Versione 1.1

HotelMgM

Daniele Calabrò
Francesco Orciuoli
Raffaele Costantino

[OBJECT DESIGN DOCUMENT (ODD)]

Indice

1.Introduzione

1.1Compromessi dell'object design

In seguito alle scelte intraprese nei documenti precedenti, l'ODD farà riferimento alle sole funzionalità del sistema implementate, le uniche funzionalità che non andremo ad implementare saranno:

- Sottosistema Amministratore:
  PacchettoGestioneCamereAmministratore:
  - Cancellare una Camera
  - Modificare una Camera

  PacchettoGestioneAccount:
  - Cancellare un Operatore
  - Modificare Operatore

- Sottosistema Operatore1:
  PachettoGestioneClienti:
  - Modificare dati Cliente
  - Stampare Fattura del Cliente
  - Visualizzare Fattura del Cliente

- Sottosistema Operatore2:
  PachettoGestioneExtra:
  - Stampare Menù Bar

- Sottosistema UtenteFinale:
  PachettoGestioneWeb:

  - Ricerca Disponibilità Camera On-Line
  - Prenotare Camera On-Line
  - Stampare Ricevuta Prenotazione

Tutte le altre funzionalità specificate nel documento RAD sono state implementate. Le soluzioni progettuali individuate nel corso della fase di System Design impongono che la gestione dei dati persistenti che risiedono all'interno del database principale vengano gestite attraverso l'utilizzo delle componenti JDBC.

Linee guida per la documentazione di interfacce

L'interfaccia del prodotto HotelMgM è composta da oggetti molto comprensibili all'utente che vanno a chiarire immediatamente la propria funzione.
Gli sviluppatori del sistema dovranno seguire alcune linee guida durante la scrittura del codice:

- I nomi dei metodi dovranno essere formulati secondo questa configurazione: nomeMetodo( )

- I metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile(), setNomeVariabile().

- I commenti alle classi, ai metodi e alle variabili di istanza dovranno seguire lo standard Javadoc, quindi iniziare con /** e terminare con */.

Sicurezza ed Efficienza

La gestione della sicurezza viene affidata all'utilizzo di username e password da parte di ogni utente interessato all'accesso e quindi all'utilizzo del sistema HotelMgM.
Con l'utilizzo di username e password ogni utente prima di utilizzare il software, è tenuto appunto ad effettuare l'autenticazione. L'utente a qual punto sarà in grado di visualizzare e quindi utilizzare la parte di software a lui dedicata in base ai suoi privilegi d'accesso.
Questa politica di permessi permette di non appesantire eccessivamente il software.

Comprensibilità del Codice

Il codice deve essere più comprensivo possibile in modo da poter essere interpretato da altri programmatori che non hanno partecipato al progetto. Una seconda motivazione è anche per non accrescere la difficoltà dello sviluppo nella fase di testing. Il codice dove necessario sarà commentato in modo da migliorare la lettura delle righe di codice.

1.2Standard Adottati

Nella scrittura di codice durante l'implementazione i programmatori dovranno attenersi alle linee guida di seguito definite:

Stile di programmazione

HotelMgM come già annunciato verrà programmato in java lo stile di riferimento seguirà quasi completamente gli standard java noti.
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 rispecchiare il più possibile le responsabilità assunte dalla classe, tutti nomi che raggruppano devono iniziare con una lettera maiuscola.

```
Es.: public class DatiAccount
```

Tutti i nomi delle variabili devono iniziare con una lettera minuscola e i nomi che contengono in seguito devono iniziare con una lettera maiuscola, invece le variabili che hanno la caratteristica di essere costanti sono scritte interamente in maiuscolo.

```
Es.: private String codiceFiscale
private static final float PERC_IVA = 20.0
```

I nomi dei metodi devono iniziare con una lettera minuscola mentre le successive parole iniziano con una lettera maiuscola. Solitamente il nome di un metodo inizia con verbo e termina con un nome che rappresenta l'oggetto che è soggetto all'azione del verbo.

Es.: `inserisci Account()`

I nomi dei metodi di accesso iniziano con get, mentre quelli di modifica iniziano con set.

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

Documentazione

Dopo le eventuali dichiarazioni di import e package ogni classe dovrà avere un commento che avrà una breve descrizione della classe. Questi commenti verranno scritti in modo da poter essere convertiti in documenti HTML con il tool JavaDoc.

Es.:
```
/** Breve descrizione dello scopo della classe
*
*/
```

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
*
*/
```

1.3Definizioni, Acronimi e abbreviazioni

Per una perfetta comprensione dell'ODD da parte di qualsiasi operatore poco esperto, vengono di seguito riportati acronimi e abbreviazioni utilizzate nel documento.

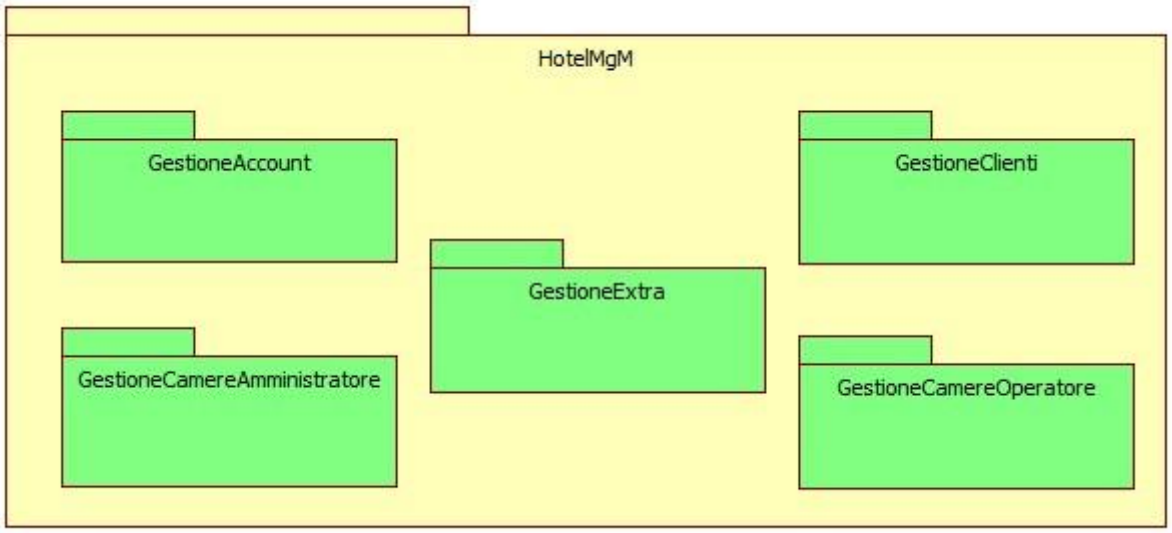
1. **RAD**: Requirements Analysis Document
2. **SDD** : System Design Document
3. **ODD**: Object Design Document
4. **DBMS**: Database Management System
5. **DB**: DataBase
6. **HotelMgM**: Nome del Sistema Software

1.4Riferimenti

HotelMgM SDD 1.2

2.Packages

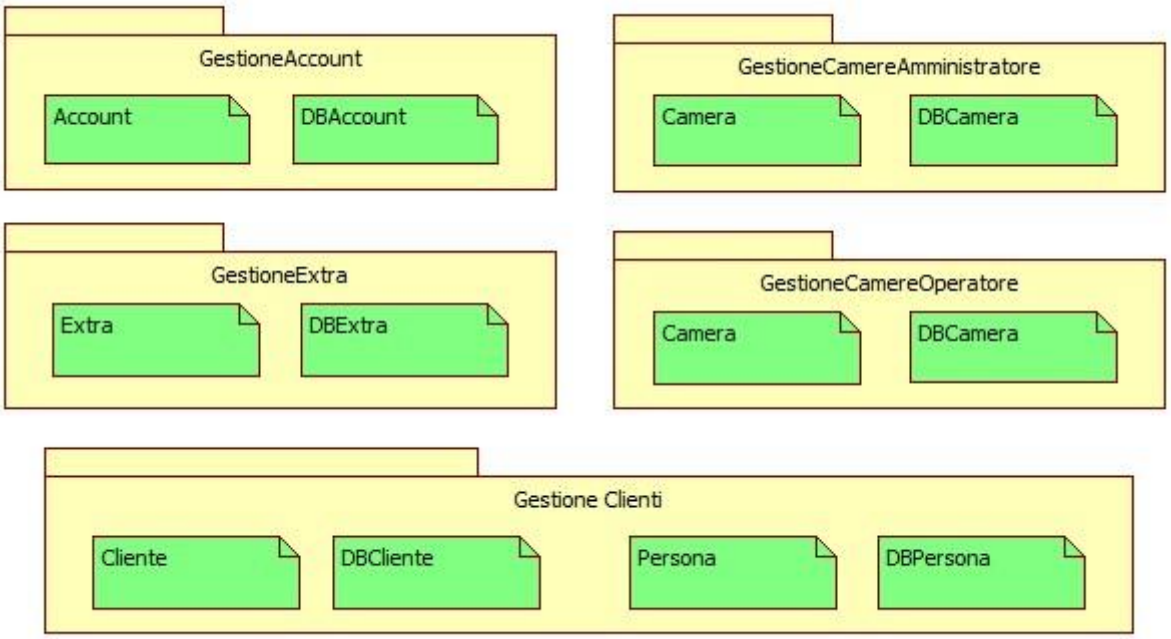
2.1HotelMgM



Il Package HotelMgM è il livello di astrazione più alto del prodotto software. Questo contiene al suo interno ogni sottopackage implementato come descritto nell' SDD.
A seguito dell'analisi del carico applicativo richiesto dalla realizzazione dei singoli sottopackage, si è scelto di implementare i seguenti sottopackage:

- GestioneAccount
- GestioneCamereAmministratore
- GestioneClienti
- GestioneCamereOperatore
- GestioneClienti
- GestioneExtra

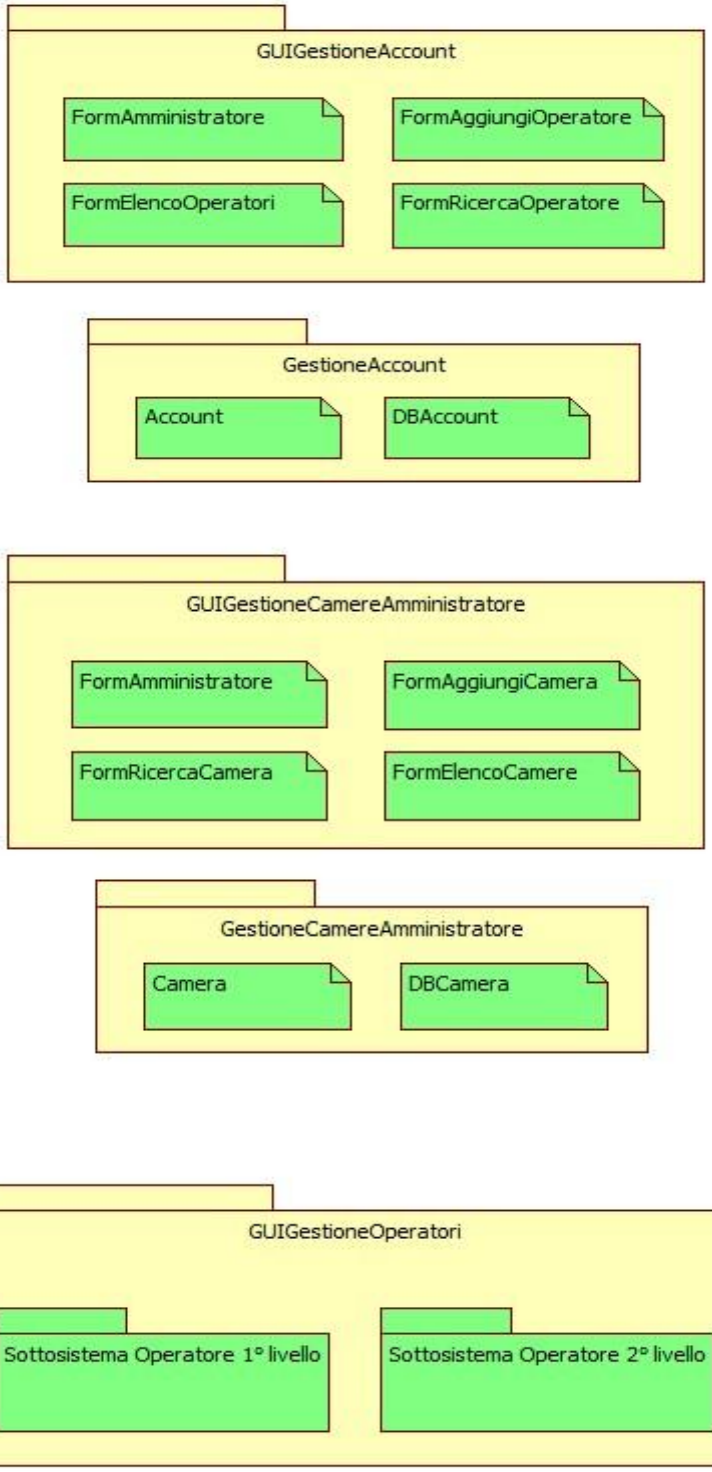
L'unico package che abbiamo deciso di escludere dall' implementazione è il package GestioneWeb.



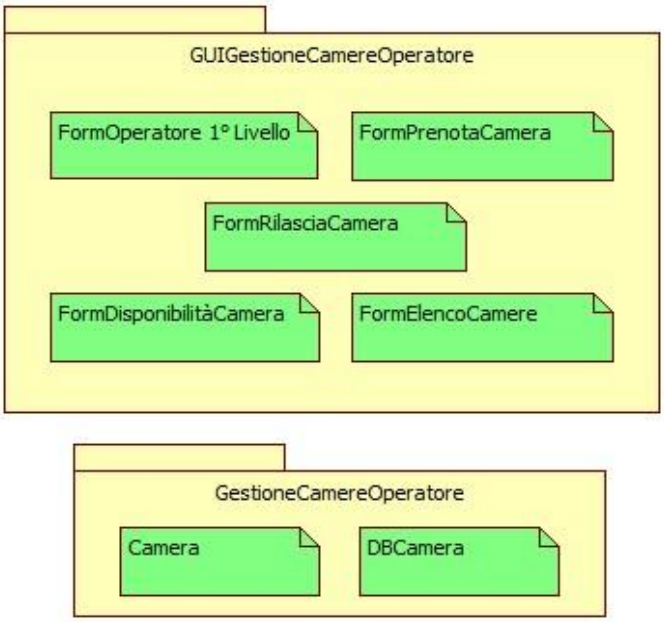
Le interfacce delle classi descritte in questo documento sono relative solo ai package descritti precedentemente.
Vi sono altre funzionalità di cui si è parlato in precedenza, necessarie per ottenere un corretto funzionamento del software. Queste funzionalità devono essere implementate in seguito, ed aggiunte al prodotto software sotto forma di classi fantoccio.
Nel prossimo passo vi sarà una descrizione più approfondita della composizione dei singoli package.
La descrizione delle funzionalità è strutturata in:

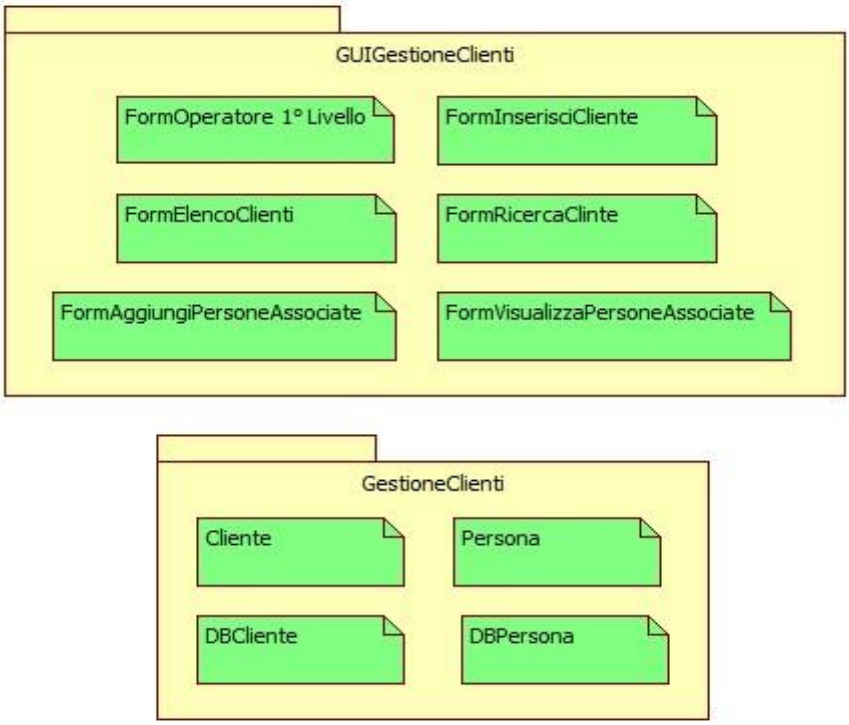
- **Gestione**: Logica Applicativa e Memorizzazione Dati
- **GUI**: Interazione con l'utente / Interfacce grafiche

2.2Gestione Sottosistema Amministratore

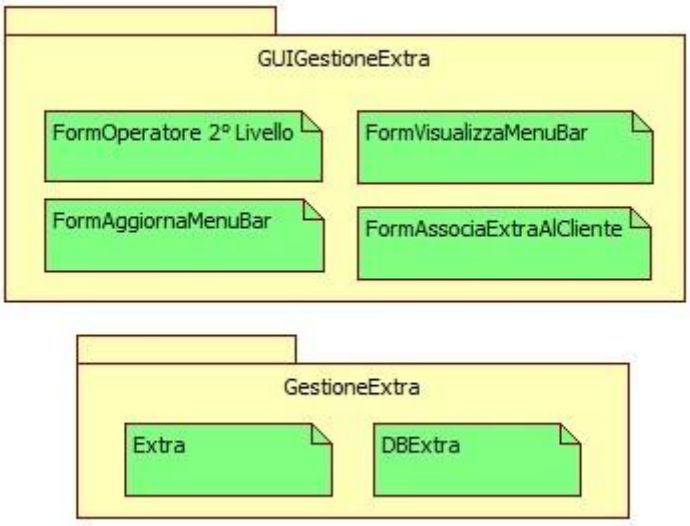


2.3Gestione Sottosistema Operatore 1° livello




GUIGestioneClienti
FormOperatore 1° Livello
FormInserisciCliente
FormElencoClienti
FormRicercaClinte
FormAggiungiPersoneAssociate
FormVisualizzaPersoneAssociate
GestioneClienti
Cliente
Persona
DBCliente
DBPersona

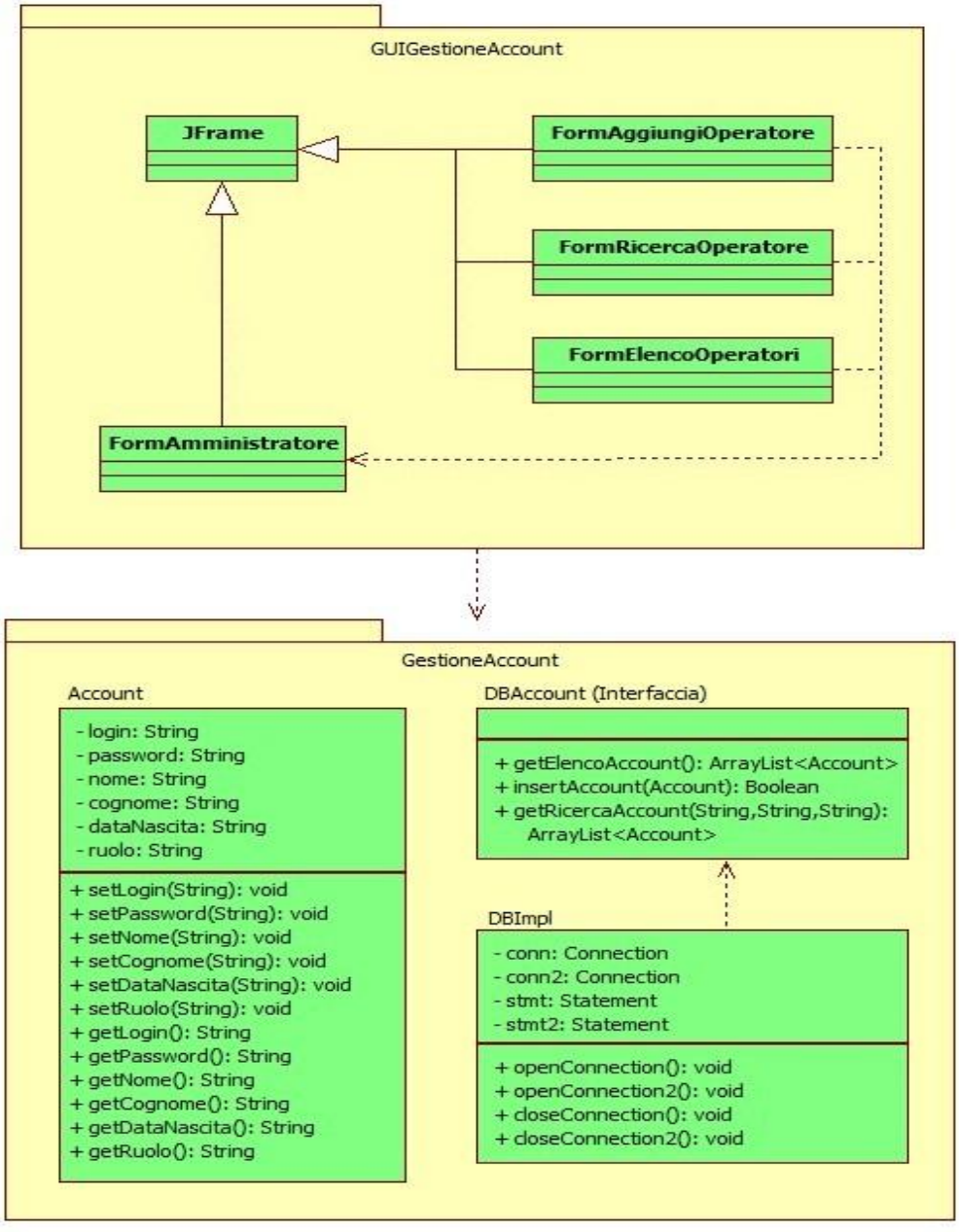
2.4Gestione Sottosistema Operatore 2° livello



3.Class interfaces

3.1Gestione Sottosistema Amministratore

3.1.1Gestione Account



Di seguito viene riportata la descrizione delle classi tramite JavaDoc:

GUIGestioneAccount:
FormAmministratore
FormAggiungiOperatore
FormElencoOperatori
FormRicercaOperatori

GestioneAccount:
Account
DBAccount
DBImpl

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:
NomeClasse: Account
Invariante: Login deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeInterfaccia: DBAccount
Invariante: -
Precondizioni: -
Postcondizioni: -

NomeClasse: DBImpl
Invariante: -
Precondizioni: + getElencoAccount(): ArrayList<Account> : gli account devono essere presenti nel database

+ getRicercaAccount(String,String,String): ArrayList<Account> : l'account ricercato deve essere presente nel database

+ openConnection(): void: il DataBase deve essere presente sul Server

+ openConnection2(): void : il DataBase deve essere presente sul Server

+ closeConnection(): void : ci deve essere una connessione al DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

Postcondizioni: + getElencoAccount(): ArrayList<Account> : getElencoAccount().size()>0

+ getRicercaAccount(String,String,String): ArrayList<Account> : account=->exists(Account.Login=Login) = true

+ getRicercaAccount(String,String,String): ArrayList<Account> : account=->exists(Account.Nome=Nome AND Account.Cognome=Cognome) = true

+ insertAccount(Account): Boolean: ((password.size()>6) && (password.size() <16))

+ insertAccount(Account): Boolean: nessun campo deve essere nullo

+ insertAccount(Account): Boolean: la data deve essere corretta

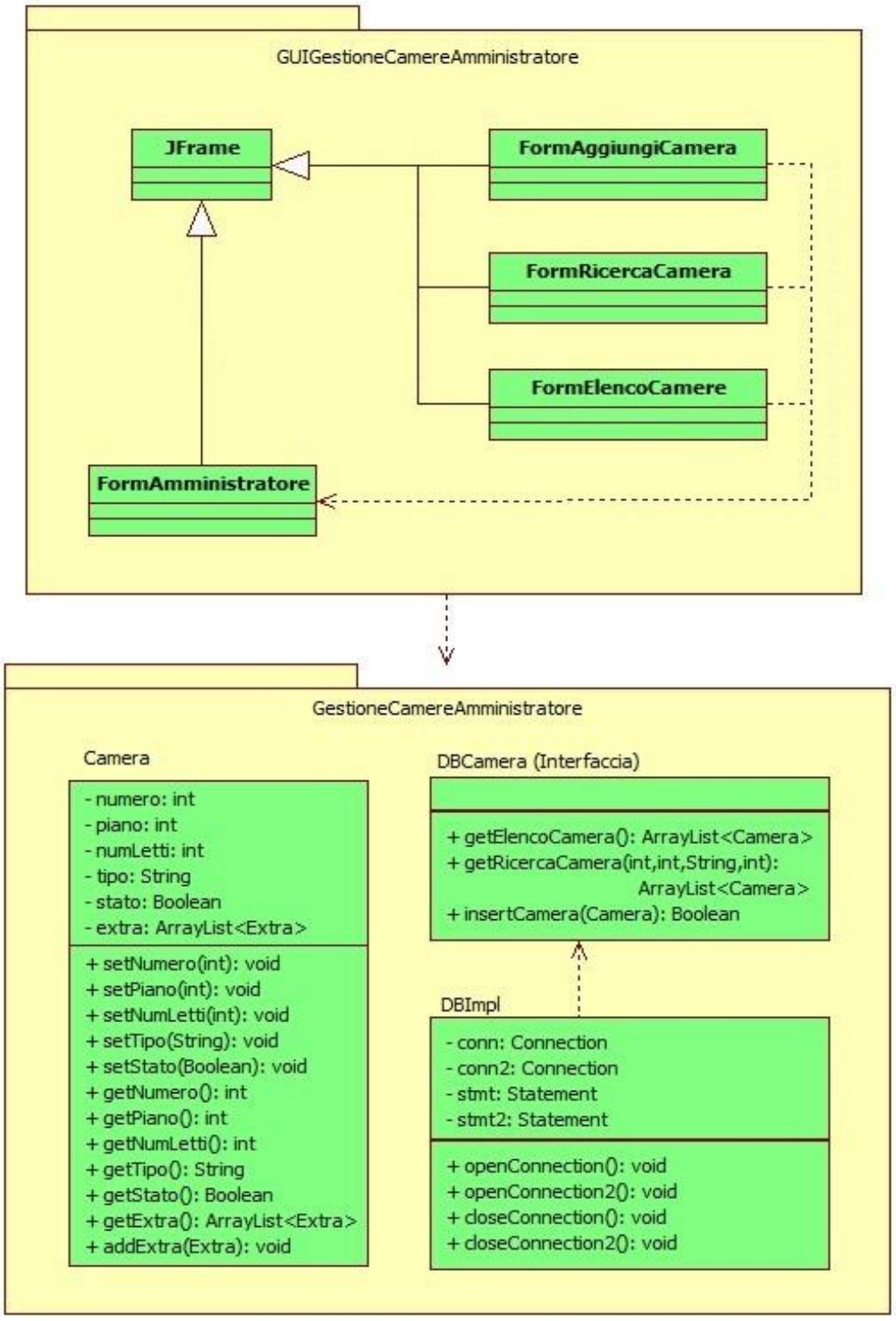
+ openConnection(): void: ((DBImpl.conn != NULL) && (DBImpl.stmt != NULL))

+ openConnection2(): void : ((DBImpl.conn != NULL) && (BImpl.stmt != NULL))

+ closeConnection(): void : ci deve essere una connessione al DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

3.1.2GestioneCamereAmministratore



Di seguito viene riportata la descrizione delle classi tramite JavaDoc:

GUIGestioneCamereAmministratore:
FormAmministratore
FormAggiungiCamera
FormRicercaCamera
FormElencoCamere

GestioneCamereAmministratore:
Camera
DBCamera

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:
NomeClasse: Camera
Invariante: Il Numero della camera deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeInterfaccia: DBCamera
Invariante: -
Precondizioni: -
Postcondizioni: -

NomeClasse: DBImpl
Invariante: -
Precondizioni: + getElencoCamera(): ArrayList<Camera> : le camere devono essere presenti nel database

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : la camera ricercata deve essere presente nel database

+ openConnection(): void: il DataBase deve essere presente sul Server

+ openConnection2(): void : il DataBase deve essere presente sul Server

+ closeConnection(): void : ci deve essere una connessione al

DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

Postcondizioni: + getElencoCamera(): ArrayList<Camera> : getElencoCamera().size()>0

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists(Camera.Numero=numero) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists(Camera.Piano=piano) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists(Camera.Tipo=tipo) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists(Camera.NumLetti=numLetti) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists((Camera.Numero=numero) AND (Camera.Piano=piano)) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists((Camera.Numero=numero) AND (Camera.NumLetti=numLetti)) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> :

camera=->exists((Camera.Numero=numero) AND (Camera.Tipo=tipo)) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists((Camera.NumLetti=numLetti) AND (Camera.Piano=piano)) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists((Camera.Tipo=tipo) AND (Camera.Piano=piano)) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists((Camera.Tipo=tipo) AND (Camera.NumLetti=numLetti)) =true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists((Camera.Numero=numero) AND (Camera.Piano=piano)AND(Camera.Tipo=tipo))=true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists((Camera.Numero=numero) AND (Camera.NumLeti=numLetti)AND(Camera.Tipo=tipo))=true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists((Camera.NumLetti=numLetti) AND (Camera.Piano=piano)AND(Camera.Tipo=tipo))=true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : camera=->exists((Camera.Numero=numero) AND (Camera.Piano=piano)AND(Camera.Tipo=tipo)AND Camera.NumLetti=numLetti))=true

+ getRicercaCamera(int,int,String,int): ArrayList<Camera> : ((Camera.numero==NULL) &&(Camera.piano==NULL) &&(Camera.tipo==NULL) &&(Camera.numLetti==NULL))

+ insertCamera(Camera): Boolean: nessun campo deve essere nullo

+ openConnection(): void: ((DBImpl.conn != NULL) && (DBImpl.stmt != NULL))

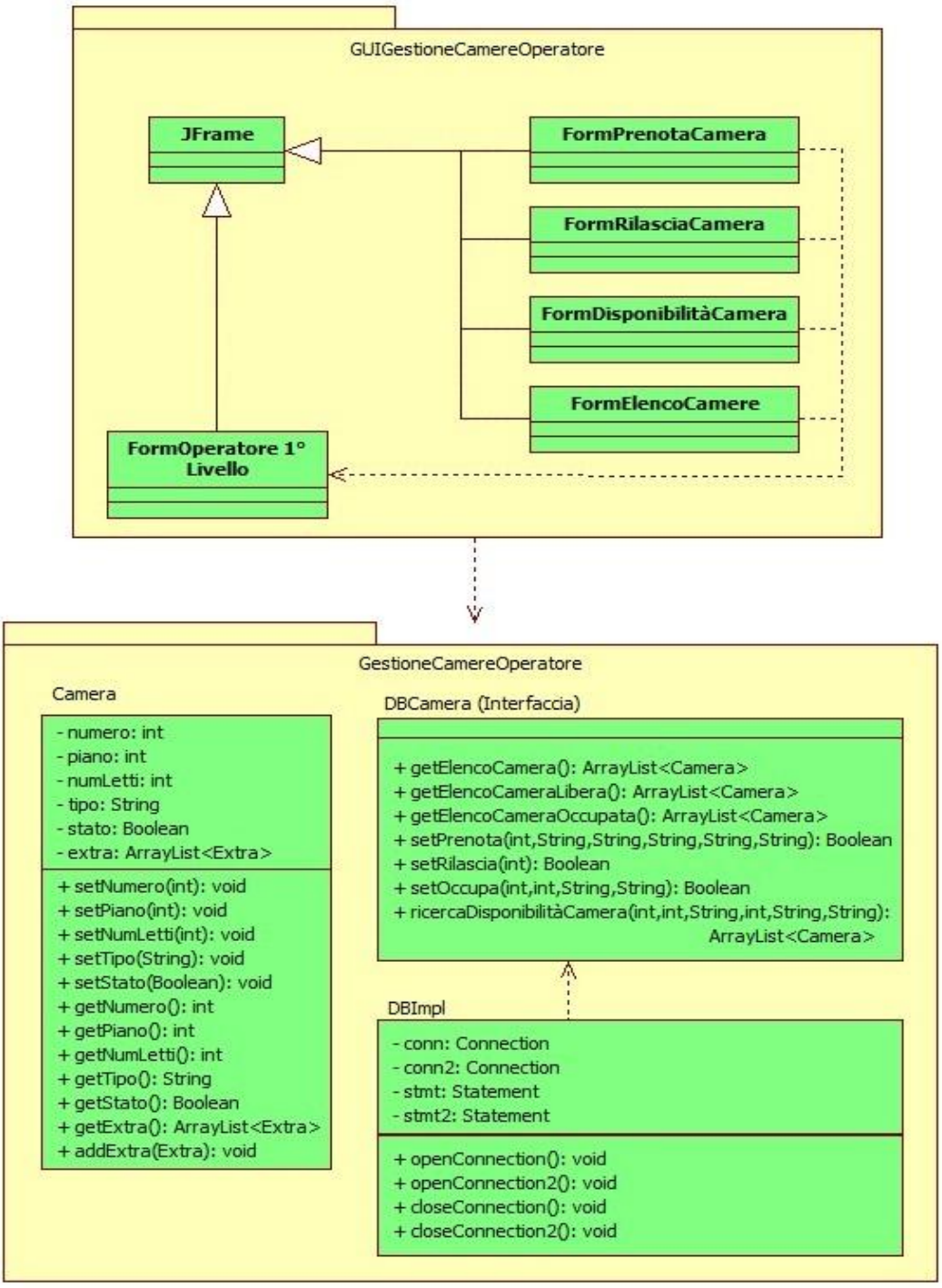
+ openConnection2(): void : ((DBImpl.conn != NULL) && (BImpl.stmt != NULL))

+ closeConnection(): void : ci deve essere una connessione al DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

3.2Gestione Sottosistema Operatore 1° livello

3.2.1GestioneCamereOperatore



Di seguito viene riportata la descrizione delle classi tramite JavaDoc:

GUIGestioneCamereOperatore:
FormOperatore 1° Livello
FormPrenotaCamera
FormRilasciaCamera
FormDisponibilitàCamera
FormElencoCamere

GestioneCamereOperatore:
Camera
DBCamera

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:
NomeClasse: Camera
Invariante: Il Numero della camera deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeInterfaccia: DBCamera
Invariante: -
Precondizioni: -
Postcondizioni: -

NomeClasse: DBImpl
Invariante: -
Precondizioni: + getElencoCamera(): ArrayList<Camera> : le camere devono essere presenti nel database

+ getElencoCameraLibera(): ArrayList<Camera> : le camere devono essere presenti nel database

+ getElencoCameraOccupata(): ArrayList<Camera> : le camere devono essere presenti nel database

+ setRilascia(int): Boolean : la camera deve essere occupata

+setOccupa(int,int,String,String): Boolean : la camera deve essere libera

+ setPrenota(int,String,String,String,String,String): Boolean : la camera deve essere libera

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String): ArrayList<Camera> : la camera ricercata deve essere presente nel database

+ openConnection(): void: il DataBase deve essere presente sul Server

+ openConnection2(): void : il DataBase deve essere presente sul Server

+ closeConnection(): void : ci deve essere una connessione al DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

Postcondizioni: + getElencoCamera(): ArrayList<Camera> : getElencoCamera().size()>0

+ getElencoCameraLibera(): ArrayList<Camera> : getElencoCamera().size()>0

+ getElencoCameraOccupata(): ArrayList<Camera> : getElencoCamera().size()>0

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
  ArrayList<Camera> :
  camera=->exists(Camera.Numero=numero) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
  ArrayList<Camera> :
  camera=->exists(Camera.Piano=piano) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
  ArrayList<Camera> :
  camera=->exists(Camera.Tipo=tipo) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
  ArrayList<Camera> :
  camera=->exists(Camera.NumLetti=numLetti) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
  ArrayList<Camera> :
  camera=->exists((Camera.Numero=numero) AND
  (Camera.Piano=piano)) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
  ArrayList<Camera> :
  camera=->exists((Camera.Numero=numero) AND
  (Camera.NumLetti=numLetti)) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
  ArrayList<Camera> :
  camera=->exists((Camera.Numero=numero) AND
  (Camera.Tipo=tipo)) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
  ArrayList<Camera> :
  camera=->exists((Camera.NumLetti=numLetti) AND
  (Camera.Piano=piano)) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):

ArrayList<Camera> :
camera=->exists((Camera.Tipo=tipo) AND
(Camera.Piano=piano)) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
ArrayList<Camera> :
camera=->exists((Camera.Tipo=tipo) AND
(Camera.NumLetti=numLetti)) =true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
ArrayList<Camera> :
camera=->exists((Camera.Numero=numero) AND
(Camera.Piano=piano)AND(Camera.Tipo=tipo))=true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
ArrayList<Camera> :
camera=->exists((Camera.Numero=numero) AND
(Camera.NumLeti=numLetti)AND(Camera.Tipo=tipo))=true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
ArrayList<Camera> :
camera=->exists((Camera.NumLetti=numLetti) AND
(Camera.Piano=piano)AND(Camera.Tipo=tipo))=true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
ArrayList<Camera> :
camera=->exists((Camera.Numero=numero) AND
(Camera.Piano=piano)AND(Camera.Tipo=tipo)AND
Camera.NumLetti=numLetti))=true

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
ArrayList<Camera> :
((Camera.numero==NULL) &&(Camera.piano==NULL)
&&(Camera.tipo==NULL) &&(Camera.numLetti==NULL))

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String):
ArrayList<Camera> : ((dataInizio!=NULL) &&(dataFine!=NULL))

+ getRicercaDisponibilitaCamera(int,int,String,int,String,String): ArrayList<Camera> : (data Inizio < dataFine)

+ setPrenota(int,String,String,String, String,String): Boolean : ((dataInizio!=NULL) &&(dataFine!=NULL) &&(dataNascita!=NULL))

+ setPrenota(int,String,String,String, String,String): Boolean : (data Inizio < dataFine)

+ setPrenota(int,String,String,String, String,String): Boolean : Nessun campo deve essere nullo

+ setPrenota(int,String,String,String, String,String): Boolean : Nessun campo deve essere nullo

+ setOccupa(int,int,String,String): Boolean : ((dataInizio!=NULL) &&(dataFine!=NULL))

+ setOccupa(int,int,String,String): Boolean : (data Inizio < dataFine)

+ setOccupa(int,int,String,String): Boolean : Nessun campo deve essere nullo

+ setRilascia(int): Boolean : Il campo NumeroCamera non deve essere nullo

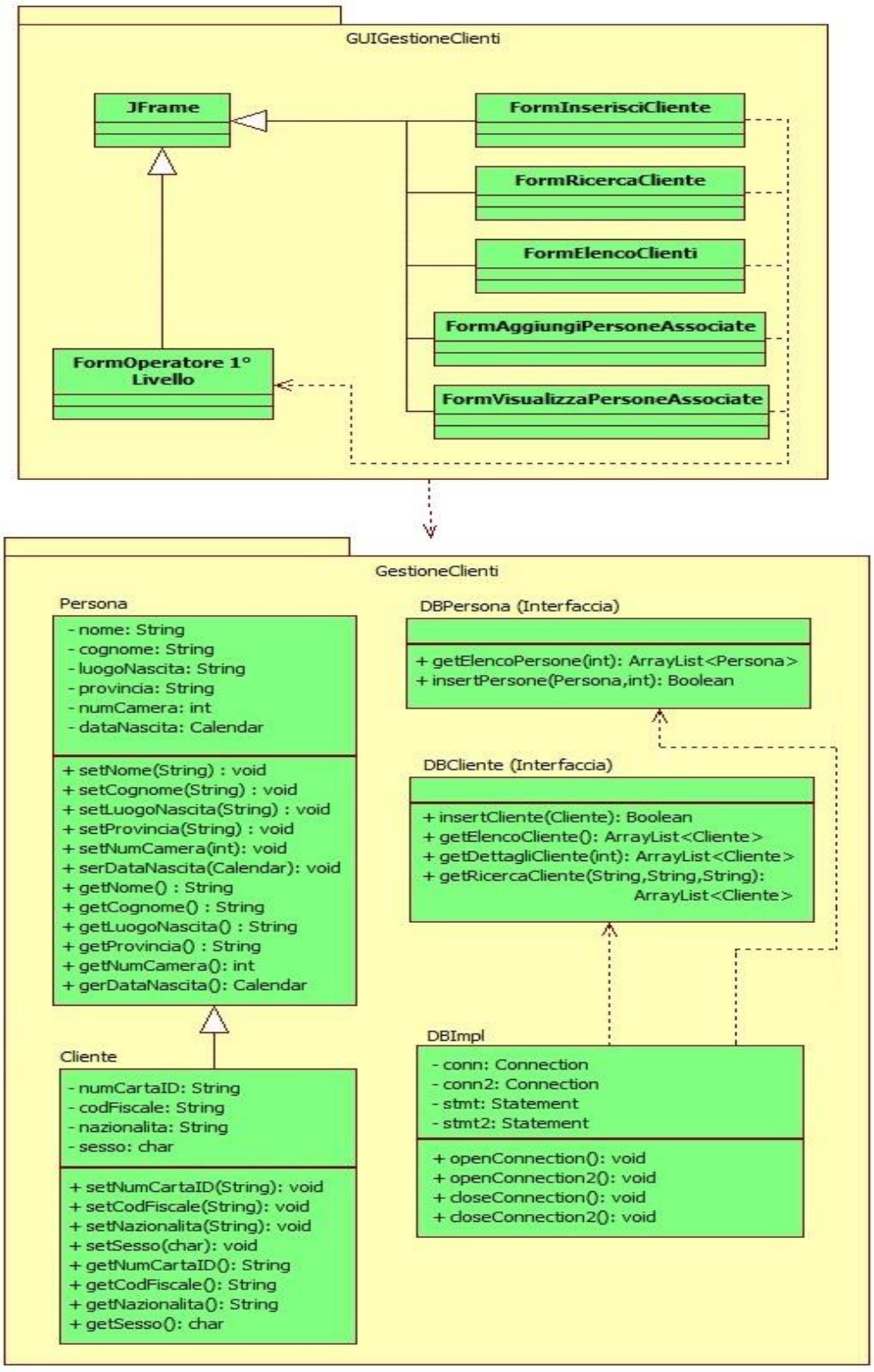
+ openConnection(): void: ((DBImpl.conn != NULL) && (DBImpl.stmt != NULL))

+ openConnection2(): void : ((DBImpl.conn != NULL) && (BImpl.stmt != NULL))

+ closeConnection(): void : ci deve essere una connessione al DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

3.2.2GestioneClienti



Di seguito viene riportata la descrizione delle classi tramite JavaDoc:

GUIGestioneClienti:
FormOperatore 1° Livello
FormInserisciClienti
FormElencoClienti
FormRicercaCliente
FormAggiungiPersoneAssociate
FormVisualizzaPersoneAssociate

GestioneClienti:
Cliente
DBCliente
Persona
DBPersona

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:
NomeClasse: Cliente
Invariante: Il Codice del cliente deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeInterfaccia: DBCliente
Invariante: -
Precondizioni: -
Postcondizioni: -

NomeClasse: Persona
Invariante: Il Codice della persona associata al cliente deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeInterfaccia: DBPersona
Invariante: -
Precondizioni: -
Postcondizioni: -

NomeClasse: DBImpl

Invariante: -

Precondizioni: + getElencoPersone(int): ArrayList<Persona> : il cliente e le persone devono essere presenti nel database

+ getElencoCliente(): ArrayList<Cliente> : i clienti devono essere presenti nel database

+ getDettagliCliente(int): ArrayList<Cliente> : il cliente deve essere presente nel database

+ getRicercaCliente(String, String,String): ArrayList<Cliente> : il cliente deve essere presente nel database

+ openConnection(): void: il DataBase deve essere presente sul Server

+ openConnection2(): void : il DataBase deve essere presente sul Server

+ closeConnection(): void : ci deve essere una connessione al DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

Postcondizioni: + getElencoPersone(int): ArrayList<Persona> : getElencoPersone(int).size()>0

+ insertPersona(Persona,int): Boolean : nessun campo deve essere nullo

+ insertCliente(Cliente): Boolean : nessun campo deve essere nullo

+ getElencoCliente(): ArrayList<Cliente> :
getElencoCliente().size()>0

+ getDettagliCliente(int): ArrayList<Cliente> :
getDettagliCliente(int).size()>0

+ getRicercaCliente(String,String,String): ArrayList<Cliente> :
cliente=->exists(Cliente.Nome=nome) =true

+ getRicercaCliente(String,String,String): ArrayList<Cliente> :
cliente=->exists(Cliente.Cognome=cognome) =true

+ getRicercaCliente(String,String,String): ArrayList<Cliente> :
cliente=->exists(Cliente.DataNascita=dataNascita) =true

+ getRicercaCliente(String,String,String): ArrayList<Cliente> :
cliente=->exists ((Cliente.Nome=nome) AND
(Cliente.Cognome=Cognome)) =true

+ getRicercaCliente(String,String,String): ArrayList<Cliente> :
cliente=->exists ((Cliente.Nome=nome) AND
(Cliente.DataNascita=dataNascita)) =true

+ getRicercaCliente(String,String,String): ArrayList<Cliente> :
cliente=->exists ((Cliente.Cognome=cognome) AND
(Cliente.DataNascita=dataNascita)) =true

+ getRicercaCliente(String,String,String): ArrayList<Cliente> :
cliente=->exists ((Cliente.Nome=nome) AND
(Cliente.Cognome=cognome) AND
(Cliente.DataNascita=dataNascita)) =true

+ getRicercaCliente(String,String,String): ArrayList<Cliente> : nessun campo deve essere nullo

+ openConnection(): void: ((DBImpl.conn != NULL) && (DBImpl.stmt != NULL))

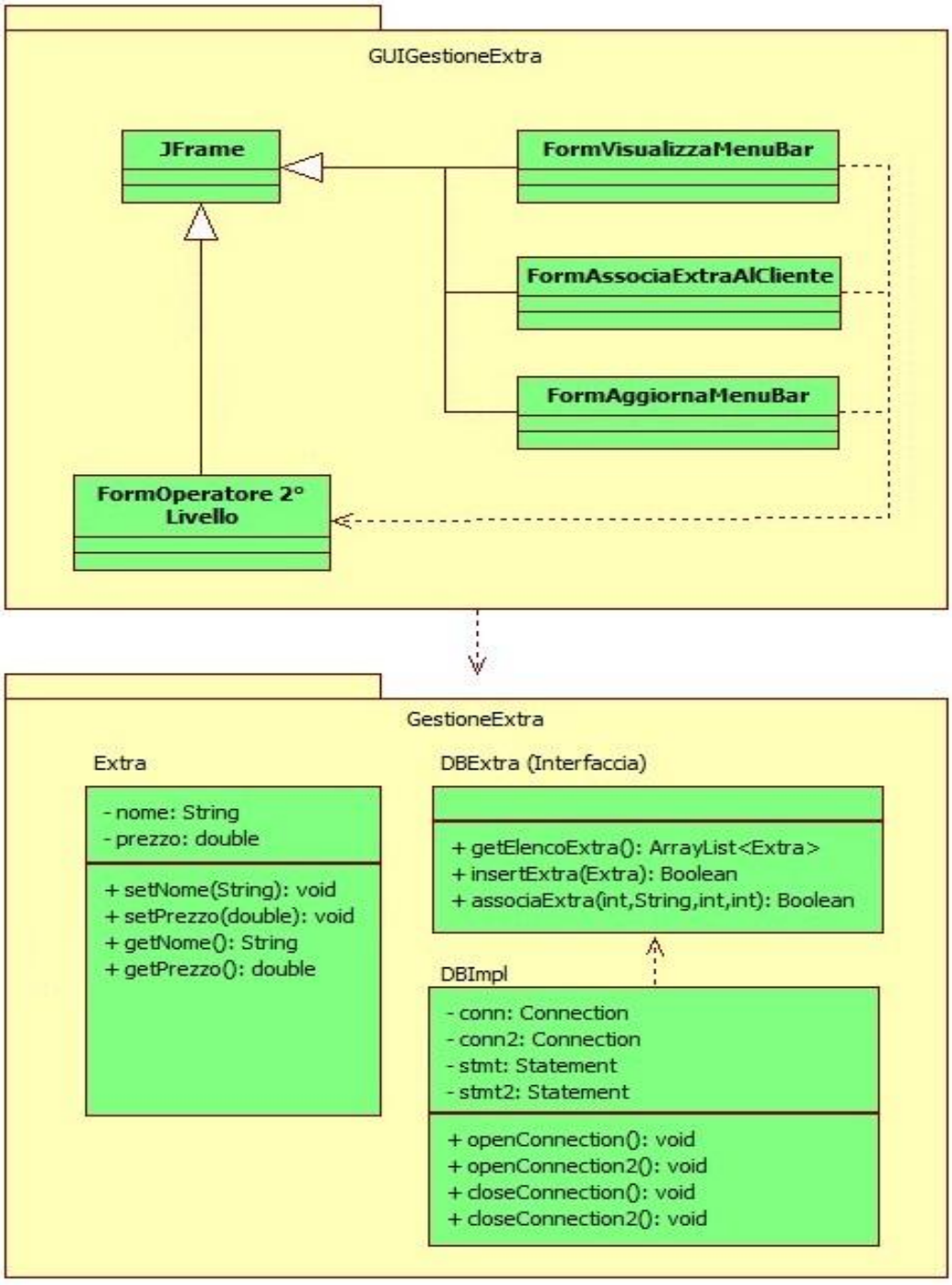
+ openConnection2(): void : ((DBImpl.conn != NULL) && (BImpl.stmt != NULL))

+ closeConnection(): void : ci deve essere una connessione al DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

3.3Gestione Sottosistema Operatore 2° livello

3.3.1GestioneExtra



Di seguito viene riportata la descrizione delle classi tramite JavaDoc:

GUIGestioneExtra:
FormOperatore 2° Livello
FormVisualizzaMenuBar
FormAggiornaMenuBar
FormAssociaExtraAlCliente

GestioneCamereOperatore:
Extra
DBExtra

Di seguito viene fornita una panoramica sui contratti legati a ciascuna classe:
NomeClasse: Extra
Invariante: Il Codice del prodotto deve essere univoco
Precondizioni: -
Postcondizioni: -

NomeInterfaccia: DBExtra
Invariante: -
Precondizioni: -
Postcondizioni: -

NomeClasse: DBImpl
Invariante: -
Precondizioni: + getElencoExtra(): ArrayList<Extra> : gli extra devono essere presenti nel database

+ associaExtra(int,String,int,int): Boolean : l'extra associato al cliente deve essere presente nel DataBase

+ associaExtra(int,String,int,int): Boolean : Il cliente a cui viene associato l'extra deve essere presente nel DataBase

+ openConnection(): void: il DataBase deve essere presente sul Server

+ openConnection2(): void : il DataBase deve essere presente sul Server

+ closeConnection(): void : ci deve essere una connessione al DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

Postcondizioni: + getElencoExtra(): ArrayList<Extra> : getElencoExtra().size()>0

+ insertExtra(Extra): Boolean : nessun campo deve essere nullo

+ associaExtra (int,String,int,int): Boolean : nessun campo deve essere nullo

+ openConnection(): void: ((DBImpl.conn != NULL) && (DBImpl.stmt != NULL))

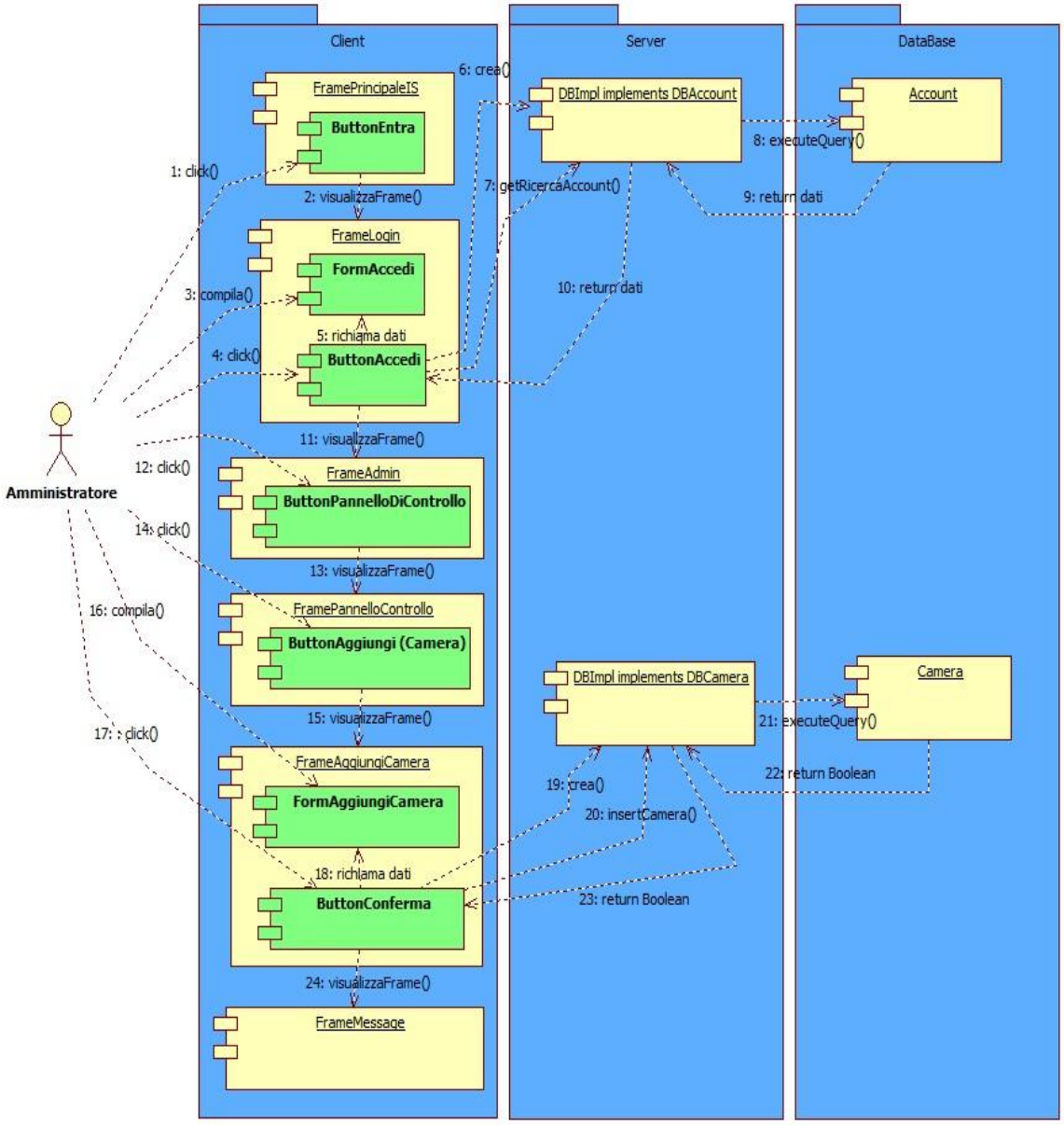
+ openConnection2(): void : ((DBImpl.conn != NULL) && (BImpl.stmt != NULL))

+ closeConnection(): void : ci deve essere una connessione al DataBase

+ closeConnection2(): void : ci deve essere una connessione al DataBase

4.Operazioni a Basso Livello

4.1Amministratore - Aggiungi Camera



1) L'Amministratore avvia il sistema HotelMgM e clicca sul ButtonEntra presente nel FramePrincipaleIS;

2) Il Sistema genera la visualizzazione del FrameLogin contenente il FormAccedi ed il ButtonAccedi;

3) L'Amministratore compila il form per l'accesso al sistema;

4) L'Amministratore clicca sul ButtonAccedi;

5) Il Sistema richiama i dati inseriti nel FormAccedi;

6) Il Sistema crea DBImpl implements DBAccount;

7) Il Sistema richiama il metodo getRicercaAccount();

8) Il Sistema esegue la query sul DataBase Account con executeQuery();

9) Il DataBase ritorna i dati a DBImpl;

10) DBImpl ritorna i dati al Sistema;

11) Il Sistema visualizza il FrameAdmin contenente il ButtonPannelloDiControllo;

12) L'Amministratore clicca sul ButtonPannelloDiControllo;

13) Il Sistema visualizza il FramePannelloControllo contenente il ButtonAggiungi (Sezione Camera);

14) L'Amministratore clicca sul ButtonAggiungi (Sezione Camera);

15) Il Sistema visualizza il FrameAggiungiCamera contenente il FormAggiungiCamera e il ButtonConferma;

16) L'Amministratore compila il FormAggiungiCamera;

17) L'Amministratore clicca sul ButtonConferma;

18) Il Sistema richiama i dati inseriti nel FormAggiungiCamera;

19) Il Sistema crea DBImpl implements DBCamera;

20) Il Sistema richiama il metodo insertCamera();

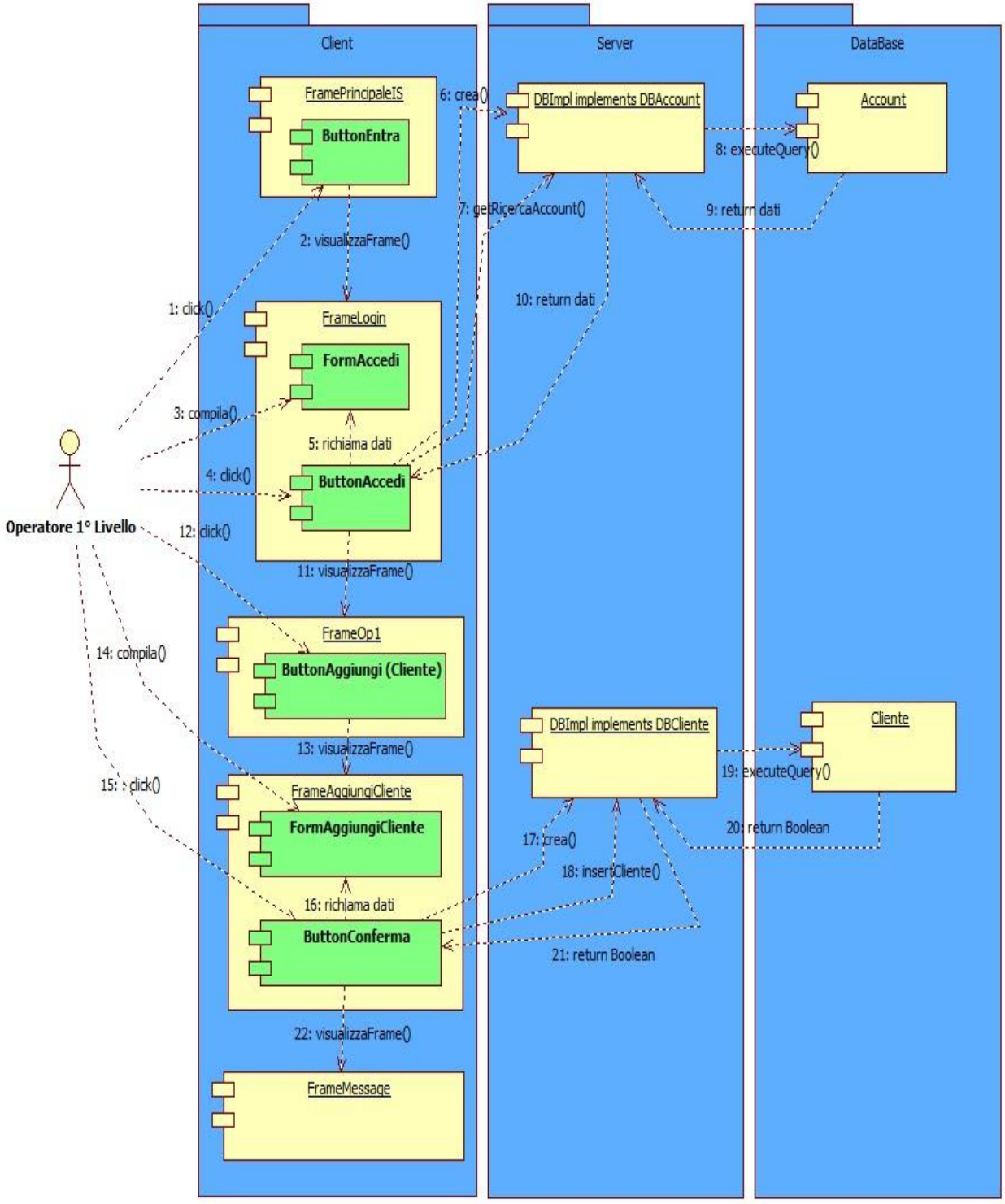
21) Il Sistema esegue la query sulla tabella Camera del DataBase con executeQuery();

22) Il DataBase ritorna un dato di controllo a DBImpl;

23) DBImpl ritorna un dato di controllo al Sistema;

24) Il Sistema visualizza il FramMessage con l'esito dell'operazione.

4.2Operatore 1° Livello - Aggiungi Cliente



1) L'Operatore di 1° livello avvia il sistema HotelMgM e clicca sul ButtonEntra presente nel FramePrincipaleIS;

2) Il Sistema genera la visualizzazione del FrameLogin contenente il FormAccedi ed il ButtonAccedi;

3) L'Operatore di 1° livello compila il form per l'accesso al sistema;

4) L'Operatore di 1° livello clicca sul ButtonAccedi;

5) Il Sistema richiama i dati inseriti nel FormAccedi;

6) Il Sistema crea DBImpl implements DBAccount;

7) Il Sistema richiama il metodo getRicercaAccount();

8) Il Sistema esegue la query sul DataBase Account con executeQuery();

9) Il DataBase ritorna i dati a DBImpl;

10) DBImpl ritorna i dati al Sistema;

11) Il Sistema visualizza il FrameOp1 contenente il ButtonAggiungi (Sezione Cliente);

12) L'Operatore di 1° livello clicca sul ButtonAggiungi (Sezione Cliente);

13) Il Sistema visualizza il FrameAggiungiCliente contenente il FormAggiungiCliente e il ButtonConferma;

14) L'Operatore di 1° livello compila il FormAggiungiCliente(Persone associate: no);

15) L’Operatore di 1° livello clicca sul ButtonConferma;

16) Il Sistema richiama i dati inseriti nel FormAggiungiCliente;

17) Il Sistema crea DBImpl implements DBCliente;

18) Il Sistema richiama il metodo insertCliente();

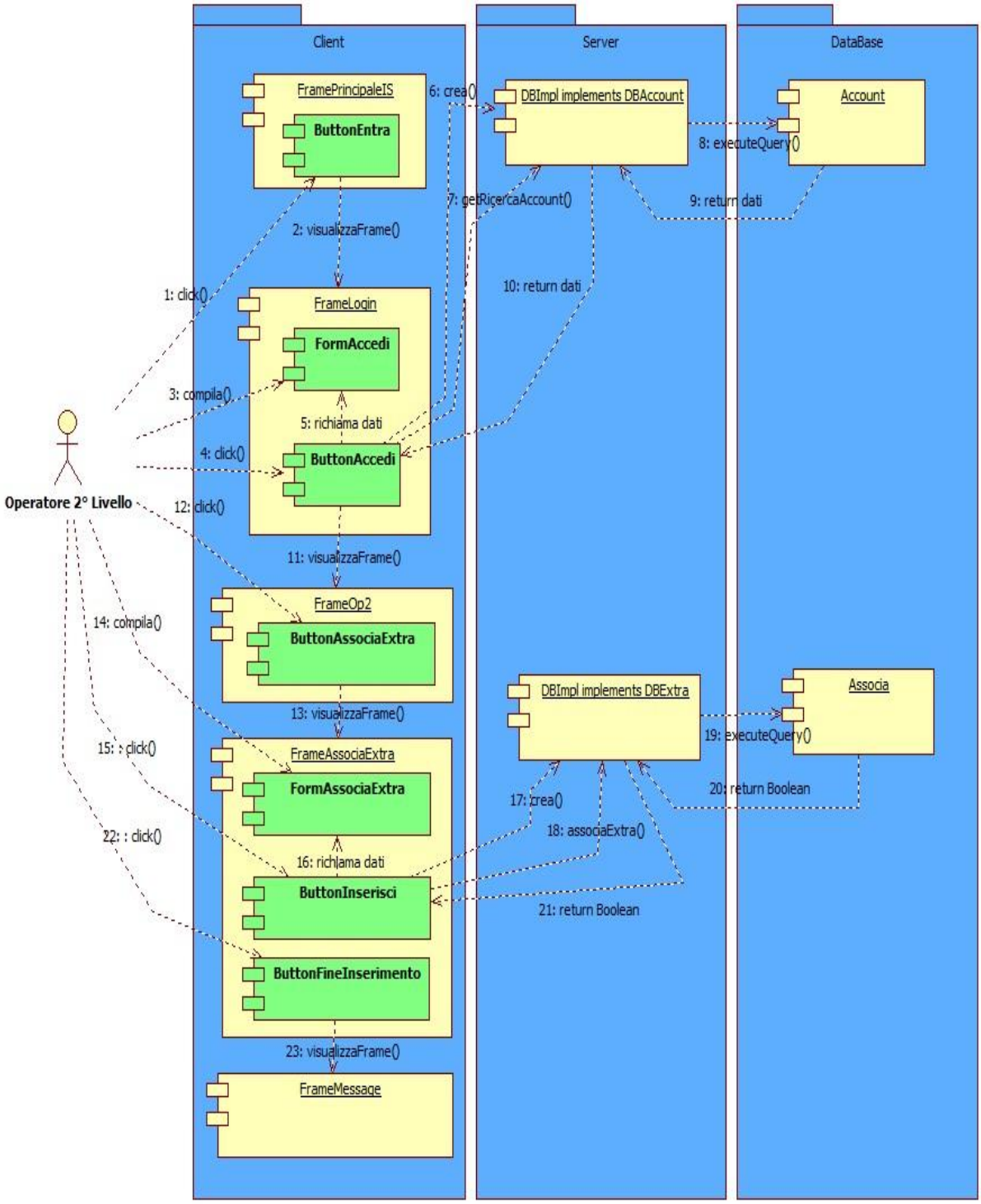
19) Il Sistema esegue la query sulla tabella Cliente del DataBase con executeQuery();

20) Il DataBase ritorna un dato di controllo a DBImpl;

21) DBImpl ritorna un dato di controllo al Sistema;

22) Il Sistema visualizza il FramMessage con l’esito dell’operazione.

4.3Operatore 2° Livello - Associa Extra



1) L'Operatore di 2° livello avvia il sistema HotelMgM e clicca sul ButtonEntra presente nel FramePrincipaleIS;

2) Il Sistema genera la visualizzazione del FrameLogin contenente il FormAccedi ed il ButtonAccedi;

3) L'Operatore di 2° livello compila il form per l'accesso al sistema;

4) L'Operatore di 2° livello clicca sul ButtonAccedi;

5) Il Sistema richiama i dati inseriti nel FormAccedi;

6) Il Sistema crea DBImpl implements DBAccount;

7) Il Sistema richiama il metodo getRicercaAccount();

8) Il Sistema esegue la query sul DataBase Account con executeQuery();

9) Il DataBase ritorna i dati a DBImpl;

10) DBImpl ritorna i dati al Sistema;

11) Il Sistema visualizza il FrameOp2 contenente il ButtonAssociaExtra;

12) L'Operatore di 2° livello clicca sul ButtonAssociaExtra;

13) Il Sistema visualizza il FrameAssociaExtra contenente il FormAssociaExtra, il ButtonInserisci e il ButtonFineInserimento;

14) L'Operatore di 2° livello compila il FormAssociaExtra;

15) L'Operatore di 2° livello clicca sul ButtonInserisci;

16) Il Sistema richiama i dati inseriti nel FormAssociaExtra;

17) Il Sistema crea DBImpl implements DBExtra;

18) Il Sistema richiama il metodo associaExtra();

19) Il Sistema esegue la query sulla tabella Associa del DataBase con executeQuery();

20) Il DataBase ritorna un dato di controllo a DBImpl;

21) DBImpl ritorna un dato di controllo al Sistema;

22) L'Operatore di 2° livello clicca sul ButtonFineInserimento quando ha terminato l'associazione degli extra al cliente;

23) Il Sistema visualizza il FramMessage con l'esito dell'operazione.