Università degli Studi di Salerno

Corso di Gestione Progetti Software

A.A. 2012/2013

Alfonso Murolo

Giulio Franco

Linda di Geronimo

Object Design Document

Versione 1.0

Storia delle revisioni

Versione	Data	Autori	Descrizione
1.0	20/11/2012	Alfonso Piscitelli, Andrea Micco, Angelo Rufino, Angelo Scafuro, Antonio Barba, Antonio Cesarano, Elisa D'Eugenio, Fabio Napoli, Ferdinando Di Palma, Francesco Durante, Francesco Durante, Francesco Nastro, Gianfranco Bottiglieri, Luca Di Costanzo, Luigi Lomasto, Marco Parisi, Mariella Ferrara	Prima versione definitiva

Riferimenti

CCJPL: , Code Conventions for the Java Programming Language, 1999, http://www.oracle.com/technetwork/java/codeconv-138413.html

HTML41: Dave Raggett, Arnaud Le Hors, W3CIan Jacobs, HTML 4.01 Specification, 1999, http://www.w3.org/TR/html401/

Indice

1 Introduzione

1.1 Tradeoff nel Design degli Oggetti

Sicurezza vs Efficienza:

La gestione della sicurezza viene affidata all'utilizzo del login iniziale in quanto va ad autenticare l'utente al quale sarà visualizzata solo la parte del software che gli appartiene, evitando così incongruenze di dati. Il login funziona in una ricerca all'interno del database nella sezione account dove il sistema cercherà se esiste il nome utente e se questo corrisponde alla password inserita. Questa politica di permessi, permette di non appesantire eccessivamente il software ed è un buon compromesso tra sicurezza ed efficienza.

Comprensibilità vs Tempo:

Il codice deve essere più comprensivo possibile in modo da poter essere interpretato da altri programmatori che non hanno partecipato al progetto. Il codice dovrà essere commentato in modo da migliorare la lettura delle righe di codice anche se l'aggiunta di commenti accrescerà il tempo necessario per completare l'implementazione.

Spazio di Memoria versus Velocità:

Il software memorizza tutte le informazioni inerenti alle differenti entità presenti all'interno del sistema in modo che il carico complessivo dei dati non influirà sulla velocità di risposta del sistema sia dal lato client che dal lato server, quindi, le operazioni delle funzionalità implementate richiederanno un brevissimo tempo di risposta. Visto che nel nostro sistema preferiamo la velocità allo spazio si è scelto un DBMS che rispecchia questa decisione.

1.2 Linee guida per la documentazione delle interfacce

Nell'implementazione del sistema, i programmatori dovranno attenersi alle linee guida di seguito definite.

1.2.1 Classi e Interfacce Java

Nella scrittura di codice per le classi Java, ci si atterrà allo standard [CCJPL] nella sua interezza, con particolare attenzione a:

1. Convenzioni sui nomi (cap. 9 dello standard);
2. Struttura dei file (cap. 3 dello standard);
3. Accesso alle variabili (sez. 10.1 dello standard)
4. Commenti speciali (sez. 10.5.4 dello standard)
5. Utilizzo degli spazi bianchi (cap. 8 dello standard)

Si praticano, inoltre, le seguenti restrizioni e variazioni:

1. All'inzio di ogni file devono essere presenti alcune informazioni strutturate come segue:

```
/*
 *-----------------------------------------------------------------
 * This file is licensed under GPL 3.0:
 * http://www.gnu.org/licenses/gpl-3.0.html
 *-----------------------------------------------------------------
 * FILE: NomeFile
 *-----------------------------------------------------------------
 * PROGETTO: Atsilo
 *-----------------------------------------------------------------
 * OWNER
 * nome autore, data creazione
 * REVISION
 * nome revisore, data revisione
 *-----------------------------------------------------------------
 */
```

2. I commenti iniziali (sez. 3.1.2 dello standard) sono obbligatori, e devono avere la forma:

```
/*
 * NomeClasse
 * Breve descrizione della classe
 */
```

3. Tutte le variabili dovrebbero essere private (private). Ogni variabile non privata deve essere preceduta da un commento, che spiega la ragione per cui tale variabile non è privata. Tutte le variabili che non vengono mai modificate dovrebbero essere dichiarate come costanti (final).
4. L'uso dell'operatore condizionale ternario è scoraggiato.
5. Si devono usare gli spazi per gestire l'indentazione.
   L'unità di indentazione è 4 spazi.
   Le tabulazioni sono di 8 spazi.

1.2.2 Base di dati

Le tabelle della base di dati dovrebbero rispettare la terza forma normale di Codd (3NF). Ove ciò non si verifichi, tale fatto deve essere riportato e motivato nella documentazione della base di dati.

Le scelte di gestione nel trattamento dell'integrità referenziale devono essere riportate e motivate nella documentazione della base di dati.

I nomi utilizzati all'interno della base di dati devono seguire le seguenti convenzioni:

1.2.3 Pagine Java lato Server (JSP)

Le pagine JSP devono, quando eseguite, produrre, in ogni circostanza, una documento conforme allo standard HTML versione 4.01 “strict” ([HTML41]).

Le parti Java delle pagine devono aderire alle convenzioni per la codifica in Java, con le seguenti puntualizzazioni:

1. Il tag di apertura (**<%**) è seguito immediatamente dalla fine della riga;
2. Il tag di chiusura (**%>**) si trova all'inizio di una riga;
3. È possibile emendare alle due regole precedenti, se il corpo del codice Java consiste in una singola istruzione:

```
<!-- Accettabile -->
<% for (String par : paragraphs) {%>
<p class='item'><% out.print(par); %></p>
<% } %>

<!-- Non accettabile -->
<p class='item'><% List<String> paragraphs = getParagraphs();
out.print(paragraphs.get(i++));%></p>
```

Il codice HTML, Javascript e CSS prodotto deve attenersi alle relative convenzioni.

1.2.4 Pagine HTML

Le pagine HTML, statiche e dinamiche, devono essere totalmente aderenti allo standard HTML versione 4.01 “strict” ([HTML41]).

Inoltre, il codice HTML statico deve utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

1. Un'indentazione consiste in 4 spazi;
2. Ogni tag deve avere un'indentazione maggiore o uguale del tag che lo contiene
   2.1. Non si dovrebbero superare gli 8 livelli di indentazione;
   2.2. Qualora si superassero gli 8 livelli, è possibile azzerare il livello di indentazione, portandolo a 0 anziché incrementarlo;
3. Sebbene l'uso dell'indentazione sia arbitrario, ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;

```
<!-- Accettabile -->
<div><span><nl>
   <li>Uno</li>
   <li>
      Due
   </li>
</nl></span></div>

<!-- Non accettabile -->
<div><span>
   <nl>
```

```
  <li>Uno</li>
  <li>
    Due
  </li>
  </nl></span>
</div>
```

4. I tag di commento devono seguire le stesse regole che si applicano ai tag normali;
5. Gli script e i fogli di stile incorporati, se non si completano in una sola riga di testo, devono partire dal livello 0 di indentazione.

I fogli di stile e gli script incorporati nelle pagine devono rispettare le stesse linee guida che si applicano ai documenti dedicati.

Le sezioni dinamiche (prodotte tramite l'oggetto out) dei documenti HTML generati da pagine JSP dovrebbero rispettare le stesse convenzioni dello HTML statico.

1.2.5 Script Javascript

Gli script che svolgono funzioni distinte dal mero rendering della pagina dovrebbero essere collocati in file dedicati.

Il codice Javascript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.

I documenti Javascript devono essere iniziati da un commento analogo a quello presente nei file Java.

Le funzioni Javascript devono essere documentate in modo analogo ai metodi Java.

Gli oggetti Javascript devono essere preceduti da un commento in stile Javadoc, che segue il seguente formato:

```
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti del costruttore (@param)
*
* Metodo nomeMetodo1
*    Descrizione breve
*    Eventuale ulteriore descrizione
*    Specifica degli argomenti (@param)
*    Specifica dei risultati (@return)
*
* Metodo nomeMetodo2
*    Descrizione breve
*    Eventuale ulteriore descrizione
*    Specifica degli argomenti (@param)
*    Specifica dei risultati (@return)
*
```

```
 * …
 */
function ClasseX(a, b, c) {
```

1.2.6 Fogli di stile CSS

Tutti gli stili non inline devono essere collocati in fogli di stile separati.

Ogni foglio di stile deve essere iniziato da un commento analogo a quello presente nei file Java.

Ogni regola CSS deve essere formattata come segue:

1. I selettori della regola si trovano a livello 0 di indentazione, uno per riga;
2. L'ultimo selettore della regola è seguito da parentesi graffa aperta ({);
3. Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori;
4. La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

Le proprietà e le regole poco chiare dovrebbero essere precedute da un commento esplicativo.

Le regole possono essere divise in blocchi concettualmente legati, preceduti da commenti in stile Javadoc, che ne spiegano lo scopo, e seguiti da 2 righe bianche.

1.2.7 Database SQL

Le tabelle del database dovrebbero essere in terza forma normale di Codd (3NF). Qualora non lo fossero, la cosa deve essere documentata e motivata nello script di creazione del database.

I nomi delle tabelle devono seguire le seguenti regole:

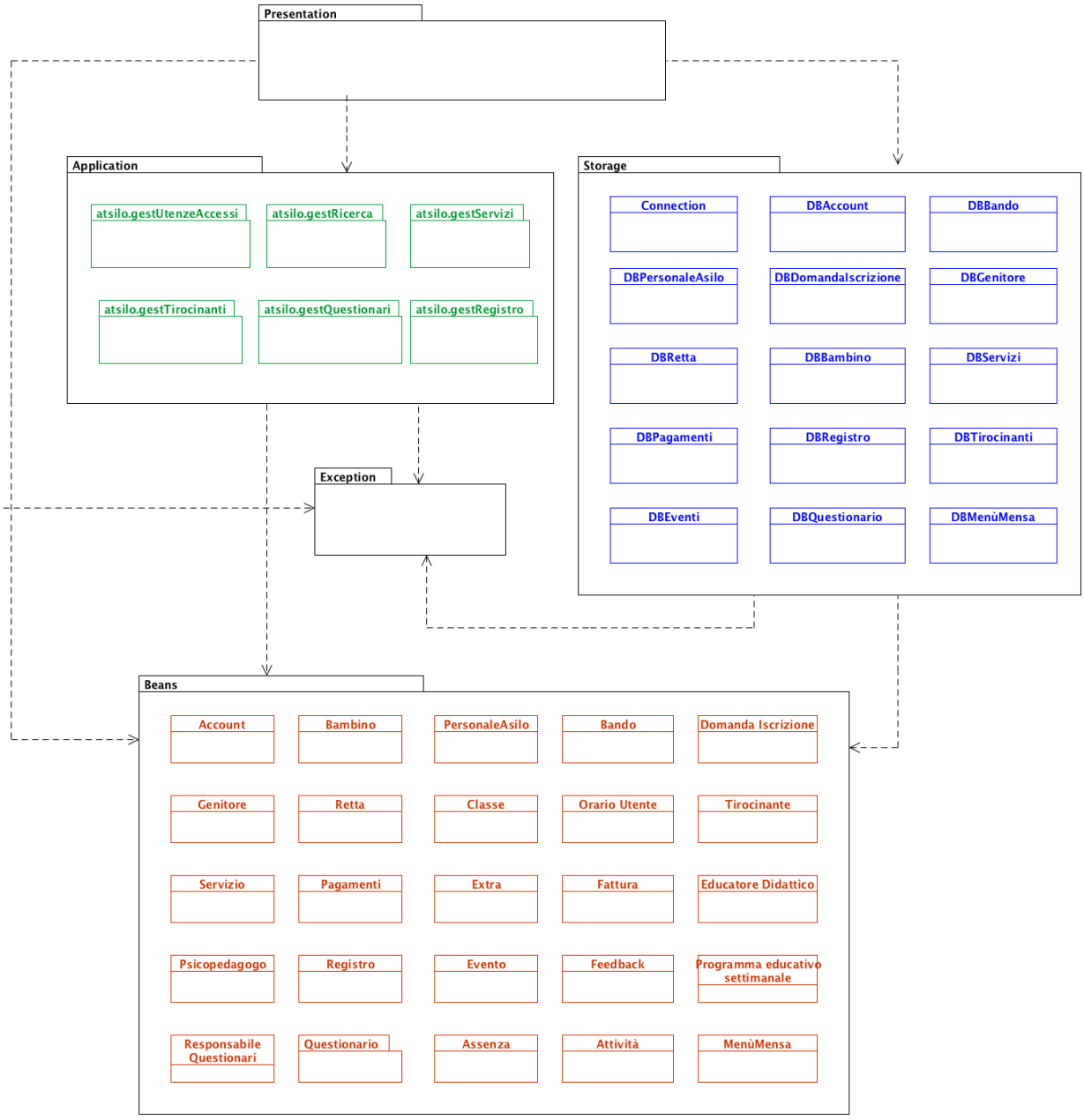
1. sono costituiti da sole lettere maiuscole;
2. se il nome è costituito da più parole, queste sono separate da un underscore (_);
3. il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

I nomi dei campi devono seguire le seguenti regole:

1. sono costituiti da sole lettere minuscole;
2. se il nome è costituito da più parole, queste sono separate da un underscore (_);
3. il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

1.3 Definizioni, acronimi e abbreviazioni

2 Pacchetti



Presentation

Il sottosistema Presentation è responsabile della gestione delle interfacce grafiche del sistema @silo, interfacce che permettono l'interazione con l' utente.

Application

Il sottosistema Application rappresenta il maggiore livello di astrazione, dove vengono implementate tutte le funzionalità che permettono la gestione, la manipolazione ed il passaggio dei dati dal livello Presentation a quello di Storage. E' suddiviso in due packages:

atsilo.gestUutenzeAccessi: package che si occupa di: gestire gli accessi al sistema, gestire la registrazione, l'inserimento e la modifica di utenti nel sistema, gestire i pagamenti.

atsilo.gestRicerca: package che si occupa della ricerca degli utenti del sistema e del loro criterio di visualizzazione.

atsilo.gestServizi: il package si occupa di tutte le operazioni relative ai servizi offerti dall'asilo agli utenti, come il servizio mensa e quello orario, consentendo la modifica e visualizzazione del servizio; permette inoltre la gestione delle fasce di utenza orarie specifiche.

atsilo.gestTirocinanti: il package si occupa di gestire il periodo di tirocinio degli studenti della facoltà di Scienze della Formazione, consentendo le operazioni di inserimento, modifica, richiesta e feedback.

*atsilo.gestQuestionari:*il package permette la gestione dei questionari inserendoli e modificandoli per ogni utente e visualizzarne le statistiche.

atsilo.gestRegistro: il package permette la gestione del registro di classe permettendo l'inserimento, la modifica e la cancellazione di eventi e attività svolte e di tener traccia delle presenze.

Storage

Il sottosistema Storage gestisce ed immagazzina i dati persistenti, e per questo presenta tutte quelle operazioni che permettono la comunicazione con il Database, come operazioni per la connessione, l'accesso ai dati e l'interrogazione (query) di questi. Le classi in cui è suddiviso questo livello sono:

- Connection
- DBAccount
- DBBando
- DBPersonaleAsilo
- DBDomandaIscrizione
- DBGenitore
- DBRetta
- DBBambino
- DBServizi
- DBPagamenti
- DBTirocinanti
- DBEventi
- DBRegistro
- DBQuestionario
- DBMenuMensa

Beans

Il sottosistema Beans permette la gestione dei dati, tramite operazione che si occupano di creare e manipolare le entità che fanno parte del sistema, entità che rappresentano un concetto ben distinto e con un proprio ruolo. Fanno parte dei Beans le classi:

- Account
- Bando
- Personale Asilo
- Domanda Iscrizione
- Genitore
- Retta
- Classe
- Bambino
- Orario Utente
- Servizi
- Richiesta
- Pagamenti
- Fattura
- Extra
- Tirocinanti
- Psicopedagogo
- Evento
- Feedback
- Programma Educativo Settimanale
- Registro
- Responsabile Questionari
- Questionario
- Assenza
- Attività
- Menù Mensa

3 Interfacce delle classi

3.1 Presentation

3.2 Application

Questa sezione include il diagramma che descrive il layer application. Esso contiene gli oggetti Control responsabili del flusso di controllo del software, i quali realizzano i design pattern descritti nel paragrafo “Riuso”.

Per ragioni di comprensibilità e di dimensione, il diagramma non è stato incluso in questo documento ma allegato nella cartella “Diagrammi allegati” col nome di “Diagramma Application.png”

3.3 Storage

Questa sezione include il diagramma che descrive il layer di storage deputato a interagire con JDBC e quindi la base dati relazionale. Esso contiene gli oggetti responsabili del reperimento e della scrittura dei dati nel database.

Per ragioni di comprensibilità e di dimensione, il diagramma non è stato incluso in questo documento ma allegato nella cartella “Diagrammi allegati” col nome di “Diagramma Storage.png”

3.4 Beans

Questa sezione include il diagramma che descrive il layer di entity deputato a immagazzinare i dati letti e da scrivere sul database. Esso contiene gli oggetti denominati “Beans”.

Per ragioni di comprensibilità e di dimensione, il diagramma non è stato incluso in questo documento ma allegato nella cartella “Diagrammi allegati” col nome di “Diagramma Beans.png”

3.5 Exceptions

Documentazione del riuso

Ritenendo necessario favorire un'alta riusabilità e una facile manutenzione del sistema per probabili modifiche future, per il sistema @silo si è ritenuto necessario usare i design pattern.

Il livello di Storage è stato implementato con l'utilizzo di un Adapter Pattern per JDBC, mentre il livello di Application è stato implementato con l'impiego dello Strategy Pattern. Inoltre @silo presenta due componenti off-the-shelf ,forum e il sistema di posta elettronica, che utilizzeranno un Adapter Pattern.

Livello di Storage : Adapter Pattern per JDBC

ManagerDB: è l'interfaccia che contiene i metodi che si useranno per gestire una qualunque tabella del database. Nel nostro caso vi è un'unica classe(**Tabella**) che implementa ManagerDB.

La classe **Tabella** ha al suo interno un oggetto **Database**.

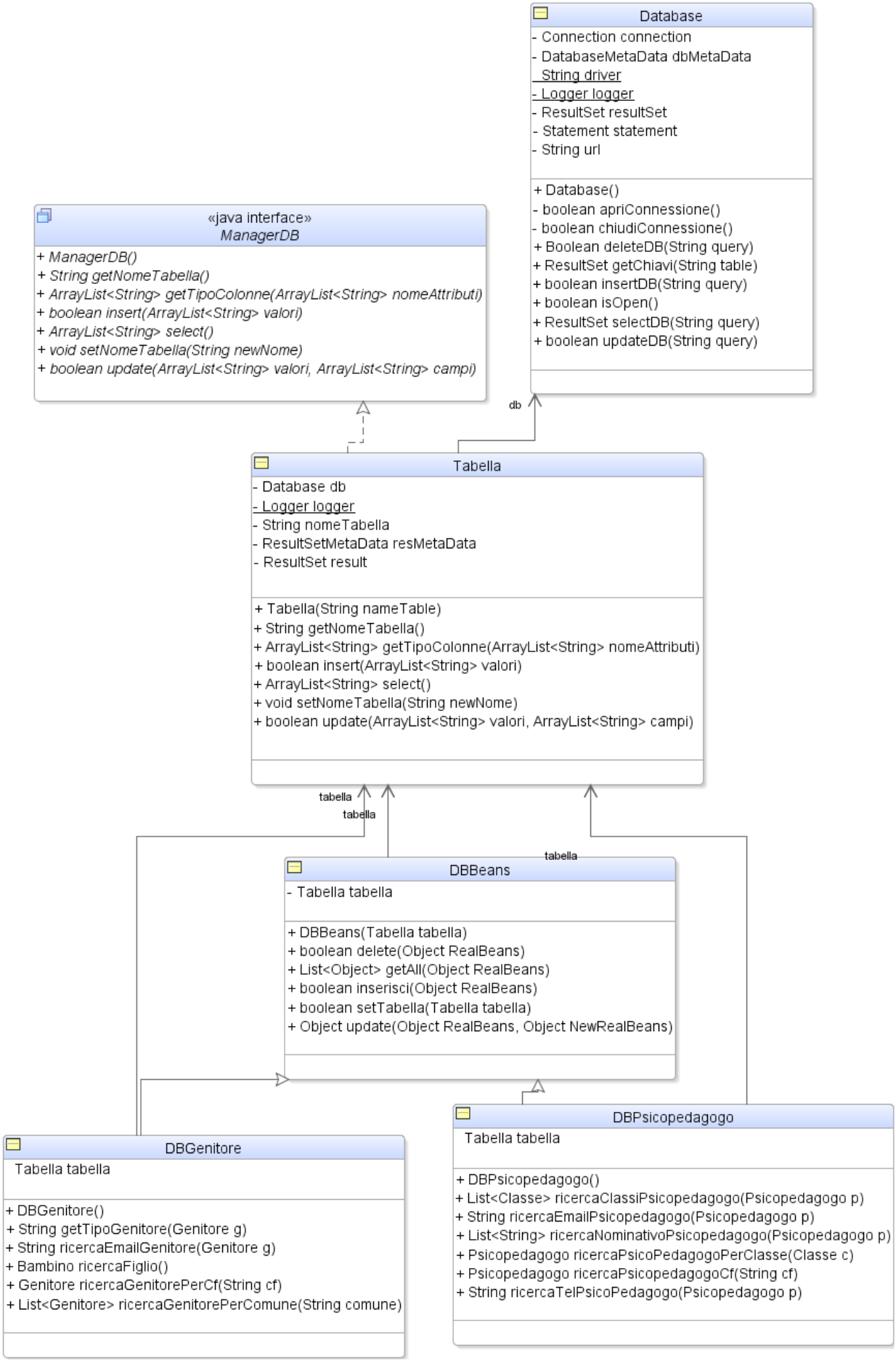
La classe **Database** si occupa di gestire la connessione al database ed al suo interno contiene dei metodi generici per eseguire delle query sul database del sistema.

DbBeans è una classe che contiene metodi per effettuare le operazioni più comuni sul database come ad esempio getAll,inserisci,update,delete ,e un oggetto di tipo Tabella, il quale si riferisce

alla tabella relativa al bean in questione. Nel caso,ad esempio, di account l'oggetto si riferirà alla tabella "Account" del database.

DB"NomeBeans" estende DbBeans,ereditando quindi i metodi comuni a tutti i beans e in ciascun DB"NomeBeans" vengono implementati dei metodi ausiliari per eseguire query specifiche sulle relative tabelle.

Di seguito è riportato il grafico dell' Adapter Pattern per JDBC:



Livello di Application : Strategy Pattern

Nel livello di Application si è deciso di utilizzare un design pattern di tipo Strategy .Questa decisione è stata presa per la presenza della ricerca avanzata che può essere effettuata da una particolare tipologia specifica di utente su un altro specifico utente. Quindi come primo passo è stato creato ContestoRicerca per selezionare la tipologia di ricerca che si andrà ad effettuare e il contesto in cui si farà,sostanzialmente in ContestoRicerca.. Il secondo passo è stata la creazione della classe AlgoritmoRicerca che, in base al tipo di utente da ricercare effettua una ricerca specifica, con un algoritmo specifico. Questo perchè:

- gli utenti che effettuano la ricerca vedono alcuni dati o tutti, in base

al tipo di utente

- gli utenti ricercati su cui viene effettuata la ricerca hanno dati

differenti in base al tipo di utente.

Di seguito è riportato il grafico del Design Pattern Strategy usato: