ODD: OBJECT DESIGN DOCUMENT

Health Care

Riferimento	2022_MT_C03
Versione	0.9
Data	09/02/2023
Destinatario	Prof.ssa F. Ferrucci, Prof. F. Palomba
Presentato da	Giovanni Borrelli, Gerardo Di Muro, Agostino Andrea Mangia, Giovanni Mercurio, Luca Pastore, Angelo Zuottolo
Approvato da	Giusy Castaldo, Mariarosaria Esposito, Francesca Perillo

REVISION HISTORY

Data	Versione	Descrizione	Autori
14/12/2022	0.1	Object design trade-offs	Agostino Andrea Mangia
14/12/2022	0.2	Componenti off-the shelf	Giovanni Borrelli
14/12/2022	0.3	Design Pattern	Luca Pastore, Angelo Zuottolo
15/12/2022	0.4	Identificazione dei Packages	Giovanni Mercurio, Gerardo Di Muro
28/12/2022	0.6	Varianti, Precondizioni e Postcondizioni	Luca Pastore, Agostino Andrea Mangia
28/12/2022	0.7	Diagramma delle Classi	Giovanni Borelli, Gerardo di Muro
30/12/2022	0.8	Drafting and Revision ODD	Giovanni Mercurio, Luca Pastore
09/02/2023	0.9	Revisione Finale	Tutti

Sommario

1. Introduzione

L'Object Design Document consente di specificare in modo dettagliato le decisioni prese in fase di analisi e di design; in particolare verranno specificati i principali trade-offs, descritte le componenti off-the-shelfs utilizzate dal sistema, le linee guida per la documentazione delle interfacce e l'individuazione dei Design Patterns. Inoltre, verranno definiti i packages, le interfacce delle classi e i diagrammi delle classi che riguardano importanti decisioni implementative.

1.1 Object design trade-offs

Spazio di memoria VS Tempo di risposta.

Considerato che lo scopo del nostro sistema è quello di fornire ai MMMG uno strumento per la rapida individuazione di malattie rare la nostra priorità è quella di minimizzare il tempo di risposta anche a discapito della memoria allocata.

Comprensibilità VS Costi.

Un aspetto importante da prendere in considerazione nel nostro sistema è che sarà utilizzato da MMMG che non necessariamente sono pratici di informatica e di software, di conseguenza la comprensibilità è un requisito fondamentale per Health Care. Tale requisito potrà essere soddisfatto utilizzando commenti per permettere una maggiore leggibilità a coloro che in un futuro potranno apportare delle modifiche al codice; quindi, daremo maggiore importanza alla comprensibilità anche se ciò comporta un aumento dei costi e del tempo di sviluppo.

Sicurezza VS Prestazioni.

Sicurezza e Prestazioni sono due aspetti che necessitano di massima attenzione. Per quanto la sicurezza sia essenziale, aumentarla potrebbe significare un peggioramento a livello di prestazioni.

A tal proposito si è dato un maggior effort sulle prestazioni che il sistema deve avere, in quanto lo scopo del nostro sistema è quello di essere il più veloce e performante possibile rispondendo in maniera efficiente alle esigenze di tutti gli utenti.

1.2 Componenti off-the-shelf

Nella realizzazione del nostro sistema software andremo ad utilizzare componenti off-the-shelf già disponibili per facilitare lo sviluppo del progetto.

Per la progettazione del lato front-end utilizzeremo HTML5, CSS3 e alcune estensioni di JavaScript.

Per quanto riguarda l'implementazione utilizzeremo come linguaggio di programmazione Java.

Ci serviremo del software MongoDB per la creazione del Database Non Relazionale.

1.3 Linee Guida per la documentazione delle interfacce

Organizzazione del codice in file

Risulta necessario che il codice per l'implementazione del sistema HealthCare venga organizzato in file.

Ciascuno dovrà avere un nome che lo identifica univocamente, essere sviluppato in base alla categoria di appartenenza.

L'implementazione del sistema dovrà essere organizzata in package seguendo linee guida dell'architettura individuata in fase di progettazione. Ogni nuovo documento creato dovrà essere nominato in modo da rendere intuitivo l'obiettivo da realizzare e dovrà essere posizionato nel package di riferimento.

Linee guida per classi e interfaccia Java

Per ogni sorgente di classe Java implementato, dovranno essere rispettati i seguenti parametri:

▪Classi e interfacce:

- ▫ La nomenclatura delle classi dovrà rispettare la notazione UpperCamelCase;
- ▫ L'identificativo delle classi non dovrà essere ambiguo e congruo allo scopo della classe;
- ▫ L'identificativo delle classi dovrà essere un nome (Es. Array) o al più un nome frasale

  (Es.ArrayList);
- ▫I nomi delle classi di test dovranno riportare il nome della classe testata, seguita dal suffisso "UT" nel caso di test di unità e "IT" nel caso di test di integrazione;

▪Costanti:

- ▫ I valori immutabili definiti in classi java dovranno essere definiti come "static final";
- ▫ I nomi di costanti dovranno essere definiti in maiuscolo;
- ▫ È ammessa la separazione tramite _ là dove necessario;

▪Variabili d'istanza e variabili locali:

- ▫ I nomi di parametri o variabili locali dovranno essere definiti secondo la notazione lowerCamelCase o al più separati dall'underscore;
- ▫ Per ogni parametro d'istanza definito nella classe dovrà essere definito il livello di visibilità;

▪Metodi:

▫ Gli identificativi dei metodi dovranno iniziare con un verbo;

▫ Per ogni metodo di una classe sarà necessario specificare il livello di visibilità;

▫ Eventuali parametri nella firma del metodo, dovranno seguire le convenzioni adottate per le variabili d'istanza;

▪Blocchi e indentazioni:

▫ Il codice dovrà essere accuratamente indentato, tramite un Tab per ogni livello d'indentazione;

▫ Le parentesi graffe per l'inizio di un nuovo blocco di codice dovranno essere riportate sulla stessa riga della definizione del blocco;

▫ Le parentesi graffe di fine blocco dovranno essere allineate con l'inizio della definizione del blocco;

▪Blocchi eccezionali:

▫Ogni blocco try catch definito all'interno di un metodo dovrà essere indentato in maniera corretta secondo le specifiche sopra riportate;

▫Insiemi di operazioni comuni tra il blocco try e seguenti blocchi catch dovranno essere riportate nel blocco di chiusura finally;

▪Annotazioni:

▫ Le annotazioni previste per una classe o per un metodo dovranno apparire una per riga, subito dopo il blocco di documentazione;

▫Eventuali annotazioni prima di un parametro della classe dovranno essere definite una per linea, al disopra del parametro stesso, senza lasciare linee vuote;

▪Commenti:

▫ Per ogni metodo della classe dovranno essere riportati blocchi di commenti che aiutino a capire il corretto flusso di operazioni del metodo;

poco

▫ Sarà necessario chiarire tramite commenti operazioni innestate o eventuali blocchi chiari in prima lettura;

▫ La specifica di commenti nel codice dovrà essere conforme ai seguenti esempi:

```
/*
 * This is          // And so          /* Or you can
 * okay.            // is this.         * even do this. */
 */
```

Linee guida per pagine HTML 5

Per ogni documento HTML 5 creato, dovranno essere rispettati i seguenti parametri:

▪Ogni documento creato dovrà riportare il tag <!doctype html> per identificare la tecnologia HTML 5 utilizzata;

▪Ogni tag aperto nel corpo del documento HTML, a meno di tag singoli, dovrà riportare il rispettivo tag di chiusura;

▪La struttura base di una pagina html (head, body), a meno di include JSP, dovrà essere rispettata;

▪Ogni documento html, in particolare nel corpo del documento, dovrà essere indentato (preferibilmente tramite 1 tabulazione per livello) ad ogni definizione di un nuovo tag;

Esempio accettabile

```
<!doctype html>
<html>
        <head>
                <title> Titolo </title>
        </head>
        <body>
                <ol>
                        <li>
                                primo
                        </li>
                </ol>
        </body>
</html>
```

Esempio non accettabile

```
<!doctype>
<html>
        <head>
                <title> Titolo </title>

        <body>
                <ol>
                        <li>primo</li>
        </ol>
        </body>
</html>
```

Linee guida per script JavaScript

▪Ogni funzione Javascript dovrà essere riportata in un documento diverso dalla pagina html inclusa;

▪I nomi di funzioni, variabili e costanti dovranno seguire le stesse specifiche definite per i documenti Java;

▪Ogni script dovrà essere incluso all' inizio del body del relativo documento HTML.

Linee guida per Fogli di stile CSS 3

▪Ogni regola CSS non inline dovrà essere riportata su un documento differente rispetto a quello della pagina html di riferimento, in modo da garantire un più facile riuso senza duplicazione;

▪Ogni regola CSS dovrà iniziare all'inizio di una nuova linea, con la specifica dei selettori della regola;

▪L'ultimo selettore di una regola CSS dovrà essere seguito dall'apertura del blocco con {.

▪L'indentazione dovrà seguire i seguenti criteri:

- Inizio di una nuova regola (#...{) e fine del blocco della regola (}) livello di indentazione 0;
- Le proprietà di ogni regola CSS dovranno essere indentate di un Tab rispetto all'inizio del blocco, e dovranno essere riportate 1 per riga.

1.4 Design Pattern

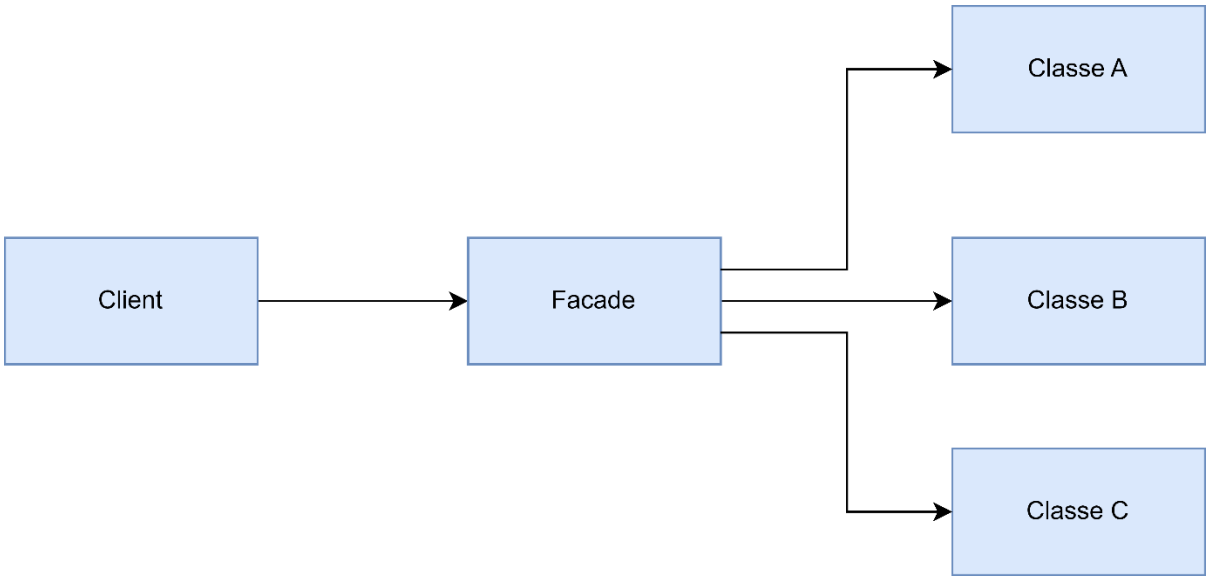
Facade

Problema

In sistemi complessi spesso si hanno tante classi che svolgono funzioni correlate e l'insieme delle interfacce può essere complesso. Può essere difficile, quindi, capire qual è l'interfaccia essenziale ai client per l'insieme di classi. Bisogna, quindi, ridurre le comunicazioni e le dipendenze dirette tra i client e i sottosistemi, quindi fornire un'interfaccia unificata al posto di un insieme di interfacce in un sottosistema. In conclusione, lo scopo è quello di definire un'interfaccia semplificata di alto livello che rende il sottosistema più leggibile e facile da utilizzare.

Soluzione

L'applicazione di questo pattern fornisce un'unica interfaccia semplificata ai client e nasconde gli oggetti del sottosistema; questo riduce la complessità dell'interfaccia e quindi delle chiamate. Facade invoca i metodi degli oggetti che nasconde, quindi il client interagirà solo con l'oggetto Facade.



Conseguenze

- Nasconde ai client l'implementazione del sottosistema.
- Promuove l'accoppiamento debole tra sottosistema e client.
- Riduce le dipendenze di compilazione in grandi sistemi. Se si desidera cambiare una classe del sottosistema, si può ri-compilare la parte di sottosistema fino al facade, quindi non i vari client.
- Non previene l'uso di client più complessi, quando occorre, che accedono ad oggetti del sottosistema.

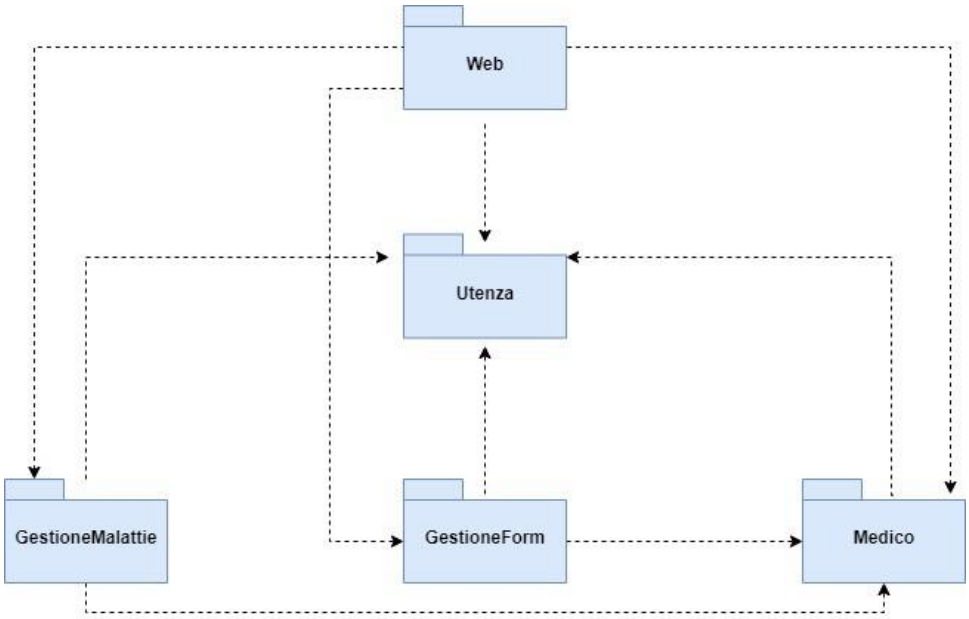
Implementazione

Per rendere gli oggetti del sottosistema non accessibili da altri client, le corrispondenti classi possono essere annidate dentro la classe Facade.

2. Packages

La divisione in package segue la divisione in sottosistemi individuata nella fase di system design. Nel package sono presenti le entità coinvolte nel sottosistema, e per ognuna di esse, abbiamo una classe repository per la logica di persistenza e una classe service per la logica di business, del sottosistema.

Il sistema sarà distribuito in packages come segue:



- Il package "Utenza" conterrà tutte le classi utili alla gestione dell'utenza.
- Il package "Medico" conterrà tutte le classi utili a gestire i dati personali del medico.
- Il package "GestioneMalattie" conterrà tutte le classi utili alla gestione delle malattie.
- Il package "GestioneForm" conterrà tutte le classi utili alla gestione dei form.
- Il package "Web" conterrà i controller.

2.1 Package utenza

Classe	Descrizione
UtenteBean.java	Classe che rappresenta le informazioni di un utente.
UserDAO.java	Si occupa del recupero e del salvataggio dei dati relativi agli utenti.
UserFacade.java	Fornisce tutti i servizi utili per la gestione dell'utenza.

2.2 Package medico

Classe	Descrizione
MMGBean.java	Classe che rappresenta le informazioni di un medico.

2.3 Package gestione malattie

Classe	Descrizione
GestioneMalattieBean.java	Classe che rappresenta le informazioni della gestione delle malattie.
SintomoBean.java	Rappresenta le informazioni relative ai sintomi.
MalattiaDAO.java	Si occupa del recupero e del salvataggio dei dati relativi alla gestione delle malattie.
SintomoDAO	Si occupa del recupero e del salvataggio dei dati relativi ai sintomi.
MalattiaFacade.java	Fornisce tutti i servizi utili per la gestione delle malattie.

2.4 Package gestione form

Classe	Descrizione
GestioneFormBean.java	Classe che rappresenta le informazioni dei form.
GestioneInterventiBean.java	Classe che rappresenta le informazioni degli interventi in un form.
InterventoDAO.java	Si occupa del recupero e del salvataggio dei dati relativi alla gestione degli interventi
FormDAO.java	Si occupa del recupero e del salvataggio dei dati relativi alla gestione dei form
FormFacade.java	Fornisce tutti i servizi utili per la gestione dei form

2.5 Package web

Classe	Descrizione
LoginServlet.java	Servlet che si occupa della funzionalità di login
LogoutServlet.java	Servlet che si occupa della funzionalità di logout
RegistrationServlet.java	Servlet che si occupa della funzionalità di registrazione di un utente
EditProfileServlet.java	Servlet che si occupa della funzionalità per modificare i dati del profilo di un MMG
ListaMalattieServlet.java	Servlet che si occupa della funzionalità per visualizzare la lista delle malattie
RicercaPerNomeServlet.java	Servlet che si occupa della funzionalità per ricercare una malattia per nome
RicercaPerSintomiServlet.java	Servlet che si occupa della funzionalità per ricercare una malattia per sintomi
AperturaFormServlet.java	Servlet che si occupa della funzionalità di creazione di un form
ApriChiudiFormServlet.java	Servlet che si occupa delle funzionalità di riapertura e chiusura di un form
DettagliFormServlet.java	Servlet che si occupa di recuperare i dettagli di un form

EliminaFormServlet.java	Servlet che si occupa della funzionalità di eliminare un form (sia aperto che chiuso)
InserimentoInterventoServlet	Servlet che si occupa della funzionalità di inserimento di un intervento ad un form di un MMG
ModificaFormServlet.java	Servlet che si occupa della funzionalità di modifica di un form da parte del MMG proprietario di esso
MyFormServlet.java	Servlet che si occupa di recuperare la lista dei form aperti e di quelli chiusi
SearchServlet.java	Servlet che si occupa del recupero di tutti i form creati
VisualizzaInterventiPersonaliServlet.java	Servlet che si occupa di recuperare tutti gli interventi fatti da un MMG a diversi form.

3. Varianti precondizioni e postcondizioni

NOME CLASSE	UtenteDAO
DESCRIZIONE	Questa classe definisce tutti i dati personali di un Medico di Medicina Generale
PRE-CONDIZIONE	Context MMG::ModificaDatiProfilo() pre: email !=null && indirizzo != null && provincia != null && comune!=null && telefono!=null && password!=null
POST-CONDIZIONE	Context MMG::ModificaDatiProfilo Post: @pre email != email || @pre indirizzo != indirizzo @pre provincia != provincia || @pre comune != comune || @pre telefono != telefono || @pre password != password
INVARIANTI	

NOME CLASSE	InterventoDAO
DESCRIZIONE	Questa classe definisce tutte le funzionalità inerenti all'intervento in un form
PRE-CONDIZIONE	Context Intervento:: InserimentoIntervento() Pre: descrizione == null Context Intervento:: CancellazioneIntervento() Pre: Intervento.exist(db.Intervento) Context Intervento:: ModificaIntervento() Pre: descrizione != null
POST-CONDIZIONE	Context Intervento::InserimentoIntervento() Post: Intervento.size() == Intervento.size()+1 Context Intervento::CancellazioneIntervento() Post: Intervento.size() == Intervento.size() - 1 Context Intervento::ModificaIntervento() Post: @pre descrizione != descrizione
INVARIANTI	Context Intervento inv: interventoId != null

NOME CLASSE	FormDAO
DESCRIZIONE	Questa classe definisce tutte le funzionalità inerenti ai form
PRE-CONDIZIONE	Context Form:: CreazioneForm() Pre: Titolo == null && descrizione == null && status==true && topic == null Context Form:: CancellazioneForm() Pre: Form.exist(db.Form) Context Form::ChiusuraForm() Pre: status == true && form.exist(db.form) Context Form:: RiattivazioneForm() Pre: status ==false && form.exist(db.form)
POST-CONDIZIONE	Context Form:: CreazioneForm() Post: form.size() == form.size() +1 Context Form:: CancellazioneForm() Post: form.size() == form.size()-1 Context Form::ChiusuraForm() Post: status == false Context Form::RiattivazioneForm() Post: status == true
INVARIANTI	Context: Form inv: formId != null

NOME CLASSE	MalattiaDAO
DESCRIZIONE	Questa classe definisce tutte le funzionalità inerenti alle Malattie Rare
PRE-CONDIZIONE	Context MalattiaRara:: RicercaPerNome() Pre: nome != null && MalattiaRara.exist(db.MalattiaRara) Context MalattiaRara:: RicercaPerSintomi() Pre: sintomo != null && Sintomo.exist(db.Sintomo)
POST-CONDIZIONE	Context MalattiaRara:: RicercaPerNome() Post: List(MalattiaRaraPerNome) Context MalattiaRara::RicercaPerSintomi

	Post: List(MalattiaRaraPerSintomi)
INVARIANTI	Context: MalattiaRara inv: malattiaRaraId != null

4. Diagramma delle classi

Per una lettura più agevole sarà allegato il file del seguente diagramma