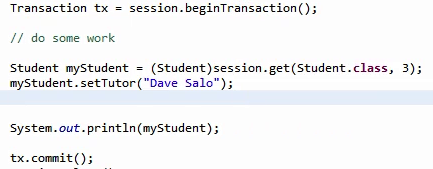
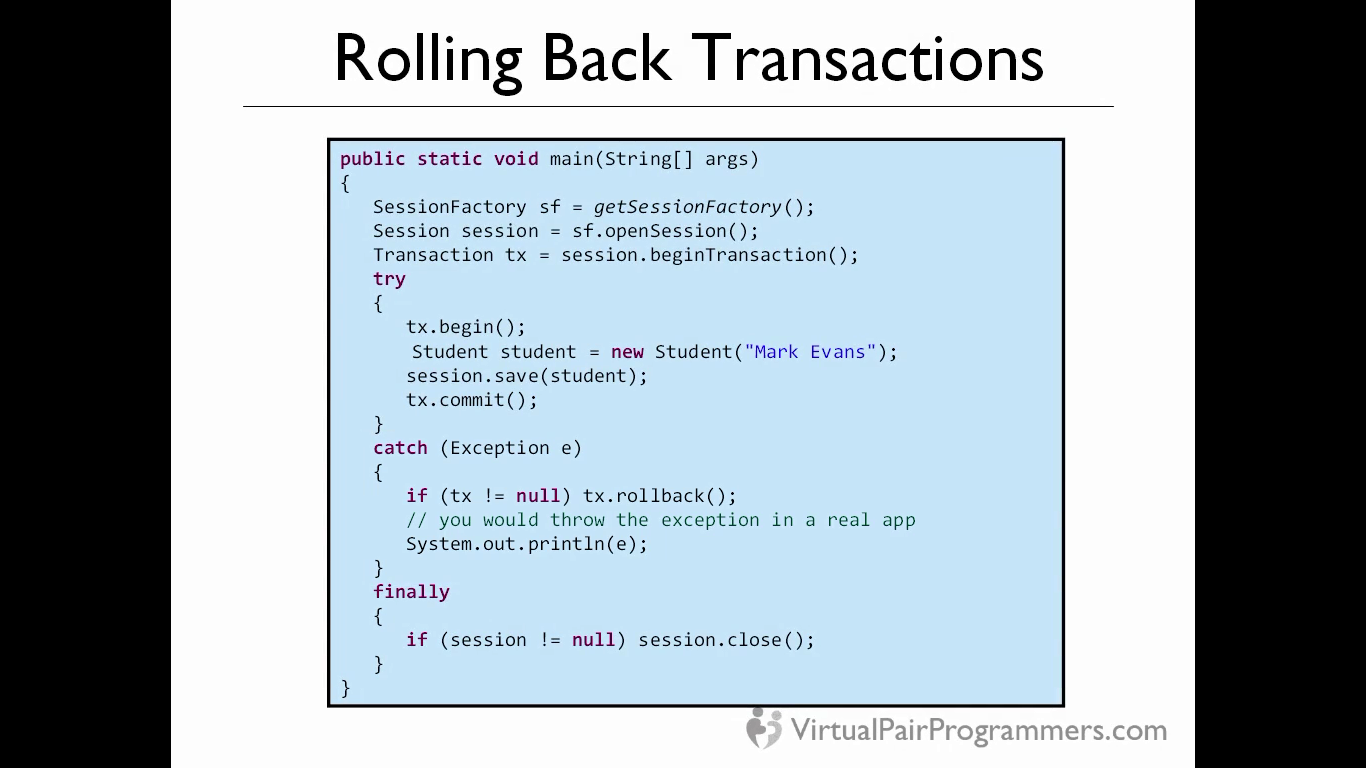
-To delete an object in Hibernate – find an object first and then the object will be persisted. Then use delete function to delete. Alternatively we can use **query** language to delete in 1 statement.



-While updating an entity, we don’t need to specify any operation on session object. Because whenever we are committing a transaction, Hibernate will check for all the objects (**automatic** **dirty check**) that were modified and issue and internal update for those objects.

*This is a big advantage of hibernate/ORM language that it intelligently checks if the object is updated or not. This is straight forward in above example but for a complex logic, we cannot be sure every time that an object is changed or not and in SQL, we may issue an unnecessary update statement*.

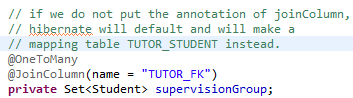
Important – When we declare a field as int(primitive), hibernate will add a not null constraints over the field. We can also use a wrapper Integer to have null values for the field.



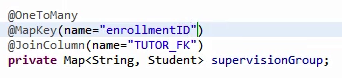
**OneToMany annotations(unidirectional)**

<http://stackoverflow.com/questions/11938253/jpa-joincolumn-vs-mappedby>

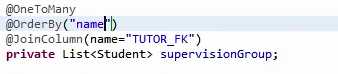
@joinColumn is used in two ways for unidirectional mapping. When used in “Many “side, this is used to change the column name. When used in “One” side, this mapping will force the foreign constraint on the “Many” side of the relationship instead of the link table. The below code is only for unidirectional relationship as the Student entity has no link for Tutor class below.



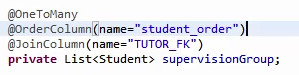
A oneToMany can be mapped by using a Map collection also. Here we will have to define a @MapKey annotation to tell hibernate that while building a map, we will use enrollmentID from Student entity as a key of the map. So when finding a tutor, hibernate will build a map.



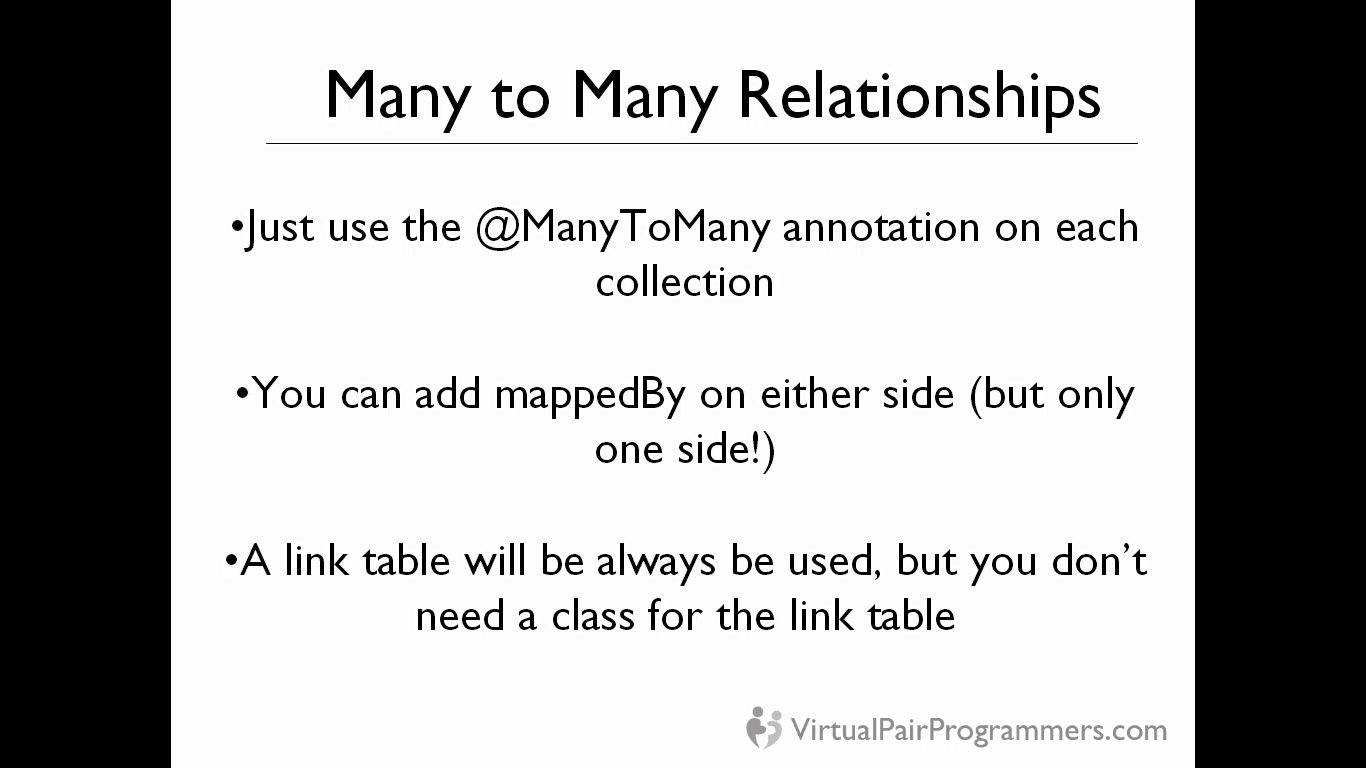
We can also use a List interface which will identify the order of inserts. Additionally we have an @OrderBy annotation to work with.

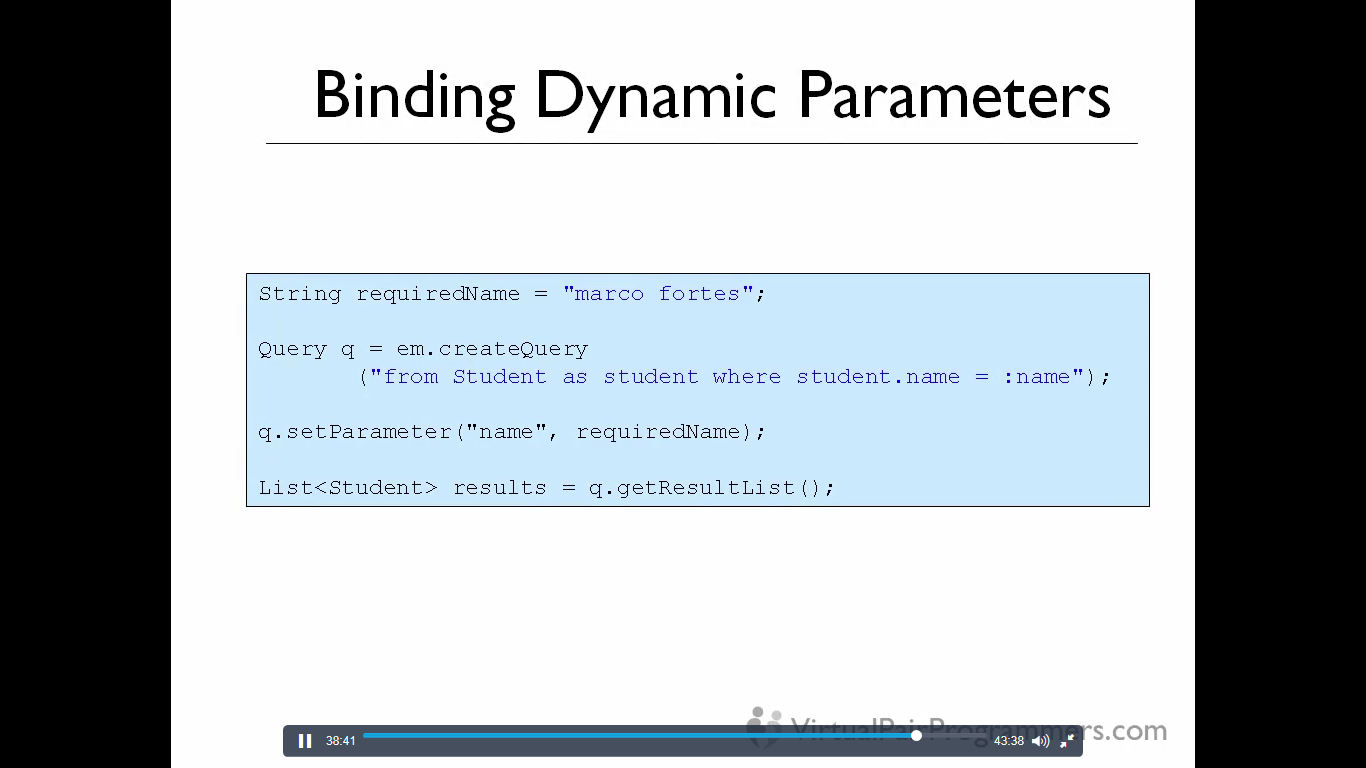


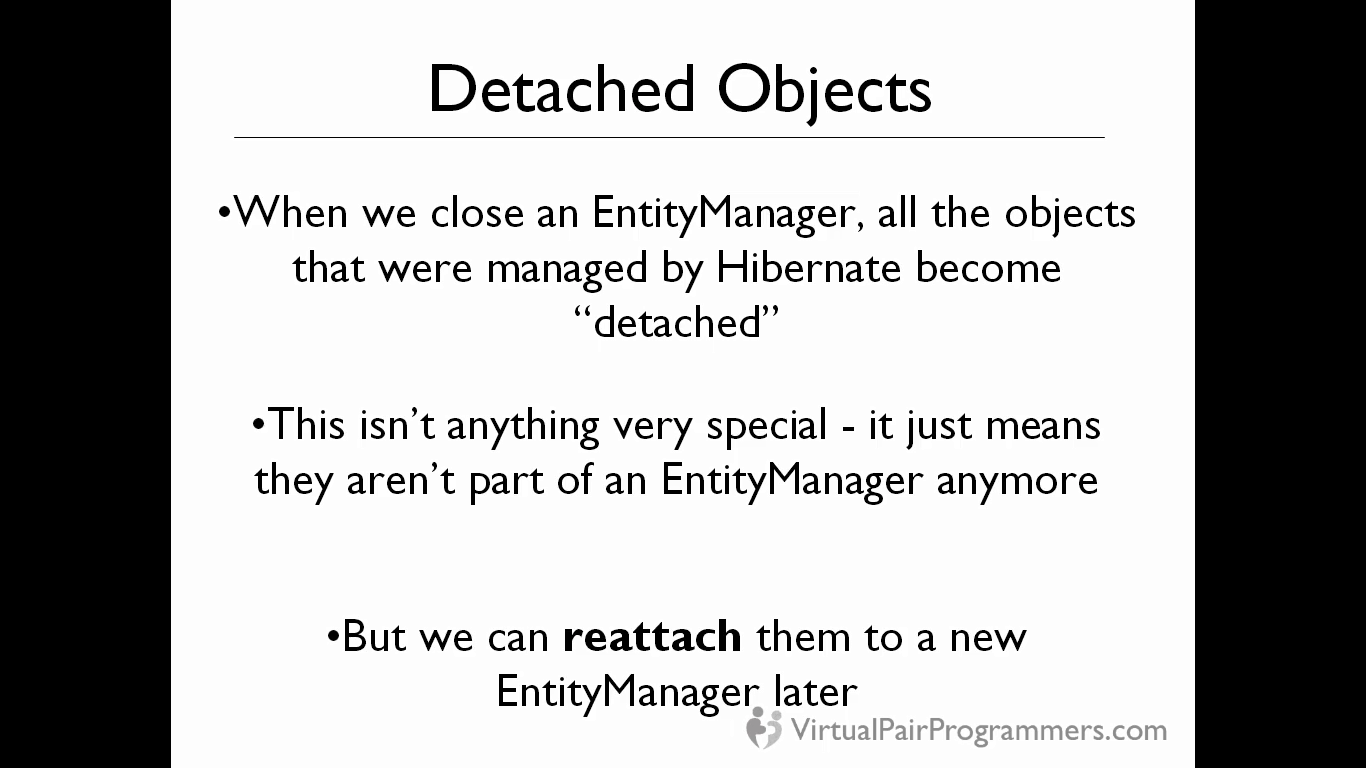
We can also provide an additional column to tract the order by using @orderColumn



**OneToMany annotations (Bidirectional)**







When we have a process which is long waiting process, we should close our entity manager as that is not thread safe.

But once we close the EM, all the objects that were held by the EM, will be released and any updates on those objects will not result in dirty checking and hence will not be reflected in DB.

So we can use a detached object concept, then when we are finished with the work, we can again reattach and detached object, then it will be picked by ORM and all the updates will be executed.

