Università degli Studi di Salerno
Corso di Ingegneria del Software

FreedomAir
ODD
Versione 0.5

Data: 28/12/2017

Progetto: FreedomAir Documento: ODD	Versione: 0.5 Data 28/12/2017

Partecipanti:

Nome	**Matricola**
Ferraioli Rosa	0512103450
Alfonso Del Gaizo	0512103462

Scritto da:	Ferraioli Rosa, Alfonso Del Gaizo

Revision History

Data	Versione	Descrizione	Autore
20/12/2017	0.1	Introduzione	Ferraioli Rosa
22/12/2017	0.2	Design Pattern	Del Gaizo Alfonso
23/12/2017	0.3	Paragrafo 2, Packages	Ferraioli Rosa, Del Gaizo Alfonso
27/12/2017	0.4	Class interfaces	Ferraioli Rosa
28/12/2017	0.5	Invarianti, post-condizioni, pre-condizioni e Ocl	Del Gaizo Alfonso

INDICE DEI CONTENUTI

1.Introduzione

1.1 Object Design trade-offs.....3

1.2 Linee guida di documentazione di interfaccia.....4

1.3 Definizioni acronimi e abbreviazioni.....5

1.4 Design Pattern 6

2. Packages 7

3. Classes interfaces.....12

1. Introduzione

L'object design è il processo che si occupa di aggiungere dettagli all'analisi dei requisiti e di prendere decisioni di implementazione. I progettisti devono scegliere i diversi modi per implementare il modello di analisi, con l'obiettivo di minimizzare il tempo di esecuzione, la memoria e altre misure di costo.

1.1 Object Design trade-offs

Questo documento si propone di evidenziare gli obiettivi principali:

- La dimensione complessiva dipende dalla memoria utilizzata per il mantenimento del database. Verrà utilizzato il programma MySql; in presenza di grandi quantità di dati, non vi sarà un rallentamento del sistema.

- A qualsiasi sollecitazione fornita dagli attori, il sistema deve rispondere entro 30 secondi oppure in caso di errore presentare un messaggio di errore.

- Il sistema deve sfruttare al meglio la rete senza sovraccaricarla inutilmente

Interfaccia vs Usabilità

L'interfaccia grafica si basa sull'utilizzo di form. Infatti, saranno utilizzati elementi grafici che renderanno immediatamente intuibile il contesto a cui appartengono, in modo da rendere facilmente deducibile l'utilizzo del Sistema in termini di flusso degli eventi.

Sicurezza vs Efficienza

La gestione della sicurezza si basa sull'autenticazione tramite username e password: ogni utente registrato accede alle funzionalità previste dal suo profilo in accordo alle politiche di sicurezza. Le autorizzazioni vengono definite, quindi, da una politica di permessi.

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, al fine di facilitare la comprensione. Quindi la fase di sviluppo del progetto è stata tale da coprire tutte le sue funzionalità allungando però i tempi di consegna.

1.1.1 Linee guida e convenzioni:

I meccanismi per la gestione delle eccezioni sono racchiusi per la maggior parte nella gestione della connessione tra le applicazioni e il database.

1.1.2 Casi limite (Boundary Cases):

Il database che abbiamo utilizzato, fa in modo che i campi possano avere un valore massimo di lunghezza, evitando così di incorrere in eccezioni.

Meccanismi gestione delle eccezioni:

La scelta di utilizzare le eccezioni è stata fatta per garantire il controllo del sistema, infatti sono in grado di affrontare qualsiasi tipo di problema. Ad esempio:

1) Username e password errate:

L'utente inserisce l'username e la password nei campi di testo criptati e fa clic su "Login".
L'interfaccia utente comunica con sistema per verificare se l'username esiste già, o e se la password è corretta o meno. Se il l'username non esiste o la password non è corretta, si visualizza un allarme con "Username o password Errati".

1.2 Linee guida di documentazione di interfaccia

Regole che utilizzi per le classi, metodi, le variabili e i packages.

Tipi	Regole	Esempi
Packages	Per i packages utilizziamo tutto in minuscolo.	default package
Classes	Le classi sono costituire con “class” + il nome della classe, la quale ha la prima lettera del nome in maiuscolo.	class Voli; class Annunci;
Methods	I metodi utilizzati sono verbi, con la prima lettera della prima parola in minuscolo e la prima lettera della seconda parola in maiuscolo.	insericiVolo (); modificaDatiUtente();
Variables	Le costanti vengono scritte in maiuscolo, mentre le variabili interne alle classi tutte in minuscolo e le variabili globali la prima lettera in maiuscolo.	int CodiceId; string Username; double Prezzo; int Posto;

1.3 Definizioni, acronimi e abbreviazioni.

- username = nome con cui accede al Sistema l’utente.
- password= chiave per accedere al Sistema.
- mySql= programma per la gestione del Database.

1.4 Design Pattern

Data Access Object

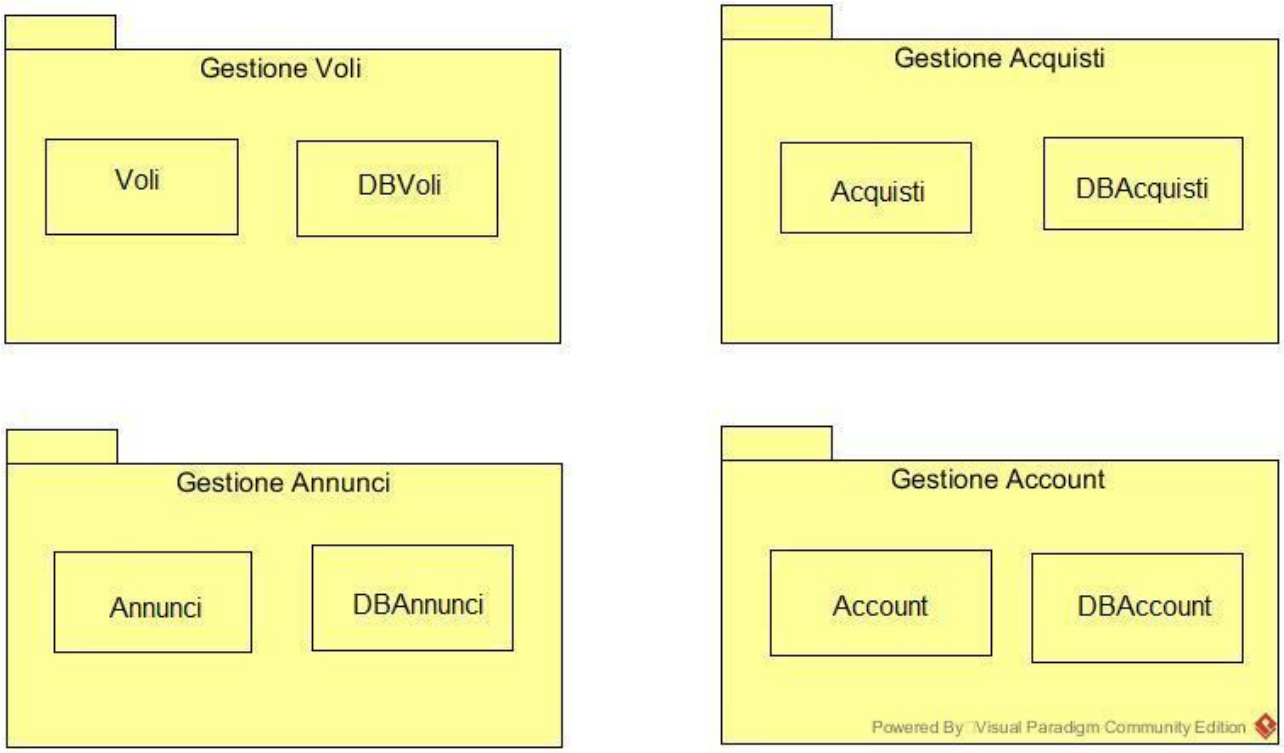
In informatica, nell'ambito della programmazione Web, il *DAO* (*Data Access Object*) è un pattern architetturale .
Si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un RDBMS, usata principalmente in applicazioni web sia di tipo Java EE sia di tipo EJB, per stratificare e isolare l'accesso ad una tabella tramite query(poste all'interno dei metodi della classe) ovvero al *data layer* da parte della *business logic* creando un maggiore livello di astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic.

2. Packages

Le funzionalità elencate saranno implementate nel nostro sistema:

- Gestione Voli
- Gestione Annunci
- Gestione Acquisti
- Gestione Account

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

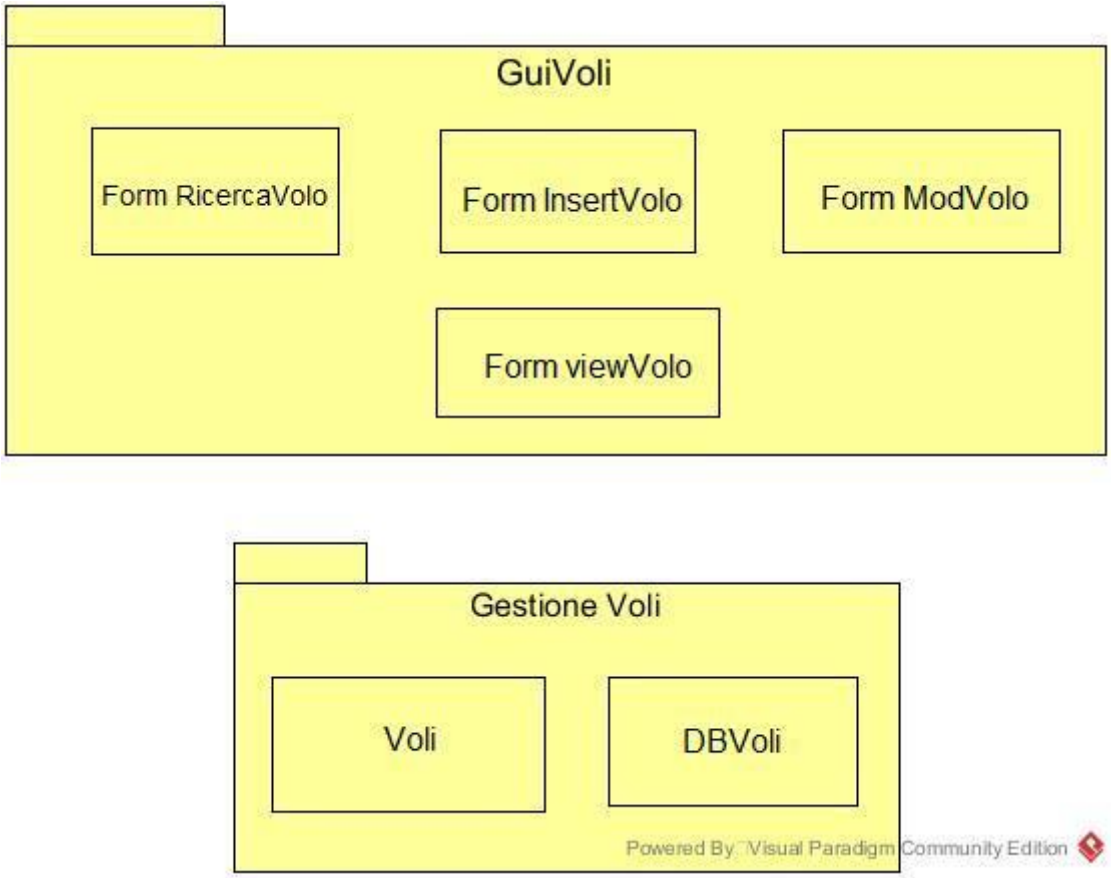


Nel successivo paragrafo vengono elencate graficamente le classi che compongono ciascun package. Una loro descrizione approfondita (metodi e campi) è fornita al punto 3.

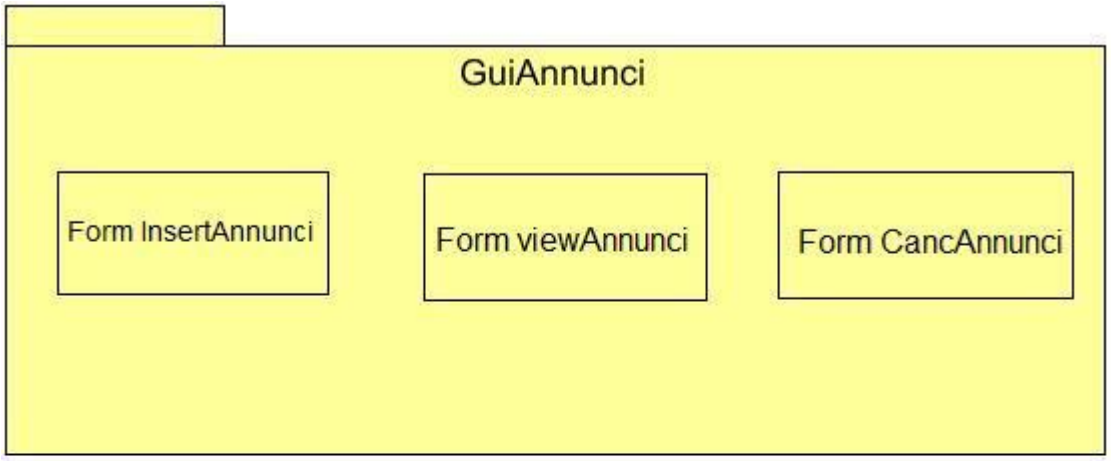
Poiché, utilizzando JDBC, la parte relativa alla logica applicativa del pacchetto e quella relativa alla memorizzazione sono strettamente correlate si è preferito "accorpare" l'ApplicationLogicLayer e lo StorageLayer, definiti nel SDD. Per ciascun gruppo di funzionalità, sono presenti, quindi, due pacchetti, ognuno composto da diverse classi:

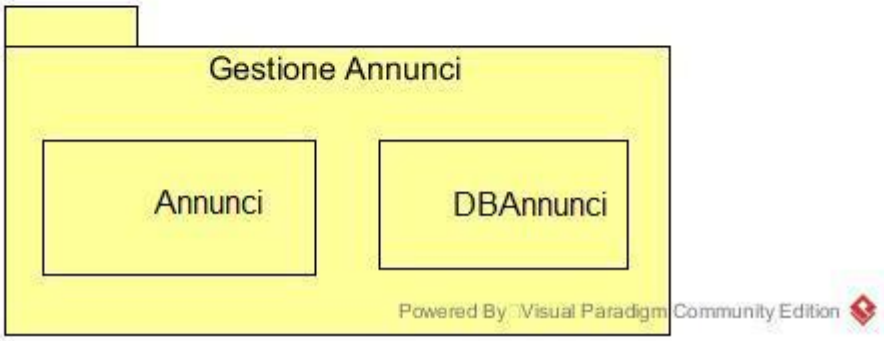
- GestioneXxx, che si occupa, appunto, della logica applicativa e della memorizzazione dati
- GUIXxx, che si occupa dell'interazione con l'utente (in pratica, delle interfacce grafiche).

2.1 Gestione Voli

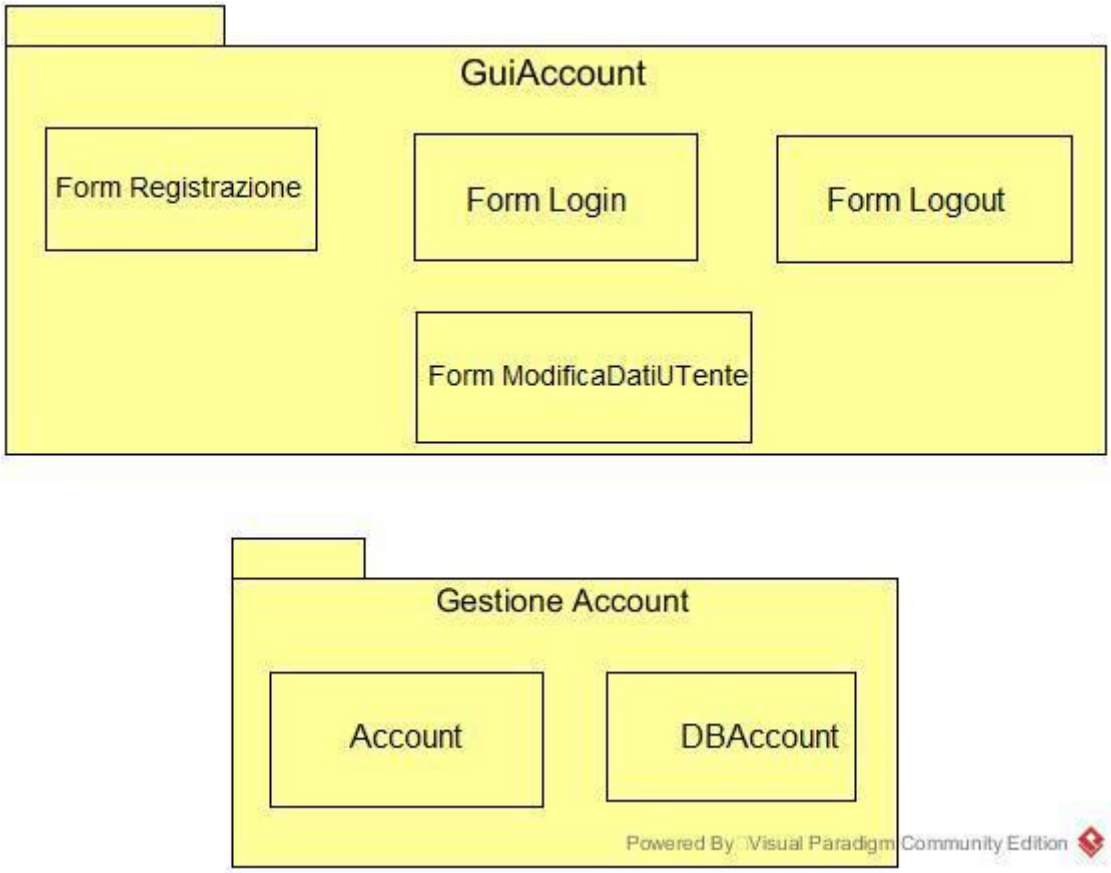


2.2 Gestione Annunci

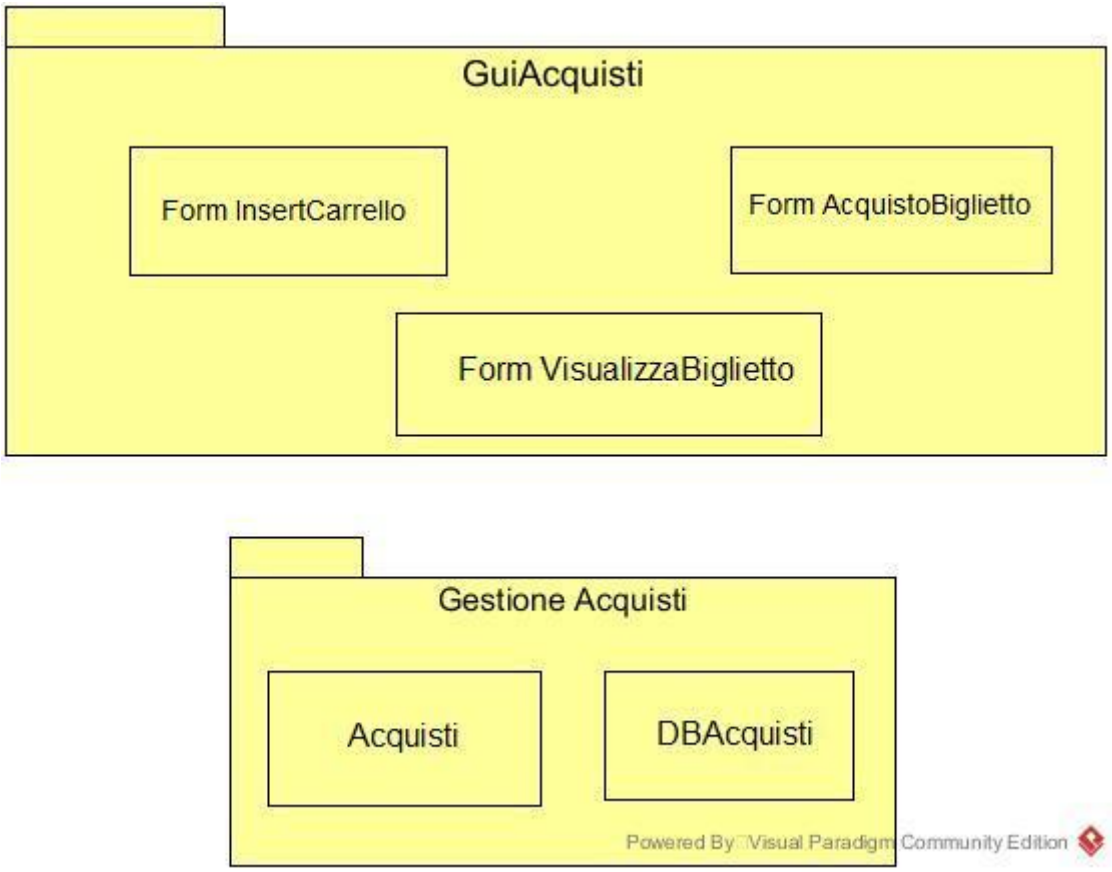




2.3 Gestione Account

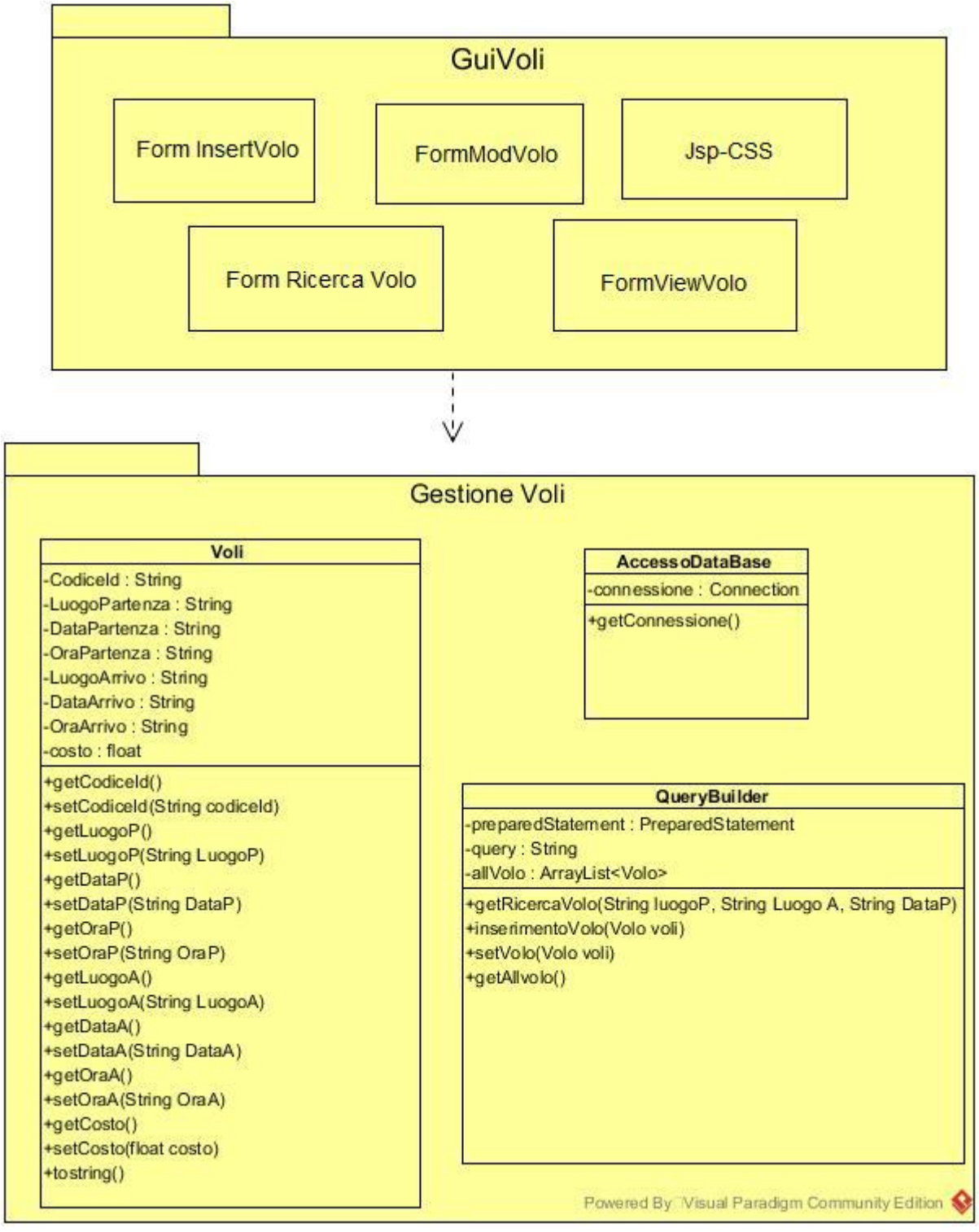


2.4 Gestione Acquisti



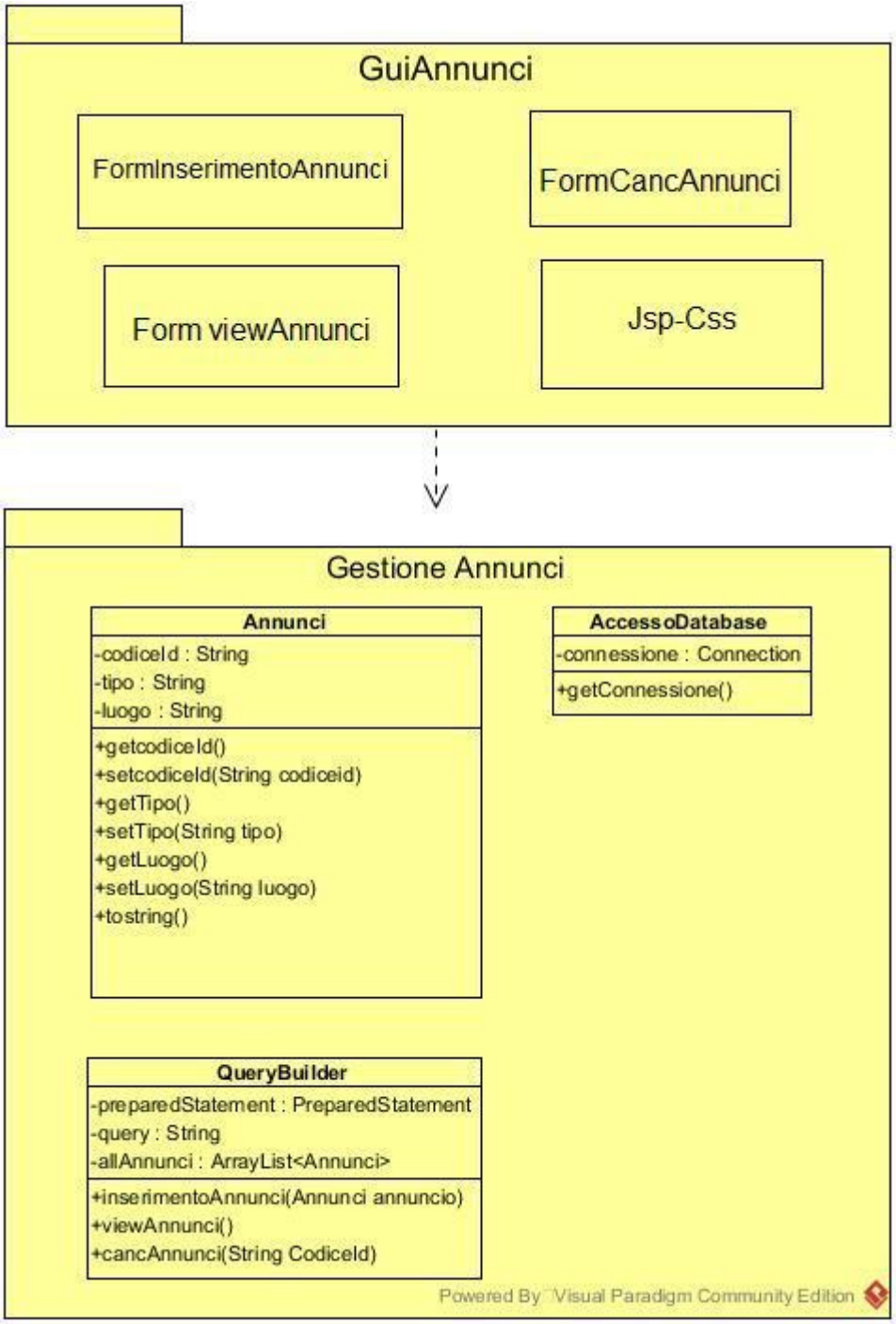
3. Class Interfaces

3.1 GestioneVoli



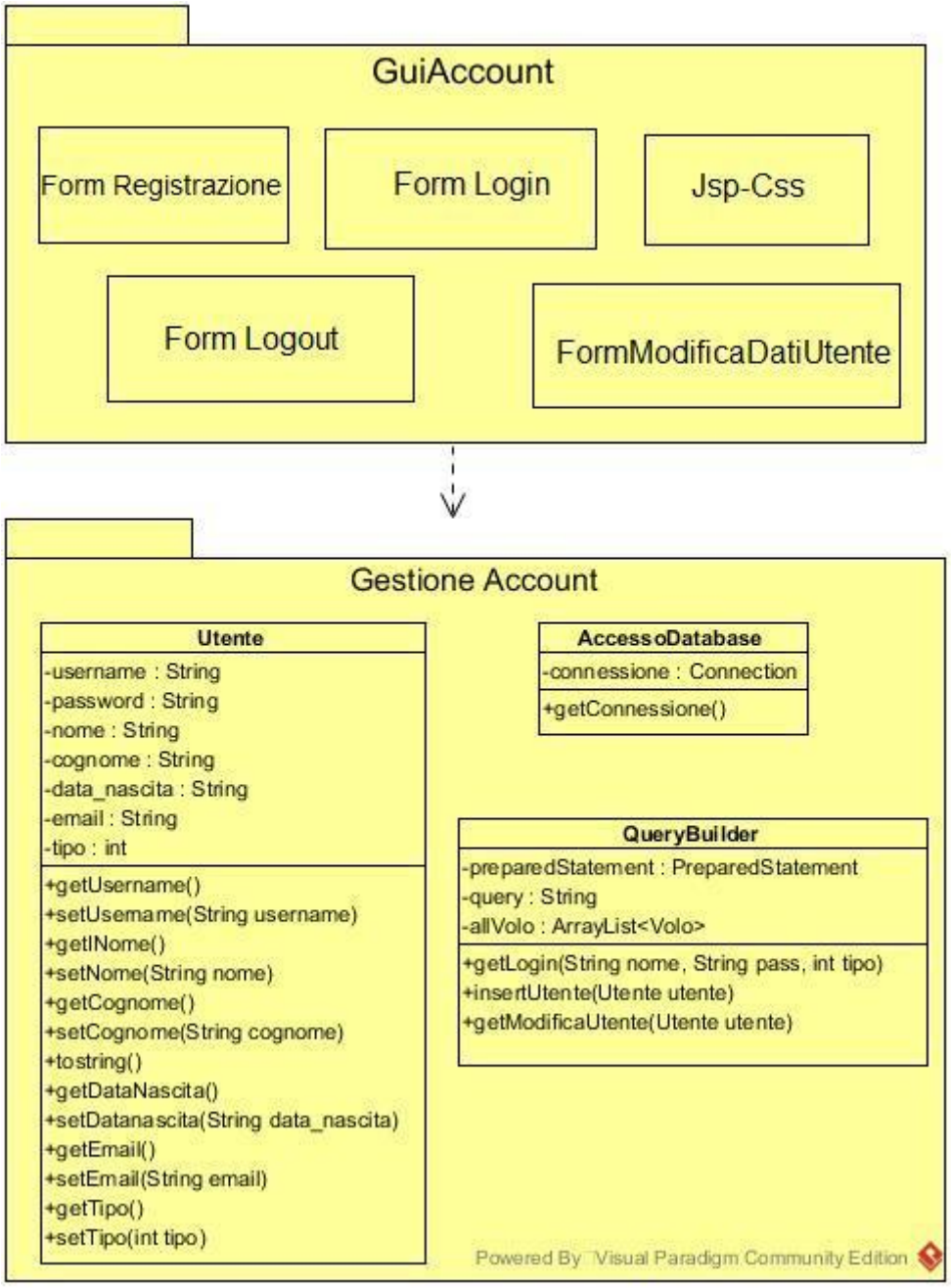
Nome	GestioneVoli
Descrizione	GestioneVoli permette i voli , permettendo dunque l' inserimento , la ricerca e la visualizzazione dei
Pre-Condizione	context GestioneVoli::getRicercaVolo(String luogo_partenza, String data_partenza) pre: luogo_partenza!=null and and data_partenza->size()=10 context GestioneVoli::setDataPartenza(String dataPartenza) pre: dataPartenza!=null and and dataPartenza->size()=10
Post-Condizione	
Invarianti	context GestioneVoli inv : codiceId != null and codiceId>=1

3.2 GestioneAnnunci



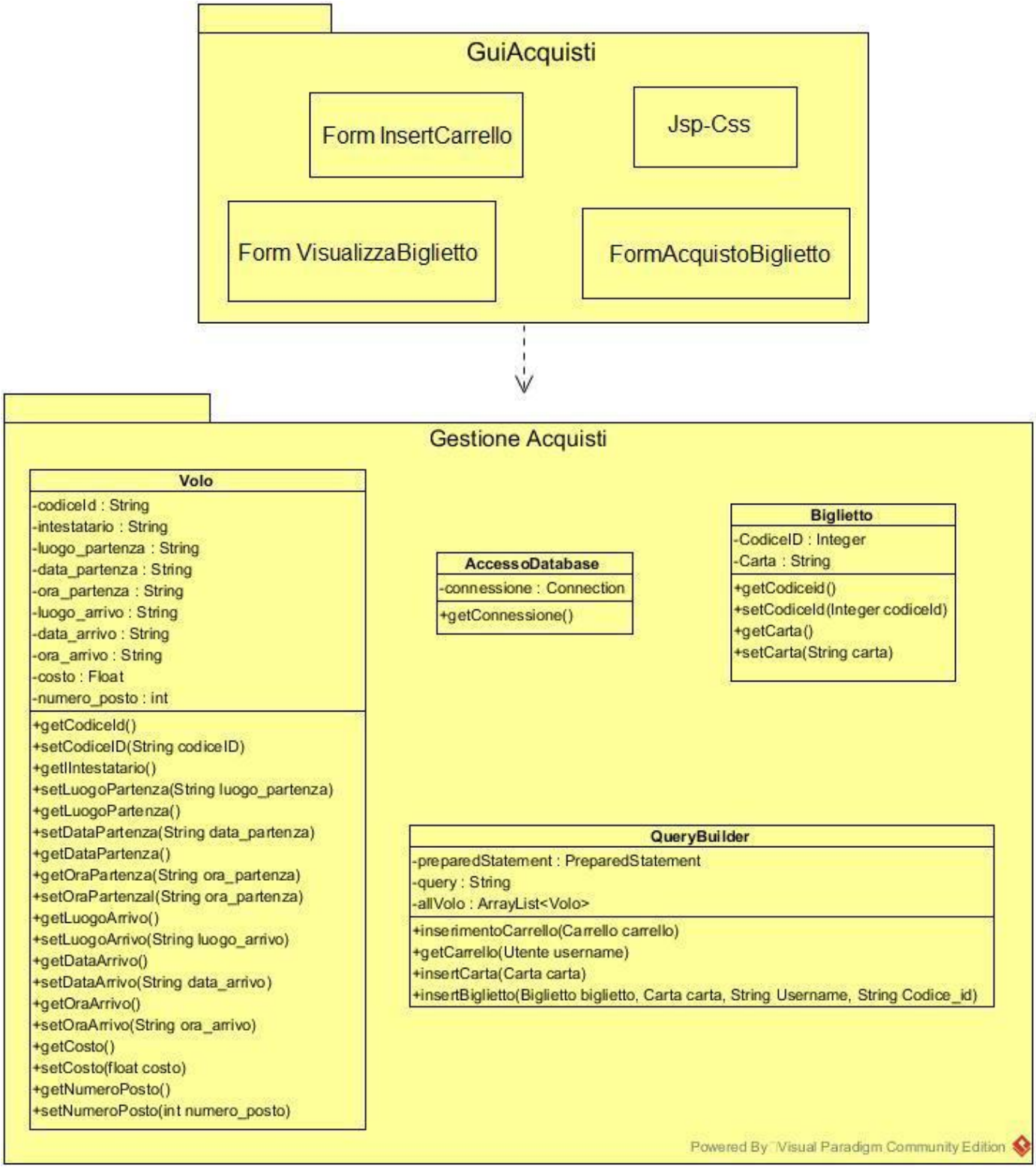
Nome	GestioneAnnunci
Descrizione	GestioneAnnunci permette la gestione degli annunci pubblicitari ,
Pre-Condizione	context GestioneAnnunci ::setCodiceId(String codiceId) pre: codiceId != null and codiceId ->size()>1
Post-Condizione	context GestioneAnnunci ::getTipo() post: tipo!= null and tipo->size()>0
Invarianti	

3.3 GestioneAccount



Nome	GestioneAccount
Descrizione	Questa classe gestisce gli account, permettendo l'accesso, l'uscita, e la registrazione di un nuovo account
Pre-Condizione	context GestioneAccount::getLogin(String username, String password , int tipo) pre: username!=null and username->size() >1 and password =! null and tipo=0 or tipo=1 context GestioneAccount::setDataNascita(String dataNascita) pre : dataNascita->size() =10 context GestioneAccount::setEmail(String email) pre : email->size() <=20
Post-Condizione	
Invarianti	

3.4 Gestione Acquisti


GuiAcquisti
Form InsertCarrello
Jsp-Css
Form VisualizzaBiglietto
FormAcquistoBiglietto
Gestione Acquisti
Volo
-codiceId : String
-intestatario : String
-luogo_partenza : String
-data_partenza : String
-ora_partenza : String
-luogo_arrivo : String
-data_arrivo : String
-ora_arrivo : String
-costo : Float
-numero_posto : int
+getCodiceId()
+setCodiceID(String codiceID)
+getIlntestatario()
+setLuogoPartenza(String luogo_partenza)
+getLuogoPartenza()
+setDataPartenza(String data_partenza)
+getDataPartenza()
+getOraPartenza(String ora_partenza)
+setOraPartenzal(String ora_partenza)
+getLuogoArrivo()
+setLuogoArrivo(String luogo_arrivo)
+getDataArrivo()
+setDataArrivo(String data_arrivo)
+getOraArrivo()
+setOraArrivo(String ora_arrivo)
+getCosto()
+setCosto(float costo)
+getNumeroPosto()
+setNumeroPosto(int numero_posto)
AccessoDatabase
-connessione : Connection
+getConnessione()
Biglietto
-CodiceID : Integer
-Carta : String
+getCodiceid()
+setCodiceId(Integer codiceId)
+getCarta()
+setCarta(String carta)
QueryBuilder
-preparedStatement : PreparedStatement
-query : String
-allVolo : ArrayList<Volo>
+inserimentoCarrello(Carrello carrello)
+getCarrello(Utente username)
+insertCarta(Carta carta)
+insertBiglietto(Biglietto biglietto, Carta carta, String Username, String Codice_id)
Powered By Visual Paradigm Community Edition

Nome	GestioneAcquisti
Descrizione	GestioneAquisti permette la gestione degli acquisti dei biglietti relativi ai voli effettuati dell'utenza del sistema mediante anche l'utilizzo del Carrello .
Pre-Condizione	context GestioneAcquisti::insertBiglietto(Biglietto biglietto, String carta , String Username , String Codice:id) pre: username!=null and username->size() >1 and carta =! null and carta->size() >=12 and CodiceId !=null context GestioneAcquisti::getCarrello(String username) pre : username->size() >1
Post-Condizione	context GestioneAcquisti::getCodiceId() post : codiceId != null
Invarianti	