
Università degli Studi di Salerno

Corso di Gestione dei Progetti Software

A.A. 2015/2016

Giovanni De Costanzo


**MTP:** Sistema di gestione della programmazione didattica

# Object Design Document

*Object Design Document*

*19/01/2016*

*Versione 1.2*

## Coordinatori del progetto

| |
|---|
| Prof. ssa Filomena Ferrucci - Top Manager |
| Giovanni De Costanzo - Project Manager |

## Partecipanti

| |
|---|
| Vincenzo Belmonte |
| Francesco Vicidomini |
| Giovanni Buglione |
| Domenico Iannone |

## Revision History

| ***Data*** | ***Versione*** | ***Descrizione*** | ***Autori*** |
|---|---|---|---|
| 11/12/2015 | Versione 1.0 | Prima Stesura | Team Members |
| 21/12/2015 | Versione 1.1 | Seconda stesura | Team Members |
| 19/01/2016 | Versione 1.2 | Terza stesura | Domenico Iannone |

# SOMMARIO

# 1. INTRODUZIONE

Questo documento ha lo scopo di essere un supporto all'implementazione e di produrre un modello capace di integrare in modo coerente e preciso tutte le funzionalità individuate in fase di analisi dei requisiti.

## 1.1 COMPROMESSI DELL'OBJECT DESIGN

- **Comprensibilità vs costi:** Un aspetto molto importante è dato alla comprensibilità del codice, in quanto ogni classe e metodo devono essere facilmente interpretabili anche da chi non ha collaborato al progetto.
- **Prestazioni vs costi:** Si è scelto di utilizzare materiale fornito gratuitamente tramite licenze universitarie e Open Source non avendo a disposizione nessun budget con lo scopo di rendere il sistema performante ed efficiente.
- **Costi vs mantenimento:** L'utilizzo di materiale open source e del linguaggio PHPDoc rende il sistema facilmente modificabile e manutenibile. Trattandosi di un sistema distribuito i costi di manutenzione saranno alti.
- **Interfaccia vs easy-use:** Si è scelto di realizzare un'interfaccia grafica cercando di rendere questa, intuitiva e semplice da utilizzare per l'utente grazie all'uso di icone e form, permettendo una facile gestione del database.
- **Memoria vs efficienza:** Si è cercato di dare un giusto compromesso tra efficienza e memoria, cercando di eliminare le ridondanze e di rendere subito disponibili i dati.
  Nel diagramma E/R è stato cercato di ridurre al minimo il livello di accoppiamento delle varie entità.
- **Sicurezza vs efficienza:** Nel sistema ogni richiesta viene validata attraverso l'uso di sessioni e un controllo del livello di utenza.
  Questa caratteristica potrebbe far aumentare il tempo di risposta del sistema soprattutto in situazioni di carico elevato, ma è necessario effettuare tali controlli per rispettare i requisiti iniziali del sistema.
- **Tempo di risposta vs Spazio di memoria:** La scelta di utilizzare un DB relazionale è scaturita dai diversi vantaggi che se ne derivano:
  - Gestione consistente dei dati
  - Tempo di risposta basso rispetto all'utilizzo del file system
  - Accesso veloce e concorrente ai dati

### 1.2 Definizioni, acronimi e abbreviazioni

| Acronimo | Descrizione |
|---|---|
| RAD | Requirement Analysis Document |
| GUI | Graphical User Interface |
| SW | Software |
| HW | Hardware |
| SQL | Structured Query Language |
| SDD | System Design Document |
| DBMS | Database Management System |
| P.C.D. | Presidente del consiglio didattico |

### 1.3 Riferimenti

- SDD System Document Design 1.2
- RAD Requirement Analysis Document 4.1

## 2. Linee guida per l'implementazione

### 2.1 Nomi di file

Si utilizzano le seguenti estensioni per i file:

- per i sorgenti PHP è .php;
- per i fogli CSS è .css;
- per i sorgenti JAVASCRIPT è .js.

### 2.2 Organizzazione dei file

Un file è composto da sezioni separate da linee bianche e da commenti opzionali che identificano ogni sezione.
File più lunghi di 2000 linee sono ingombranti e devono essere evitati.

#### 2.2.1 File sorgenti

Ogni file sorgente PHP contiene una singola classe pubblica o un'interfaccia. Quando ci sono classi e interfacce private associate con la classe pubblica, è possibile inserirle nello stesso file sorgente della classe pubblica. La classe pubblica deve essere la prima classe o interfaccia nel file.
I file sorgenti PHP hanno la seguente struttura:

- commento di inizio che elenca il nome della classe, descrizione, autore e informazioni sulla versione, informazioni di copyright:

```
/**
 * Nome della classe
 *
 * Descrizione
 *
 * Author
 * Informazioni di versione
 *
 * 2015 - Copyright by MTPGroup
 */
```

- Dichiarazioni di classe e interfaccia: l'ordine in cui le dichiarazioni di una classe o interfaccia devono apparire è il seguente:
    - commento di documentazione della classe/interfaccia;
    - istruzione "class" o "interface";
    - variabili di classe (static): prima le variabili di classe public, poi quelle protected e infine quelle private;
    - variabili di istanza: prima quelle public, poi quelle protected e infine quelle private;
    - costruttori
    - metodi: raggruppati in base alla loro funzionalità per rendere più semplice la lettura e la comprensione del codice.

## 2.3 INDENTAZIONE

Una unità di indentazione equivale a una tabulazione.

### 2.3.1 Lunghezza delle linee

Cercare di evitare linee più lunghe di ottanta caratteri, perché esse non vengono ben gestite da molti terminali e strumenti software. Per la documentazione si utilizza una più corta lunghezza di linea, generalmente non più di settanta caratteri.

### 2.3.2 SPOSTAMENTO DI LINEE

Quando un'espressione supera la lunghezza della linea, occorre spezzarla secondo le seguenti regole:

- interrompere la linea dopo una virgola;
- interrompere la linea prima di un operatore;
- allineare la nuova linea con l'inizio dell'espressione nella linea precedente.

## 2.4 COMMENTI

I siti scritti in PHP possono avere due tipi di commenti: commenti d'implementazione e commenti di documentazione.
I commenti d'implementazione sono delimitati da / *...*/ e //. I commenti di documentazione (doc comments) sono delimitati da /**...*/. I doc comments possono essere estratti in file HTML utilizzando PHPDoc. I commenti di implementazione sono dei mezzi per commentare il codice o per commentare una particolare implementazione. I doc comments vengono utilizzati per descrivere la specifica del codice che può essere letta anche da uno sviluppatore che non ha il codice in mano.
I commenti dovrebbero essere usati per dare una panoramica del codice e per fornire informazioni aggiuntive non comprensibili leggendo direttamente il codice. I commenti devono contenere solo informazioni rilevanti per leggere e comprendere il programma.

I commenti non dovrebbero essere inclusi in grandi riquadri tracciati con asterischi o altri caratteri, né dovrebbero includere caratteri speciali come backspace.

### 2.4.1 FORMATTAZIONE COMMENTO DI IMPLEMENTAZIONE

I programmi possono avere tre tipi di commenti di implementazione:

- commenti di blocco: sono usati per fornire descrizioni di file, metodi, strutture dati e algoritmi. I commenti di blocco possono essere usati all'inizio di ogni file e prima di ogni metodo. Possono inoltre essere usati in altri punti, come all'interno dei metodi. Un commento di blocco dovrebbe essere preceduto da una linea bianca di separazione dal codice.

```
/**
 * Commento di blocco
 */
```

- commenti a linea singola: sono brevi commenti che possono apparire su una singola linea di codice ed indentati al livello del codice che seguono. Se un commento non può essere scritto su una linea singola, deve seguire il formato di commento di blocco. Un commento a linea singola deve essere preceduto da una linea bianca. Ecco un esempio:

```
while ($i!=0){

    /*Commento*/
    ...

}
```

- Commenti di fine linea: Il delimitatore di commento // può commentare una linea completa o una parte di essa. Non dovrebbe essere usato su più linee consecutive per commenti testuali; comunque, può essere usato su più linee consecutive per commentare sezioni del codice. Ecco un esempio:

```
if($i==0){
    i++;//commento
    ...
}
```

### 2.4.2 COMMENTI DI DOCUMENTAZIONE

Descrivono classi PHP, interfacce, costruttori, metodi e campi. Ogni doc comment è compreso all'interno dei delimitatori di commento /**...*/, con un commento per ogni classe, interfaccia o membro. Questo commento deve apparire solo prima della dichiarazione:

```
/**
* La classe Person....
*/
class Person {
    ...
}
```

Un doc comment si compone di una descrizione seguita da un blocco di tag.
I tag da utilizzare sono @author, @exception, @param, @return, @see.

## 2.5 DICHIARAZIONI

Per favorire il commento di una dichiarazione, saranno scritte una per linea.

```
public $i; //Commento
public $j; //Commento
```

è preferito rispetto a

```
public $i,$j; //Commento
```

Un'altra alternativa accettabile è usare le tabulazioni, cioè:

```
public $i;          //Commento
public $j;          //Commento
public $variabile;  //Commento
```

### 2.5.2 INIZIALIZZAZIONE

Provare ad inizializzare le variabili locali nel punto in cui sono dichiarate. L'unica ragione per non inizializzare una variabile dove è dichiarata è se il suo valore iniziale dipende da un calcolo che prima occorre eseguire.

### 2.5.3 POSIZIONE

Mettere le dichiarazioni all'inizio dei blocchi. Un blocco è un codice racchiuso entro parentesi graffe aperta e chiusa.

### 2.5.4 DICHIARAZIONE DI CLASSE E INTERFACCIA

Quando si codificano classi e interfacce, si dovrebbero rispettare le seguenti regole di formattazione:

- non mettere spazi tra il nome del metodo e la parentesi "(" che apre la lista dei parametri;
- la parentesi graffa aperta "{" si trova alla fine della stessa linea dell'istruzione di dichiarazione;
- La parentesi graffa chiusa "}" deve essere indentata con la corrispondente istruzione di apertura, eccetto il caso in cui c'è un'istruzione vuota, allora la "}" dovrebbe essere immediatamente dopo la "{".

## 2.6 ISTRUZIONI

### 2.6.1 ISTRUZIONI SEMPLICI

Ogni linea deve contenere al massimo un'istruzione.

```
$i++;               //OK
$j++;               //OK
$i=$i+$j; $j=$i+5;  //NO
```

### 2.6.2 ISTRUZIONI COMPOSTE

Le istruzioni racchiuse dovrebbero essere indentate ad un ulteriore livello rispetto all'istruzione composta.

La parentesi graffa aperta dovrebbe stare alla fine della linea che inizia l'istruzione composta, mentre la parentesi graffa chiusa dovrebbe iniziare una linea es essere indentata verticalmente con l'inizio dell'istruzione composta.

### 2.6.3 ISTRUZIONI RETURN

Un'istruzione return con un valore non dovrebbe usare parentesi, a meno che queste rendano in qualche modo il valore ritornato più ovvio.

### 2.6.4 ISTRUZIONI IF, IF-ELSE, IF-ELSE-IF ELSE

La classe di istruzioni if-else deve avere la seguente forma

```
if (cond){
    istruzioni
}

if (cond){
    istruzioni
}
else{
    istruzioni
}

if (cond1){
    istruzioni
}
else if (cond2){
    istruzioni
}
else{
    istruzioni
}
```

### 2.6.5 ISTRUZIONI FOR

Un'istruzione for dovrebbe avere la seguente forma

```
for(inizializzazione;condizione;aggiornamento){
    istruzioni
}
```

### 2.6.6 Istruzioni while

Un'istruzione while dovrebbe avere la seguente forma

```
while(condizione){
  istruzioni
}
```

### 2.6.7 ISTRUZIONI DO-WHILE

Un'istruzione do-while dovrebbe avere la seguente forma

```
do{
    istruzioni
}while(cond);
```

### 2.6.8 ISTRUZIONI SWITCH

Un'istruzione switch dovrebbe avere la seguente forma

```
switch(valore) {

   case valore 1:
     istruzioni
   break;
   case valore 2:
     istruzioni
   break;

   ...

   case valore n:
     istruzioni
   break;

   default:
     istruzioni
}
```

### 2.6.9 ISTRUZIONI TRY-CATCH

Un'istruzione try-catch dovrebbe avere la seguente forma

```
try{
   istruzioni
} catch (eccezione){
   istruzioni
}
```

Un'istruzione try-catch può inoltre essere seguita da finally, che viene eseguita indipendentemente dal fatto che il blocco try sia stato o meno completato con successo.

```
try{
   istruzioni
} catch (eccezione){
   istruzioni
}finally{
   istruzioni
}
```

## 2.7 SPAZI BIANCHI

### 2.7.1 LINEE BIANCHE

Due linee bianche dovrebbero essere sempre usate nelle seguenti circostanze:

- fra sezioni di un file sorgente;
- fra definizioni di classe e interfaccia.

Una linea bianca dovrebbe essere sempre usata nelle seguenti circostanze:

- fra metodi;
- fra le variabili locali in un metodo e la sua prima istruzione;
- prima di un commento di blocco o a singola linea;
- fra sezioni logiche all'interno di un metodo

### 2.7.2 SPAZI BIANCHI

Spazi bianchi dovrebbero essere usati nelle seguenti circostanze:

- una parola chiave seguita da una parentesi dovrebbe essere separata da uno spazio;
- uno spazio bianco non dovrebbe essere usato fra il nome di un metodo e le sue parentesi d'apertura;
- uno spazio bianco dovrebbe essere interposto dopo le virgole nelle liste di argomenti;
- tutti gli operatori binari eccetto . (operatore punto) dovrebbero essere separati dai loro operandi tramite spazi.

## 2.8. CONVENZIONI DI NOMI

### 2.8.1 CLASSI

I nomi di classe dovrebbero essere sostantivi, con lettere minuscole e, sia la prima lettera del nome della classe sia la prima lettera di ogni parola interna, deve essere maiuscola. Cercare di rendere i nomi delle classi semplici, descrittivi e che rispettino il dominio applicativo. Usare parole intere evitando acronimi e abbreviazioni.

### 2.8.2 INTERFACCE

I nomi di interfaccia seguono le stesse regole dei nomi di classi.

### 2.8.3 METODI

I nomi dei metodi devono essere verbi con iniziale minuscola e a gobba di cammello.
Cercare di rendere i nomi dei metodi semplici, descrittivi e che rispettino il dominio applicativo. I nomi dei metodi non devono iniziare con caratteri di underscore o di dollaro.

### 2.8.4 VARIABILI

Tutte le variabili e le istanze di classe devono essere scritte con iniziale minuscola e a gobba di cammello. La scelta di un nome deve essere mnemonica e deve rispettare il dominio applicativo.
I nomi di variabili di un solo carattere dovrebbero essere evitati.

### 2.8.5 COSTANTI

I nomi delle variabili dichiarate come costanti di classe devono essere scritte in lettere tutte maiuscole con le parole separate da underscore. La scelta di un nome deve essere mnemonica e deve rispettare il dominio applicativo.

## 2.9 CONSUETUDINI DI PROGRAMMAZIONE

### 2.9.1 FORNIRE ACCESSO A VARIABILI DI ISTANZA O DI CLASSE

Non rendere pubblica una variabile di istanza o di classe senza una buona ragione. Le variabili di istanza devono essere scritte o lette attraverso delle chiamate a metodi.

### 2.9.2 RIFERIRE VARIABILI E METODI DI CLASSE

Evitare di usare un oggetto per accedere a variabili o metodi di classe static. Usare invece il nome della classe.

### 2.9.3 ASSEGNAMENTO DI VARIABILI

Evitare di assegnare a più variabili lo stesso valore in una sola istruzione.
Non usare l'operatore di assegnamento in un punto in cui può essere facilmente confuso con l'operatore di uguaglianza.
Non usare assegnamenti innestati nel tentativo di migliorare le prestazioni a tempo di esecuzione.
Questo è compito del compilatore.

### 2.9.4 PARENTESI

E' generalmente una buona idea usare le parentesi liberamente in espressioni che coinvolgono operatori misti per evitare problemi di precedenza degli operatori.

### 2.9.5 VALORI RITORNATI

Provare a rendere la struttura del programma aderente alle proprie intenzioni.

## 2.10 SCRIPT JAVASCRIPT

Gli script che svolgono funzioni diverse dalla definizione degli stili della pagina dovrebbero essere collocati in file dedicati.
Il codice Javascript deve seguire le stesse convenzioni per il layout e i nomi del codice PHP.
I commenti verranno inseriti secondo le stesse modalità descritte per il codice PHP.

## 2.11 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 PHP.
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 PHPDoc, che ne spiegano lo scopo, e seguiti da 2 righe bianche.

## 2.12 DATABASE SQL

Le tabelle del database dovrebbero essere in terza forma normale. 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:

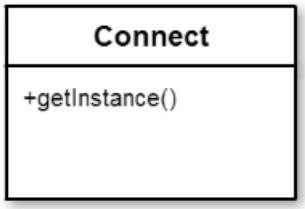
1. sono costituiti da sole lettere minuscole;
2. se il nome è costituito da più parole, queste sono separate da un underscore (_);
3. il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

# 3. DESIGN PATTERN

Sono utilizzati nello sviluppo dell'applicazione web MTP due design pattern: il Singleton Pattern e lo Strategy Pattern.

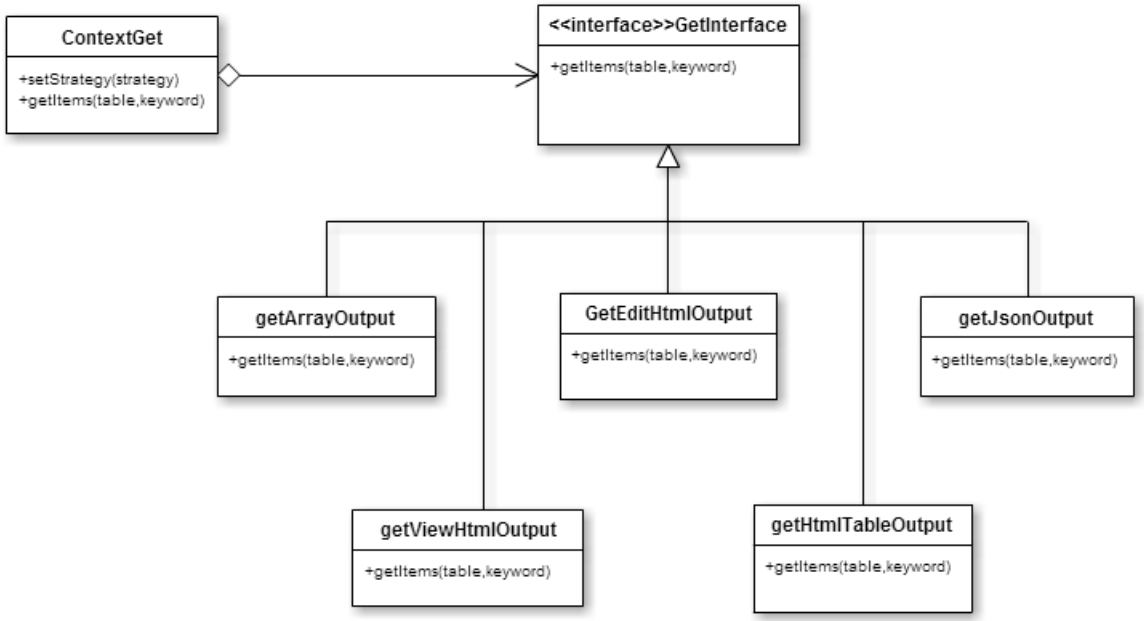
## 3.1 USO DEL SINGLETON PATTERN

La web application che il team intende progettare utilizza il design pattern Singleton per gestire la connessione col Database.
Il Singleton Pattern, nello specifico, permette di creare una e una sola istanza di connessione.



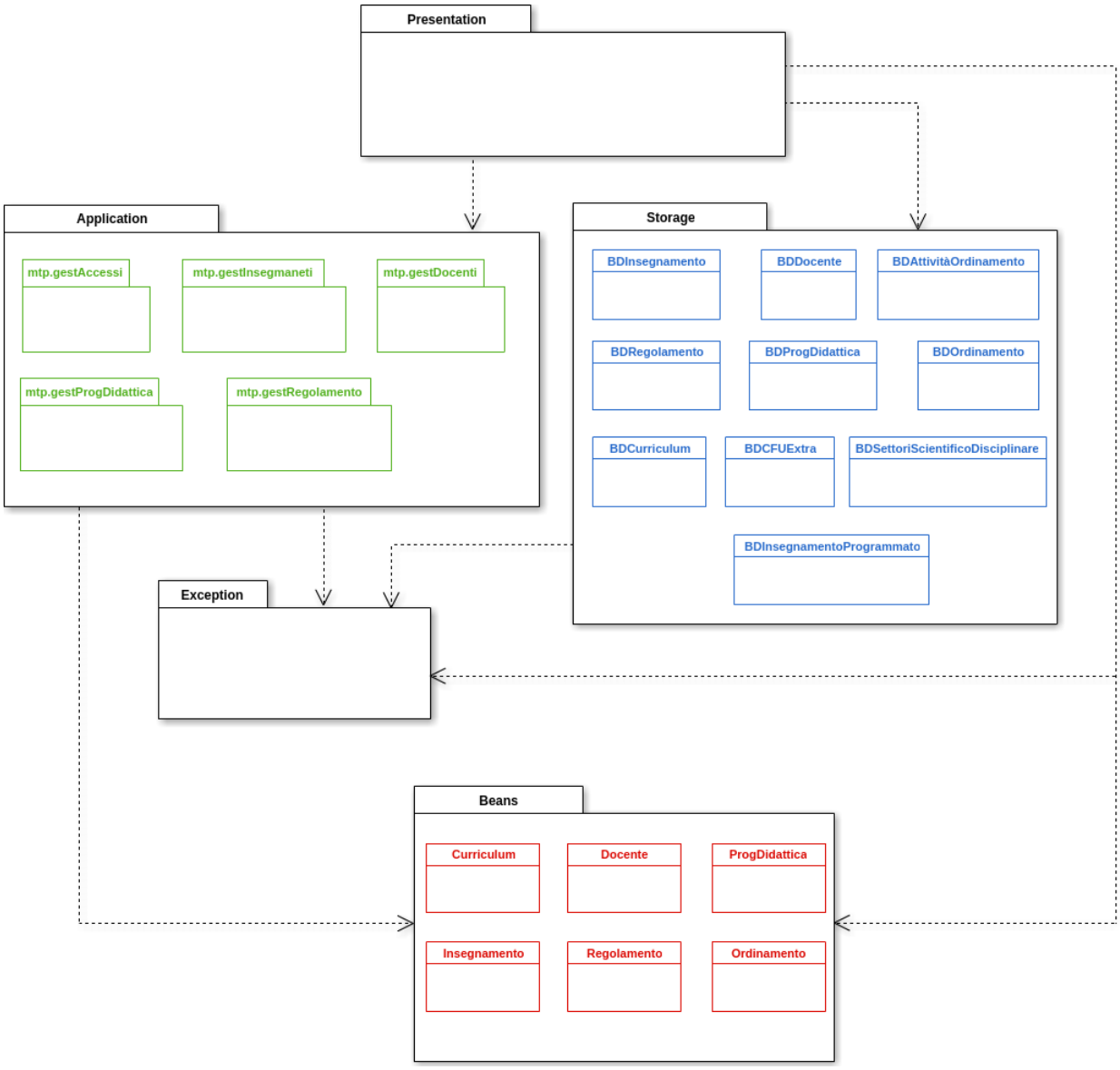
## 3.2 USO DELLO STRATEGY PATTERN

Lo Strategy Pattern verrà utilizzato per implementare le chiamate ad interrogazioni SQL nei casi in cui devono essere costruite le tabelle ed effettuate visualizzazioni.



# 4. PACKAGES

Di seguito la divisione in packages del sistema:



## 4.1 PRESENTATION

Il sottosistema Presentation ha il compito di gestire le interfacce grafiche dell'intero sistema. Esse permettono l'interazione con l'utente.

## 4.2 APPLICATION

Nel sottosistema Application vengono implementate tutte le funzionalità che permettono di manipolare, gestire e far transitare i dati dal sottosistema Presentation al sottosistema Storage.
È suddiviso nei seguenti packages:

*mtp.gestAccessi:* package che si occupa della gestione degli accessi al sistema, della registrazione, dell'inserimento e la modifica di utenti nel sistema.
*mtp.gestInsegnamenti:* package che permette di eseguire le operazioni di inserimento, ricerca e modifica degli insegnamenti.
*mtp.gestDocenti:* package che si occupa di eseguire le operazioni di inserimento, ricerca e modifica degli insegnamenti.
*mtp.gestProgDidattica:* package che permette la gestione della programmazione didattica permettendo di inserire tutti gli insegnamenti ad essa relativi con i rispettivi docenti, di eliminarli e di modificarli.
*mtp.gestRegolamento:* package che si occupa della gestione del regolamento, permettendo di creare un nuovo curriculum con i rispettivi insegnamenti che verrà inserito all'interno del regolamento e di modificarlo.

## 4.3 STORAGE

Nel sottosistema Storage sono presenti tutti i dati persistenti del sistema e si occupa di gestire questi ultimi tramite 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:

1. BDInsegnamento
2. BDDocente
3. BDInsegnamentoProgrammato
4. BDProgrammazioneDidattica
5. BDCurriculum
6. BDRegolamento
7. BDOrdinamento
8. BDAttivitàOrdinamento
9. BDCFUExtra
10. BDSettoriScientificoDisciplinare

## 4.4 BEANS

Il sottosistema Beans si occupa di gestire i dati, tramite la creazione e la manipolazione di entità che fanno parte del sistema, entità che identificano concetti ben distinti e con un proprio ruolo. Le classi che costituiscono Beans sono:

1. Insegnamento
2. Docente
3. ProgDidattica
4. Curriculum
5. Regolamento
6. Ordinamento.

## 4.5 EXCEPTION

Il sottosistema Exception descrive il package per la gestione delle eccezioni all'interno del sistema. Esso contiene le eccezioni che alterano la normale esecuzione del codice in caso di errori.