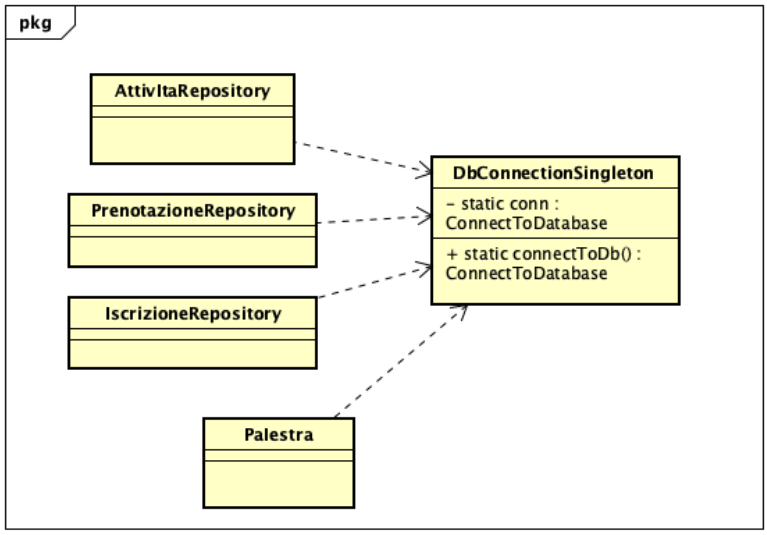
Pattern utilizzati per la progettazione

Per effettuare in maniera ottimale la progettazione sono stati implementati 3 pattern tra cui due creazionali (**Singleton e Prototype**) e uno comportamentale (**Command**).

1 . Singleton

Al fine di gestire correttamente la connessione al DB si è deciso di applicare il Singleton Pattern per creare un'unica istanza che verrà poi richiamata nella classe del metodo main() o in qualunque altra classe che ne abbia bisogno.

Diagramma delle classi Singleton:



Codice Singleton (che richiama la classe di connessione al DB → ConnectToDatabase):

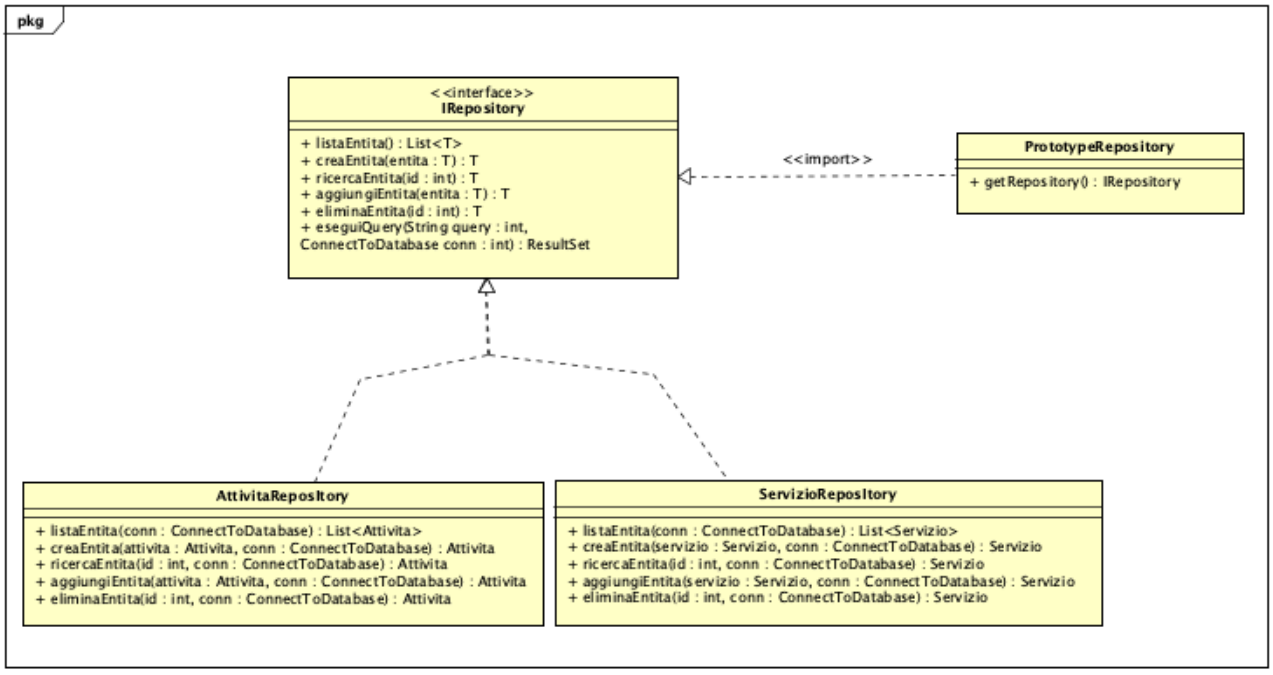
```
*/
public class DbConnectionSingleton {

    public static class Connect
    {
        private static ConnectToDatabase conn = null;  //questa è la mia instanza

        public static ConnectToDatabase connectToDb(){
            if(conn == null)
            {
                conn = new ConnectToDatabase();
            }
            return conn;
        }

    }

}
```

2 . Prototype

Al fine di poter effettuare una progettazione di tipo Generic , si è deciso di implementare il Prototype Pattern cosi da avere un'interfaccia univoca (**IRepository**) che definisca i metodi comuni necessari a tutte le classi.
Il principio base è quello di fare l'override di tali metodi nelle classi EntitaRepository (dove Entita verrà sostituito col nome delle diverse classi) e sulla base di ciò che si vuole gestire il file **RepositoryPrototype** creerà la giusta istanza della classe che potrà essere dunque utilizzata.
Ovviamente all'interno di ogni repository verranno implementati i metodi col tipo effettivo di classe.

N.B
La notazione "Repository" è stata usata al fine di identificare la corretta separazione fra lo strato di modello (Classi base come Attivita, Servizio, Socio, ecc ….) e quello di persistenza dei dati (ovvero il DB) tipico del Repository Pattern utilizzato per la progettazione di software secondo la struttura tipo del MVC (Model-View-Controller).

Diagramma delle classi Prototype:



Codice Prototype (per semplicità con una sola classe che implementa l'interfaccia, da esempio) :

```java
public interface IRepository<T> {

    public List<T> listaEntita() throws SQLException;
    public T creaEntita(T entity) throws SQLException;
    public T aggiornaEntita(T entity) throws SQLException;
    public boolean eliminaEntita(String[] chiavi) throws SQLException;
    public ResultSet eseguiQuery(String query) throws SQLException;
    public T ricercaEntita(String[] chiavi) throws SQLException;
}
```

Interfaccia -> IRepository

```java
public class PrenotazioneRepository implements IRepository<Prenotazione>  {

    List<Prenotazione> listaPrenotazione = new ArrayList<Prenotazione>();
    Statement stmt = null;
    ConnectToDatabase conn;



    @Override
    public List<Prenotazione> listaEntita() throws SQLException {...30 lines }

    @Override
    public Prenotazione creaEntita(Prenotazione entity) throws SQLException {...14 lines }

    @Override
    public Prenotazione aggiornaEntita(Prenotazione entity) throws SQLException {...13 lines }

    @Override
    public boolean eliminaEntita(String[] chiavi) throws SQLException {...18 lines }

    @Override
    public ResultSet eseguiQuery(String query) throws SQLException {...8 lines }

    @Override
    public Prenotazione ricercaEntita(String[] chiavi) throws SQLException {...25 lines }
```

Classe che implementa l'interfaccia

3 . Command

Per poter invece gestire e parametrizzare le chiamate ai metodi della repository si è deciso di applicare il pattern **Command**. Nel sistema ogni comando è visto come un oggetto separato che può essere passato e invocato (mediante il metodo execute()). L'interfaccia generica istanziata è definita come ICommand.

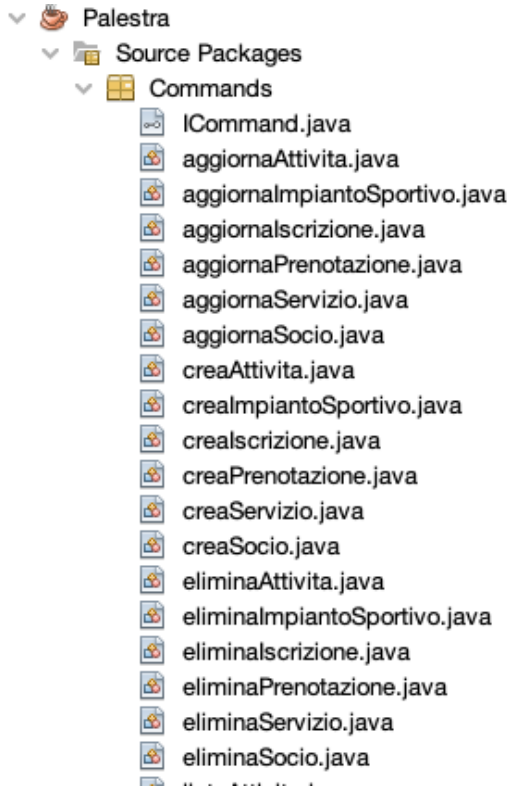
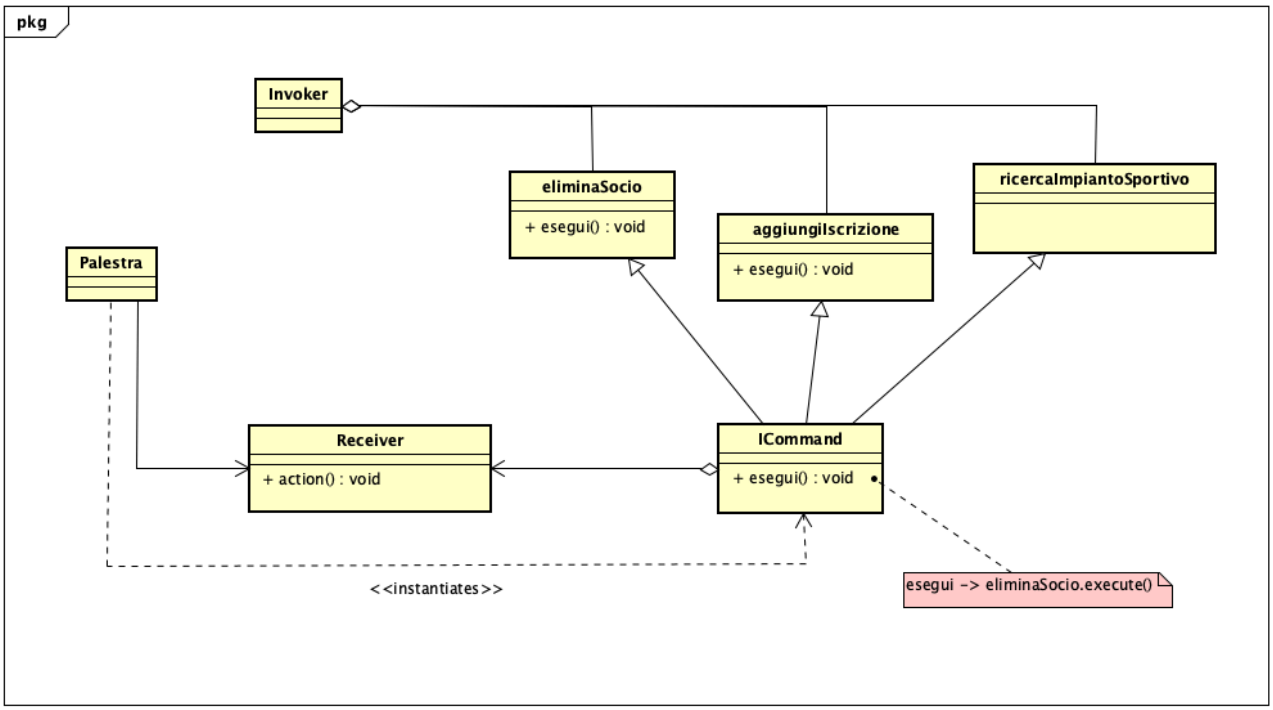


Diagramma delle classi Command (per semplicità si mettono solo alcune classi a scopo illustrativo):



Codice Command:

```
*/
public interface ICommand {

    public static void execute() throws ParseException, SQLException {};

}
```

Interfaccia -> ICommand

```
public class aggiornaAttivita implements ICommand{

    private Attivita attivita;
    ConnectToDatabase conn = new DbConnectionSingleton.Connect().connectToDb();


       static RepositoryPrototype prototypeRepo = new RepositoryPrototype();
       static Scanner sc = new Scanner(System.in);

       public aggiornaAttivita()
       {

       }

   public static void execute() throws ParseException, SQLException
   {
       prototypeRepo.setRepository("Attivita");

           System.out.println("----------------------------------------");
           System.out.println("Modifica Informazioni Attivita ");
           System.out.println("----------------------------------------");
           System.out.println("Inserisci descrizione attivita");
           String descrizione = sc.nextLine();
           System.out.println("Inserisci prezzo attivita");
           double prezzo = new Scanner(System.in).nextDouble();
           prototypeRepo.repo.aggiornaEntita(new Attivita(descrizione, prezzo));

   }

}
```

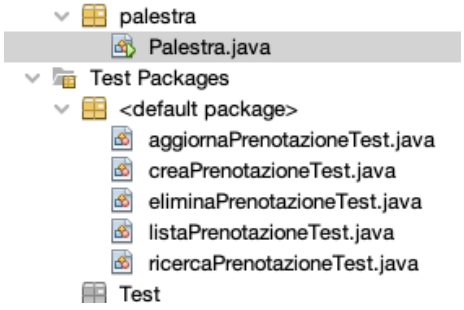
Classe che implementa l'interfaccia

Test in jUnit

Al fine di testare i metodi realizzati si è utilizzato jUnit (tramite librerie importate sul progetto realizzato con l'IDE NetBeans). Per semplicità e tempo si è deciso di effettuare il test dei metodi più significativi, relativi alla "Prenotazione di impianti sportivi".

Nello specifico i test racchiudono i classici metodi quali:

- Ricerca di una prenotazione (per sapere se un impianto è occupato in un giorno specifico)
- Aggiornamento di una prenotazione (cambio data ad esempio)
- Eliminazione della prenotazione (una volta che l'utente ha usufruito dell'impianto sportivo)
- Lista di tutte le prenotazioni (per consentire una ricerca fluida)
- Creazione di una nuova prenotazione relativa ad un impianto sportivo.



I metodi testati si trovano (nel progetto Palestra), sotto la directory Test Package, dove vengono collocati singolarmente alla creazione di un nuovo file di test jUnit.

Si procede dunque all'illustrazione di come sono stati effettuati i test.

1 Test Lista Prenotazione

```
@Test
public void testListaPrenotazione() throws SQLException  {

    RepositoryPrototype prototypeRepo = new RepositoryPrototype();
    Statement stmt;
    ConnectToDatabase conn = new ConnectToDatabase();
    Connection c = conn.connect();
    stmt = c.createStatement();
    stmt.executeUpdate("DELETE FROM prenotazione");

    prototypeRepo.setRepository("Prenotazione");

       ArrayList<Prenotazione> listPrenotazione = new ArrayList<>();
       Prenotazione prenotazione = new Prenotazione("Mario", "Rossi", "Campo Calcio", "19-07-2022");
       Prenotazione prenotazione1 = new Prenotazione("Giuseppe", "Verdi", "Campo Tennis", "20-07-2022");
       listPrenotazione.add(prenotazione);
       listPrenotazione.add(prenotazione1);
       prototypeRepo.repo.creaEntita(prenotazione);
       prototypeRepo.repo.creaEntita(prenotazione1);

       List<Prenotazione> a = prototypeRepo.repo.listaEntita();
       var d = listPrenotazione;

       for(int i = 0; i<a.size() && i<d.size(); i++){
       assertEquals(d.get(i).Nome , a.get(i).Nome);
       assertEquals(d.get(i).Cognome , a.get(i).Cognome);
       assertEquals(d.get(i).Impianto , a.get(i).Impianto);
       assertEquals(d.get(i).Data , a.get(i).Data);
       }
    }
```

2 Test Crea Prenotazione

```
@Test
public void testCreaPrenotazione() throws SQLException  {

  Statement stmt;
  ConnectToDatabase conn = new ConnectToDatabase();
  Connection c = conn.connect();
  stmt = c.createStatement();
  stmt.executeUpdate("DELETE FROM prenotazione");
  RepositoryPrototype prototypeRepo = new RepositoryPrototype();

  Prenotazione prenotazione = new Prenotazione("Francesco", "Maurici", "Campo Calcio", "19-07-2022");

  prototypeRepo.setRepository("Prenotazione");

  prototypeRepo.repo.creaEntita(prenotazione);

  Prenotazione d = new Prenotazione(" ", " ", " ", " ");

  assertFalse(prenotazione.Nome.equals(d.Nome));
}
```

3 Test Ricerca Prenotazione

```
@Test
 public void testRicercaPrenotazione() throws SQLException  {

    RepositoryPrototype prototypeRepo = new RepositoryPrototype();

    Statement stmt;
    ConnectToDatabase conn = new ConnectToDatabase();
    Connection c = conn.connect();
    stmt = c.createStatement();
    stmt.executeUpdate("DELETE FROM prenotazione");

   Prenotazione prenotazione = new Prenotazione("Daniele", "Giusti", "Campo Basket", "25-06-2022");

   prototypeRepo.setRepository("Prenotazione");

   prototypeRepo.repo.creaEntita(prenotazione);
   String[] param = new String[2];
   param[0] = "Daniele";
   param[1] = "Giusti";
   var a = prototypeRepo.repo.ricercaEntita(param);
   var d = (Prenotazione) a;
   assertTrue(prenotazione.Nome.equals(d.Nome));
   assertTrue(prenotazione.Cognome.equals(d.Cognome));
 }
```

4 Test Aggiorna Prenotazione

```
@Test
 public void testAggiornaPrenotazione() throws SQLException  {

   Statement stmt;
   ConnectToDatabase conn = new ConnectToDatabase();
   Connection c = conn.connect();
   stmt = c.createStatement();
   stmt.executeUpdate("DELETE FROM prenotazione");
   RepositoryPrototype prototypeRepo = new RepositoryPrototype();

   prototypeRepo.setRepository("Prenotazione");

   Prenotazione prenotazione = new Prenotazione("Vincenzo", "Zimbone", "Campo Calcio", "19-07-2022");
   prototypeRepo.repo.creaEntita(prenotazione);

   Prenotazione aggiornamento = new Prenotazione("Vincenzo", "Zimbone", "Campo Calcio", "27-07-2022");
   prototypeRepo.repo.aggiornaEntita(aggiornamento);

   assertTrue(prenotazione.Nome.equals(aggiornamento.Nome));
   assertTrue(prenotazione.Cognome.equals(aggiornamento.Cognome));
   assertTrue(prenotazione.Impianto.equals(aggiornamento.Impianto));
   assertFalse(prenotazione.Data.equals(aggiornamento.Data));
 }
```

5 Test Elimina Prenotazione

```
@Test
 public void testEliminaPrenotazione() throws SQLException  {

   Statement stmt;
   ConnectToDatabase conn = new ConnectToDatabase();
   Connection c = conn.connect();
   stmt = c.createStatement();
   stmt.executeUpdate("DELETE FROM prenotazione");

   RepositoryPrototype prototypeRepo = new RepositoryPrototype();

   prototypeRepo.setRepository("Prenotazione");
   Prenotazione prenotazione = new Prenotazione("Mario", "Rossi", "Campo Tennis", "19-07-2022");

   prototypeRepo.repo.creaEntita(prenotazione);
   String[] param = new String[2];
   param[0] = "Campo Calcio";
   param[1] = "19-07-2022";
   boolean a = prototypeRepo.repo.eliminaEntita(param);
   var b = true;
   assertTrue(a = b);

 }
```