ODD

Documento di Object Design

Gruppo 14

Team Manager: Maria D'Arco 0510201593

Componenti : Massimo Caruso 0510201704
Filomena Fruncillo 0510201659
Vincenzo D'Alessio 0510201560
Giuseppe Valitutto 0510201191

1. Introduzione ……..…3
1.1 Compromessi dell'object design……....3
1.2 Standard adottati……4
1.3 Definizioni, Acronimi e Abbreviazioni……7
1.4 Riferimenti……....7

2. Package FullDent……...…8
2.1 Gestione Accessi…….….11
2.2 Gestione Sale…..……...…11
2.3 Gestione Attrezzature ……12
2.4 Gestione Fornitori……13
2.5 Getione Calendario……..14

3. Class Interfaces. ……15
3.1 Gestione Sale…….……..15
2.2 Gestione Calendario…..……18
2.3 Gestione Attrezzature ……...21
2.4 Gestione Fornitori……24

1. Introduzione

1.1Compromessi dell'Object Design

Come già riportato nei precedenti documenti è stato scelto di utilizzare la componente JDBC per permettere le operazioni di connessione e comunicazione tra il software ed il database.

Interfaccia vs usabilità

L'interfaccia grafica utilizzata risulta essere molto intuitiva, quindi a prova di utenti poco esperti. Inoltre, l'ausilio di bottoni e linguette rende la gestione e la visualizzazione del lavoro più limpida.

Sicurezza vs efficienza

La sicurezza del prodotto è regolata da accessi al sistema che seguono una particolare politica di permessi: ogni utente possiede un UserId ed una Password ed, in base al dominio di competenza, viene dato l'accesso alla propria area. Le operazioni di autenticazione, non essendo onerose in termini di risorse e tempo, non affaticano il sistema e, quindi, non ne compromettono l'efficienza.

Comprensibilità vs tempo

Al fine di rendere comprensibile il codice e, di conseguenza, agevolare la fase di testing, vengono utilizzati commenti per la descrizione del codice. Tali commenti vengono scritti in uno stile standard (quello fornito per default dal compilatore java "Eclipse"). Realizzare commenti comporta un notevole aumento del tempo di sviluppo, ma l'ausilio di questi è necessario per le motivazioni precedentemente descritte.

1.2 Standard Adottati

I sviluppatori nella scrittura del codice e della documentazione ad esso associato devono necessariamente attenersi a delle linee guida per facilitare la comprensione e l'eventuale modifica, al fine di miglioramento, del codice.

Di seguito vengono specificati i standard di riferimento adottati.

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.:
```
for(i=0; i<n ; i++) {
    If((i%2)==0){
        A[i]==0
    }
}
```

Convenzioni sui nomi

I nomi delle classi identificate devono ben delineare le responsabilità della classe stessa.
I nomi delle classi non devono contenere caratteri speciali o numeri e le parole che compongono li compongono devono iniziare con la lettera maiuscola.

Es.:
```
public class DatiPersonale {
    …
}
```

I nomi delle variabili istanza devono iniziare con una lettera minuscola, se il nome della variabile è composto da più parole allora la prima parola verrà scritta intermante in maiuscolo mentre le successive parole contenute nel nome devono invece iniziare con una lettera maiuscola.

Diversamente i nomi di variabili costanti vengono scritti completamente in maiuscolo.

Le parole che compongono il nome di una variabile, a differenza di quelli che compongono una classe, devono essere separate dal simbolo di underscore (“ _ ”)

Es.:
```
public class DatiPersonale {

    public String codiceFiscale;
    private static final int PREZZO_SERVIZIO = 100;
        …
}
```

I nomi dei metodi devono identificare la distinta operazione che il metodo deve compiere. I nomi dei un metodi devono iniziare con una lettera minuscola, e ogni successiva parola contenuta deve invece iniziare con una maiuscola. In genere il nome di un metodo inizia con un verbo e prosegue con il nome di una classe e non vi sono presenti caratteri speciali.

Es.:
```
public class DatiPersonale {

    Public inserisciPersonale(){
        …
    }
}
```

Inoltre è bene ricordare che i nomi dei metodi di accesso devono iniziare con `get`, mentre i metodi di modifica con `set` e i metodi che controllano il valore di variabili booleane devono, invece, iniziare con `is`.

Es.:
```
public class DatiPersonale {

    public String getNomePersonale(){
    …
    }


    public void setNomePersonale( Sting newNome){
    …
    }

   public boolean isCurriculumVitae(){
    …
   }
}
```

Documentazione

In primis si sottolinea che un qualsiasi commento, curato in una classe, deve essere scritto in modo da poter essere convertito in documento HTML con il tool Javadoc

Le documentazioni legate ad ogni classe definita devono contenere, oltre alle eventuali dichiarazioni di import e package di appartenenza, un commento che specifica i scopi e le responsabilità della classe, nonché il nome dello sviluppatore ed un eventuale versione della classe.

Un qualsiasi commento, delineato in una classe, deve essere scritto in modo da poter essere convertito in documento HTML con il tool Javadoc

Es.:
```
public class DatiPersonale {

    /** Breve descrizione dello scopo della classe
    *
    *@author nome dello sviluppatore
    *@version versione
    *
    */

    …

}
```

Un breve commento viene inserito anche ad un metodo, che implementa una operazione di nota. Vanno specificati lo scopo del metodo, gli argomenti da passare, il valore di ritorno e le eventuali eccezioni.

Es.:
```
public class DatiPersonale {

    Public inserisciPersonale(){

        /** 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
            *
            */


        }
        …
    }
```

Infine è possibile inserire, in alcuni punti, commenti inline.

Es.:
```
public class DatiPersonale {

        Public inserisciPersonale(){
            …
            //ciclo sugli oggetti
            …
        }
        …
    }
```

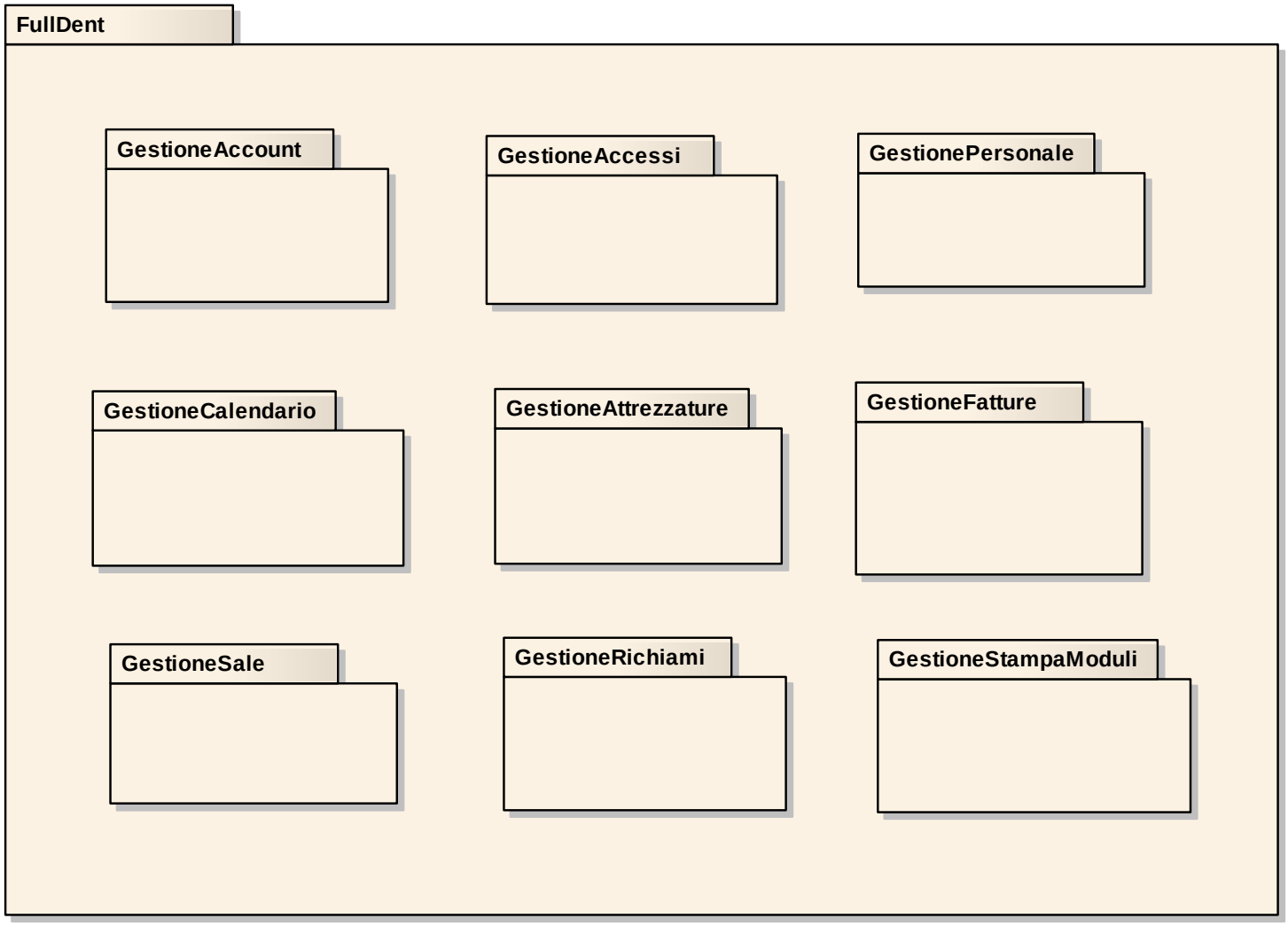
1.3 Definizioni, acronimi e abbreviazioni

API = Application Programming Interface
JDBC = Java DataBase Connectivity
SDD = System Design Document
ODD = Object Design Document

1.4.Riferimenti

FullDent SDD .

2. Packages FullDent

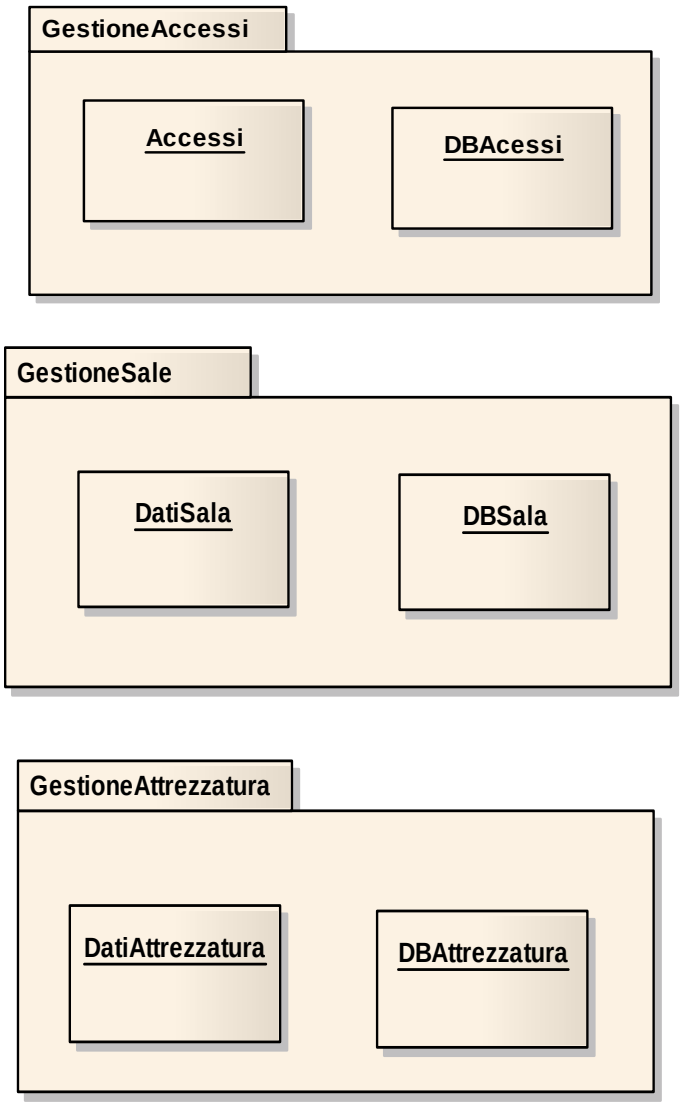


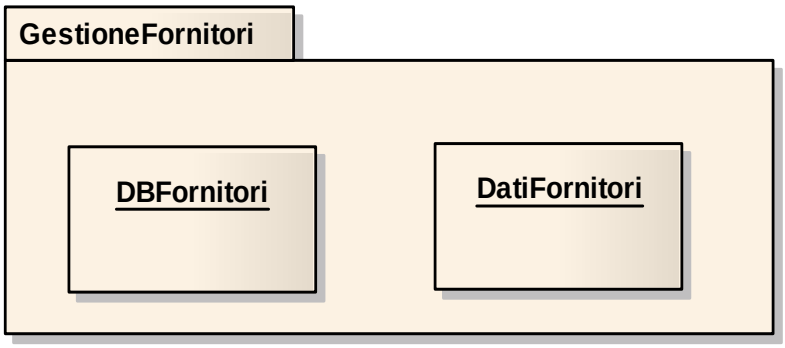
La figura, precedentemente illustrata, descrive la visione generale del pakage di FullDent che comprende i sottopackages relativi alle varie gestioni previste nel progetto software e individuati nell' SDD di FullDent.

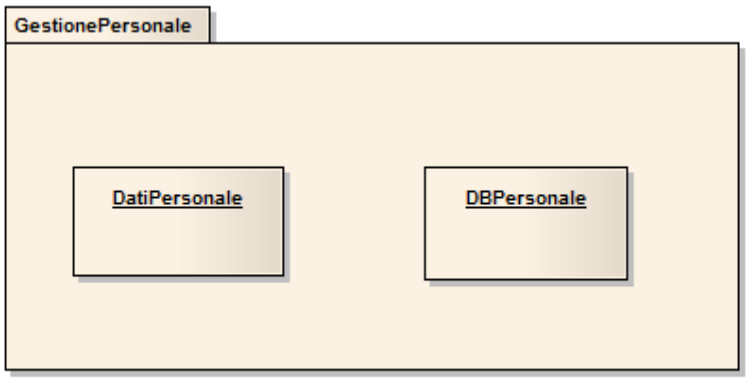
Tra le funzionalità precedentemente illustrate, in accordo con il docente/committente, si è deciso di realizzare:

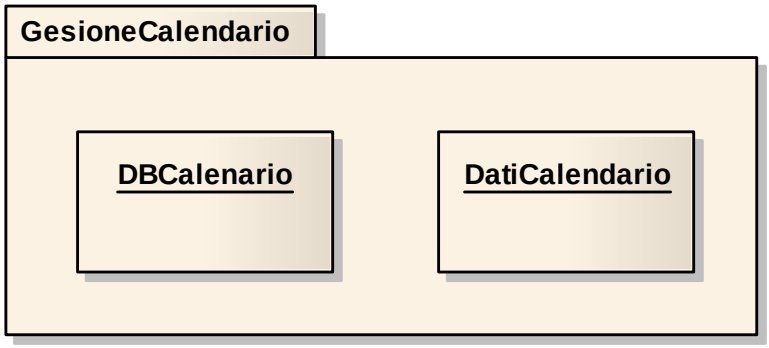
- Gestione Accessi;
- Gestione Sale;
- Gestione Personale;
- Gestione Attrezzature;
- Gestione Fornitori;
- Gestione Calendario.

Di seguito verrà riportata una quadro grafico delle funzionalità implementate e che successivamente verrà raffinato.









In questo documento verranno descritte e realizzate nei dettagli le classi e le interfacce che identificano le funzionalità sopra descritte. Tale scelta è dovuta al fine di garantire un coretto funzionamento del software a scopi dimostrativi.

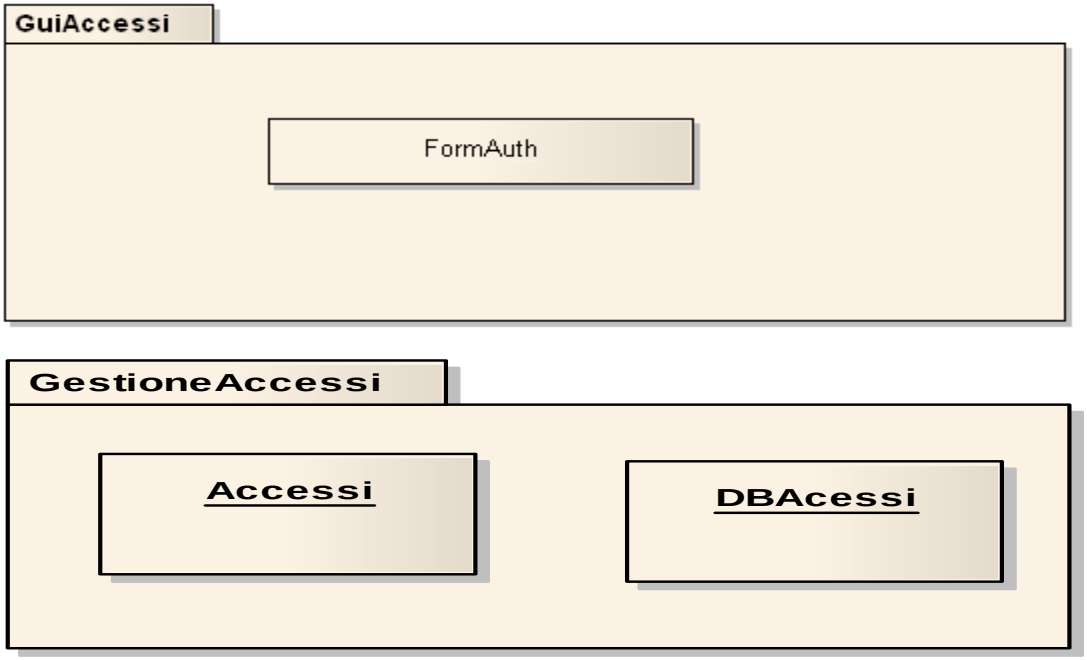
Verranno successivamente implementate le altre funzionalità (in modo molto superficiale) necessarie al funzionamento e al test dimostrativo del software.

Le classi che compongono ciascun package verranno illustrate graficamente nei paragrafi successivi.

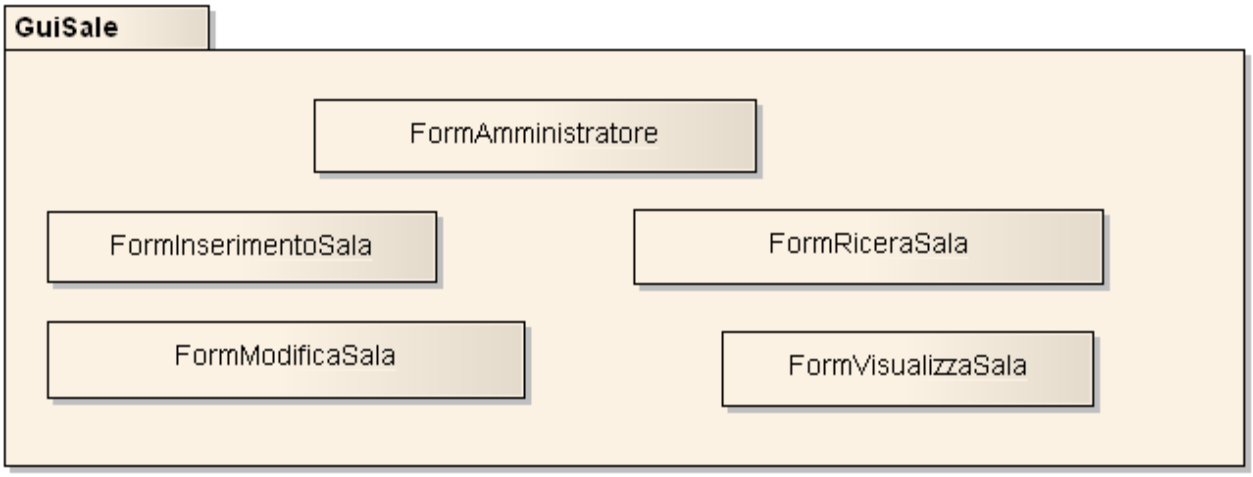
Allo scopo di rendere più pulita la gestione delle funzionalità, si è deciso di raggruppare in un unico package le classi che gestiscono la logica applicativa (ApplicationLogicLayer) e la memorizzazione dei dati (StorageLayer). Quindi, per ogni funzionalità si avranno due package suddivisi nel seguente modo:

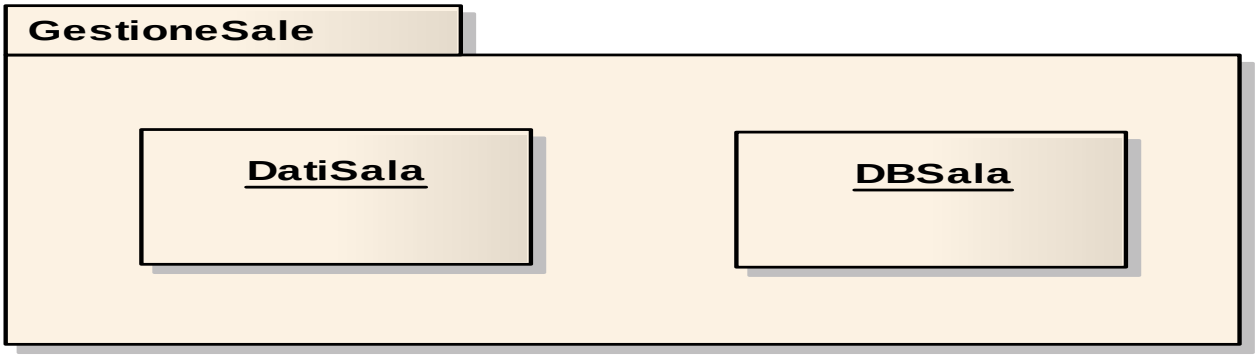
- GestioneXXX : sezione riguardante la logica applicativa e la memorizzazione dei dati.
- GuiXXX : sezione contenente le interfacce grafiche per l'interazione User/Sistema.

2.1 Gestione Accessi

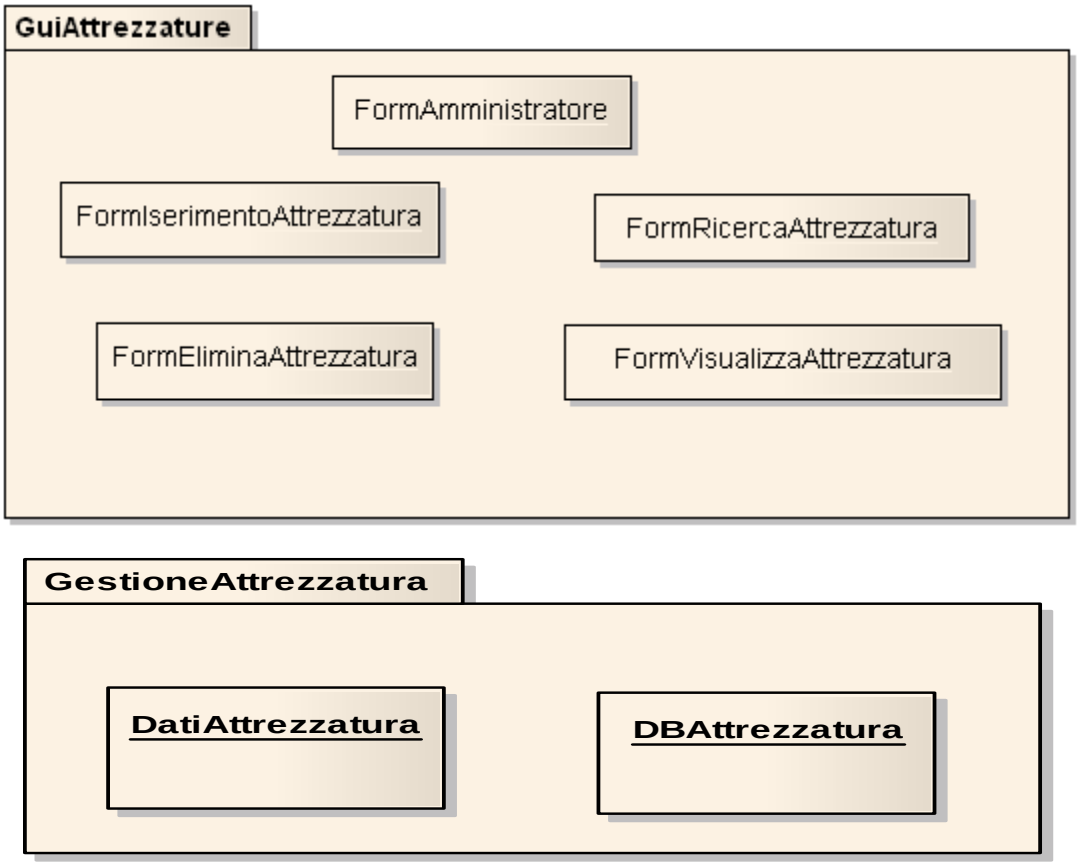


2.2 Gestione Sale

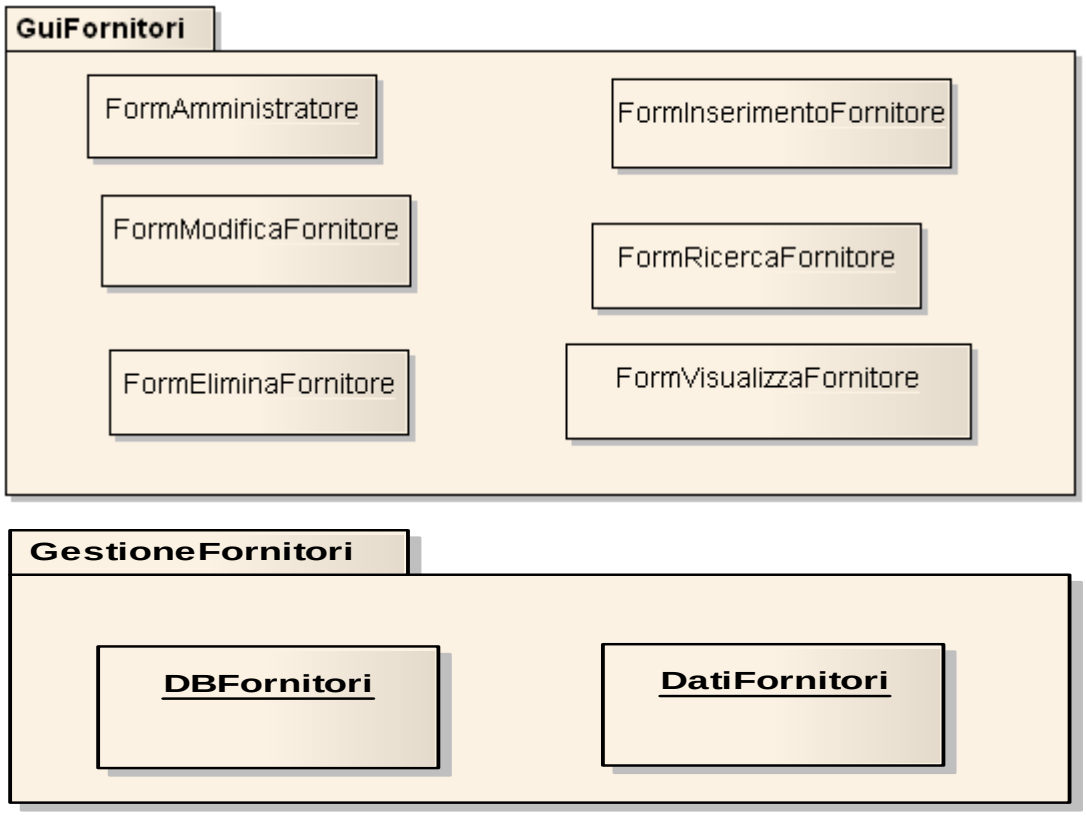




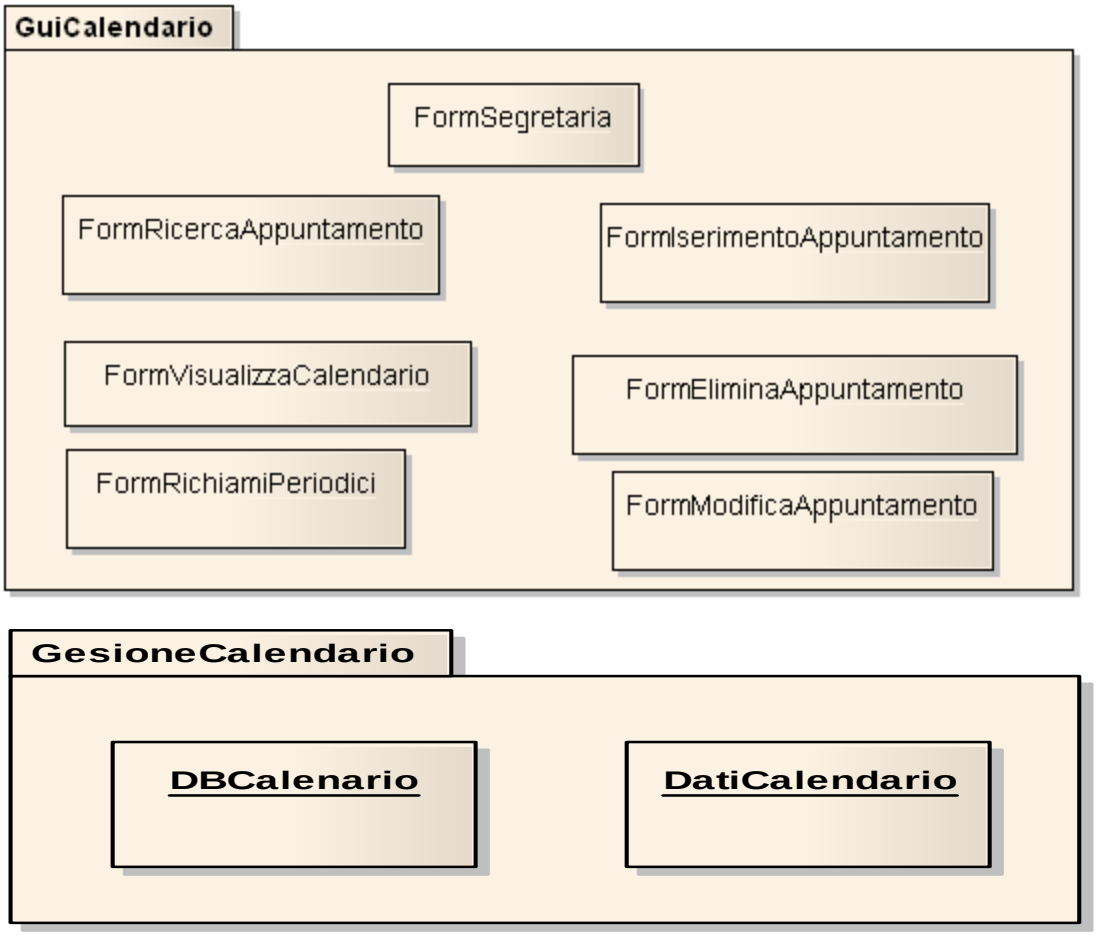
2.3 Gestione Attrezzature



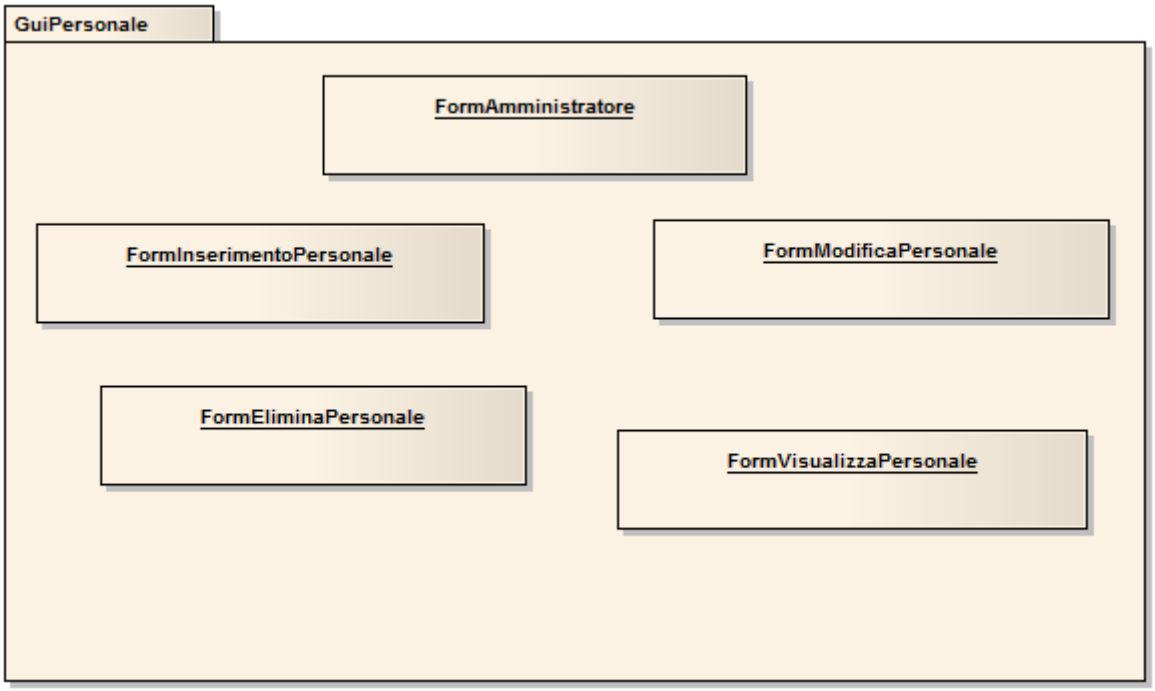
2.4 Gestione Fornitori

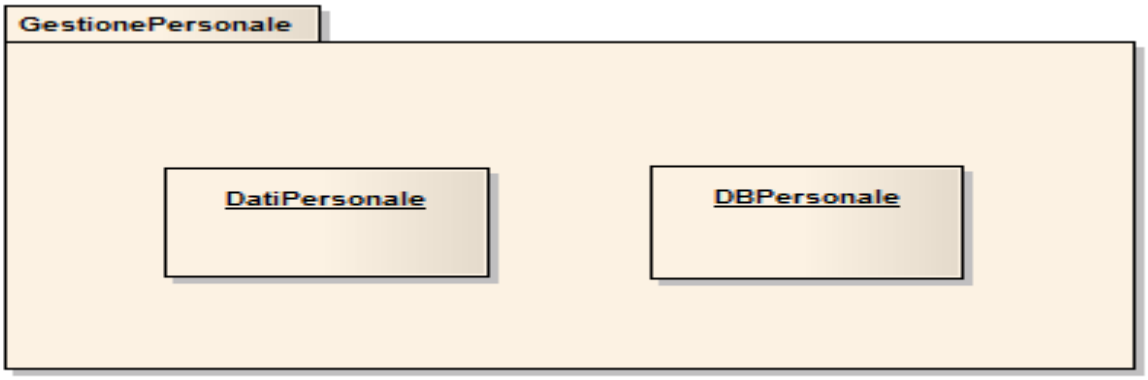


2.5 Gestione Calendario



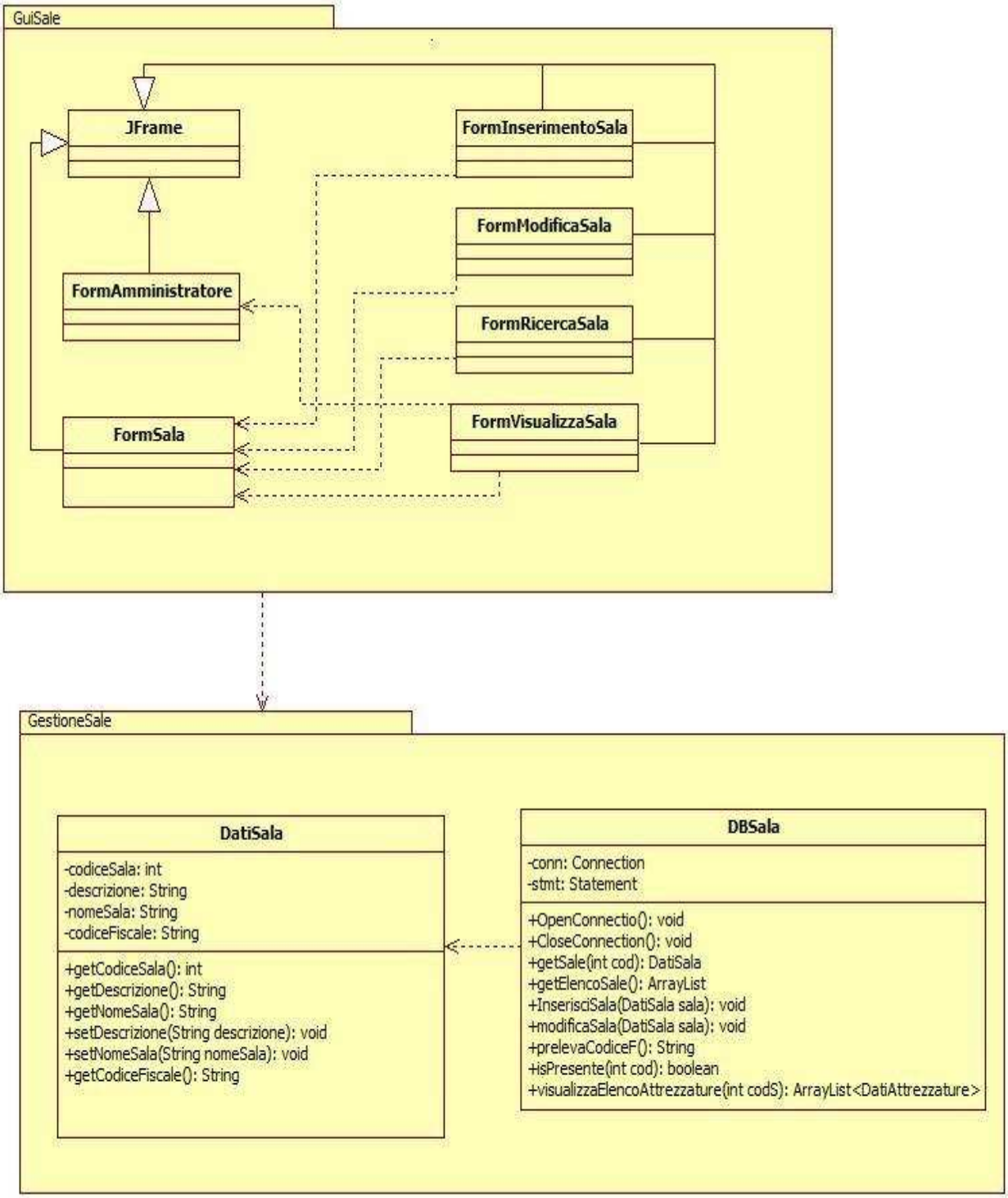
2.6 Gestione Personale





3.Class Interfaces

3.1 GestioneSale



Di seguito viene fornito un resoconto sui contratti legati a ciascuna classe:

NomeClasse: DatiSala

Invarianti: In una sala il codice è unico e identifica la sala.

PreCondizione: -

PostCondizione: +setNomeSala(String nomeSala): nomeSala devrebbe essere differente dal precedente.

OCL Standard:

Context DatiSala inv ::getCodiceSala() not null.

Context DatiSala :: set NomeSala(String nomeSala)
post:nomeSala<>@getNomeSala().

NomeClasse: DBSala

Invarianti: -

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

+isPresente(int cod):il codice della sala deve essere diverso da null.

+getSala(int cod) :cod deve riferirsi al codice di una sala presente nel DataBase.

+inserisciSala: (DatiSala sala): sala non deve essere presente nel database.

+modificaSala : (DatiSala sala):sala deve essere presente nel database.

PostCondizione: -

OCL Standard:
Context DBSala::getSala(int cod) pre:
codice >0 int AND not null.

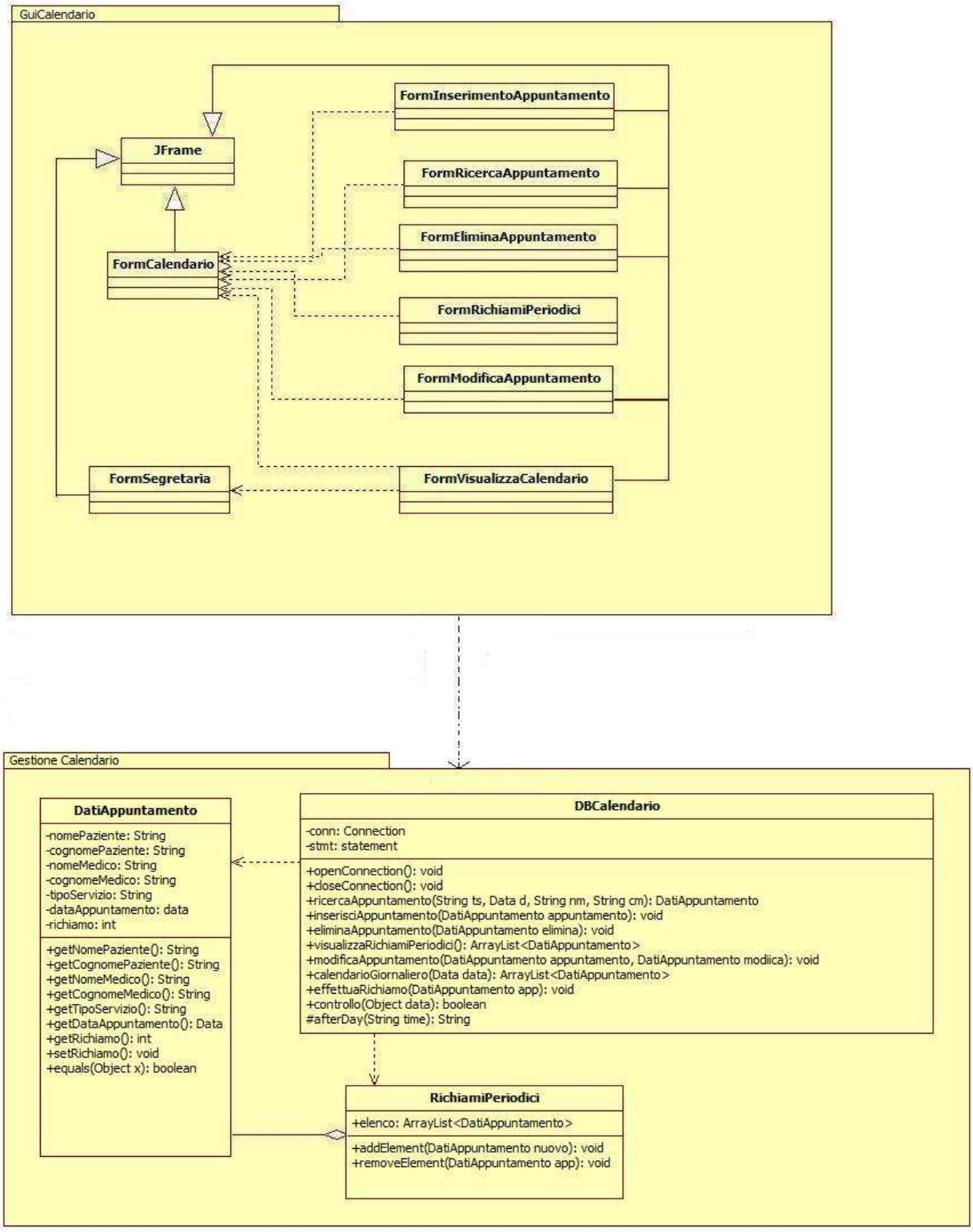
Context DBSala::inserisciSala(DatiSala sala) pre:
not isPresente(int cod)

Context DBSala::modificaSala(DatiSala sala) pre:
isPresente(int cod).

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

Context DBSala::isPresente(int cod) pre:
cod<>null.

3.2 Gestione Calendario



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

NomeClasse: DatiAppuntamento

Invarianti: Nel momento in cui la segretaria accede al sistema la dataAppuntamento deve essere quella corrente non può essere una data successiva o precedente al giorno in cui accede al sistema.

Pre-condizione: +getTipoServizio():il tipo del servizio non può essere nullo.

+getDataAppuntamento():la data dell'appuntamento,compresa l'ora,non può essere nulla e il formato deve essere GG\MM\AAAA HH:MM.

Post-condizione: -

OCL Standard:

Context DatiAppuntamento inv: getDataAppuntamento>= data corrente

Context DatiAppuntamento:: getTipoServizio() pre:getTipoServizio not null

ContextDatiAppuntamento::getDataAppuntamento() pre:: getDataAppuntamento() not null AND getDataAppuntamento().controllo (Object data).

NomeClasse:DBCalendario

Invarianti: -

Pre-condizione:

+closeConnection: la connessione deve essere aperta

+inserisciAppuntamento(DatiAppuntamento appuntamento):l'appuntamento che viene inserito non deve essere già presente nel DataBase.

+ricercaAppuntamento(DatiAppuntamento appuntamento):l'appuntamento da ricercare deve essere presente nel DataBase.

+eliminaAppuntamento(DatiAppuntamento appuntamento):l'appuntamento da eliminare deve essere presente nel DataBase.

Post-condizione:

+modificaAppuntamento(DatiAppuntamento appuntamento).il nuovo appuntamento è differente da quello precedente.

OCL Standard:

Context DBCalendario :: inserimentoAppuntamento(DatiAppuntamento appuntamento) pre: not isPresente(appuntamento).

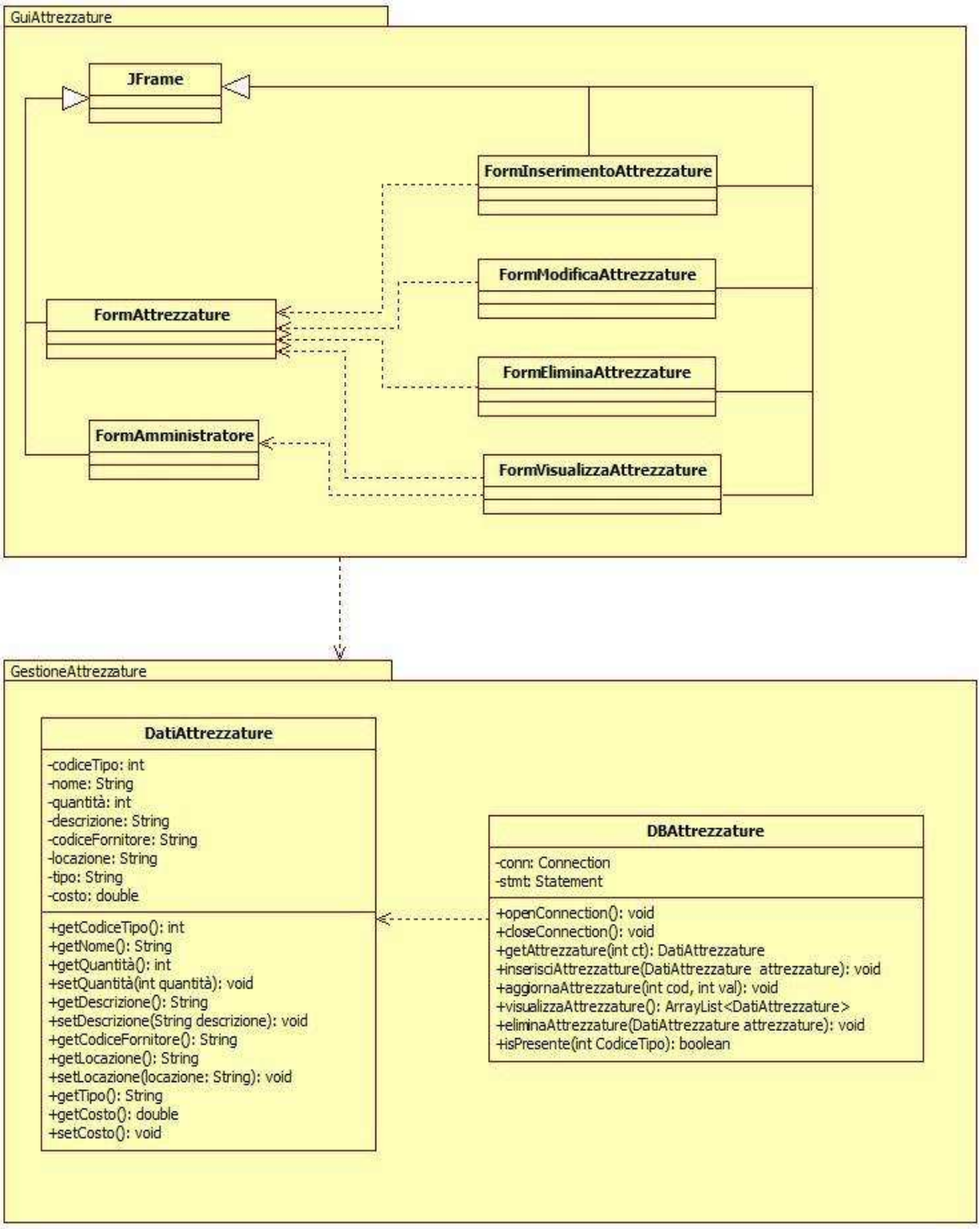
Context DBCalendario ::ricercaAppuntamento(DatiAppuntamento appuntamento) pre: isPresente(appuntamento)

Context DBCalendario::eliminaAppuntamento(DatiAppuntamento appuntamento) pre: isPresente(appuntamento)

Context DBCalendario::modificaAppuntamento(DatiAppuntamento appuntamento) post: not equals(precedente).

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

3.3 Gestione Attrezzature



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

Nome Classe: DatiAttrezzature

Invarianti: -

Pre-Condizione: + getCodiceTipo():il codice tipo è unico per ogni attrezzature.

+set Quantità(quantità): la quantità deve essere un numero maggiore di zero, perché se uguale a zero non sarebbero presenti attrezzature nel centro.

Post-Condizione: -

OCL Standard :

Context DatiAttrezzature: getCodiceTipo(attretture) pre:

attrezzature.CodiceTipo<>null

Context DatiAttrezzature: set Quantità(attrezzature) pre:

attrezzature.quantità > 0

*NomeClasse:*DBAttrezzature

Invarianti: all'inserimento di una nuova attrezzature deve essere sempre specificata la locazione di quest'ultima in modo che l'amministratore possa avere sempre un riferimento "fisico" della posizione, dell'attrezzatura, per futuri reperimenti o spostamenti.

Pre-Condizioni:

+inserisciAttrezzature(Datiattrezzature attrezzature): il codice tipo delle attrezzature non deve essere nullo.

+aggiorna Attrezzature(int cod,int val):il codice deve essere presente nel Database.

+getAttrezzature(int ct): il valore di ct inserito non deve essere un valore nullo.

+isPresente(int codiceTipo):il codiceTipo dell'attrezzatura deve essere diverso da null.

+ closeConnection(): la connessione deve essere aperta.

Post-Condizioni: -

OCL Standard :

Context DBAttrezzature inv: getLocazione()<>null.

Context DBAttrezzature:: inserisciAttrezzature(DatiAttrezzature attrezzature) pre:

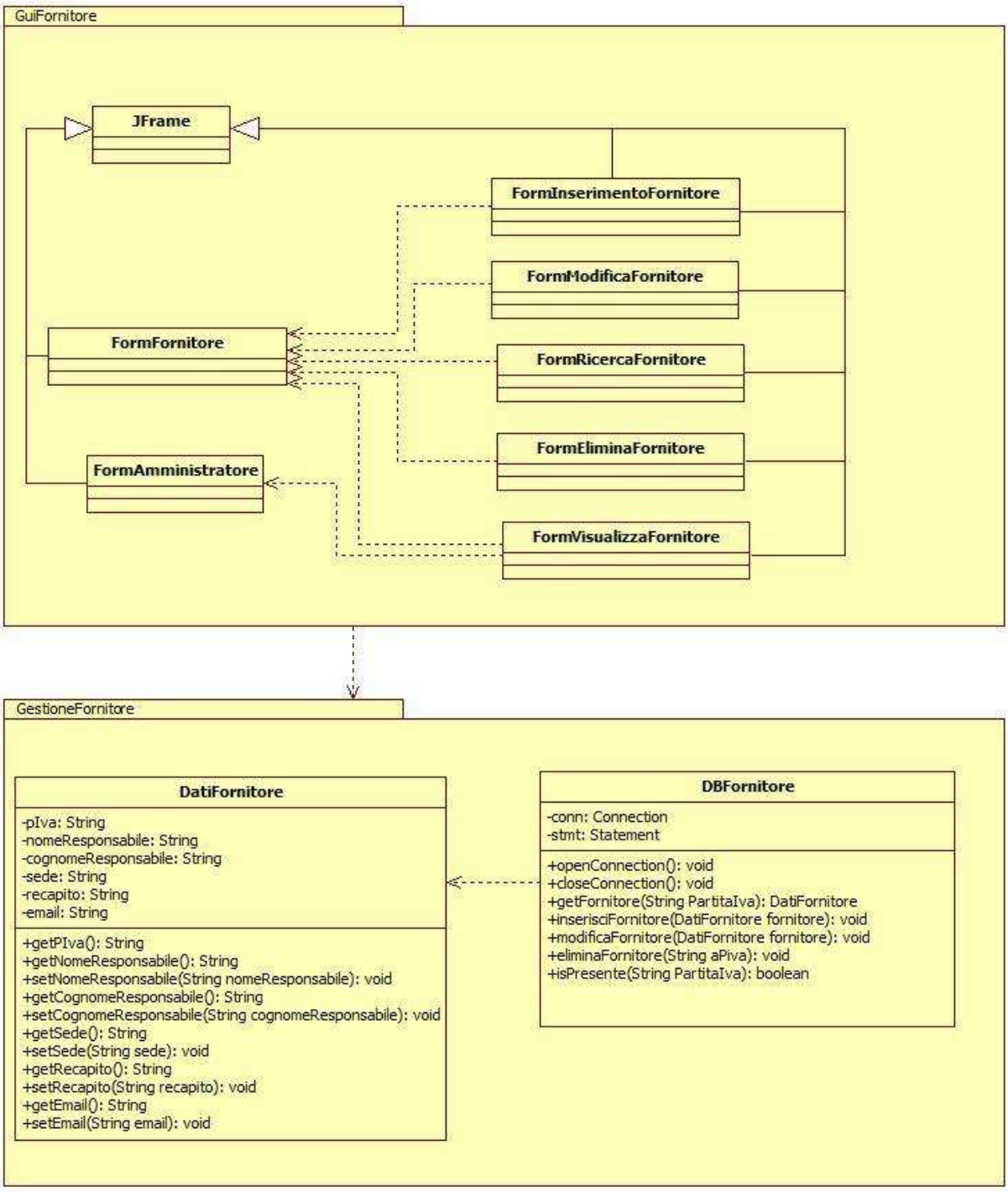
attrezzatura.codiceTipo <> null .

Context DBAttrezzature:: getAttrezzature(int ct) pre: ct <> null.

Context DBAttrezzature::aggiornaAttrezzature(int cod,int val) pre: cod<>null.

Context DBSala::isPresente(int codiceTipo) pre: codiceTipo<>null.
Context DBAttrezzature:: closeConnection() pre: connection.status=open

3.4 Gestione Fornitori



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

Nome Classe: DatiFornitore

Invarianti: la partita IVA è unica per ogni fornitore.

Pre-Condizione: +setNomeResponsabile(String nomeResponsabile): il nome del responsabile non può essere nullo e non è lo stesso che si va a modificare

+: +setCognomeResponsabile(String cognomeResponsabile): il cognome del responsabile non può essere nullo e non è lo stesso che si va a modificare

Post-Condizione: -

OCL Standard :

Context DatiFornitore: getPIva() inv: getPIva<> null.

Context DatiFornitore: setNomeResponsabile(String nomeResponsabile) pre:

nomeResponsabile not null AND not isPresente(nomeResponsabile).

Context DatiFornitore: setCognomeResponsabile(String cognomeResponsabile) pre:

cognomeResponsabile not null AND not isPresente(cognomeResponsabile).

NomeClasse: DBFornitore

Invarianti: -

Pre-Condizioni: +inserisciFornitore(DatiFornitore fornitore): il fornitore non deve essere nullo.

+getFornitore(String PartitaIva): non deve ritornare un valore nullo.

+ closeConnection(): la connessione deve essere aperta.

+modificaFornitore(DatiFornitore fornitore): il fornitore deve essere presente nel DataBase e non è lo stesso che si va a modificare.

+eliminaFornitore(String aPiva): aPiva deve essere presente nel DataBase.

+isPresente(String PartitaIva):PartitaIva del fornitore deve essere diverso da nullo.

Post-Condizioni:-

OCL Standard :

Context DBFornitore:: inserisciFornitore(DatiFornitore fornitore) pre:

fornitore<> null .

Context DBFornitore:: getFornitore(String PartitaIva) pre:

PartitaIva <> null.

Context DBFornitore:: modificaFornitore(DatiFornitore fornitore) pre:

IsPresente(fornitore) AND fornitore not equals(precedente)

Context DBFornitore:: eliminaFornitore(String aPiva) pre:

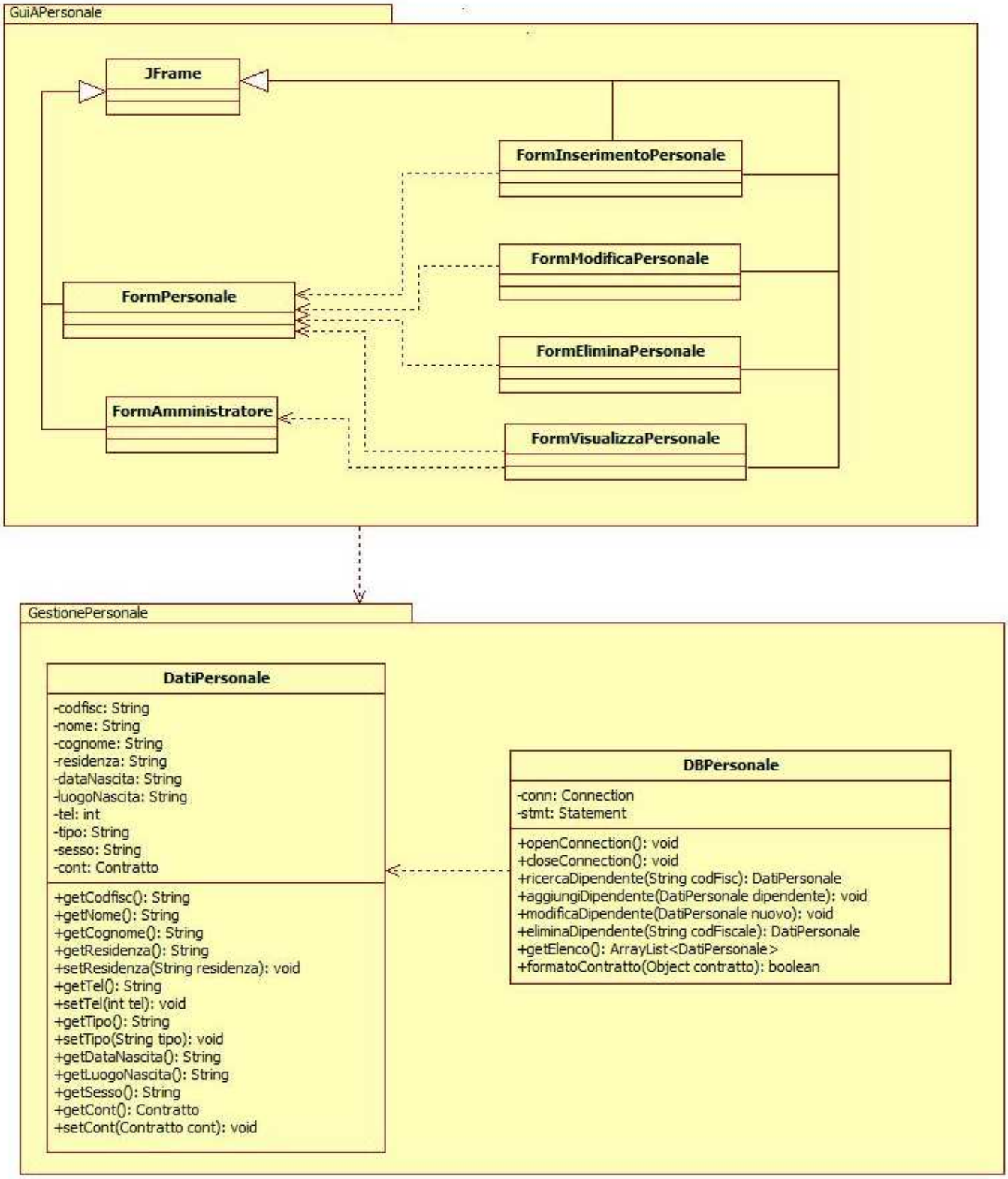
IsPresente(String PartitaIva).

Context DBSala::isPresente(String PartitaIva) pre:
partitaIva<>null.

Context DBFornitore:: closeConnection() pre:

connection.status=open

3.5 Gestione Personale



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

Nome Classe: DatiPersonale

Invarianti: codice fiscale è unica per ogni dipendente.

Pre-Condizione: +setCont(Contratto cont):cont deve essere inserito prima cont.tipo poi cont.salario e infine cont.data.

Post-Condizione: -

OCL Standard :

Context DatiPersonale: getCodisc() inv:

getCodfisc<> null

Context DatiPersonale:+ setCont(Contratto cont):pre:

cont.formatoContratto(Object contratto).

NomeClasse:DBPersonale

Invarianti: -

Pre-Condizioni: +ricercaDipendente (String codFisc): il valore codFisc deve essere presente nel DataBase.

+ closeConnection(): la connessione deve essere aperta.

+modificaDipendente(DatiPersonale nuovo): il valore di nuovo non può essere nullo e non è lo stesso che si va a modificare.

+eliminaDipendente(DatiPersonale codFiscale): il valore di codFiscale deve essere presente nel DataBase.

Post-Condizioni:-

OCL Standard :

Context DBFornitore:: modificaDipendente(DatiPersonale nuovo) pre:

nuovo<>null AND nuovo not equals(precedente)

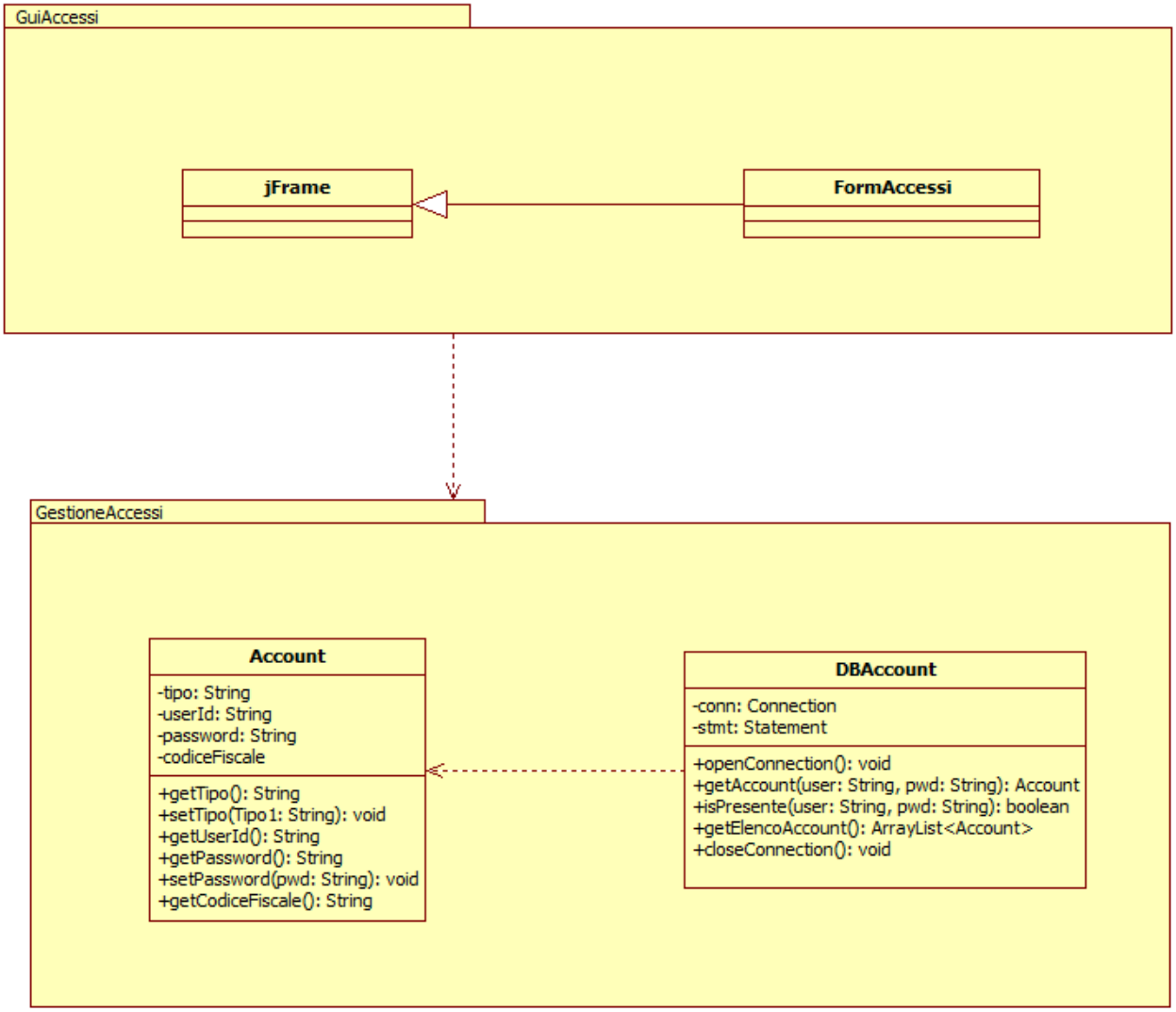
Context DBFornitore:: eliminaDipendente(DatiPersonale codFiscale)
pre:IsPresente(codFiscale).

Context DBFornitore:: ricercaDipendente(String codFisc) pre:IsPresente(codFisc).

Context DBFornitore:: closeConnection() pre:

connection.status=open

3.6 Gestione Account



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

Nome Classe: Account

Invarianti: l'userId e password sono unici per ogni account, e non possono essere nulli.

Pre-Condizione: -

Post-Condizione: -

OCL Standard :

Context Account: +getUserId() inv:

getUserId <>null.

Context Account: + getPassword() inv:

getPassword() <>null AND getPassword() >0.

NomeClasse:DBAccount

Invarianti: -

Pre-Condizioni: +getAccount(String userId, String password): l'account ricercato deve essere presente del DataBase.

+closeConnection(): la connessione deve essere aperta.

+isPresente(String userId, String password):l'userId e la password devono essere presenti nel DataBase.

Post-Condizioni: -

OCL Standard :

Context DBAccount:: isPresente(String userId, String password) pre:
userId <> null AND password <> null;

Context DBAccount:: getAccount(String userId, String password) pre:
account=>exists(Account.userID=user AND
Account.password= password)=true.

4. Glossario

Terms	Definiscion
FullDent	Nome del sistema sviluppato
Utente Generico	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).
Amministratore	Termine che identifica l'utente che accede al sistema come Amministratore e può gestire tutte le funzionalità di amministrazione permesse dal prodotto software.
Segretaria/o	Termine che identifica l'utente che accede al sistema come Segretaria e può gestire tutte le funzionalità di segreteria permesse dal prodotto software.
Medico	Termine che identifica l'utente che accede al sistema come Medico. Il termine raggruppa lo staff medico che opera nel centro (odontoiatri, igienisti etc.)
Operatore	Termine che identifica i vari dipendenti che lavorano nel centro (inservienti, medici,segretari,assistenti etc.)
Dati Personale/Pazienti	Termine con cui si intendono i dati non medici relativi ad un paziente o i dati relativi ad un membro del personale medico.
Cartella Clinica	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.
Fornitori	Termine che identifica i grossisti, dai quali il centro acquista i prodotti necessari per l'operato del centro.

Atrezzatura/e	Termine che identifica una qualsiasi apparecchiatura presente nel centro.
Sala/Stanza	Una stanza facente parte del centro e comprendente una serie attrezzature.
Report	Documento stampato contenente dati relativi ad una cartella clinica o ad un membro del personale medico.
Form	Componente di base per la creazione dell'interfaccia dell'applicazione del prodotto software
Account	Termine utilizzato per descrivere i dati di un utente registrato nel sistema (userID, password e tipo).
Hardware	Parti fisiche legate al sistema sviluppato
	L'insieme di applicazioni e programmi facenti parte del sistema sviluppato
Desig goals	Qualità individuate nel software che devono essere ottimizzate
Accoppiamento (Coupling)	Unita di misura per esprimere l'interdipendenza tra due sottosistemi o classi
Mapping	Funzione matematica che fa corrispondere a un elemento (o un insieme di elementi) di un modello ogni elemento di un altro modello. Nel caso specifico il termine mapping viene utilizzato, nel documento proposto, per illustrare la corrispondenza tra hardware e software
Boundary conditions	Condizioni particolari che devono essere gestite dal sistema proposto
client	Macchina che funziona da terminale e che inoltra le richiesta al Server

server	Macchina sulla quale è memorizzata gran parte del sistema e che fornisce servizi al Client
Three-tiers	Tipo di architetture basata su tre strati e utilizzata nella progettazione del sistema proposto
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 sistema
GUI	Acronimo di Graphic User Interface (interfaccia grautente), ossia l'insieme di oggetti che l'utente vede sullo schermo e con 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
Accessi(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 condividano 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 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