JPA :

It guides how to use POJO to interact with Persistant src (DB)

$$\begin{tikzcd}\text{POJO} \arrow[r, leftrightarrow, dashed] & \text{TABLE}\end{tikzcd}$$

JPA is just a spec (still needs implementation) : JSR

Hibernate (implementation) based on JPA

# uses JDBC in backend:

auto : connection, state, ResultSet, SQL Query, transaction

Hibernate alt to SQL ~ HQL (Object Oriented)

Implementing JPA support

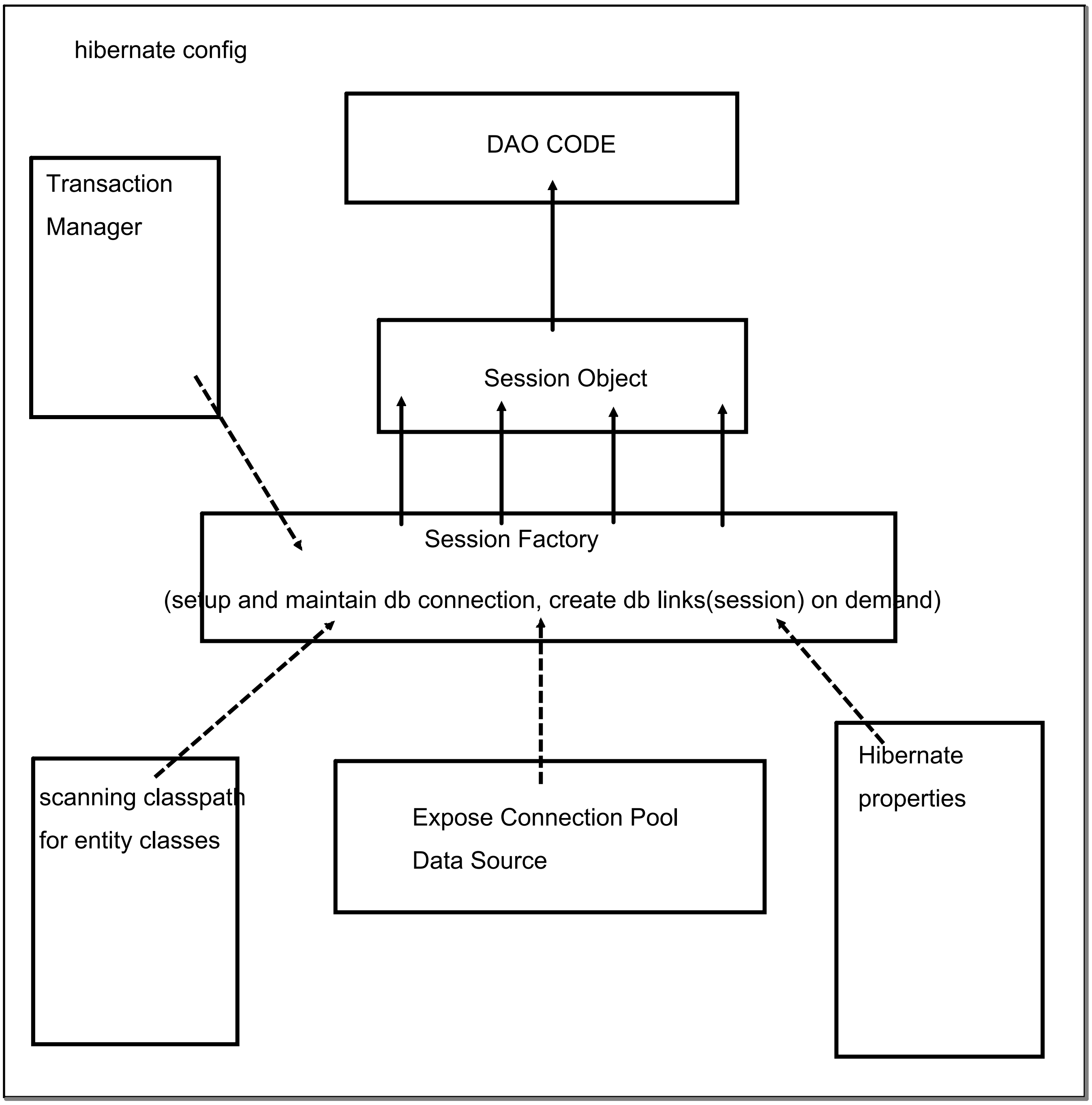
- # Use appropriate JPA ann. to map Java Object with DB schema (entity class)
  (mapping table field with class fields)
- # provide appropriate configure for datasource

Dependency :

1. Hibernate (ORM)
2. jdbc-connection
3. Connection Pool

Hibernate :

- # hibernate project (hibernate-core)
- # do need plumbing API
  - # spring-orm ( helps to connect Spring bean with Hibernate ORM)
  - # spring-tx ( provide sync between application context of spring with Hibernate tx)


hibernate config
DAO CODE
Transaction Manager
Session Object
Session Factory
(setup and maintain db connection, create db links(session) on demand)
scanning classpath for entity classes
Expose Connection Pool Data Source
Hibernate properties

Hibernate uses HQL ------> SQL (dialect) mysql_dialect

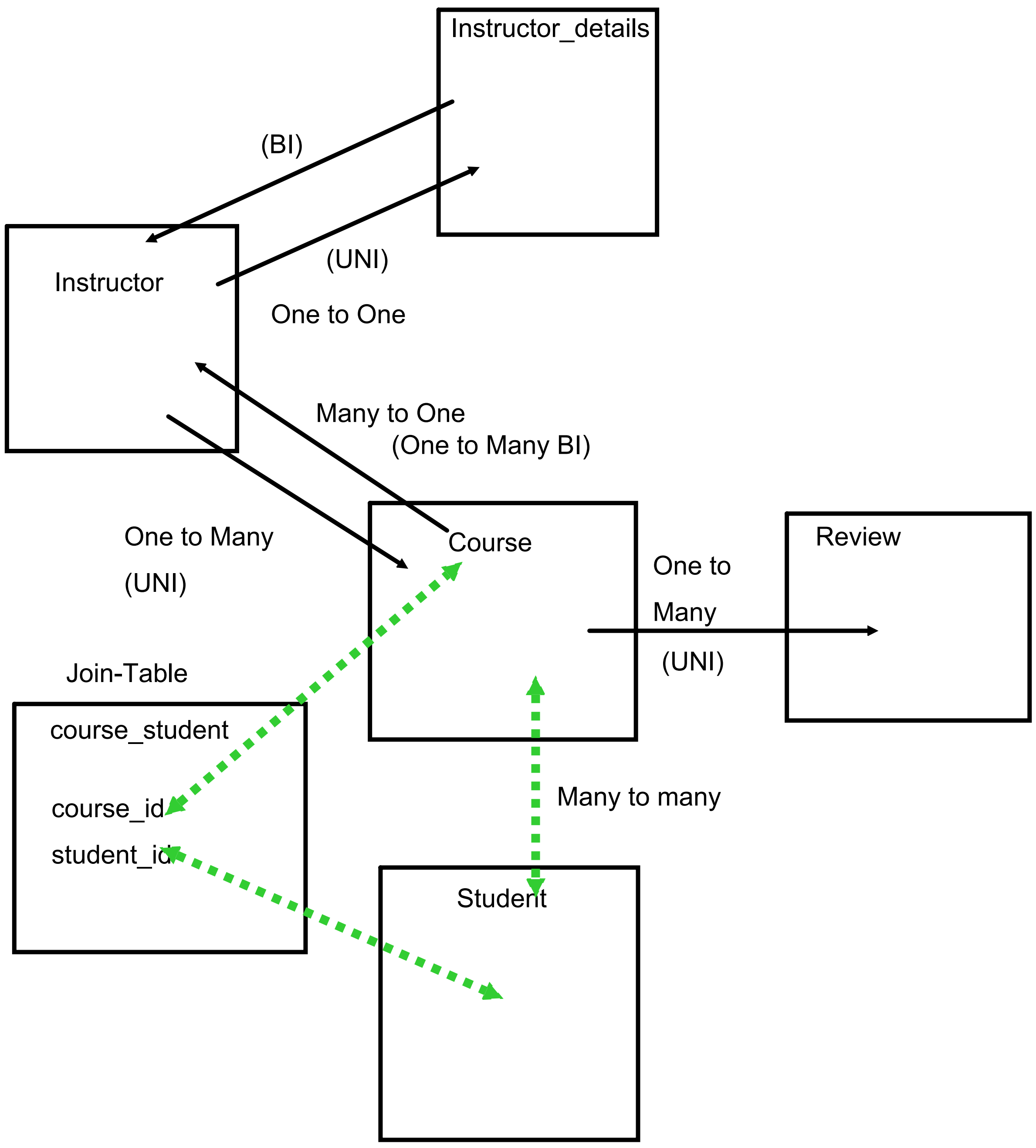
# need to have method that config and exposes the txmanager to be used by springframework

    # to set the sessionFactory instance for transaction

    # all Session object from SessionFactory will have trans support

# need to add annotation to config class to enable the transaction

JPA annotation for Relation mapping in entity classes

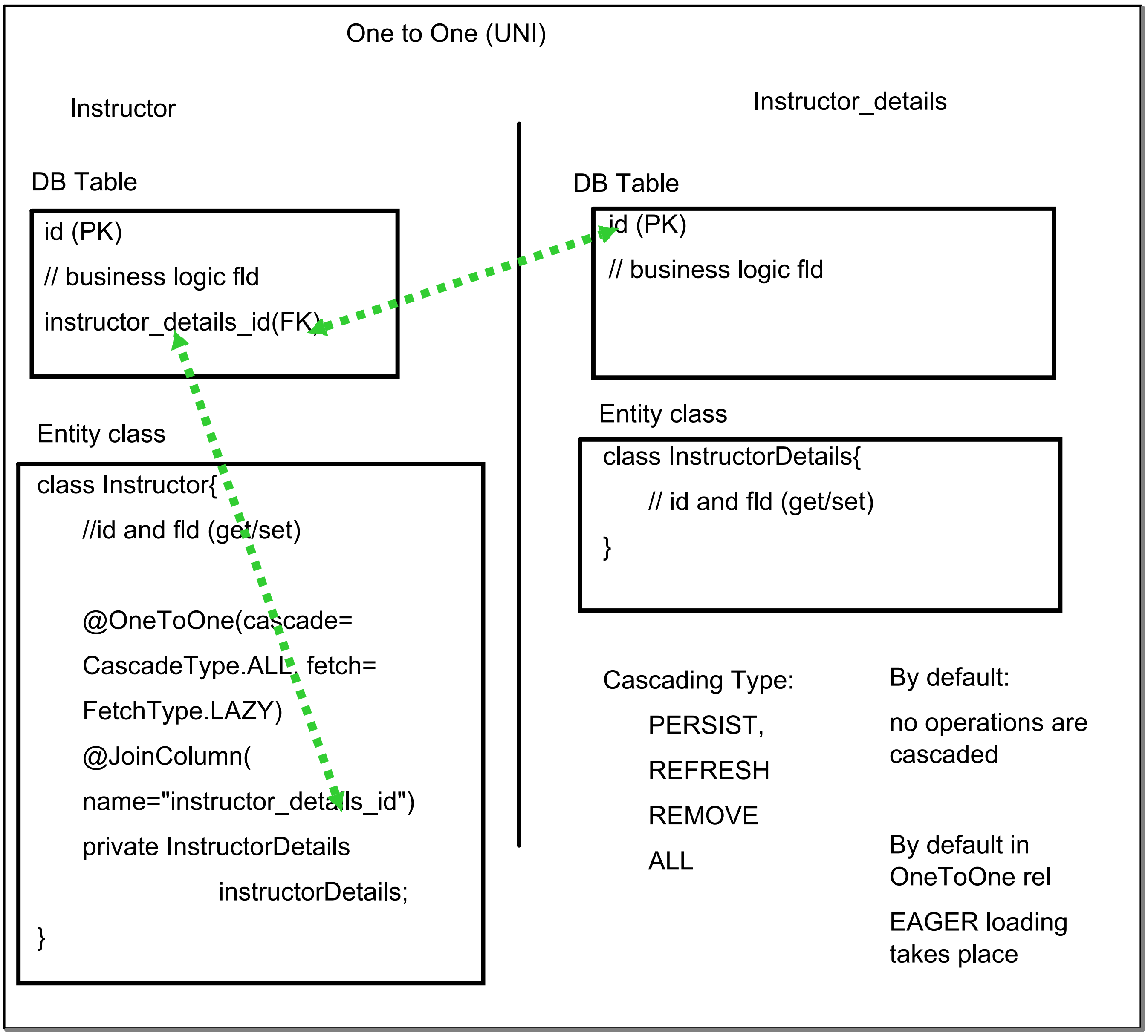


Basic DB relation req

equivalent mapping in entity classes using JPA annotations

Base Concepts

1. DB : primary and foreign keys
2. Implementations:
   - Cascading : (applying same operation on related entities)
   - Fetching : :
     - Eager  : will fetch everything related to that entity
     - Lazy : fetch on demand



One to One (UNI)

Instructor

DB Table

```
id (PK)
// business logic fld
instructor_details_id(FK)
```

Entity class

```
class Instructor{
    //id and fld (get/set)

    @OneToOne(cascade=
    CascadeType.ALL, fetch=
    FetchType.LAZY)
    @JoinColumn(
    name="instructor_details_id")
    private InstructorDetails
                instructorDetails;
}
```

Instructor_details

DB Table

```
id (PK)
// business logic fld
```

Entity class

```
class InstructorDetails{
    // id and fld (get/set)
}
```

Cascading Type:

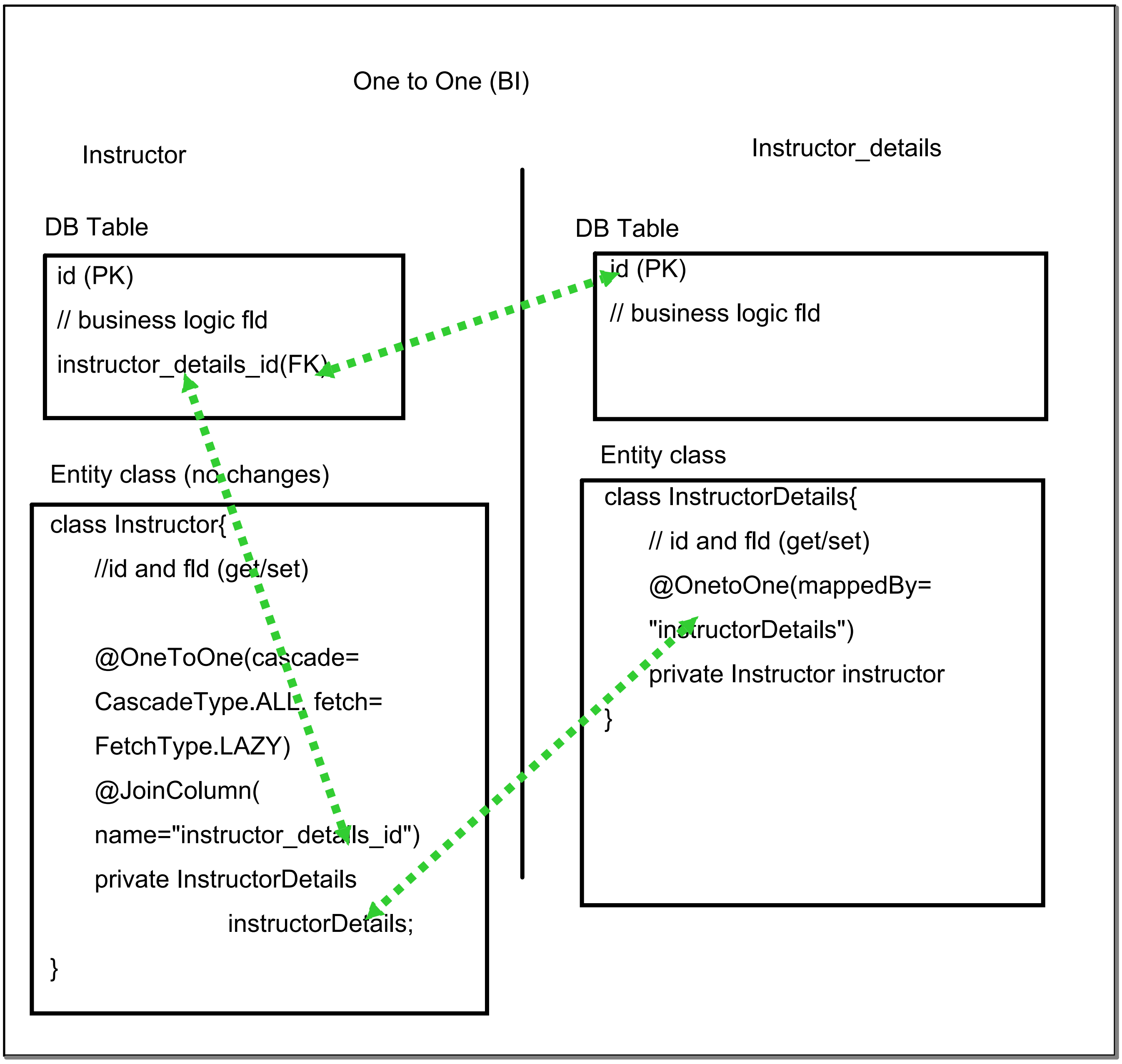
- PERSIST,
- REFRESH
- REMOVE
- ALL

By default:

no operations are cascaded

By default in OneToOne rel

EAGER loading takes place



One to One (BI)

Instructor

DB Table

```
id (PK)
// business logic fld
instructor_details_id(FK)
```

Entity class (no changes)

```
class Instructor{
      //id and fld (get/set)

      @OneToOne(cascade=
      CascadeType.ALL, fetch=
      FetchType.LAZY)
      @JoinColumn(
      name="instructor_details_id")
      private InstructorDetails
                    instructorDetails;
}
```

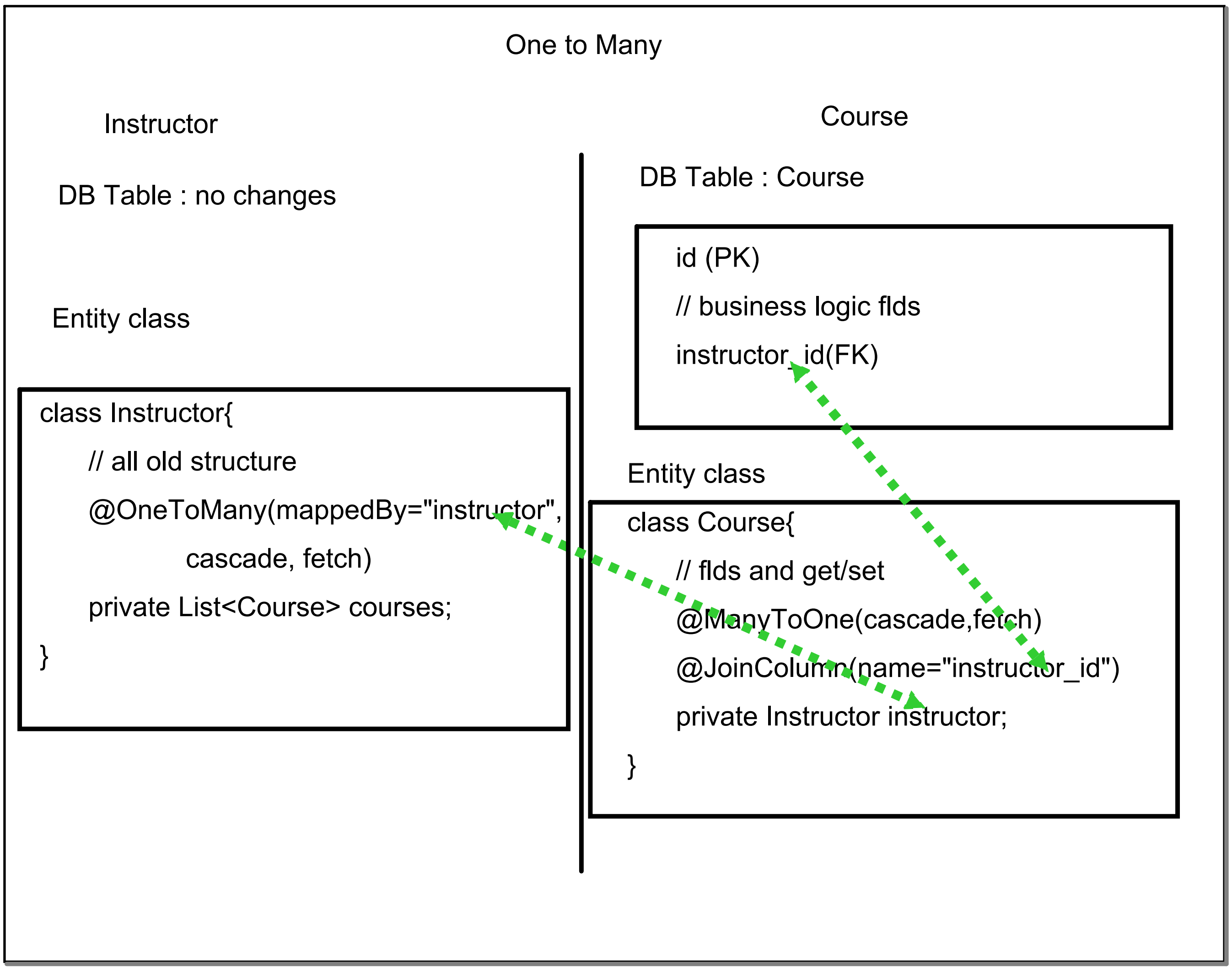
Instructor_details

DB Table

```
id (PK)
// business logic fld
```

Entity class

```
class InstructorDetails{
      // id and fld (get/set)
      @OnetoOne(mappedBy=
      "instructorDetails")
      private Instructor instructor
}
```



## One to Many

Instructor

DB Table : no changes

Entity class

```
class Instructor{
      // all old structure
      @OneToMany(mappedBy="instructor",
                  cascade, fetch)
      private List<Course> courses;
}
```

Course

DB Table : Course

```
id (PK)
// business logic flds
instructor_id(FK)
```

Entity class

```
class Course{
      // flds and get/set
      @ManyToOne(cascade,fetch)
      @JoinColumn(name="instructor_id")
      private Instructor instructor;
}
```

Default Fetching type

- OneToOne : EAGER
- OneToMany : LAZY
- ManyToOne : EAGER
- ManyToMany : LAZY

ManyToMany : third table (join-table) containing fk with other 2 tables
: no entity class for it is created

Course

DB Table : no changes

Entity class

```
class Course{
    // all old composition
    @ManyToMany(cascade,fetch)
    @JoinTable(
    name="course_student",
    joinColumns=
        @JoinColumn(name="course_id"),
    inverseJoinColumns=
        @JoinColumn(name="student_id")
    )
    private List<Student> students;
}
```

@JoinTable tell Hibernate:

1. Look as course_id in join-table
2. for other side (inverse), look at student_id in JT
3. use this info to find relationship

Entity class

```
class Student{
    // all std composition
    @ManyToMany(cascade,fetch)
    @JoinTable(
    name="course_student",
    joinColumns=
        @JoinColumn(name="student_id"),
    inverseJoinColumns=
        @JoinColumn(name="course_id")
    )
    private List<Course> courses;
}
```