**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

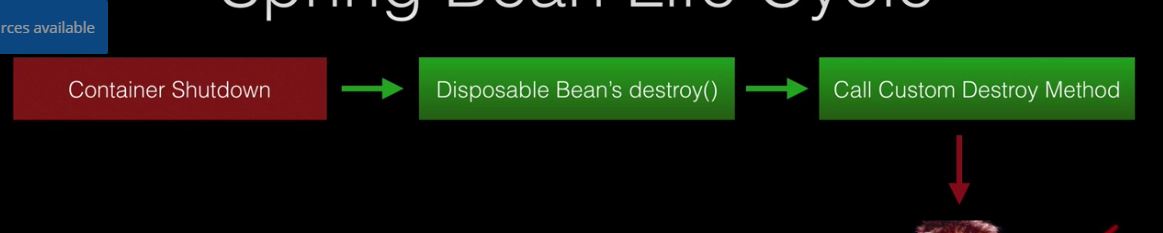
but if the value null it will throw an exception because It don’t know which profile choosen.

So We need a default profile by using this code

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

**Spring Bean Life Cycle(Important)**



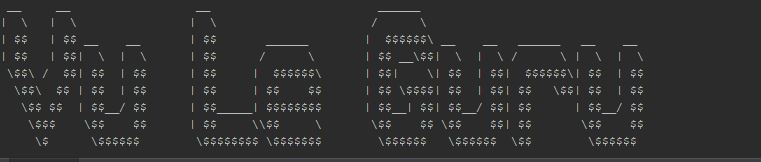


**Custom Banner Spring Boot**

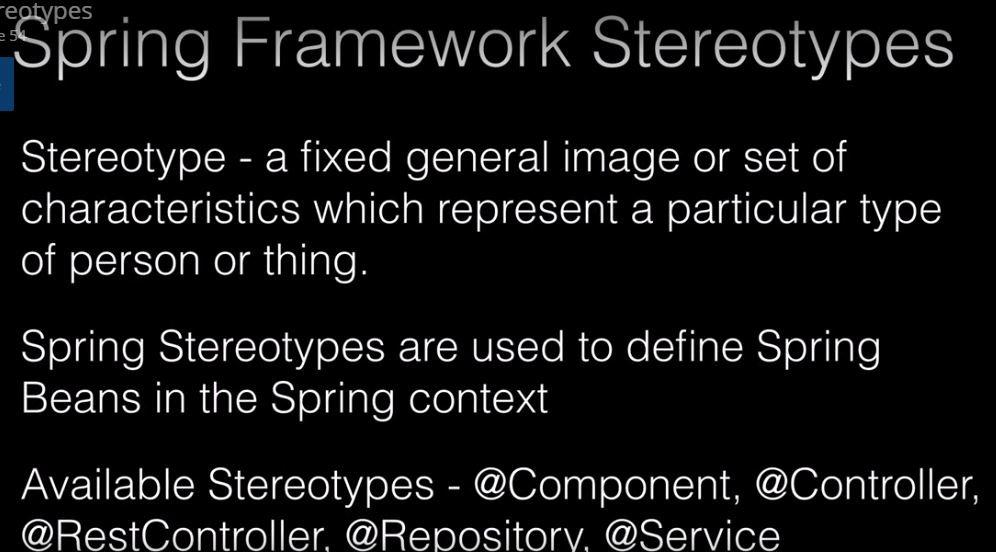
Go this this website and choose texts and fronts you want:

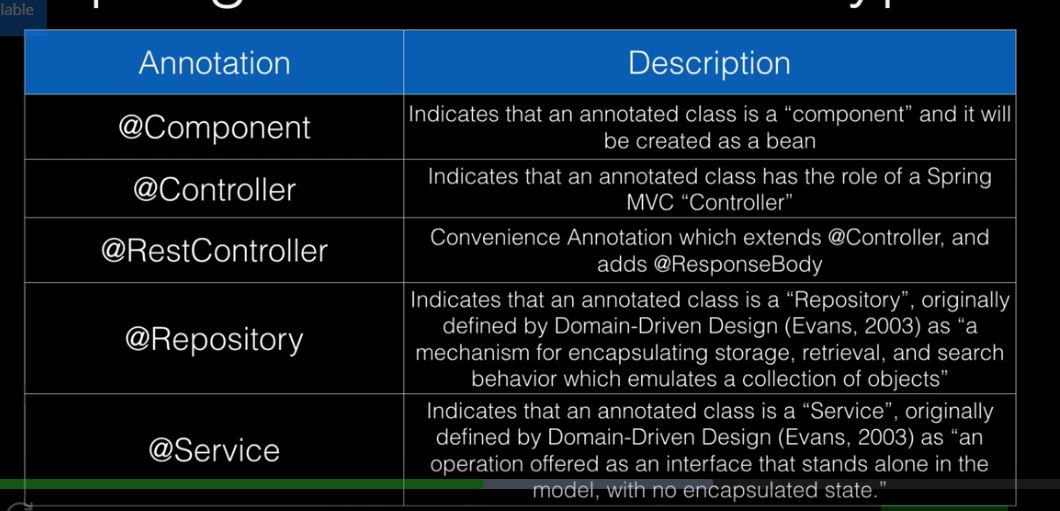
<http://patorjk.com/software/taag/>

Go to resource folder of your IDE and create new file: banner.txt and copy text above



**Spring Framework Stereotypes**

****

****

**Component Scan**

If we declare something out of spring boot structure. It would be error, because spring context don’t see the bean

* + - * To fix this one

@ComponentScan(basePackages = {"guru"})

Use component Scan at these package where you want to scan components

**Java Configuration Example**

@Configuration  
public class ChuckConfiguration {  
  
 @Bean  
 public ChuckNorrisQuotes chuckNorrisQuotes() {  
 return new ChuckNorrisQuotes();  
 }  
}

This is a way to config spring framework by java base class: declare @Configuration annotation and if we want to declare a bean for spring context use @bean annotation.

**Spring Boot Configuration**

@SpringBootApplication contain: @Configuration, @EnableAutoConfiguration, @ComponentScan

@EnableAutoConfiguration will bring a lot of configuration classes in supplied Spring boot Jars. But we can specify classes to exclude with:

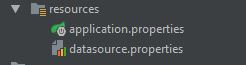
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})

**External Properties:**

Overview:

Externalizing properties are something very important especially when you’re in a big enterprise application because you’re going to be doing externalizing things like database , connect URL, database, users, password, things like that. So you can move externalizing to many environment: from development, to QA, to production and have these different environment.

**Property source:**



@Configuration  
@PropertySource("classpath:datasource.properties")  
public class DataUserConfiguration {  
  
 @Value("${guru.username}")  
 private String username;  
  
 @Value("${guru.password}")  
 private String password;  
  
 @Value("${guru.url}")  
 private String url;  
  
 @Bean  
 public FakeDataUser fakeDataUser() {  
 return new FakeDataUser(this.username,this.password,this.url);  
 }  
  
 @Bean  
 public static PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {  
 return new PropertySourcesPlaceholderConfigurer();  
 }  
  
}

DataSource is externalizing file, this file can changed at every development environment: production, development and QA. To work with this file we have some configuration:

@Configuration for the file work with it

@PropertySource(“classpath:…”) This is annotation use to read properties file from resources forder

PropertySourcePlaceholderConfigurer bean : this is important bean to map value from externalizing source to properties

@Value: to specify the data is mapped to property

**Spring Environment Property**

@Autowired  
Environment env;

env.getProperty("guru.username")

we can use environment to get property, because we have system property



env.getProperty("JAVA\_HOME")

**Multiple properties file**

Spring framework supports multiple property files. Although we have multiple property file but properties in property file must unique.

@PropertySource({"classpath:datasource.properties", "classpath:jms.properties"})

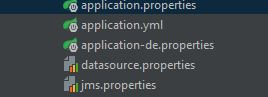
**Spring boot application.properties**

This file is available by default

**Spring boot YAML**

Spacing is very very important in yaml. Application.properties and Application.yml equal hierarchy.

**Spring boot profile properties**



spring.profiles.active=de  
  
guru.username=Vu  
guru.password=vudeptrai123  
guru.url=xvideos.com

In application.properties file we use spring.profile.active=de to change environment via configuration files. we can override properties in application.property and change the value of it in application-de.property. spring.profile.active help us to change the environment but we must create new files with syntax application-[profile].properties

guru.username=Vu\*\*IN\*\*GEMANY  
guru.password=vudeptrai123\*\*IN\*\*GEMANY  
guru.url=xvideos.com\*\*IN\*GEMANY

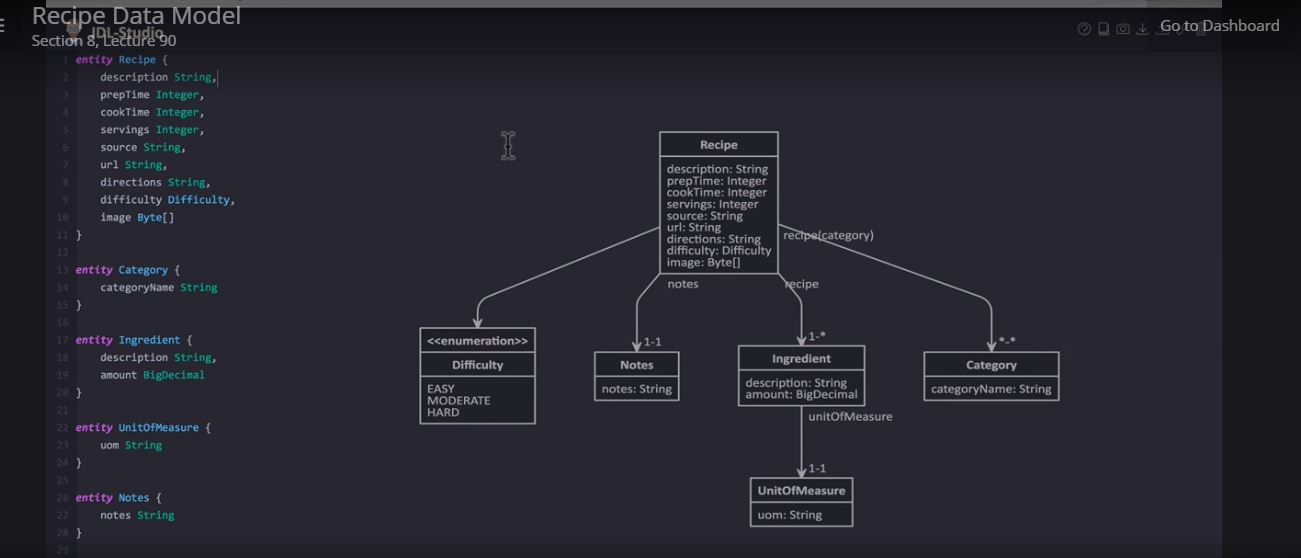
But with yml file we don’t need to create many file. Because we can separate into small files in one yml.

By use “---“

**guru:  
 jms:  
 username:** JMS Username  
 **password:** somepass  
 **url:** SomeURL  
  
--- //here  
**spring.profiles:** de  
  
**guru:  
 jms:  
 username:** JMS Username\*\*\*IN\*\*\*GEMANY  
 **password:** somepass\*\*\*IN\*\*\*GEMANY  
 **url:** SomeURL\*\*\*IN\*\*\*GEMANY

**JPA Data Modeling with Spring and Hibernate**

**Create Recipe Data Model**



**Forking application in GitHUb**

Forking the source code during this course, because it will get a copy of the source code over your repository and then that builds up some history in your repository and you can share with a potential employer. It give you a chance to show your source code that you did in the project

**One to One**

Lob is used for large objects, in relational database we can use clobs for character large objects or blobs for binary large objects

**One To Many**

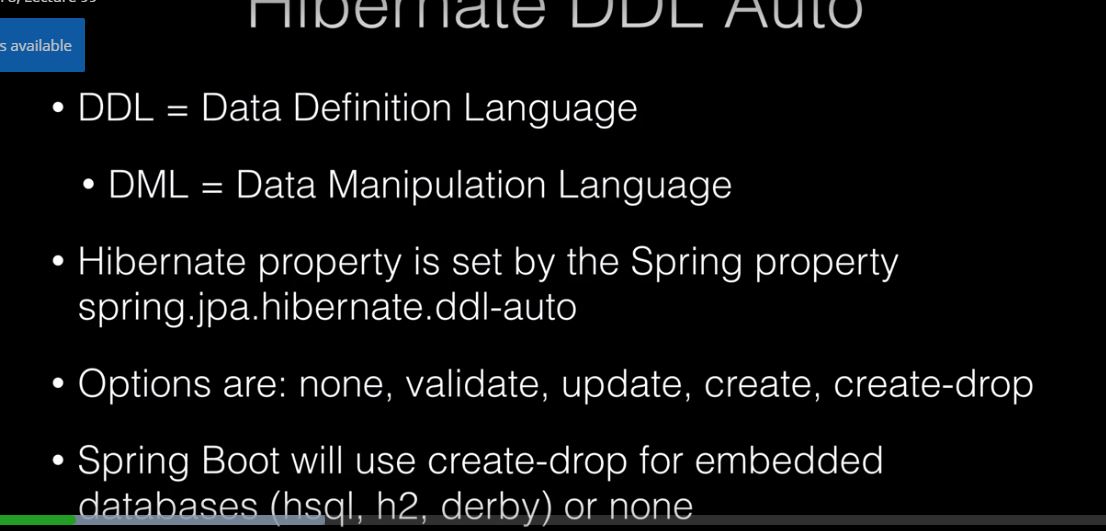
@OneToMany(cascade = CascadeType.*ALL*, mappedBy = "recipe")  
private Set<Ingredient> ingredients;

**Many To Many**

@ManyToMany  
@JoinTable(name = "recipe\_category", joinColumns = @JoinColumn(name= "recipe\_id"),  
 inverseJoinColumns = @JoinColumn(name = "category\_id"))  
private Set<Category> categories;

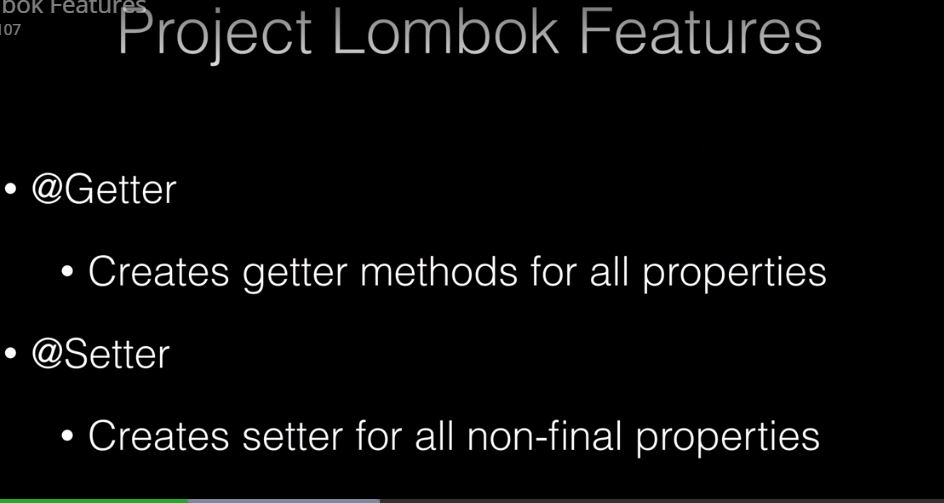
@ManyToMany(mappedBy = "categories")  
private Set<Recipe> recipes;

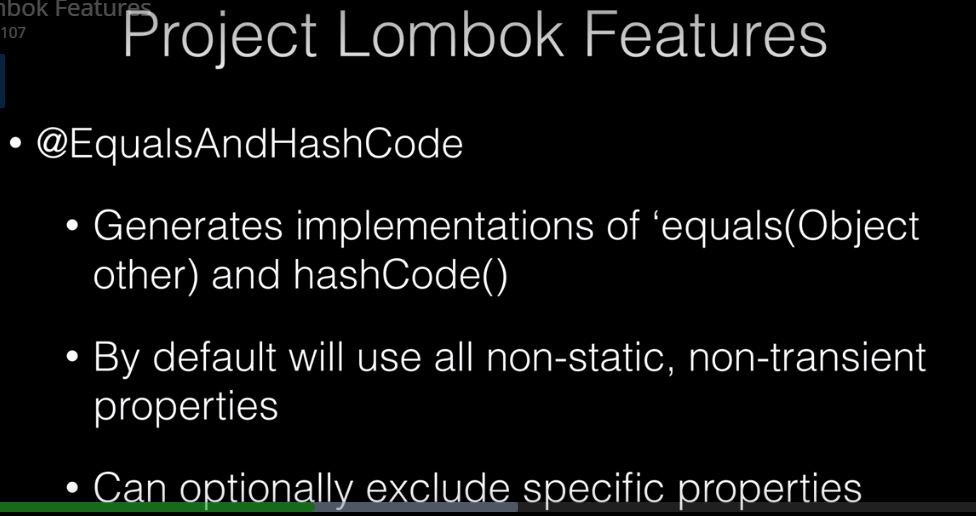
**Hibernate DDL Auto**

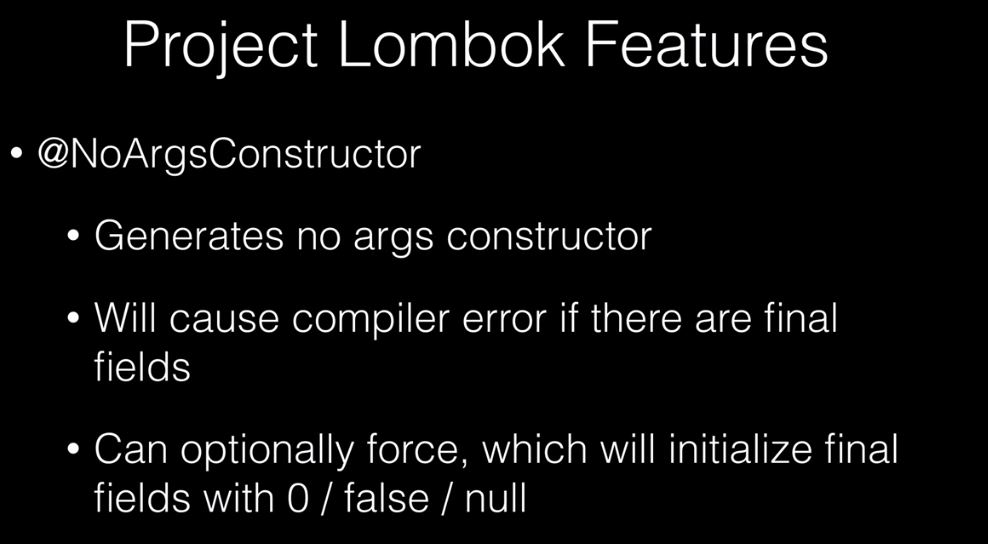
****

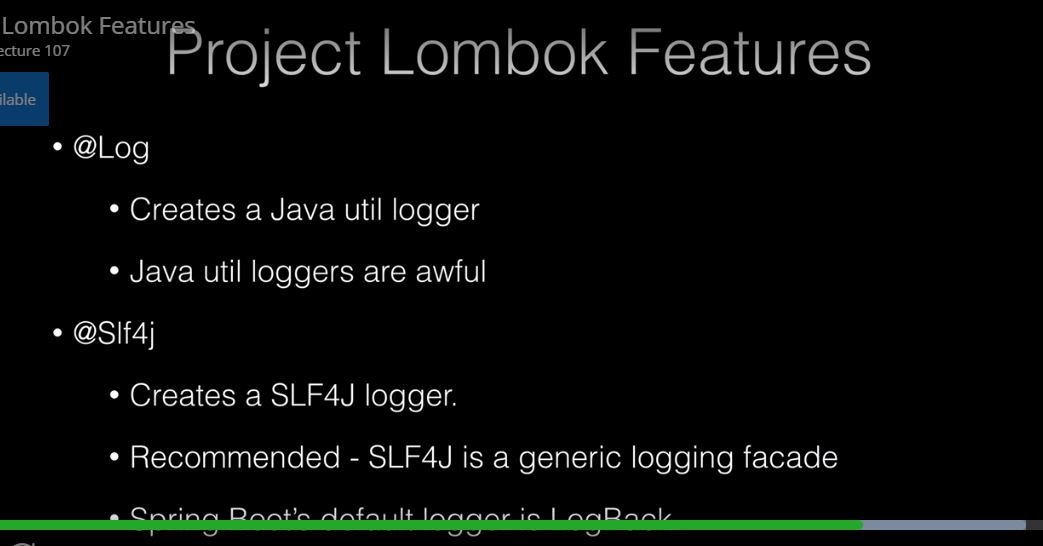
Spring boot is default create-drop with h2 memory database

**Project Lombok**

****

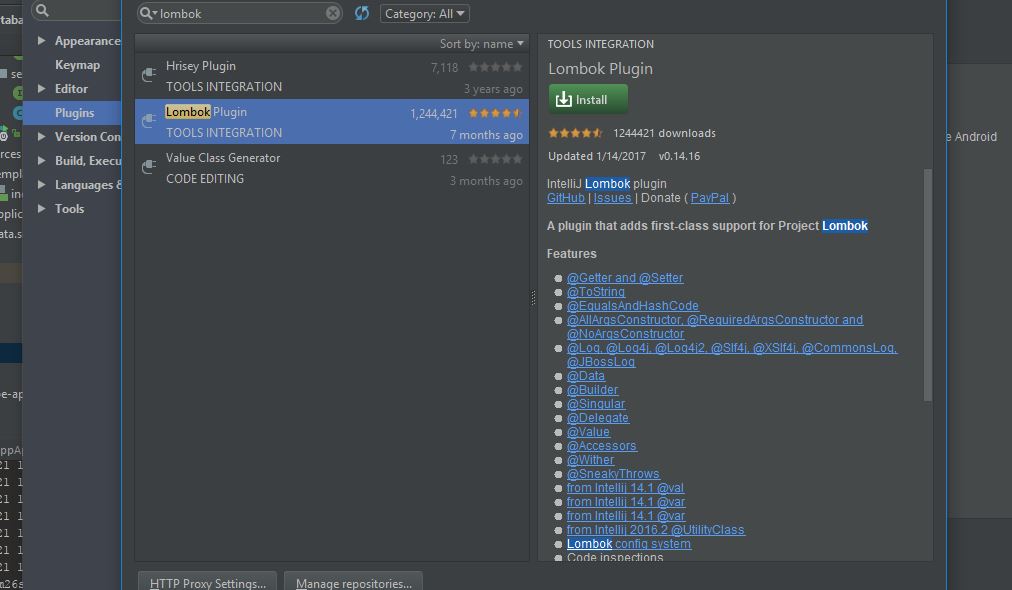
****

****

****

**Adding Project Lombok and IDE Configuration**

<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
</dependency>



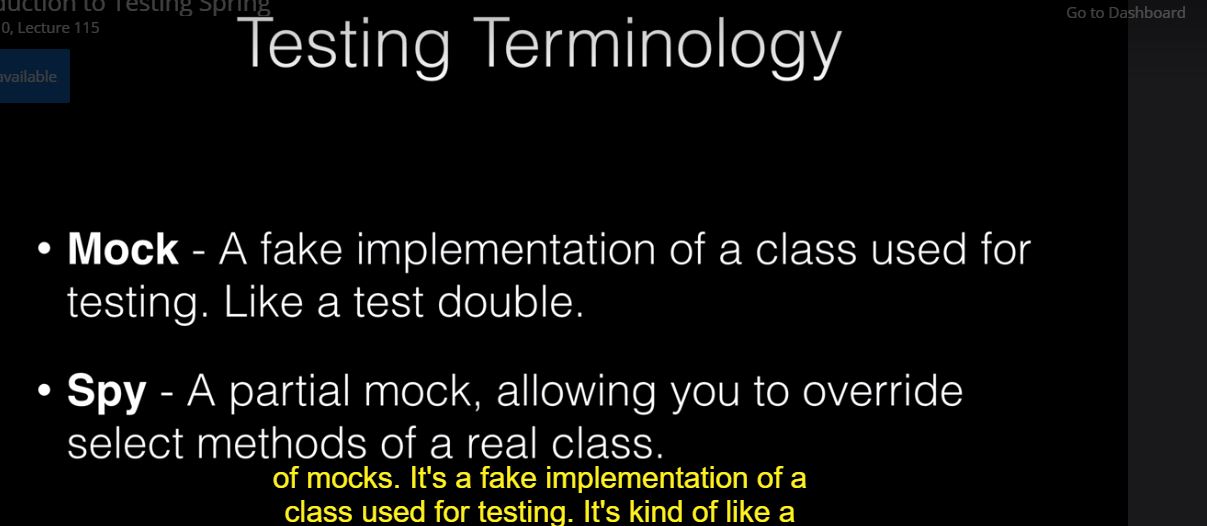
@Slf4j  
@Service  
public class RecipeServiceImpl implements RecipeService {  
  
 private RecipeRepository recipeRepository;  
  
 public RecipeServiceImpl(RecipeRepository recipeRepository) {  
 this.recipeRepository = recipeRepository;  
 }  
  
 @Override  
 public Set<Recipe> getRecipes() {  
 *log*.debug("I'm in the service");  
   
 Set<Recipe> recipes = new HashSet<>();  
 recipeRepository.findAll().iterator().forEachRemaining(recipes::add);  
 return recipes;  
 }  
}

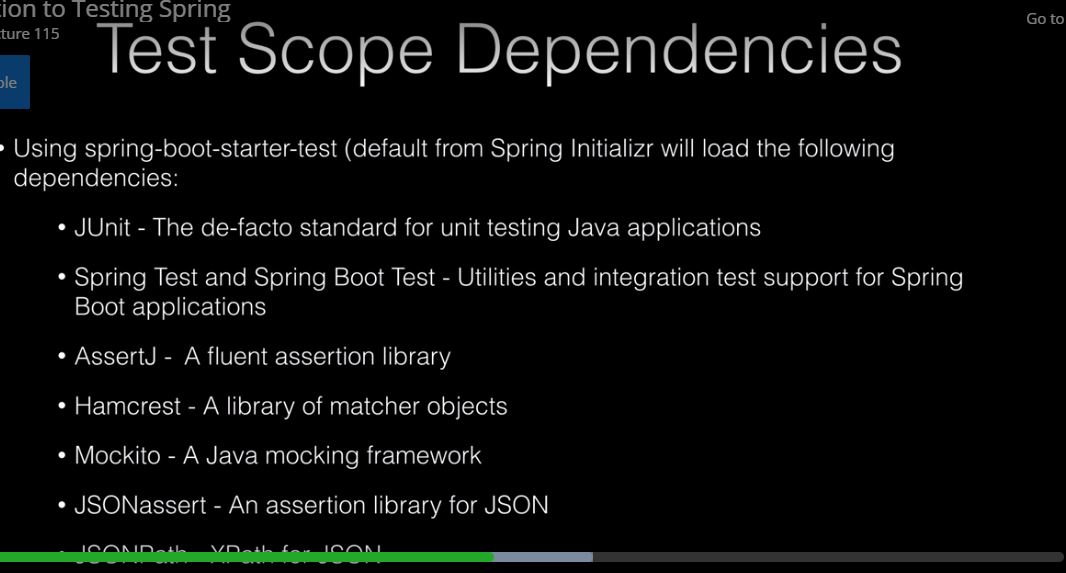
Project Lombok can use Slf4j to log actions

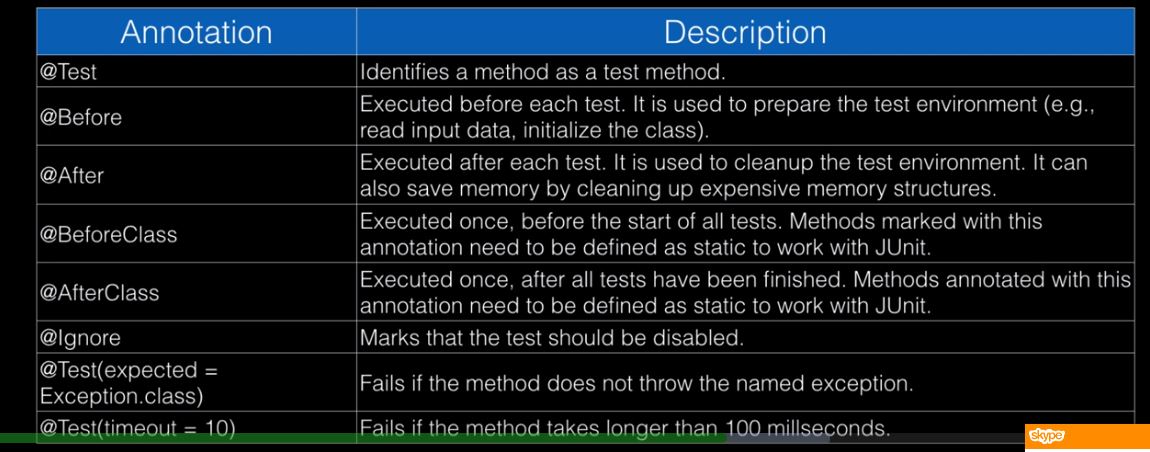
Bi-direction in Project Lombok will be error at equal and hash code so we must exclude some property at equal and hashcode

@Data  
@EqualsAndHashCode(exclude = {"recipe"})  
@Entity  
public class Ingredient {

**Introduction Testing Spring**

****





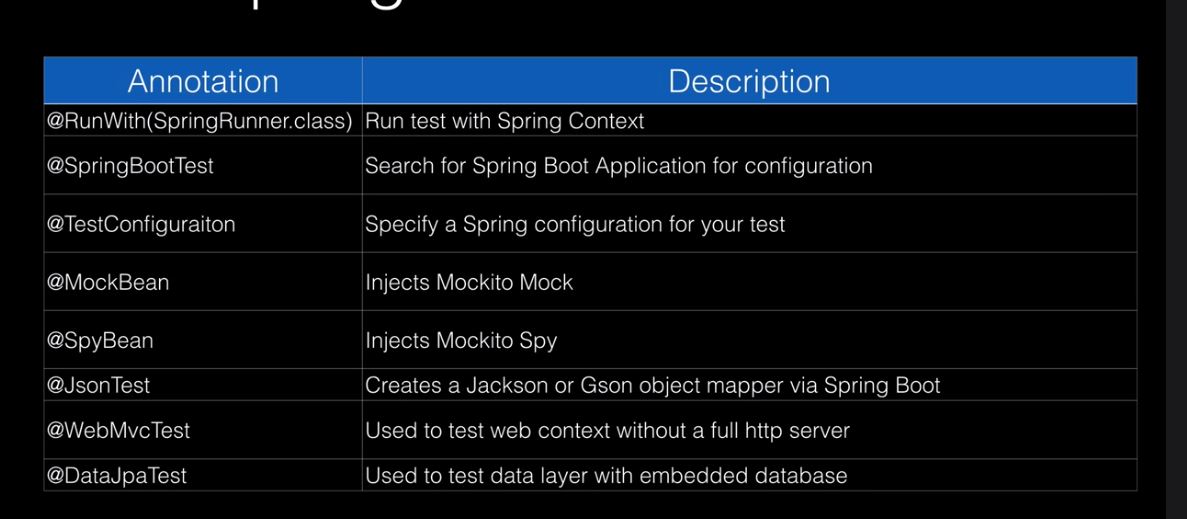
**Create a Junit Test**

Our spring boot project has already:

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
</dependency>

@RunWith(SpringRunner.class)  
@SpringBootTest  
public class Spring5RecipeAppApplicationTests {  
  
 @Test  
 public void contextLoads() {  
 }  
  
}

This is integration test actually. Not a Junit Test.



@Before  
public void setUp() {  
 category = new Category();  
}  
  
@Test  
public void getId() throws Exception {  
 category.setId(4l);  
 *assertEquals*(new Long(4l), category.getId());  
}  
  
@Test  
public void getDescription() throws Exception {  
 category.setDescription("This is category");  
 *assertEquals*("This is category", category.getDescription());  
}  
  
@Test  
public void getRecipes() throws Exception {  
 category.setRecipes(new HashSet<>(Arrays.*asList*(new Recipe())));  
 *assertNotNull*(category.getRecipes().iterator().next());  
}

**Mockito Mocks**

public class RecipeServiceImplTest {  
  
 RecipeServiceImpl recipeService;  
  
 @Mock  
 RecipeRepository recipeRepository;  
  
 @Test  
 public void getRecipes() throws Exception {  
 }  
  
}

We are trying to test for RecipeServiceImpl, in this class have method getRecipes() and we will test for this method. But in this method, call recipeRepository to find All Recipe, so we use mockito to test it. But it not finish

public class RecipeServiceImplTest {  
  
 RecipeServiceImpl recipeService;  
  
 @Mock  
 RecipeRepository recipeRepository;  
  
 @Before  
 public void setUp() throws Exception {  
 MockitoAnnotations.*initMocks*(this);  
 }  
  
 @Test  
 public void getRecipes() throws Exception {  
 }  
  
}

Add one more method setup and this above line MockitoAnnotations.initMocks(this) to tell mockito give me a mock recipe repository

@Test  
public void getRecipes() throws Exception {  
  
 *when*(recipeService.getRecipes()).thenReturn(new HashSet<>(Arrays.*asList*(new Recipe())));  
  
 *verify*(recipeRepository, *times*(0)).findAll();  
  
 Set<Recipe> recipes = recipeService.getRecipes();  
  
 *assertEquals*(recipes.size(), 1);  
  
 *verify*(recipeRepository, *times*(1)).findAll();  
}

recipeService.getRecipes() inside when, it only template, it isn’t really call. It is reason that when we verify recipeRepitory times(0) findAll method, it is true, because it never call inside this method

**Test Mock MVC**

@Test  
public void testMockMVC() throws Exception {  
 MockMvc mockMvc = MockMvcBuilders.*standaloneSetup*(indexController).build();  
 mockMvc.perform(*get*("/")).andExpect(*status*().isOk()).andExpect(*view*().name("index"));  
}

**Spring integration Test**

@RunWith(SpringRunner.class)  
@DataJpaTest  
public class UnitOfMeasureRepositoryITTest {  
  
 @Autowired  
 UnitOfMeasureRepository unitOfMeasureRepository;  
  
 @Before  
 public void setUp() throws Exception {  
 }  
  
 @Test  
 public void findByDescription() throws Exception {  
  
 Optional<UnitOfMeasure> uom = unitOfMeasureRepository.findByDescription("Teaspoon");  
  
 Assert.*assertEquals*("Teaspoon", uom.get().getDescription());  
 }  
  
}

DataJpaTest to test datalayer with embedded database

**Form Thymeleaf**

<form th:object="${recipe}" th:action="@{/recipe/}">

<input type="text" class="form-control" th:field="\*{description}"/>

th:field same as path=”” in JSP to binding two way data

<input type="hidden" th:field="\*{id}"/>

Id is also binding but not show

**Test at Controller**

@Test  
public void testGetNewRecipeForm() throws Exception {  
  
 *when*(recipeToRecipeCommand.convert(*any*())).thenReturn(new RecipeCommand());  
  
 mockMvc.perform(*get*("/recipe/new"))  
 .andExpect(*status*().isOk())  
 .andExpect(*model*().attributeExists("recipe"));  
}  
  
@Test  
public void testPostNewRecipeForm() throws Exception {  
  
 RecipeCommand command = new RecipeCommand();  
 command.setId(1L);  
 command.setDescription("American");  
 *when*(recipeService.saveRecipeCommand(*any*())).thenReturn(command);  
  
 mockMvc.perform(*post*("/recipe/")  
 .contentType(MediaType.*APPLICATION\_FORM\_URLENCODED*)  
 .param("id", "")  
 .param("description", "American")  
 )  
 .andExpect(*status*().is3xxRedirection())  
 .andExpect(*view*().name("redirect:/recipe/1/show"));  
 }  
  
 @Test  
 public void getRecipe() throws Exception {  
 Recipe recipe = new Recipe();  
 recipe.setId(1L);  
  
 *when*(recipeService.getRecipe(*anyLong*())).thenReturn(recipe);  
  
 Assert.*assertEquals*("recipe/show", controller.getRecipe(1L, model));  
  
 *verify*(recipeService, *times*(1)).getRecipe(*anyLong*());  
 *verify*(model, *times*(1)).addAttribute(*eq*("recipe"), *eq*(recipe));  
 }

@Before  
public void setUp() throws Exception {  
 MockitoAnnotations.*initMocks*(this);  
 controller = new RecipeController(recipeService, categoryService, recipeToRecipeCommand);  
 mockMvc = MockMvcBuilders.*standaloneSetup*(controller).build();  
}