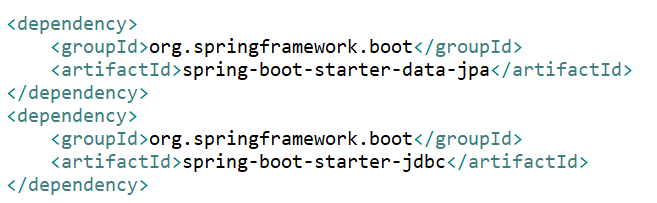
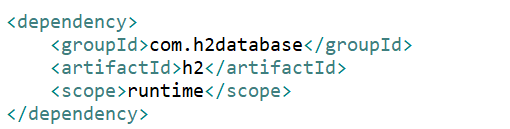
1. Introduction
   1. JPA stands for Java Persistence API
   2. JPA used to do mapping between object and tables.
2. Spring JDBC to JPA
   1. Add dependency in pom xml file



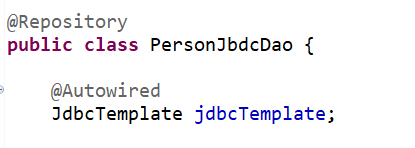
* 1. Add dependency for In-Memory DB



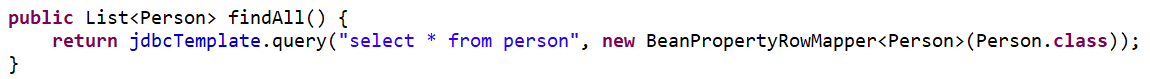
* 1. Enable console for In-Memory DB

spring.h2.console.enabled=true

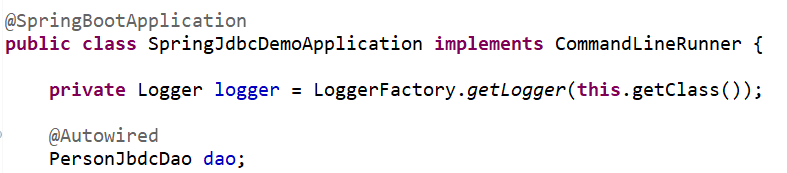
* 1. Run some query after starting database using data.sql (under resource folder)
  2. Implement JDBC in Spring
     1. Create class then @Autowired JdbcTemplate to execute query.



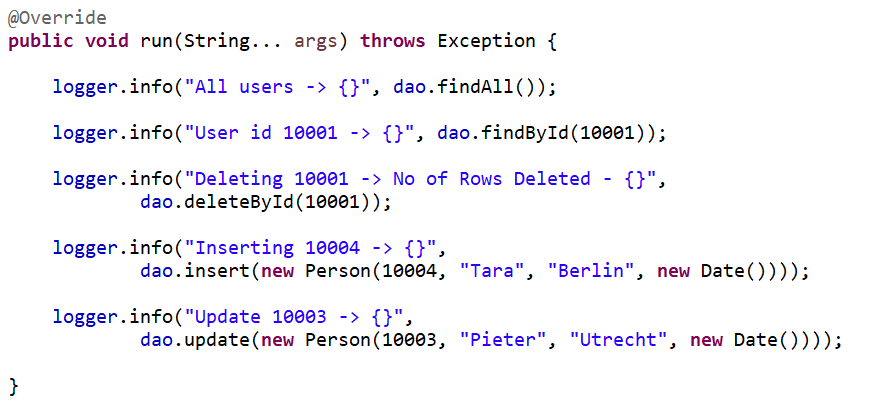
* + 1. Find All method using default row mapper



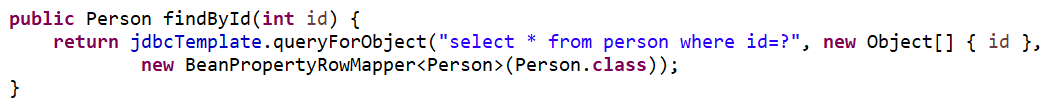
* 1. Executing method using CommandLineRunner
     1. Using CommandLineRunner (interface), we can execute jdbc method at the startup of the application.
     2. Implement CommandLineRunner in Application class and @Autowired JDBC Dao Class



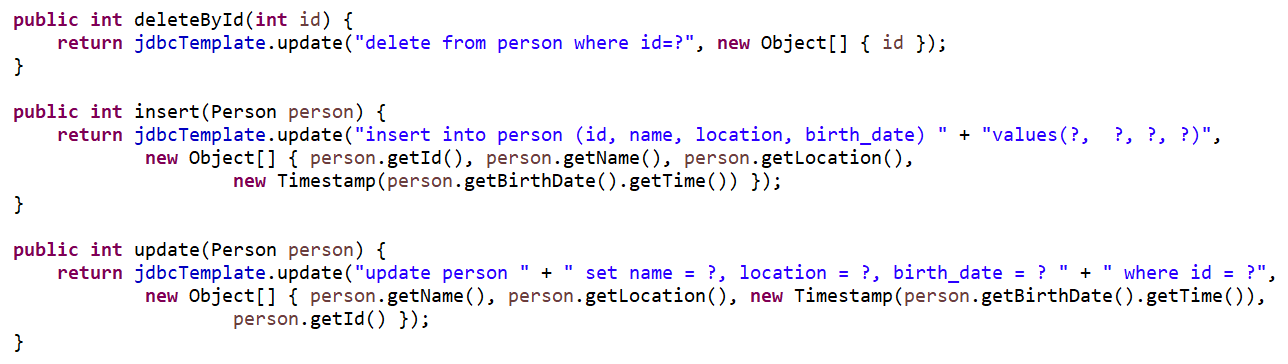
* + 1. Override run() method



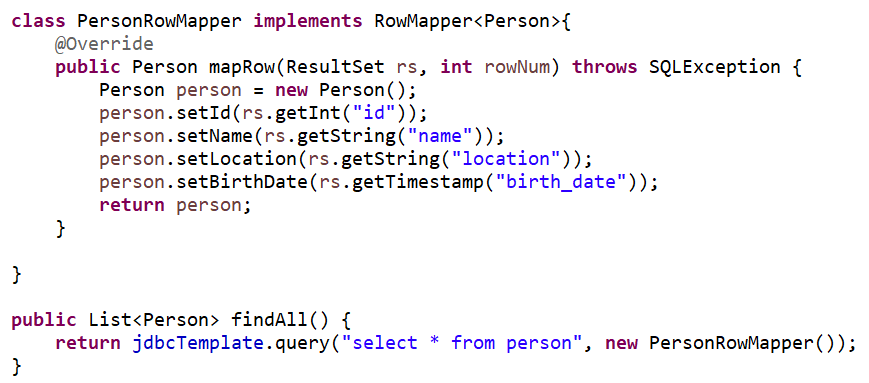
* 1. JDBC Template Functions
     1. The queryForObject() method: used to execute select query



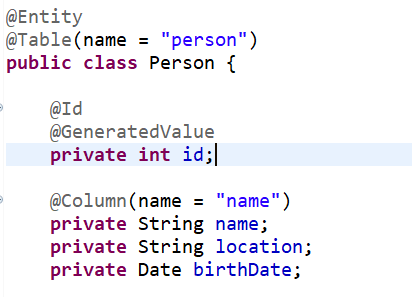
* + 1. The update() method: used to execute insert, update and delete query



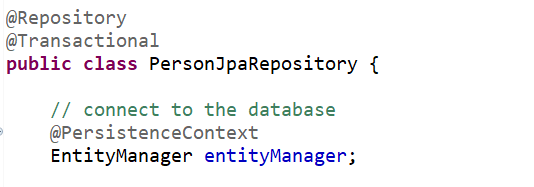
* 1. Creating a custom Spring JDBC RowMapper

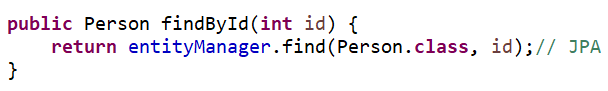


1. Introduction to JPA
   1. Define Entity using @Entity annotation at class level. Use @Table(name = “person”) to map with table name which is optional. Use @Column(name = ”location”) to map with column name which is optional.

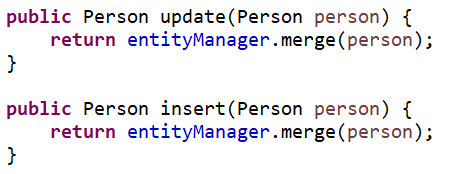


* 1. Implementing JPA Repository with Hibernate
     1. Use @Transactional annotation to connect with databases
     2. Use @PersistenceContext to create EntityManager object and then use EntityManager object to execute query

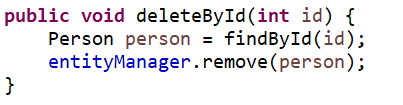




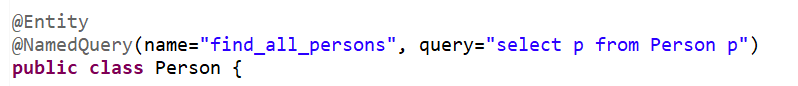
* + 1. Execute Insert and update query



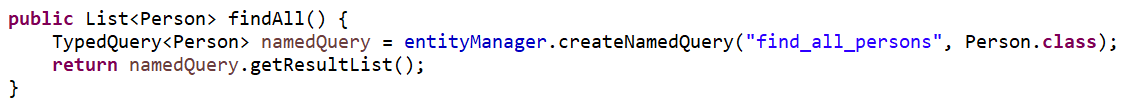
* + 1. Execute delete query



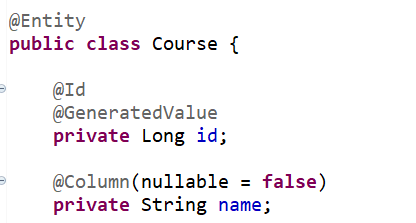
* + 1. Named query:
       1. Use @NamedQuery in Entity class at class level



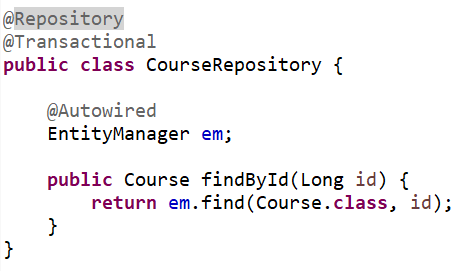
* + - 1. Execute named query using createNamedQuery() method



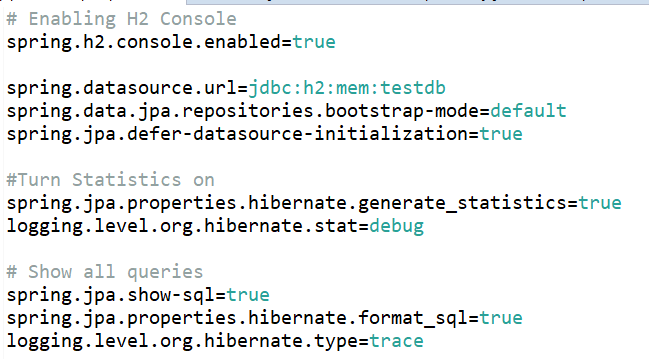
1. JPA and Hibernate
   1. Creating JPA Entity Course



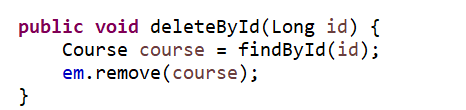
* 1. Create findById using JPA Entity Manager



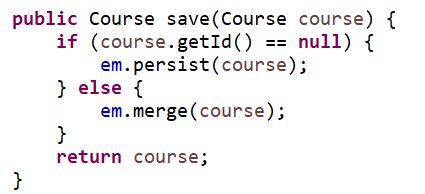
* 1. Add properties in application properties file



* 1. Execute delete query



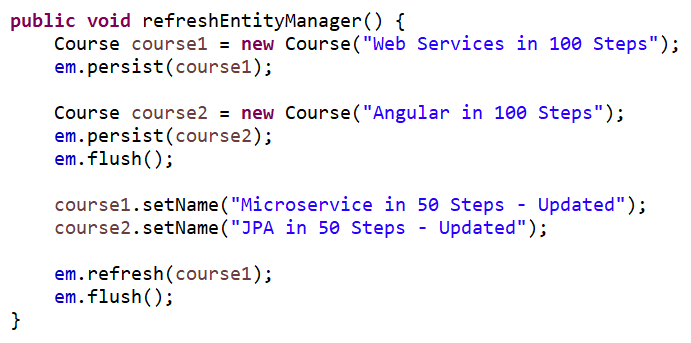
* 1. Execute save and update query
     1. The persist() method: it is used to add object in persistence context
     2. The merge() method: it is used to update existing object values in persistence context



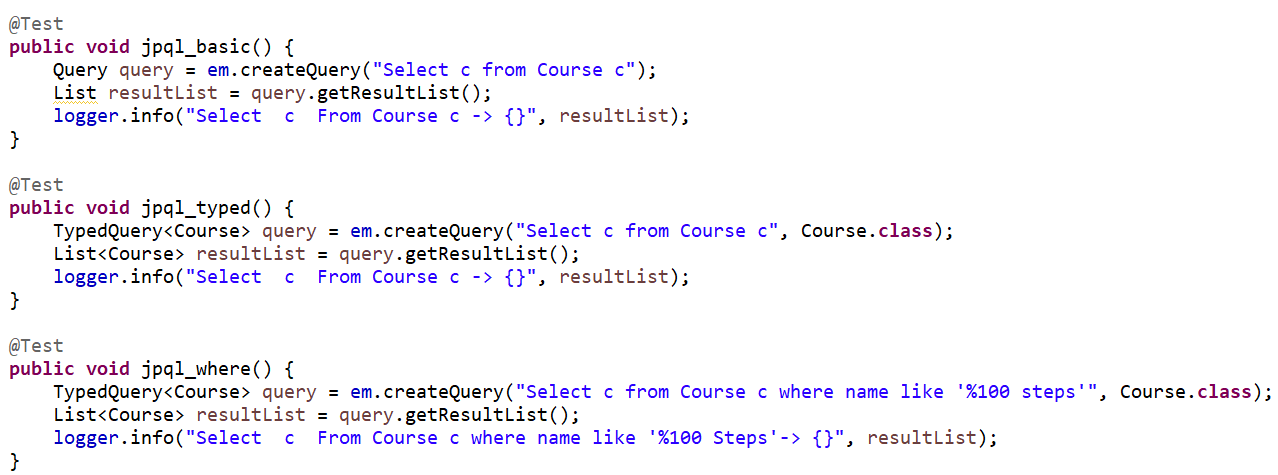
* 1. The flush(), clear() and detach() methods
     1. The clear() method: it detach all objects from persistence context so no changes flush into database.
     2. The detach() method: it detach specific object from persistence context
     3. The flush() method: it is used to persist all the changes until now down to the database in the middle of a transaction as well.



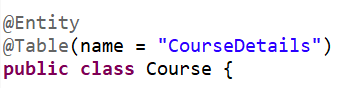
* 1. Entity manager methods – refresh
     1. The refresh() method: it is used to refresh data from database into persistence context.



* 1. JPQL – Basics
     1. JPQL stands for Java Persistent Query Language
     2. SQL queries from tables, whereas JPQL queries from entities.
     3. All queries that we write, those are converted into SQL queries by JPA that is hibernate.
     4. JPQL used to run normal SQL queries.
     5. Use createQuery() to execute JPQL



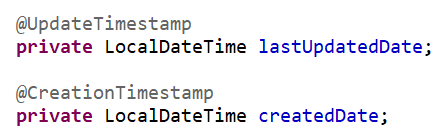
* 1. JPA and Hibernate annotation
     1. @Table annotation



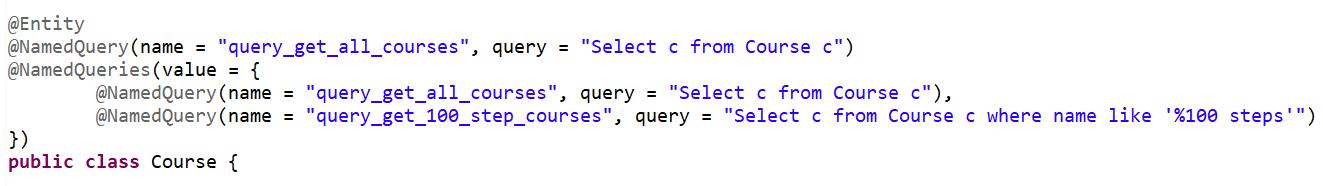
* + 1. @Column annotation

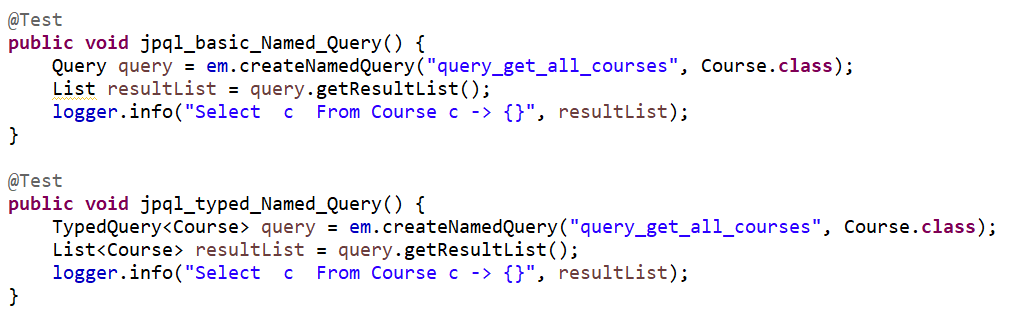


* + 1. @UpdateTimestamp and @CreationTimestamp annotation

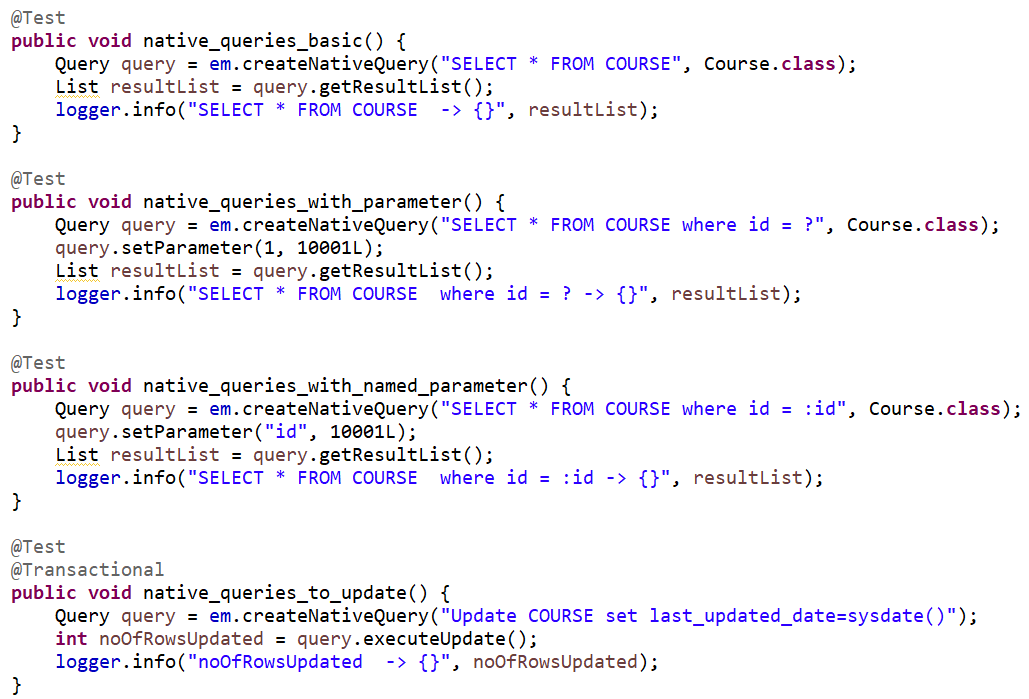


* + 1. @NamedQuery and @NamedQueries annotation
       1. We can hardcode query and then later use wherever we want. So, it can be reuse multiple time.
       2. Using this we can assign name for any query and same use in createNamedQuery().

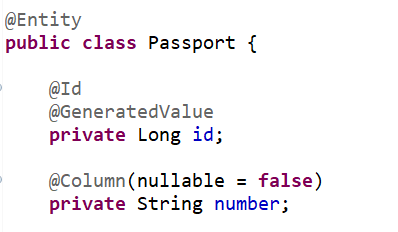
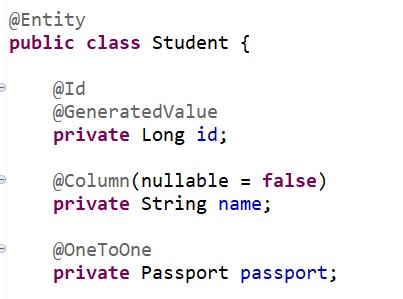




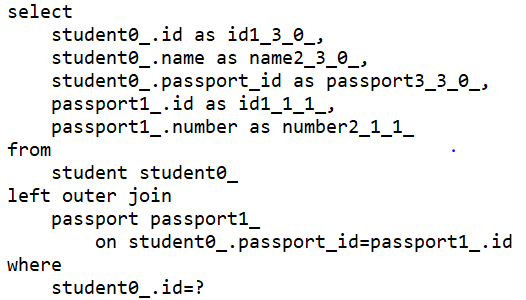
* 1. Native Queries – Basics
     1. It is used to run Native SQL queries.
     2. We can execute native SQL query using createNativeQuery() method
     3. Pass dynamic value using setParamter() method



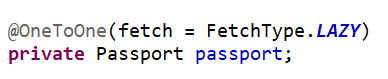
1. Establishing Relationships with JPA and Hibernate – OneToOne
   1. Use @OneToOne annotation to create one to one relationship between two table like using primary key as password\_id in student table.

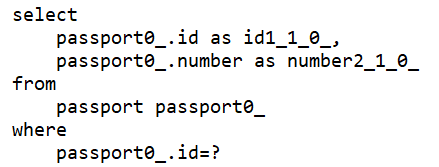
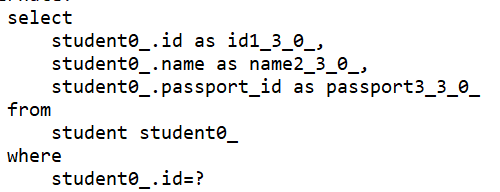


* 1. When we fetch student using id then same time it fetches passport as well, which is called **Early Fetching**. It run below query because of @OneToOne mapping.

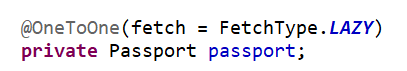


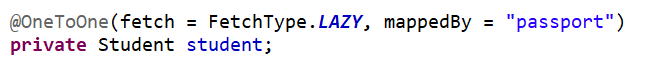
* 1. Lazy Fetching – In this case passport only fetched when we use them.





* 1. Transaction, Entity Manager and Persistence Context
     1. Transaction – Every operation should succeed, or nothing should succeed (roll back). Use @Transaction annotation at class level or can be used at function level.
     2. Persistence Context - The persistent context is the place where all the entities which you are operating upon are being stored. Persistent context is created at the start of the transaction and kill as soon as the transaction is ended. In hibernate session == persistence Context.
     3. Entity Manager - The way we interact with the persistence context is by using an entity manager. In hibernate session factory == entity Manager
  2. OneToOne Mapping – Bidirectional Relationship





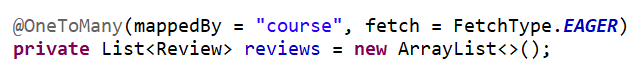
* 1. By default, @OneToOne mapping is Early Fetching.

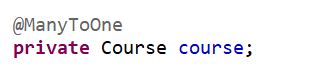
1. FAQ about Hibernate and JPA
   1. @Transaction annotation can be used when we do multiple operation in single function.

In case of repository, we can directly add @Transaction annotation at class level.

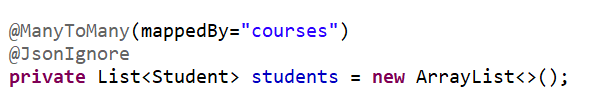
* 1. Read only function don’t need to @Transaction annotation as read only function transaction just after executing select query. But in case of lazy fetching, we should use @Transaction with read only functions as well

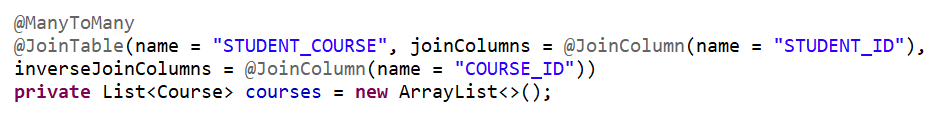
1. Establishing Relationships with JPA and Hibernate
   1. @ManyToOne or @OneToMany annotation





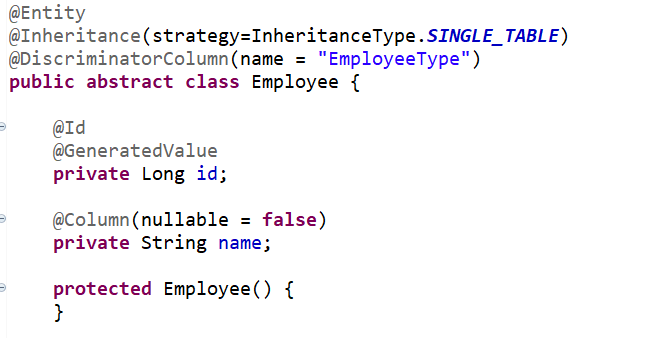
* 1. By default, @OneToMany Mapping is Lazy Fetching and @ManyToOne mapping is Early Fetching.
  2. @ManyToMany annotation

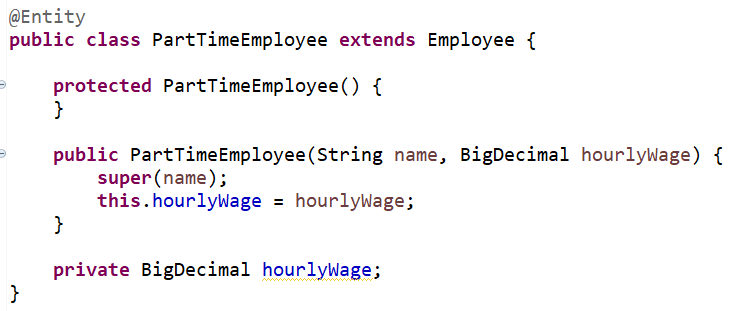


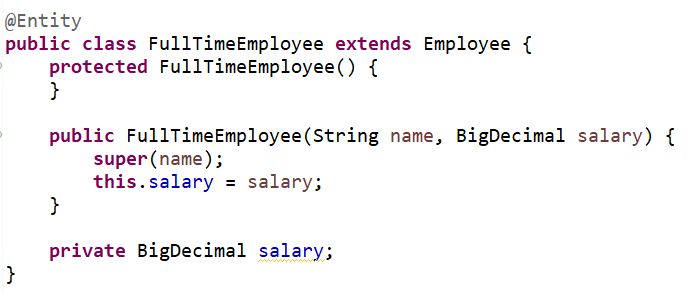


* 1. By default, @ManyToMany mapping is Lazy Fetching

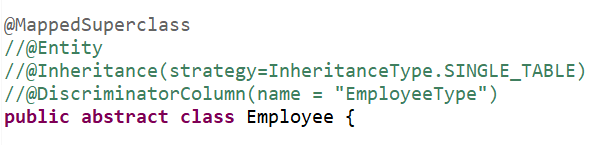
1. Inheritance Hierarchies with JPA and Hibernate
   1. By default, it merges both child class into single table (SINGLE\_TABLE inheritance type).
   2. Inheritance type: SINGLE\_TABLE, TABLE\_PER\_CLASS, JOINED







* 1. @MappedSuperClass
     1. It can be mapped in the same way as an entity, except that mappings will only apply to it subclasses.



1. Queries with Entities using JPQL
2. Queries with Java API – Criteria Queries
3. Spring Data JPA
4. Caching