3.5 Dichiarazioni

- **Numero per linea**: è preferita la dichiarazione su singola riga poiché rende in codice più leggibile e favorisce l'inserimento di commenti;
- **Inizializzazione**: tranne qualche caso particolare, le inizializzazioni di oggetti e variabili locali avverranno sulla stessa riga in cui sono dichiarati;
- **Posizione:** mettere le dichiarazioni all'inizio dei blocchi. Si preferisce non dichiarare le variabili al loro primo uso, tranne nel caso di indici da utilizzare in cicli for. Evitare le dichiarazioni locali che possono oscurare dichiarazioni a più alto livello;
- **Dichiarazioni di classe e interfaccia:** si dovrebbero rispettare le seguenti regole di formattazione:
    1. Non mettere spazi tra il nome del metodo e la parentesi"(" che apre la lista dei parametri;
    2. La parentesi graffa aperta "{" si trova alla fine della stessa linea dell'istruzione di dichiarazione;
    3. La parentesi graffa chiusa "}" inizia su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell'interfaccia.

3.6 Istruzioni

3.6.1 Istruzioni semplici

Ogni linea deve contenere al massimo una sola istruzione.

3.6.2 Istruzioni composte

Le istruzioni racchiuse all'interno di un blocco (esempio: if), devono essere indentate di un'unita all'interno dell'istruzione composta. La parentesi di apertura del blocco deve trovarsi alla fine della riga dell'istruzione composta e separata da essa da uno spazio. La parentesi di chiusura del blocco deve trovarsi allo stesso livello di indentazione dell'istruzione composta. Anche le istruzioni composte formate da un'unica istruzione devono essere racchiuse da parentesi.

3.6.3 Istruzioni return

All'interno dell'istruzione di return non deve trovarsi un'istruzione di operazione tra due o più oggetti.

3.6.4 Istruzioni if, if-else, if-else-if else

Le istruzioni if usano la seguente forma

```
if (index < 0){
    index = 0;
}

if (index + 1 < 0){

}
else if ((index + 1) == 0){

}
else {

}
```

Le istruzioni dell'if, anche se singole, devono essere racchiuse tra parentesi graffe.

3.6.5 Istruzioni for

L'istruzione for deve avere la seguente forma

```
for (int i = 0; i < 5; i++) {

}
```

In fase di inizializzazione è possibile dichiarare una variabile di iterazione.

3.6.6 Istruzioni while

L'istruzione while deve avere la seguente forma

```
int i = 0;
while (i < 5) {

}
```

3.6.9 Istruzioni try-catch-finally

L'istruzione try-catch deve avere la seguente forma

```
FileWriter fWriter = null;
try {
    fWriter = new FileWriter("file.txt");
    fWriter.write("testo");
}
catch (IOException e) {
    e.printStackTrace();
}
finally {
    try {
        if (fWriter != null){
            fWriter.close();
        }
    }
    catch (IOException e) {
        e.printStackTrace();
    }
}
```

L'istruzione finally è opzionale.

3.7 Spazi bianchi

3.7.1 Linee bianche

Due linee bianche dovrebbero essere sempre usate nelle seguenti circostanze:

- Fra sezioni di un file sorgente
- Fra definizioni di classe e interfaccia

Una linea bianca dovrebbe essere sempre usata nelle seguenti circostanze:

- Fra metodi
- Fra le variabili locali in un metodo e la sua prima istruzione
- Prima di un commento di blocco o a singola linea
- Fra sezioni logiche all'interno di un metodo

3.7.2 Spazi bianchi

Spazi bianchi dovrebbero essere usati nelle seguenti circostanze:

- Una parola chiave seguita da una parentesi dovrebbe essere separata da uno spazio;
- Uno spazio bianco dovrebbe essere interposto dopo le virgole nelle liste di argomenti;
- Tutti gli operatori binari eccetto . (operatore punto) dovrebbero essere separati dai loro operandi tramite spazi. Gli spazi bianchi non dovrebbero mai separare gli operatori unari come l'operatore meno, l'incremento e il decremento.

3.8. Convenzioni di nomi

3.8.1 Classi

I nomi di classe devono essere sostantivi, con lettere minuscole e, sia la prima lettera del nome della classe sia la prima lettera di ogni parola interna, deve essere maiuscola. Cercare di rendere i nomi delle classi semplici, descrittivi e che rispettino il dominio applicativo. Usare parole intere evitando acronimi e abbreviazioni (a meno che l'abbreviazione sia più usata della forma lunga, come URL o HTML). Non dovrebbero essere usati underscore per legare nomi. Per le Servlet è necessario far finire il nome della classe con il suffisso Servlet.

3.8.2 Interfacce

I nomi di interfaccia iniziano con la lettera I e seguono le stesse regole dei nomi di classi.

3.8.3 Metodi

I nomi dei metodi iniziano con una lettera minuscola (non sono consentiti caratteri speciali) Inoltre dovranno essere semplici, descrittivi e che rispettino il dominio applicativo.

3.8.4 Variabili

Tutte le variabili e le istanze di classe non devono iniziare con caratteri di underscore o dollaro. La scelta di un nome deve essere mnemonica e deve essere coerente con il dominio applicativo. I nomi di variabili di un solo carattere dovrebbero essere evitati, tranne in alcuni casi.

3.8.5 Costanti

I nomi delle variabili dichiarate come costanti di classe devono essere scritte in lettere tutte maiuscole con le parole separate da underscore. La scelta di un nome deve essere mnemonica e deve rispettare il dominio applicativo.

3.9 Consuetudini di programmazione

3.9.1 Fornire accesso a variabili di istanza o di classe

È possibile accedere alle variabili di istanza solo tramite i metodi forniti dalla classe. Si potrà accedere in maniera diretta solo alle costanti di classe.

3.9.2 Riferire variabili e metodi di classe

Evitare di usare un oggetto per accedere a variabili o metodi di classe static. Usare invece il nome della classe.

3.9.3 Assegnamento di variabili

Evitare di assegnare a più variabili lo stesso valore in una sola istruzione. Non usare l'operatore di assegnamento in un punto in cui può essere facilmente confuso con l'operatore di uguaglianza. Non usare assegnamenti innestati nel tentativo di migliorare le prestazioni a tempo di esecuzione.

3.9.4 Parentesi

E' generalmente una buona idea usare le parentesi liberamente in espressioni che coinvolgono operatori misti per evitare problemi di precedenza degli operatori.

3.9.5 Valori ritornati

Provare a rendere la struttura del programma aderente alle proprie intenzioni.

4. Definizioni, acronimi e abbreviazioni

Acronimi e abbreviazioni

Acronimo	Definizione
DB	Database (sistema di memorizzazione per l'archiviazione dei dati permanenti)
DBMS	Database Management System (gestore del sistema di memorizzazione)
HTML	HyperText Markup Language (linguaggio di markup utilizzato per la definizione della struttura della pagina web)
HTTP	HyperTextTransferProtocol (Protocollo per la gestione di richieste e risposte scambiate tra client e server)
JAVA	Linguaggio orientato agli oggetti
JAVASCRIPT	Linguaggio di scripting utilizzato lato client per rendere il portale web dinamico
Layout	Impaginazione struttura grafica del portale e dell'applicazione
MYSQL	DBMS utilizzato

SQL	Structured Query Language (linguaggio per l'invocazione delle richieste al database)
Query	Interrogazione al database
RAD	Requirementes Analysis Document
SDD	System Design Document

Definizioni
Database: sistema che memorizza i dati.
Utente: è un utente generico che ha accesso al sito, e può essere "registrato" o "non registrato".
Utente registrato: è un utente che può prendere in prestito libri dal catalogo.
Utente non registrato: può solamente visualizzare informazioni, ma non può effettuare prestiti.
Utente loggato: utente che ha effettuato il login.
Utente non loggato: utente che non ha effettuato il login.
Amministratore: è il bibliotecario. Può modificare il catalogo e gestire i prestiti e gli utenti.
Amministratore loggato: amministratore che ha effettuato il login.
Form: è una "scheda di compilazione" per l'inserimento dei dati, dove l'utente inserisce e invia i dati al server.
Login: è la procedura di accesso (procedura di autenticazione) ad un sistema informatico o ad un'applicazione informatica con le credenziali registrate al sito web.
Logout: è la procedura di uscita da un sistema informatico o un'applicazione informatica.
Password: è una parola chiave con cui l'utente accede al sito web.
Registrazione: è la procedura di salvataggio dei dati dell'utente nel database con i quali può procedere agli acquisti.
Carrello: oggetto contente i libri che l'utente vuole prendere in prestito.
Catalogo: è l'insieme dei libri disponibili.
Package: collezione di classi e interfacce correlate.
Classe: è identificabile come un tipo di dato astratto.
Interfaccia: è un tipo analogo alla classe, ma vincolato a non definire l'implementazione dei propri metodi.
Metodo: è un sottoprogramma associato in modo esclusivo ad una classe e che rappresenta.
Variabile: è un insieme di dati modificabili situati in una porzione di memoria destinata a contenere dei dati **.**
Costante: identifica una porzione di memoria il cui valore non varia nel corso dell'esecuzione di un programma.

5. Riferimenti

Per realizzare il progetto, sono stati utilizzati:

- slide del docente;
- B. BRUEGGE, A.H. DUTOIT, OBJECT ORIENTED SOFTWARE ENGINEERING – USING UML, PATTERNS AND JAVA, PRENTICE HALL, 3D EDITION, 2009;
- RAD e SDD di Easybook.

6. Packages

Individuiamo 5 packages in cui dividere il codice, uno per ogni sotto-sistema:
-carrelloPackage
-libriPackage
-utentiPackage
-prestitoPackage

Il package carrelloPackage contiene i file “AggiungiLibroCarrelloServlet”, “CarrelloBean.java”,“RimuoviLibroCarrelloServlet.java”, responsabili dell’interazione con il carrello.Ha dipendenze con libriPackage.

Il package libriPackage contiene LibroBean.java, LibriManager.java , AggiungiLibroCatalogoServlet.java,RimuoviLibroCatalogoServlet.java,VisualizzaCatalogoServlet.java, Esso contiene tutte quelle classi .java responsabili delle operazioni riguardanti i libri. Ha dipendenze con carrelloPackage.

Il package utentiPackage contiene i file UtenteBean.java, UtentiManager.java, LoginServlet.java,LogoutServlet.java,RegistraServlet.java,RimuoviUtenteServlet.java.
Esso contiene tutte quelle classi responsabili di quelle operazioni riguardanti gli utenti e l’amministratore.

Il package prestitiPackage contiene le classi VisualizzaPrestitiServlet.java, PrestitiManager.java, InvioPrestitoServlet.java,ChiusuraPrestitoServlet.java.
Esso contiene le classi responsabili di gestire i prestiti.
Ha dipendenze con libriPackage e carrelloPackage.

7. Class interfaces

Per questa sezione, si fa riferimento al JAVADOC di Easybook.

8. Team

Nome	**Matricola**
Andrea Di Lucia	0512103222

9. Storia delle versioni

DATA	VERSIONE	DESCRIZIONE	AUTORE
14/12/2017	1.0	Completamento ODD	Andrea Di Lucia
14/01/2018	2.0	Revisione finale	Andrea Di Lucia