

ODD *version 2.0*



Documento di Object Design

Gruppo 7

Team Manager:	Roberto Di Russo	0510201086
Componenti:	Giovanni Maggiolini	0510201080
	Umberto Annunziata	0510201230
	Francesco Nicolao	0510200141
	Claudio Gargiulo	0510200990
	Vincenzo Iuliano	0510201428

Revision History

Data	Versione	Descrizione	Autore
07/05/08	ODD 0.1 raw	Definizione della parte del sistema da implementare	Tutti i componenti del gruppo
08/05/08	ODD 0.2 raw	Introduzione: compromessi, standard adottati, acronimi e definizioni Definizione dei packages	Tutti i componenti del gruppo
09/04/08 - 10/05/08	ODD 0.3 raw	Definizione delle interfacce delle classi	Tutti i componenti del gruppo
12/05/08	ODD 1.0	Glossario Integrazione e correzione del lavoro svolto	Tutti i componenti del gruppo
03/06/08	ODD 2.0	Correzione e revisione finale	Tutti i componenti del gruppo

ODD

Progetto di un sistema software per la gestione dei posti letto presso una clinica privata

1. Introduzione.....	pag.4
1.1 Compromessi dell'object design.....	pag.4
1.2 Standard adottati.....	pag.5
1.3 Definizioni, Acronimi e Abbreviazioni.....	pag.7
1.4 Riferimenti.....	pag.7
2. Package EasyClinique.....	pag.8
2.1 Gestione Pazienti.....	pag.11
2.2 Gestione Cartelle Cliniche.....	pag.11
2.3 Gestione Degenze.....	pag.12
2.4 Gestione Accessi.....	pag.12
3. Class Interfaces.....	pag.13
3.1 Gestione Pazienti.....	pag.13
3.2 Gestione Cartelle Cliniche.....	pag.16
3.3 Gestione Degenze.....	pag.19
3.4 Gestione Accessi.....	pag.21
4. Glossario.....	pag.23

1. Introduzione

1.1. Compromessi dell'Object Design

In seguito alle precedenti scelte progettuali, è emersa l'esigenza di utilizzare componenti off-the-shelf per sviluppare il sistema.

In particolare, verrà utilizzata la componente JDBC per permettere connessione e la comunicazione tra il software ed il database.

Interfaccia vs usabilità

L'interfaccia grafica si basa sull'utilizzo di schede (linguette), pulsanti e form ed ha una impostazione semplice e intuitiva che permette un uso facile del sistema così da rendere immediata l'attività anche ai meno esperti.

Sicurezza vs efficienza

La gestione della sicurezza si basa sull'autenticazione tramite userID e password: ogni utente può accedere solo alla parte del sistema a cui l'accesso è autorizzato.

Le autorizzazioni vengono definite, quindi, da una politica di permessi.

(vedi SDD EasyClinique 2.0).

Ciò permette di non rallentare eccessivamente il sistema, pur non trascurando la sicurezza.

Comprensibilità vs tempo

Il codice deve essere quanto più comprensibile possibile, per facilitare la successiva fase di testing.

Le classi ed i metodi devono essere interpretabili anche da chi non ha preso parte al progetto.

Il codice viene corredato da commenti scritti in uno stile standard (vedi punto 1.2), al fine di facilitare la comprensione del sorgente.

Seppur necessaria, tuttavia questa caratteristica causa un rallentamento nello sviluppo del progetto.

1.2.Standard adottati

Nella scrittura del codice e della documentazione gli sviluppatori devono attenersi a delle linee guida di seguito definite.

Stile di programmazione

L'indentazione del codice deve essere di quattro spazi, l'equivalente di un "tab". La parentesi graffa che chiude un blocco deve essere in una riga riservata.

```
Es.: if (i==0) {  
        red=blue;  
    }
```

Convenzioni sui nomi

I nomi delle classi devono descrivere il più possibile la responsabilità della classe; tutte le parole contenute nel nome devono iniziare con una lettera maiuscola.

```
Es.: public class DatiPaziente
```

I nomi delle variabili di istanza devono iniziare con una lettera minuscola, e tutte le successive parole contenute nel nome devono invece iniziare con una maiuscola.

Le costanti devono essere scritte completamente in lettere maiuscole, e le parole che ne compongono il nome devono essere separate con un "_", simbolo di underscore.

```
Es.: private String codiceFiscale  
    private static final int DISTINTIVO_MATR_MEDICO=054
```

I nomi dei metodi devono iniziare con una lettera minuscola, e ogni successiva parola contenuta nel nome deve invece iniziare con una maiuscola.

Solitamente il nome di un metodo inizia con un verbo, e prosegue con il nome di una classe.

```
Es.: inserisciCartellaClinica()
```

I nomi dei metodi di accesso devono iniziare con `get`, mentre i metodi di modifica con `set`.

I metodi che controllano il valore di variabili booleane devono, invece, iniziare con `is`.

Es.: `getNome() ;`
`setCognome() ;`
`isRicoverato() ;`

Documentazione

Dopo le eventuali dichiarazioni di `import` e `package` di appartenenza, ogni classe deve avere un commento che ne specifica scopi e responsabilità, il nome dello sviluppatore ed, eventualmente, la versione. Il commento deve essere scritto in modo da poter essere convertito in documento HTML con il tool Javadoc.

Es.:
`/** Breve descrizione dello scopo della classe`
`*`
`*@author nome dello sviluppatore`
`*@version versione`
`*`
`*/`

Prima di ogni metodo va inserito un breve commento convertibile in Javadoc.

Vanno specificati lo scopo del metodo, gli argomenti da passare, il valore di ritorno e le eventuali eccezioni.

Es.:
`/** Breve descrizione dello scopo del metodo`
`*`
`*@param param1 descrizione del primo argomento`
`*@param param2 descrizione del secondo argomento`
`*@return descrizione del valore di ritorno`
`*@throws NomeEccezione descrizione delle condizioni in`
`cui il metodo lancia un'eccezione`
`*`
`*/`

In alcuni punti potranno essere inseriti commenti inline.

Es.: //ciclo sugli oggetti

1.3.Definizioni, acronimi e abbreviazioni

API= Application Programming Interface

JDBC= Java DataBase Connectivity

SDD= System Design Document

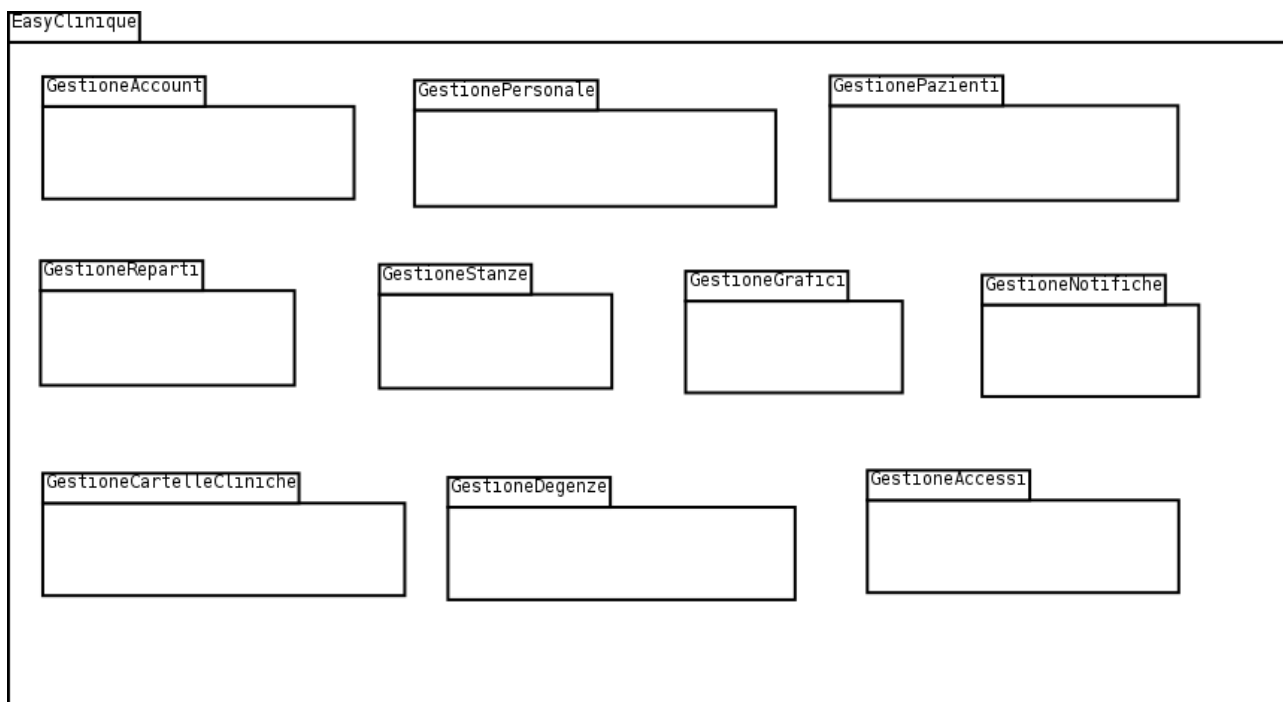
ODD= Object Design Document

1.4.Riferimenti

EasyClinique SDD 2.0.

2. Packages

EasyClinique



Il package generale EasyClinique contiene al suo interno i sottopackages relativi alle varie gestioni previste nel progetto software e definite nell' EasyClinique SDD.

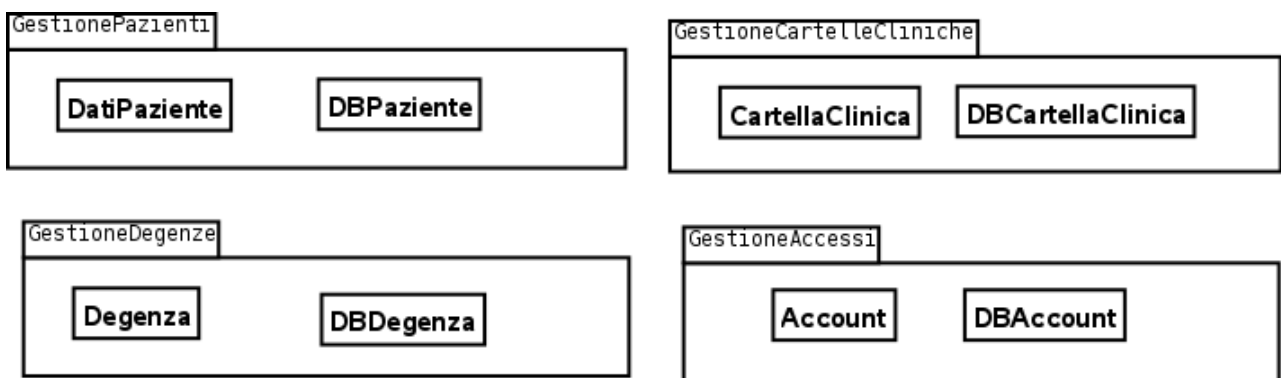
Rispetto a quest'ultimo documento, sono state ritenute necessarie delle modifiche per non appesantire l'implementazione del sistema. Si tratta principalmente di accorpamenti di sottopackages, di seguito riportati.

- CreazioneCartelleCliniche viene collassato nel package GestioneCartelle Cliniche.
La distinzione tra l'uso delle varie funzionalità viene effettuata attraverso la politica degli accessi specificata nel System Design Document.
- RicercaPaziente viene collassato nel package GestionePazienti.
Per la distinzione dell'uso delle funzionalità, vale lo stesso discorso fatto per la GestioneCartelleCliniche.

In accordo con il docente/committente, è stato deciso che delle funzionalità elencate graficamente sopra verranno implementate:

1. GestionePazienti
2. GestioneCartelleCliniche
3. GestioneDegenze
4. GestioneAccessi

Di seguito, viene riportata una prima visione grafica, successivamente raffinata, delle funzionalità implementate



Dunque, nel presente documento vengono descritte solo le interfacce delle classi relative all' implementazione delle suddette funzionalità.

Per garantire un corretto funzionamento del software a scopi dimostrativi, le altre funzionalità descritte nei documenti precedenti verranno implementate in un secondo momento, sottoforma di stub, ovvero classi “fantoccio”, il cui fine è semplicemente quello di permettere, appunto, un'esecuzione dimostrativa del software.

Nel successivo paragrafo vengono elencate graficamente le classi che compongono ciascun package. Una loro descrizione approfondita (metodi e campi) è fornita al punto 3.

Si noti che per ciascuna funzionalità si è preferito “accorpare” l' ApplicationLogicLayer e lo StorageLayer, definiti nel SDD. La scelta è stata effettuata in quanto, utilizzando JDBC, la parte relativa alla logica applicativa del pacchetto e quella relativa alla memorizzazione sono strettamente correlate.

Per ciascun gruppo di funzionalità, sono presenti, quindi, due pacchetti, ognuno composto da diverse classi:

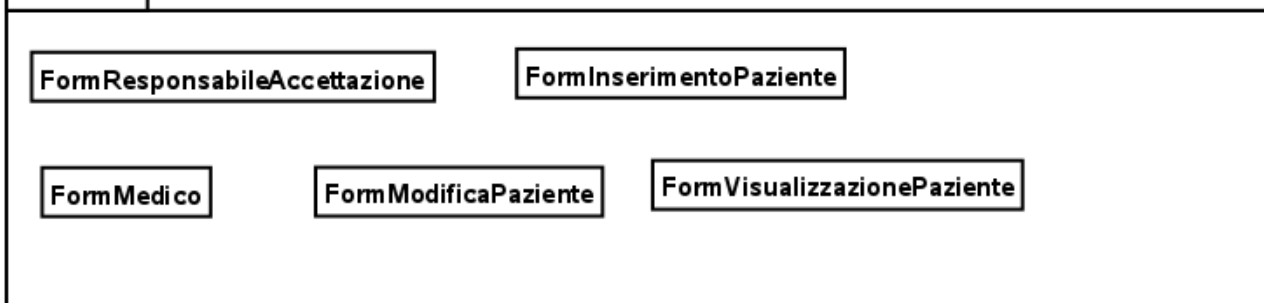
- GestioneXXX, che si occupa, appunto, della logica applicativa e della memorizzazione dati
- GUIXXX, che si occupa dell'interazione con l'utente (in pratica, delle interfacce grafiche).

In questo livello, si è preferito implementare le classi relative alle interfacce grafiche come estensioni della classe JFrame.

Inoltre, come è possibile notare nel punto 3., queste classi sono prive di metodi e variabili di istanza: ciò è giustificabile dal fatto che la costruzione dell'interfaccia può essere fatta direttamente nel costruttore, utilizzando delle variabili locali per i componenti (pulsanti, combo box etc).

2.1 GestionePazienti

GUIPazienti

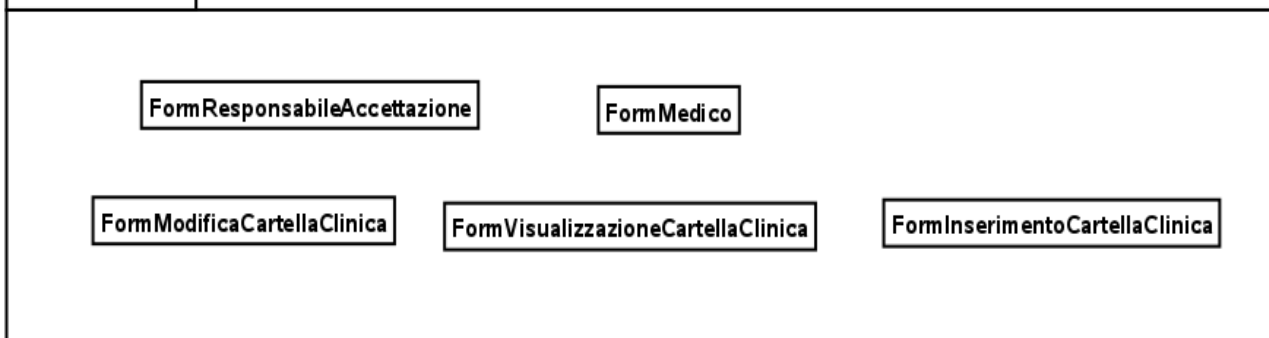


GestionePazienti

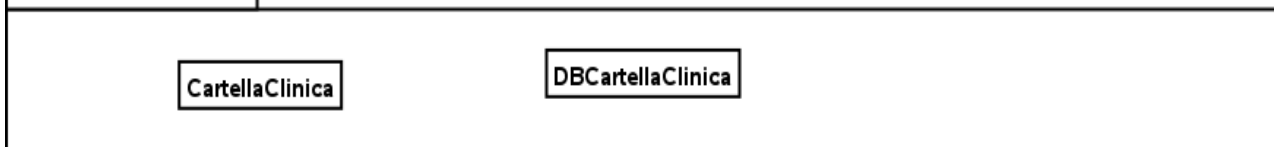


2.2 GestionCartelleCliniche

GUICartellaClinica



GestioneCartelleCliniche



2.3 GestioneDegenze

GuiDegenza

FormResponsabileAccettazioni

FormDimissionePaziente

FormCambioReparto

FormAssegnazionePostoLetto

GestioneDegenze

Degenza

DBDegenza

2.4 GestioneAccessi

GuiAccessi

FormAuth

GestioneAccessi

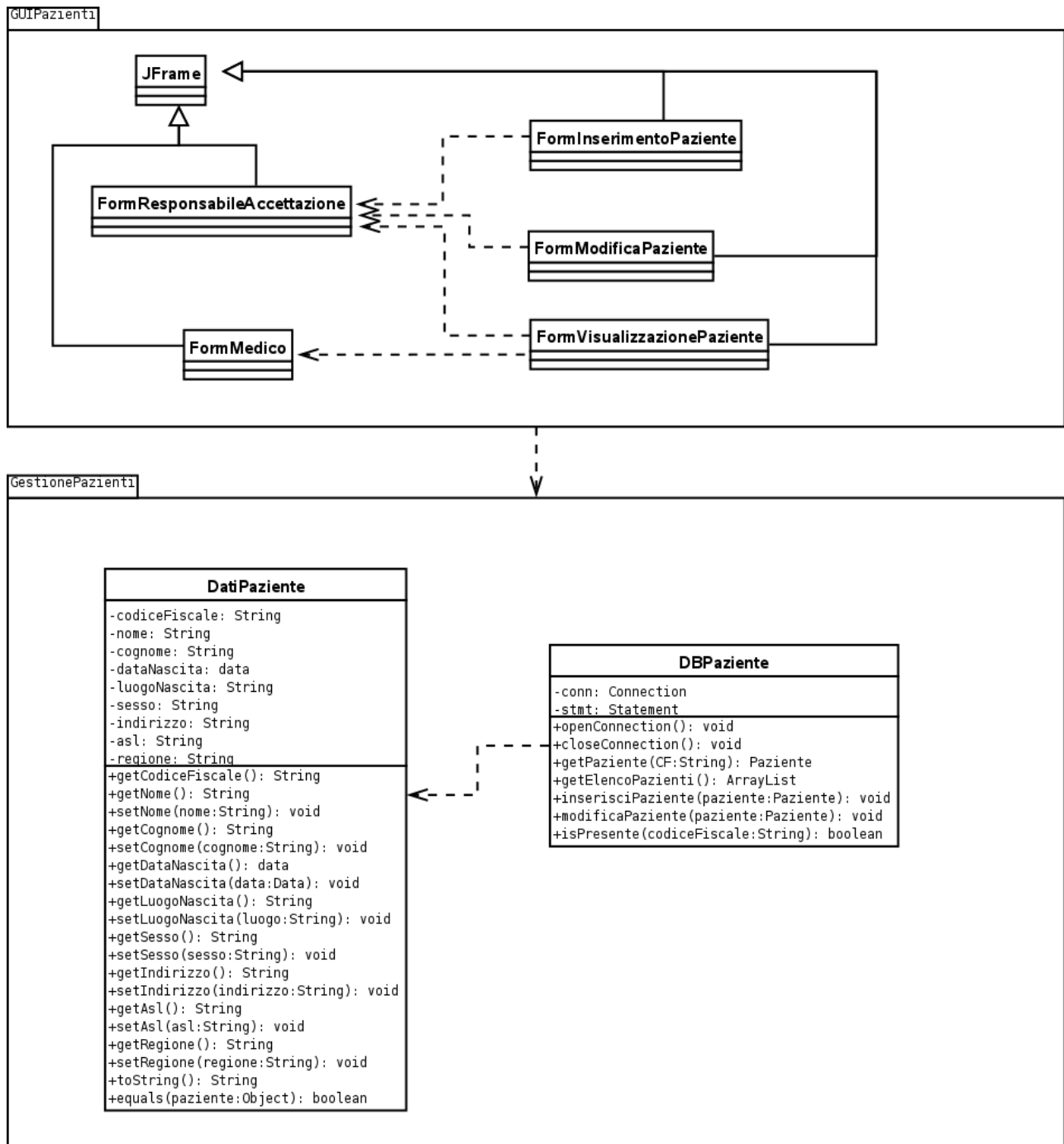
Accesso

DBAccesso

La descrizione approfondita dei package e delle classi viene fornita in Javadoc al link [Index](#)

3. Class Interfaces

3.1 GestionePazienti



Le descrizioni delle classi sono state realizzate tramite Javadoc. Di seguito sono riportati i collegamenti alle relative documentazioni delle varie classi:

GestionePazienti

[DatiPaziente](#)

[DBPaziente](#)

GuiPazienti

[FormInserimentoPaziente](#)

[FormModificaPaziente](#)

[FormVisualizzazionePaziente](#)

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

Nome Classe: DatiPaziente

Invarianti: il codice fiscale è unico.

Precondizioni: +setDataDiNascita(dataNascita): dataNascita dovrebbe essere precedente alla data attuale.

Postcondizioni: -

OCL Standard:

Context DatiPaziente::setDataDiNascita(dataNascita) pre:
dataNascita<Date.now()

Nome Classe: DBPaziente

Invarianti: -

Precondizioni: +inserisciPaziente(paziente): il codice fiscale di paziente deve essere diverso dai codici fiscali di tutti gli altri pazienti presenti.
+getPaziente(cf): il codice fiscale inserito deve essere associato ad un record del database.
+closeConnection(): la connessione deve essere aperta.
+getElencoPazienti(): nel database deve essere presente almeno un paziente.

Postcondizioni: -

OCL Standard:

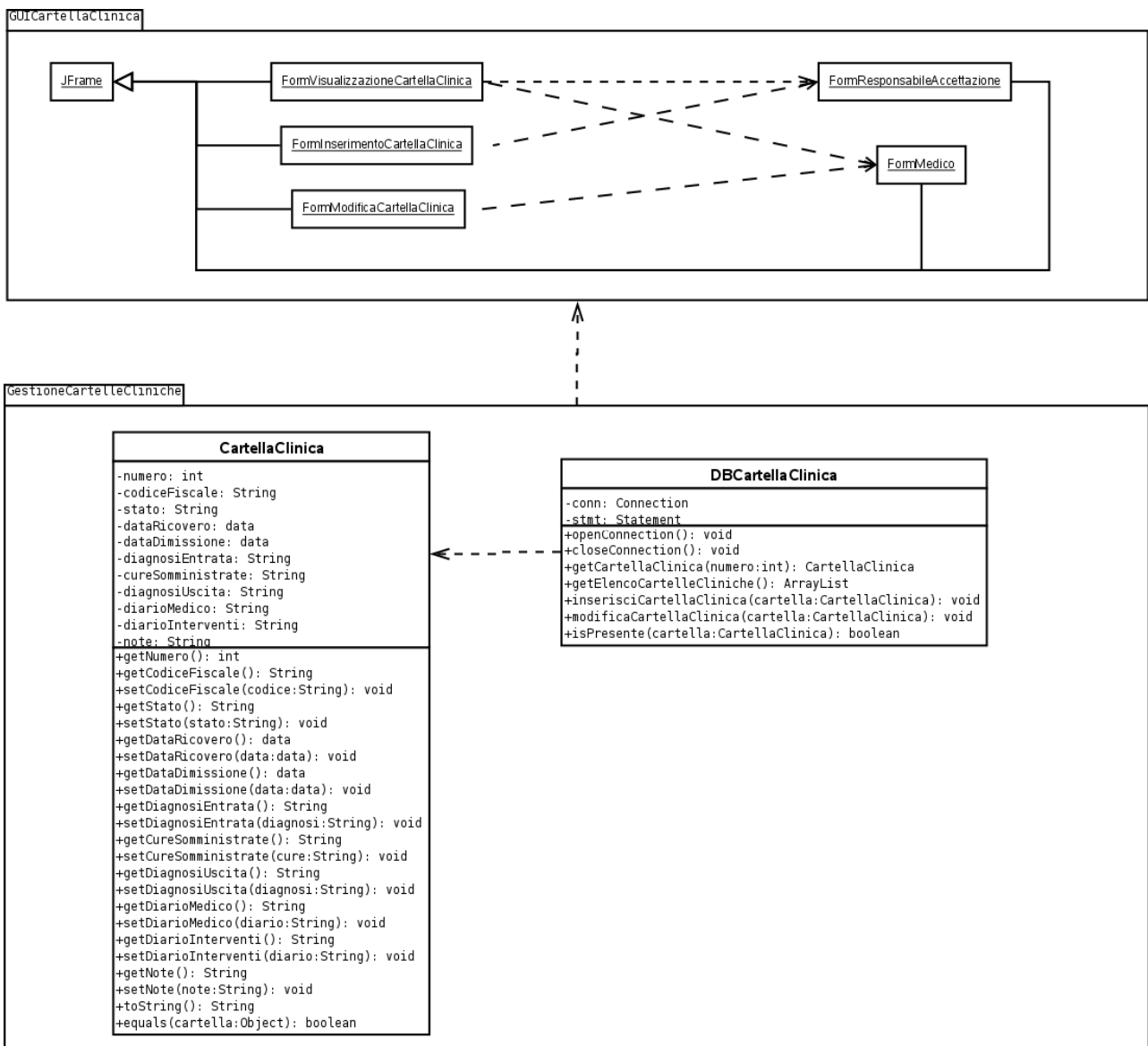
Context DBPaziente::inserisciPaziente(paziente) pre:
 paziente.codiceFiscale<>null AND not isPresente(paziente.codiceFiscale)

Context DBPaziente::getPaziente(cf) pre:
 isPresent(cf)

Context DBPaziente::closeConnection() pre:
 connection.status=open

Context DBPaziente::getElencoPazienti() pre:
 getElencoPazienti().size()>0

3.2 GestioneCartelleCliniche



Le descrizioni delle classi sono state realizzate tramite Javadoc. Di seguito sono riportati i collegamenti alle relative documentazioni delle varie classi:

GestioneCartelleCliniche:

[CartellaClinica](#)
[DBCartellaClinica](#)

GuiCartelleCliniche:

[FormInserimentoCartellaClinica](#)
[FormModificaCartellaClinica](#)
[FormVisualizzazioneCartellaClinica](#)

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

Nome Classe: CartellaClinica

Invarianti: Alla cartella clinica è associato un numero positivo unico.
 la cartella clinica è associata ad un paziente (identificato tramite codice fiscale);
 la data di ricovero è sempre antecedente la data di dimissione.

Precondizioni: +getCodiceFiscale(): il codice fiscale deve essere stato impostato.

Postcondizioni: +setCodiceFiscale(nuovoCodiceFiscale): il nuovo codice fiscale dovrebbe essere differente dal precedente.

OCL Standard:

Context CartellaClinica inv: getNumero()>=0
 getCodiceFiscale()<>null
 getDataRicovero()<getDataDimissione()

Context CartellaClinica::getCodiceFiscale() pre:
 codiceFiscale <> null

Context CartellaClinica::setCodiceFiscale(nuovoCodiceFiscale) post:
 nuovoCodiceFiscale<>@getCodiceFiscale()

Nome Classe: DBCartelleCliniche

Invarianti: -

Precondizioni: +getCartellaClinica(numero): la prima parte di numero deve essere maggiore o uguale di zero e corrispondere ad una cartella clinica presente nel database.

+isPresente(cartellaClinica): cartellaClinica deve essere diverso da null.

+inserisciCartellaClinica(cartellaClinica): cartellaClinica deve non essere presente nel database.

+getElencoCartelleCliniche(): nel database deve essere presente almeno una cartellaClinica.

Postcondizioni: -

OCL Standard:

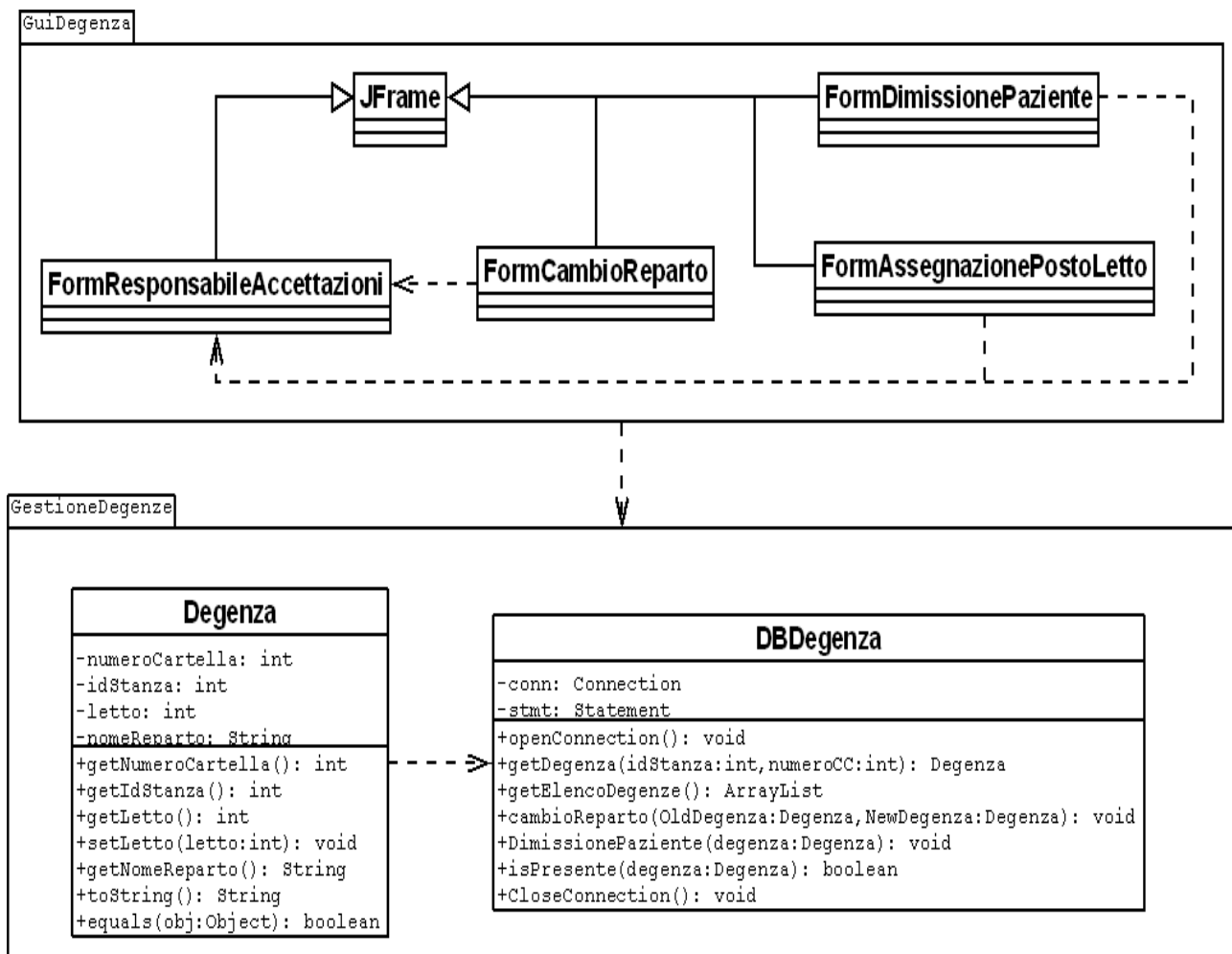
Context DBCartelleCliniche::getCartellaClinica(numero) pre:
numero>=0 AND isPresente(numero)

Context DBCartelleCliniche::isPresente(cartellaClinica) pre:
cartellaClinica<>null

Context DBCartelleCliniche::inserisciCartellaClinica(cartellaClinica) pre:
not contains(cartellaClinica)

Context DBCartelleCliniche::getElencoCartelleCliniche() pre:
getElencoCartelleCliniche().size()>0

3.3 GestioneDegenze



Le descrizioni delle classi sono state realizzate tramite Javadoc. Di seguito sono riportati i collegamenti alle relative documentazioni delle varie classi:

GestioneDegenze

[Degenza](#)

[DBDegenza](#)

GuiDegenze

[FormDegenza](#)

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

Nome Classe: Degenza

Invarianti: alla degenza è sempre associata una cartella clinica ed un posto letto di una determinata stanza.

Precondizioni: -

Postcondizioni: +setLetto(letto): il nuovo posto letto dovrebbe essere differente da quello precedente.

OCL Standard:

Context Degenza inv:

cartellaClinica<>null AND postoLetto<>null AND stanza <> null

Context Degenza::setLetto(letto): letto<>@letto

Nome Classe: DBDegenza

Invarianti: -

Precondizioni: +closeConnection(): la connessione deve essere aperta.

+getElencoDegenze(): nel database deve essere presente almeno una degenza.

Postcondizioni: -

OCL Standard:

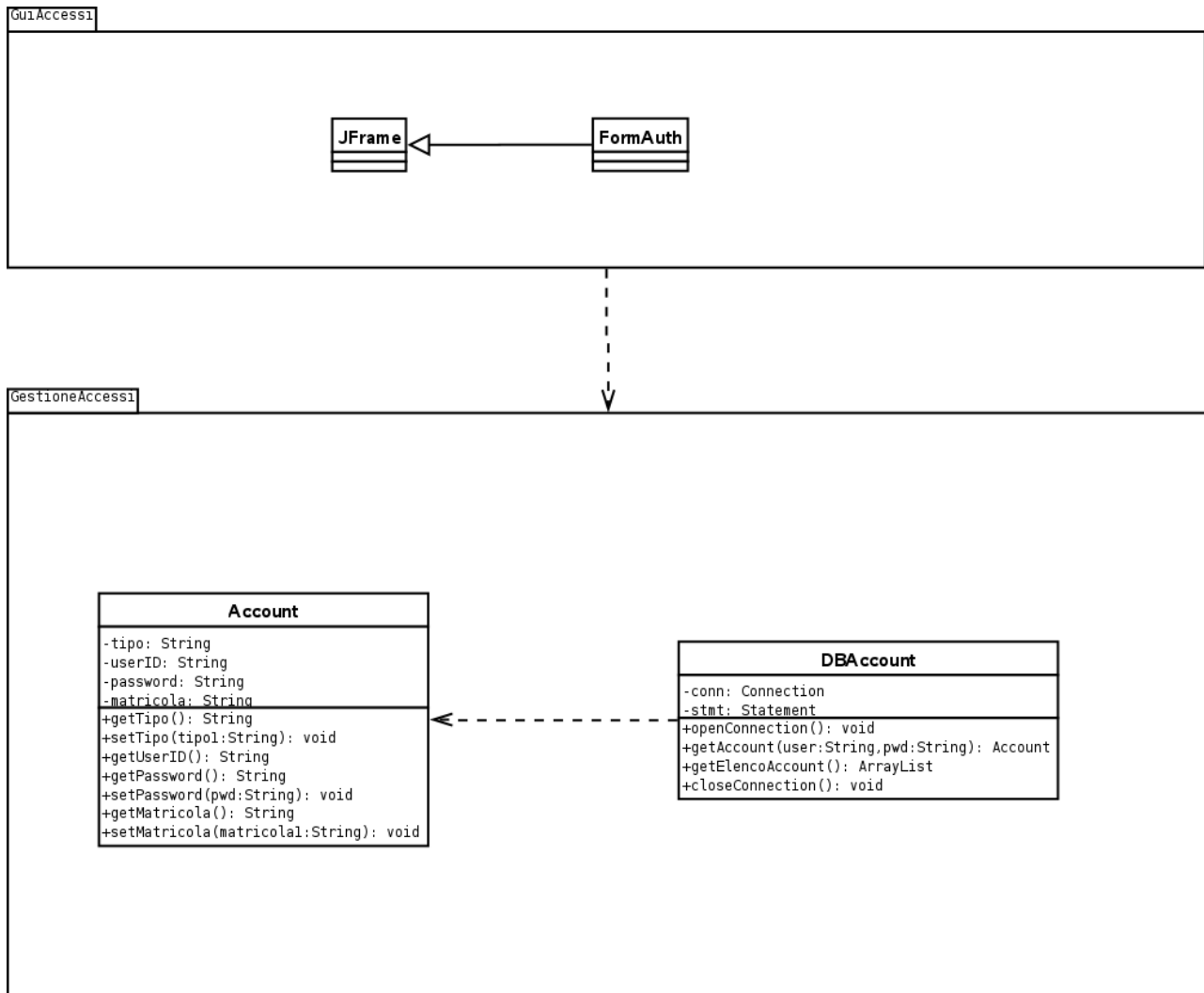
Context DBDegenza::closeConnection() pre:

connection.status=open

Context DBDegenza::getElencoDegenze() pre:

getElencoDegenze().size()>0

3.4 GestioneAccessi



Le descrizioni delle classi sono state realizzate tramite Javadoc. Di seguito sono riportati i collegamenti alle relative documentazioni delle varie classi:

GestioneAccessi

[Account](#)

[DBAccount](#)

GuiAccessi

[FormAuth](#)

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:

Nome Classe: Account

Invarianti: Ad ogni account è associato sempre il numero di matricola di un utente presente nel sistema (non sono ammesse matricole fittizie).

Precondizioni: -

Postcondizioni: +setPassword(pwd): la nuova password non deve violare il vincolo di integrità della coppia (userID, password).

OCL Standard:

Context Account inv:

matricola=->exists(Utente.matricola=matr).

Context Account::setPassword(pwd) post:

pwd<>@pwd AND ->exists(pswrd=pwd)=false

Nome Classe: DBAccount

Invarianti: -

Precondizioni: +getAccount(user, password): l'account ricercato deve essere presente nel database.

+getElencoAccount(): nel database deve essere presente almeno un account.

Postcondizioni: -

OCL Standard:

Context DBAccount::getAccount(user, password) pre:

account=->exists(Account.userID=user AND

Account.password=password)=true

Context DBAccount::getElencoAccount() pre:

getElencoAccount().size()>0

4. Glossario

Terms	Definitions
EasyClinique	Nome del sistema software sviluppato.
UtenteGenerico	Termine che identifica uno qualsiasi degli utenti registrati nel sistema; viene utilizzato nella descrizione di funzionalità accessibili ad ogni tipo di utente (login, logout, consultazione manuale utente, ricerca e visualizzazione dati non medici relativi ai pazienti).
Amministratore	Termine che identifica colui che si occupa di registrare, modificare e cancellare gli account degli utenti di altro tipo.
DirigenteSanitario	Termine che identifica colui che si occupa di: gestire i dati relativi al personale ospedaliero (ossia medici, paramedici e responsabili dell'accettazione); visualizzare grafici e statistiche sull'operato della struttura; a gestire i dati relativi ai reparti e alle stanze.
Dirigente	Termine usato in modo interscambiabile con "Dirigente sanitario".
Personale/staff medico	Termine collettivo che identifica medici e paramedici e responsabile sanitario.
ResponsabileAccettazione	Termine che identifica colui che gestisce l'assegnazione dei pazienti alle stanze. È lui che gestisce i dati non medici relativi ai pazienti.
Responsabile	Termine usato in modo interscambiabile con "Responsabile accettazione".
Medico	Termine che identifica colui che si occupa della gestione delle cartelle cliniche dei pazienti; è l'unico tipo di utente che può modificarle.
Paramedico	Termine collettivo che identifica infermieri e operatori sanitari. Può soltanto visionare le cartelle cliniche dei pazienti e i loro dati.

CartellaClinica	Termine con cui si intendono i dati medici relativi ad un paziente. Può essere visionata solo da paramedici e medici, e modificata soltanto da questi ultimi.
Dati personale/paziente	Termine con cui si intendono i dati non medici relativi ad un paziente o i dati relativi ad un membro del personale medico.
Grafico	Un grafico raffigurante informazioni di interesse sulla struttura.
Reparto	Un reparto della struttura, costituito da un insieme di stanze.
Stanza	Una stanza facente parte di un reparto e comprendente una serie di posti letto.
Posto letto	Posto letto che è incluso in una stanza e può essere assegnato a un paziente.
Account	Termine utilizzato per descrivere i dati di un utente registrato nel sistema (userID, password e tipo). I termini userID e userName sono, come di consueto, utilizzati in modo interscambiabile.
Design goals	Qualità del software che devono essere ottimizzate (definite nel System Design Document).
Three-tiers	Tipo di architettura basata su tre strati, utilizzata nella progettazione del sistema.
Java	Il linguaggio di programmazione utilizzato per sviluppare il software.
SQL	Acronimo di Structured Query Language, è il linguaggio di interrogazione usato per interagire con il database.
GUI	Acronimo di Graphic User Interface (interfaccia grafica utente), ossia l'insieme di oggetti che l'utente vede sullo schermo e con cui interagisce.
JDBC	Acronimo di Java DataBase Connectivity. Componente che permette ai programmi scritti in linguaggio Java di interagire con un database.

Form	Oggetto facente parte della GUI. Consiste in una finestra con campi di testo che l'utente deve riempire, e un pulsante per sottomettere le informazioni inserite.
Auth (autenticazione)	Processo con cui il sistema verifica se un utente è autorizzato ad accedere, e quali funzionalità può utilizzare.
Interfaccia	Nome che in Java indica la descrizione astratta di un insieme di dati e delle operazioni che possono essere compiute su di esso. Le operazioni non sono implementate.
Classe	Nome che in Java identifica un insieme di oggetti che condividono proprietà statiche (attributi) e comportamento (metodi). Una classe realizza un'interfaccia se implementa tutte le operazioni definite in essa.
Metodo	Operazione che può essere compiuta su di un oggetto.
Firma	Descrizione di un metodo: comprende il nome, il tipo di oggetto che restituisce, e gli oggetti che vengono passati come parametri.
Package	Nome che in Java indica un insieme di classi e interfacce che contribuiscono ad uno scopo comune.