ALTRA LOGICA

LoginData

```
private String name;
private String password;

public LoginData();
public LoginData(Name name, Password pass);

public String getName();
public String getPassword();

public String setName();
public String setPassword();
```

RegisterData

```
private String name;
private String password;

public LoginData();
public LoginData(Name name, Password pass);

public String getName();
public String getPassword();

public String setName();
public String setPassword();
```

PrenotazioneData

```
private String datiPrenot;
private String ...;

public PrenotazioneData();
public PrenotazioneData(...);

public String getdatiPrenot();
public String get...();

public String setdatiPrenot();
public String set...();
```

Altro

```
ALTRO
ALTRO
```

Questi Dovranno Essere Semplici Bean, non credo che potremo permetterci di avere Classi "utenti" così grosse(come quelle che abbiamo nell'altro foglio)

Una soluzione che mi è venuta in mente è distinguere "UserData" (Il Bean dati da RDB) da "User"(la classe generica utente)

QUESTA PORZIONE DIPENDE MOLTO DALL'ERA SECONDO ME

Dominio

PersistanceToModelFacade(Singleton)

```
Optional<T> getAccount(long id);          //login
void saveAccount(T t);                    //create
void updateAccount(T t, String[] params); //modifica

void savePrenotazione(T t);   //nuova prenotazione
Optional<T> getPrenotazione(long id); //view cliente
... Tutti i metodi che sono sicuro serviranno
```

La classe Facade Singleton dello strato di persistenza

L'oggetto che comunicherà col dominio

<<Interface>> IAccountDAO<T>

```
Optional<T> getAccount(long id);
List<T> getallAccounts();
void saveAccount(T t);
void updateAccount(T t, String[] params);
void deleteAccount(T t);
```

<<Interface>> IPrenotazioniDAO<T>

```
Optional<T> getPrenotazione(long id);
List<T> getAllPrenotazioni();
void savePrenotazione(T t);
void updatePrenotazione(T t, String[] params);
void deletePrenotazione(T t);
```

<<Interface>> ITurniDAO<T>

```
Optional<T> getTurno(long id);
List<T> getAllTurni();
void saveTurno(T t);
void updateTurno(T t, String[] params);
void deleteTurno(T t);
```

Le interfacce IDAO Che specificano i CRUD

Usiamo un generics su ciascuna interfaccia per renderla più vaga possibile?

Perchè molteplici interfacce e non una che implementi i CRUD generici(di cui fare l'override nelle classi concrete)?

Perchè non è detto che si usi solo il DB per memorizzare info, con un ottica vista verso la modifica nel futuro, è meglio avere una struttura facilmente ampliabile

AccountRdbDAO

```
- List<T> t;
Optional<T> getAccount(long id);
List<T> getallAccounts();
void saveAccount(T t);
void updateAccount(T t, String[] params);
void deleteAccount(T t);
```

PrenotazioniRdbDAO

```
- List<T> t;
Optional<T> getPrenotazione(long id);
List<T> getAllPrenotazioni();
void savePrenotazione(T t);
void updatePrenotazione(T t, String[]
params);
void deletePrenotazione(T t);
```

TurniRdbDAO

```
- List<T> t;
Optional<T> getTurno(long id);
List<T> getAllTurni();
void saveTurno(T t);
void updateTurno(T t, String[] params);
void deleteTurno(T t);
```

Concretizzazioni DAO, gli oggetti con le query ecc...

Qui ad esempio, per rendere il progetto più fico potremmo avere una diversa Persistenza Dati per i turni

TURNI ANCORA DA DEFINIRE

DB Connection

```
Tutti gli attributi visti in DBConnection

private static void init(T t);
public static Connection startConnection(T t, String[] params);
public static boolean isOpen(Connection Conn);
public static Connection closeConnection(Connection Conn);
```

Copia Incolla da file JDBC Prof, non mi sembra ci sia nulla da modificare

Gestione connessioni/connettore mySQL

JDBC DAO

RDB

DataBase Relazionale descritto dall'ERA di Ibra