**NGUYEN HOAN VU LE (GURU IN THE FUTURE)**

What is JPA?

* JPA stands for Java Persistence API
* JPA is the official API for working with relational data in Java
* JPA is only a specification(JPA is not a concrete implementation)
* JPA is a bridge from Java’s object world to how data stored in relational databases
* ORM- Object Relational Mapping
* JPA offers Java developers database independence
* One API will support many relational database
* Hibernate is implementation of JPA

@Entity annotation to make a POJO class are an entity

CrudRepository<Author, Long>

ApplicationListener<ContextRefreshedEvent>

@Component: make a spring bean now it wired to spring context

An Interface extends CrudRepository: it mean this interface wired to spring context

An interface must extends not implement

**ManyToMany relationship:**

@ManyToMany  
@JoinTable(name = "author\_book", joinColumns = @JoinColumn(name = "book\_id"),  
 inverseJoinColumns = @JoinColumn(name="author\_id"))  
private Set<Author> authors = new HashSet<>();

@ManyToMany(mappedBy = "authors")  
private Set<Book> books = new HashSet<>();

The author is owner

**OneToOne relationship:**

@OneToOne  
private Publisher publisher;

**Spring MVC**

* Spring Controller:
  + Annotate Controller Class with @Controller, this register the class as a Spring bean and as a Controller in Spring MVC
  + To Map methods to http request paths use @RequestMapping
* Thymeleaf:
  + Thymeleaf is a Java template engine
  + Release in July 2011
  + Rapidly gaining popularity in the Spring community
  + Thymeleaf is a natural template engine(natural meaning you can view templates in your browser)
  + You work with JSP you know you need to compile and deploy and restart tomcat to see your changes which is no fun at all
* xmlns:th="http://www.thymeleaf.org"
  + This above is namespace for thymeleaf tag. It is very important.
* <tr th:each="book: ${books}">
  + Loop in thymeleaf

**SOLID PRINCIPALS**

* Why we use it?
  1. OOP is a powerful concept
  2. But, OOP does not always lead to quality software
  3. The 5 principles focus on dependency management
  4. Poor dependency management leads to code that is brittle, fragile, and hard to change
  5. Proper dependency management leads to quality code that is easy to maintain.

SOLID include:

1. Single Responsibility Principle:

* Every Class should have a single responsibility.
* There should never be more than one reason for a class change
* Your classes should be small. No more than a screen full of code
* Avoid ‘god’ classes
* Split big class into small classes

1. Open closed priciple:

* Your classes should be open for extension.
* But closed for modification
* You should be able to extend a classes behavior, without modifying it.
* Use private variables with getters and setters – ONLY when you need them.
* Use abstract base classes

1. Liskov substitution principle: Square is rectangle but rectangle isn’t square.
2. Interface Segregation Principle:

* Many client specific interfaces are better than one “general purpose” interface
* Notice relationship to the Single Responsibility Principle

1. Dependency Inversion Principle:

**DEPENDENCY INJECTION**

* Dependency Injection is where a needed dependency is injected by another object.
* The class being injected has no responsibility in instantiating the object being injected
* Some say you avoid declaring objects using ‘new’.

IOC: is a technique to allow the dependencies to be injected at runtime

Concrete Class vs Interface:

Generally DI with concrete classes should be avoid, DI via interfaces is highly preferred

**Different Between DI and Ioc:**

Ioc and Dependency easy confused

DI refer much to composition of your classes. How you design your code

Ioc is technique allow…at run time. It is runtime environment of your code this is your code running when start up

**The Java Persistence API, or JPA is the standard way of mapping Java objects to relational database tables (aka ORM).**

**Why Constructor Dependency Injection is preferred than Setter Dependency Injection?**

Because Setter Dependency you can miss out the code to set the dependencies into but it not have any compile error. But at the runtime your code will throws exception.

**If a service Interface have more than one implementation, when spring application start , it will throw exception. To solve this problem we must use @Qualifier.**

@Autowired

@Qualifier(“getterGreetingService”)

@Autowired

@Qualifier(“constructorGreetingService”)

If not use Qualifier you must use name of every implementation of service is lower case of Service Class(Short cut way)

Intent(muc tieu). **But remember somebody read your code latter so it’s better to use qualifier**

**@Primary:**

This is a technique you can use when you have multiple bean of the same type and you want one of them to go in by default

**@Profile**:

@Profile(“es”) and need to active it in application.properties. Spring profile have high application in switch between many databases to use

Application.properties by using

spring.profiles.active=… No profile active

but if the value null it will throw an exception because It don’t know which profile choosen.

So We need a default profile by using this code

@Profile({"en", "default"})

**Spring Bean Life Cycle(Important)**