Corso di Ingegneria del Software

OBJECT DESIGN DOCUMENT

CINEFORUM

Anno accademico: 2020/2021

Versione 1.0

Scritto da:	Andreozzi Giuseppe, Marcantuono Vincenzo, Vignola Michele

Partecipanti

Nome	Matricola
Andreozzi Giuseppe	0512105790
Marcantuono Vincenzo	0512105886
Vignola Michele	0512105748

Revision History

Data	Versione	Descrizione	Autore
17/12/2020	0.3	Aggiunta sezione 3 “Interfacce delle classi”	Marcantuono Vincenzo
30/12/2020	0.6	Aggiunta sezione 1 “Introduzione”	Vignola Michele
30/12/2020	0.9	Aggiunta sezione 2 “Packages”	Andreozzi Giuseppe
30/12/2020	1.0	Revisione generale	Tutti

Sommario

1. Introduzione

1.1 Trade offs

Compatibilità vs costi

Si preferisce aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte nel progetto o le persone coinvolte che non hanno lavorato a quella parte in particolare. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorare la comprensibilità agevola la manutenibilità e anche il processo di modifica.

Costi vs mantenimento

Il sistema può essere facilmente modificato ed implementato con nuove funzioni e corretto in presenza di errori, grazie all'uso di materiale open source e con l'utilizzo di javadoc.

Interfacce vs Easy-use

L'interfaccia, grazie ad una impostazione semplice e intuitiva, permette facile utilizzo (Easyuse) delle principali funzionalità del sistema anche per gli utenti meno esperti.

Componenti off-the-shelf

Per il progetto software che si vuole realizzare facciamo uso di componenti off-the-shelf, che sono componenti software disponibili sul mercato per facilitare la creazione del progetto.
Per il sistema che si vuole realizzare faremo uso di un framework per la realizzazione dell'interfaccia grafica. Il framework scelto per l'interfaccia grafica è Bootstrap, un framework open source che contiene una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Questo contiene modelli di progettazione basati su HTML e CSS, per le varie componenti dell'interfaccia, come moduli, bottoni e navigazione, così come alcune estensioni opzionali di JavaScript.

1.2 Linee guida per la documentazione dell'interfaccia

Per rendere il codice più estensibile e manutenibile è opportuno sottomettere le regole di implementazione, in modo che eventuali correzioni nella logica dell'applicazione possano essere apportate prima di imbattersi nella sintassi degli strumenti scelti.

Classi e interfacce Java

La convenzione utilizzata per i nomi delle variabili è la seguente: camelCase.

Essa consiste nello scrivere parole composte in modo che ogni parola al centro della frase inizia con una lettera maiuscola.

Quando si codificano classi e interfacce Java, si dovrebbero rispettare le seguenti regole di formattazione:

1. Non si devono inserire spazi tra il nome del metodo e la parentesi tonda "(" che apre la lista dei parametri.
2. La parentesi graffa aperta "{" si deve trovare sulla stessa linea dell'istruzione di dichiarazione.
3. La parentesi graffa chiusa "}" si troverà su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell'interfaccia.

I nomi delle classi devono essere sostantivi con l'iniziale maiuscola. I nomi delle classi dovrebbero essere semplici, descrittivi e inerenti al dominio applicativo. Non devono essere usati underscore per legare nomi.

I nomi dei metodi iniziano con una lettera minuscola, non sono consentiti caratteri speciali e dovranno essere semplici, descrittivi e inerenti al dominio applicativo.

I nomi dei metodi dovranno seguire la notazione camelCase descritta sopra.

Pagine lato Server (JSP)

Le pagine JSP quando eseguite dovranno produrre un documento conforme allo standard HTML 5.

Il codice Java presente nelle JSP deve aderire alle convenzioni descritte precedentemente (regole di formattazione), con le seguenti specifiche:

1. Il tag di apertura (<%) è seguito immediatamente dalla fine della riga;
2. Il tag di chiusura (%>) si trova all'inizio di una riga;
3. È possibile evitare le due regole precedenti, se il corpo del codice Java consiste in una singola istruzione (<%= %>).

Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML 5 e il codice deve essere indentato, per facilitare la lettura, secondo le seguenti regole:

1. Un' indentazione consiste in una tabulazione;
2. Ogni tag deve avere un' indentazione maggiore del tag che lo contiene;

3. Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;

4. I tag di commento devono seguire le stesse regole che si applicano ai tag normali.

Script Javascript

Gli Script in Javascript devono rispettare le seguenti convenzioni:

1. Gli script che svolgono funzioni distinte dal rendering della pagina dovrebbero essere collocati in file dedicati.
2. Il codice Javascript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.
3. Le funzioni Javascript devono essere documentate in modo analogo ai metodi Java.

Fogli di stile CSS

I fogli di stile (CSS) devono seguire la seguente convenzione:

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

Inoltre ogni regola CSS deve essere formattata come segue:

1. I selettori della regola si devono trovare nella stessa riga;
2. L'ultimo selettore della regola è seguito da parentesi graffa aperta "{";
3. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
4. La regola è terminata da una parentesi graffa chiusa "}" collocata da sola su una riga;

Database SQL

I nomi delle tabelle devono seguire le seguenti regole:

1. Devono essere costituiti da sole lettere minuscole;
2. Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

I nomi dei campi delle tabelle devono seguire le seguenti regole:

1. Il nome composto da più termini può utilizzare underscore (_) come separatore;
2. Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

Design Pattern

MVC

Il design pattern MVC consente la suddivisione del sistema in tre blocchi principali: Model, View e Controller.

Il Model modella i dati del dominio applicativo e fornisce i metodi di accesso ai dati persistenti. Il View si occupa della presentazione dei dati all'utente e di ricevere da quest'ultimo gli input. Infine il Controller riceve i comandi dell'utente attraverso il View e modifica lo stato di quest'ultimo e del Model.

DAO (Data Access Object) Pattern

Il DAO pattern è utilizzato per il mantenimento di una rigida separazione tra le componenti del Model e il Controller in un'applicazione basata sul paradigma MVC.

Il pattern sarà composta da:

- Data Access Object Interfaccia: interfaccia che definisce le operazioni che saranno performante sugli oggetti del model.
- Data Access Object classe concreta: classe che implementa l'interfaccia descritta sopra.
- Model Object: JavaBean contenenti i metodi get/set per salvare i dati ritirati attraverso le classi DAO.

Commenti

I commenti di implementazione sono dei mezzi per commentare il codice o per commentare una particolare implementazione. I commenti dovrebbero essere usati per dare una panoramica del codice e per fornire informazioni aggiuntive che non sono prontamente disponibili nel codice stesso. I metodi devono essere preceduti da un commento, o più precisamente da una documentazione che riporti l'obiettivo che si vuole raggiungere, con il nome/i dell'autore/i. Inoltre bisogna commentare, giustificare le eventuali decisioni particolari o i calcoli.

Dichiarazioni

Posizionare le dichiarazioni all'inizio del blocco del codice. Non dichiarare le variabili al loro primo uso, può portare incomprensioni verso il programmatore e rendere la manutenibilità del codice più complessa e di difficile comprensione.

Indentazione

L'indentazione deve essere effettuata con un TAB e a prescindere dal linguaggio usato per la produzione del codice, ogni istruzione deve essere opportunamente indentata.
Esempio:

```
<html>
<head>
</head>
<body>
</body>
</html>
```

Deve essere sostituita da:

```
 <html>
        <head>

        </head>

        <body>

        </body>
</html>
```

1.3 Definizioni, acronimi e abbreviazioni

- HTML: linguaggio di mark-up per pagine web;
- CSS: linguaggio usato per definire la formattazione di pagine web;
- Framework: software di supporto allo sviluppo web;
- Javascript: linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client;
- Eclipse: ambiente di sviluppo integrato multi-linguaggio e multi-piattaforma;
- Off-The-Shelf: servizi esterni di cui viene fatto utilizzo da terzi;
- JSP(JavaServer Pages): è una tecnologia di programmazione web in Java per lo sviluppo della logica di presentazione (tipicamente secondo il pattern MVC) di applicazioni web;
- MVC: acronimo di Model-View-Controller, è un pattern architetturale molto diffuso nello sviluppo di sistemi software;
- Bootstrap: è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il web;
- AJAX: acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive;
- camelCase: è una tecnica di naming delle variabili e dei metodi adottata dal linguaggio Java;
- Servlet: le servlet sono classi scritte in linguaggio Java che operano all'interno di un server web;
- Tomcat: Apache Tomcat è un web server open source. Implementa le specifiche JavaServer Pages (JSP) e le servlet, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java.

1.4 Riferimenti

- Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, (2nd edition), Prentice-Hall, 2003;
- SDD_Cineforum.

2.Packages

Il sistema viene diviso in 3 package:

- Control, che implementa i sottosistemi: Gestione Autenticazione, Gestione Lista, Gestione Film, Gestione Utente e Gestione Sito. Inoltre contiene le servlet che elaborano i dati ricevuti dalle JSP, ottenuti dall'interazione con l'utente, manipola i dati persistenti tramite il Model e inoltra alla View adatta per la presentazione;
- Model, che implementa il sottosistema Storage e contiene i JavaBean e i DAO per accedere ai dati persistenti sul database;
- View, che implementa il sottosistema Presentation e contiene le pagine JSP che si occupano di fornire l'interfaccia utente.

2.1 Package Control

Nome	Descrizione
AdminFilter	Controlla l'accesso alle pagine riservate all'amministratore.
CheckPasswordAdminControl	Controlla che la password dell'amministratore, inserita per confermare una operazione, sia corretta.
CommentoFilmControl	Gestisce l'inserimento di un commento su un film.
ConsultaListeControl	Genera la lista di utenti, scelti casualmente, da mostrare nella pagina di "Consulta altre liste".
FilmControl	Si occupa di gestire le request o per accedere alla raccolta film, recuperando tutti i film dal database, oppure quelle per accedere alla pagina dedicata al film.
GestioneFilmControl	Gestisce le operazioni di inserimento, modifica e cancellazione film effettuate da un amministratore.
GestioneRichiesteControl	Gestisce l'operazione di valutazione delle richieste fatte da parte dell'amministratore, salvando l'esito e/o recuperando le richieste dal database.
GestioneUtentiControl	Gestisce le operazioni di cancellazione utente o rank-up effettuate da un amministratore su un account di un utente.
GetPictureControl	Si occupa di recupera l'immagine di un film per mostrarla nell'interfaccia utente.
InfoAccountControl	Gestisce le operazioni relative all'account di un utente: modificare la password e cancellare la lista.
InitDBListener	Inizializza la connessione al DB quando si avvia la web application.
ListaControl	Recupera dal database i film presenti nella lista di un utente.
ListaOperationControl	Gestisce le operazioni di inserimento, modifica e eliminazione di un film nella lista di un utente.
LogControl	Gestisce il login degli utenti.
LogoutControl	Gestisce il logout degli utenti.
RetrieveFilmListaControl	Recupera dal database i film ricercati dall'utente tramite una barra di ricerca.

RichiestaFilmControl	Gestisce l'inserimento nel database delle richieste, effettuate da un utente, di aggiungere un film nel sistema.
SearchFilmAdminControl	Recupera dal database i film ricercati dall'amministratore tramite una barra di ricerca nella pagina per la gestione dei film.
SearchFilmControl	Recupera dal database i film ricercati dall'utente tramite una barra di ricerca nella pagina della raccolta film.
SignControl	Gestisce la registrazione degli utenti.
UserFilter	Controlla l'accesso alle pagine riservate all'utente registrato.

2.2 Package Model

Nome	Descrizione
CommentoFilm	JavaBean per un commento inserito da un utente su un film.
CommentoFilmDAO	Implementazione del pattern DAO per il JavaBean CommentoFilm.
DAOInterface	Interfaccia che definisce i metodi del pattern DAO per i JavaBeans.
DAOInterfaceDoublePK	Interfaccia che estende "DAOInterface" e definisce il metodo per recuperare un singolo elemento dal database utilizzando le due primary key.
DAOInterfaceSinglePK	Interfaccia che estende "DAOInterface" e definisce il metodo per recuperare un singolo elemento dal database utilizzando l'unica primary key.
DBConnectionPool	Definisce la pool di connessioni per la web application.
Film	JavaBean per un film presente nel sistema.
FilmDAO	Implementazione del pattern DAO per il JavaBean Film.
Lista	JavaBean per un elemento della lista di un utente.
ListaDAO	Implementazione del pattern DAO per il JavaBean Lista.
RichiestaFilm	JavaBean per una richiesta di aggiunta film effettuata da un utente.
RichiestaFilmDAO	Implementazione del pattern DAO per il JavaBean RichiestaFilm.
Utente	JavaBean per un utente registrato al sistema, inclusi gli amministratori.
UtenteDAO	Implementazione del pattern DAO per il JavaBean Utente.

2.3 Package View

Nome	Descrizione
gestionefilm	Rappresenta la pagina per la gestione dei film.
gestionerichieste	Rappresenta la pagina per la gestione delle richieste di aggiunta film.
gestioneutenti	Rappresenta la pagina per la gestione degli utenti.
infoaccount	Rappresenta la pagina in cui gli utenti possono modificare le informazioni del proprio account.
richiestafilm	Rappresenta la pagina per richiedere l'aggiunta di un film nel sistema.
consultaliste	Rappresenta la pagina che mostra l'elenco degli utenti scelti casualmente per accedere alla loro lista di film.
homepage	Rappresenta l'homepage della web application.
lista	Rappresenta la pagina della lista personale di un utente.
login	Rappresenta la pagina per effettuare il login.
navbar	Rappresenta la navigation bar che viene mostrata in tutte le pagine del sistema.
paginafilm	Rappresenta la pagina dedicata ad un film.
raccoltafilm	Rappresenta la pagina che contiene la raccolta dei film presenti nel sistema.
signup	Rappresenta la pagina per effettuare la registrazione al sistema.

3. Interfacce delle classi

NOME CLASSE	UTENTE
DESCRIZIONE	**Questa classe rappresenta l'utente registrato.**
ATTRIBUTI	- Username: String - Password: String - Email: String - Ruolo :String
SIGNATURE DEI METODI	- getUsername() :String - setUsername(username :String) - getPassword() :String - setPassword(password :String) - getEmail() :String - setEmail(email :String) - getRuolo() :String - setRuolo(ruolo :String)
PRE CONDIZIONI	
POST CONDIZIONI	
INVARIANTI	

NOME CLASSE	COMMENTO FILM
DESCRIZIONE	**Questa classe rappresenta un commento rilasciato su un film.**
ATTRIBUTI	- IdCommento: int - Commento: String - Orario: String - Username: String - CodiceFilm: int
SIGNATURE DEI METODI	- getidCommento() :int - setidCommento(id :int) - getCommento () :String - setCommento(commento :String) - getOrario() :String - setOrario(orario :String) - getUsername() :String - setUsername(username :String) - getCodiceFilm() :int - setCodiceFilm(codiceFilm :int)
PRE CONDIZIONI	
POST CONDIZIONI	
INVARIANTI	

NOME CLASSE	FILM
DESCRIZIONE	**Questa classe rappresenta un film presente.**
ATTRIBUTI	- CodiceFilm: int - Titolo: String - Genere: String - DataUscita: String - Descrizione: String - Durata: short - Immagine: byte[]
SIGNATURE DEI METODI	- getCodiceFilm() :int - setCodiceFilm(codiceFilm :int) - getTitolo() :String - setTitolo(titolo :String) - getGenere() :String - setGenere(genere :String) - getDataUscita () :String - setDataUscita(dataUscita :String) - getDescrizione() :String - setDescrizione(descrizione :String) - getDurata() :short - setDurata(durata :short) - getImmagine() :byte[] - setImmagine(immagine :byte[])
PRE CONDIZIONI	
POST CONDIZIONI	
INVARIANTI	

NOME CLASSE	RICHIESTA AGGIUNTA FILM
DESCRIZIONE	**Questa classe rappresenta una richiesta per aggiungere un film.**
ATTRIBUTI	- IdRichiesta: int - Commento: String - Esito: String - Titolo: String - Username: String
SIGNATURE DEI METODI	- getidRichiesta() :int - setidRichiesta(id :int) - getCommento () :String - setCommento(commento :String) - getEsito() :String - setEsito(esito :String) - getTitolo() :String - setTitolo(titolo :String) - getUsername() :String - setUsername(username :String)
PRE CONDIZIONI	
POST CONDIZIONI	
INVARIANTI	

NOME CLASSE	LISTA
DESCRIZIONE	**Questa classe rappresenta una lista.**
ATTRIBUTI	- CodiceFilm: int - Username: String - Voto: short - Categoria: String
SIGNATURE DEI METODI	- getCodiceFilm() :int - setCodiceFilm(codiceFilm :int) - getUsername() :String - setUsername(username :String) - getVoto() :short - setVoto(voto :short) - getCategoria() :String - setCategoria(categoria :String)
PRE CONDIZIONI	
POST CONDIZIONI	
INVARIANTI	