Spring

Spring boot

Pros: Spring → Spring boot

Bootstrap: **Initiation configurations**

Embedded tomcat

SpringBootAppliacation.run ()

Auto Configurations: **changes are easy**

Every Spring **starter** & **app.pro**

Faster developing/ Easy testing

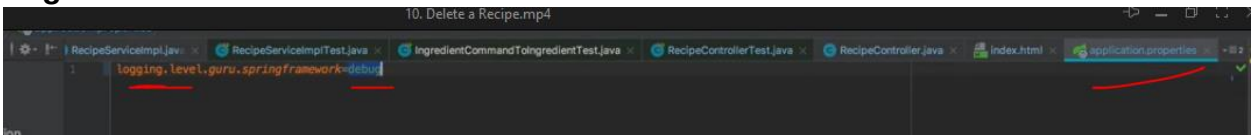
Spring boot handles configurations, and developers handle convention(codes)

SpringBoot **Bootstrap**:

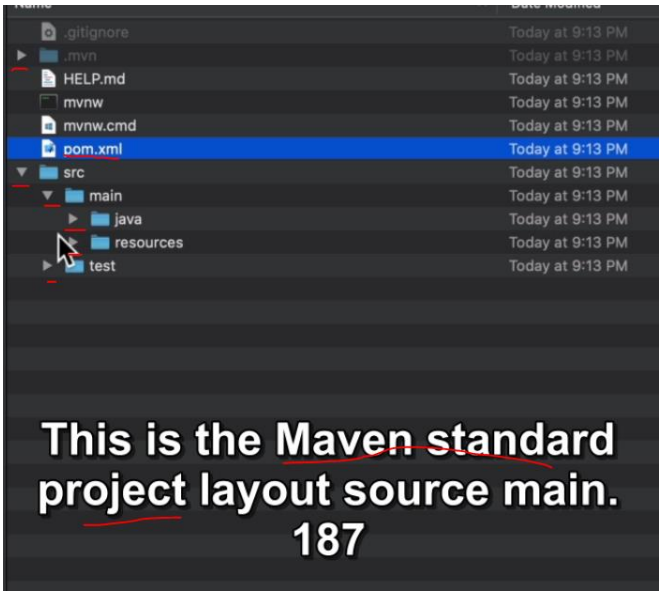
SpringBoot، **Initiation configures** دارد که با صدا شدن SpringBootAppliacation.**run ()** اجرا میشود. ولی قبلا مثلا SpringIOC تنظیمات زیادی در سطحServlet Container نیاز داشت که دولوپر باید از اول ایجاد میکرد. حالا یکی از Initiation configures هایی که springboot انجام میدهد (SpringBootAppliacation.**run ()**) برای همین SpringIOC هست.

Every Spring **starter =** auto configure for Spring Framework (**IOC**+DI+...) **+ another** ability (web + AOP + Security + ORM +…)

Log level

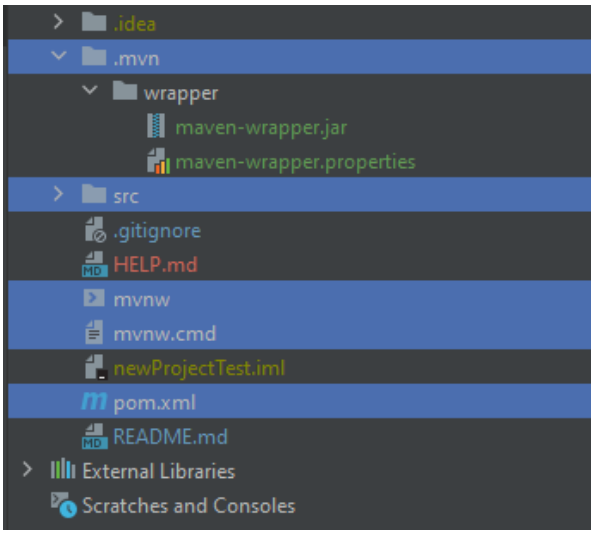


maven



.21Spring Pet Clinic - **Multi-Module** Maven Builds

adjust gitignore in first push



Data loader (in bootstrap)

اگر DB نداشته باشیم این روش خوب است. اگر DB داشته باشیم، روش های initialize DB توسط hiber و spring مهیا شده است.

Filldb.info

1- Run() of CommandLineRunner

پر کردن یک collection روی ram، با jpa یا مستقیم

(hiber) Repository jpa بجای دیتا بیس از Map استفاده شده ، Entityها روی ram هستند.

```java
@Component
public class DataLoader implements CommandLineRunner {

    private final OwnerService ownerService;
    private final VetService vetService;

    public DataLoader(OwnerService ownerService, VetService vetService) {
        this.ownerService = ownerService;
        this.vetService = vetService;
    }

    @Override
    public void run(String... args) throws Exception {

        Owner owner1 = new Owner();
        owner1.setId(1L);
        owner1.setFirstName("Michael");
        owner1.setLastName("Weston");

        ownerService.save(owner1);

        Owner owner2 = new Owner();
        owner2.setId(2L);
        owner2.setFirstName("Fiona");
```

```java
/**
 * Created by jt on 12/23/19.
 */
@Component
public class BootStrapData implements CommandLineRunner {

    private final AuthorRepository authorRepository;
    private final BookRepository bookRepository;

    public BootStrapData(AuthorRepository authorRepository, BookRepository bookRepository) {
        this.authorRepository = authorRepository;
        this.bookRepository = bookRepository;
    }

    @Override
    public void run(String... args) throws Exception {

        Author eric = new Author("Eric", "Evans");
        Book ddd = new Book( title: "Domain Driven Design",  isbn: "123123");
        eric.getBooks().add(ddd);
        ddd.getAuthors().add(eric);

        authorRepository.save(eric);
        bookRepository.save(ddd);

        Author rod = new Author("Rod", "Johnson");
        Book noEJB = new Book( title: "J2EE Development without EJB",  isbn: "3939459459");
        rod.getBooks().add(noEJB);
        noEJB.getAuthors().add(rod);

        System.out.println("Started in Bootstrap");
        System.out.println("Number of Books: " + bookRepository.count());
    }
}
```

2- **onApplicationEvent () 0f @Component ApplicationListener**

پر کردن یک collection روی ram، با jpa یا مستقیم

```java
@Component
public class RecipeBootstrap implements ApplicationListener<ContextRefreshedEvent> {

    private final CategoryRepository categoryRepository;
    private final RecipeRepository recipeRepository;
    private final UnitOfMeasureRepository unitOfMeasureRepository;

    public RecipeBootstrap(CategoryRepository categoryRepository, RecipeRepository recipeRepository, Uni
        this.categoryRepository = categoryRepository;
        this.recipeRepository = recipeRepository;
        this.unitOfMeasureRepository = unitOfMeasureRepository;
    }

    @Override
    public void onApplicationEvent(ContextRefreshedEvent event) {
        recipeRepository.saveAll(getRecipes());
    }

    private List<Recipe> getRecipes() {

        List<Recipe> recipes = new ArrayList<>( initialCapacity: 2);
```

we have **a lazy initialization exception** and what's happening is in our bootstrap class, we are doing this getRecipes () method and what's going to happen in that, each time we go in and out of the **Spring Data JPA repositories**, we are going in and out of a **Hibernate transaction.** So the lazy collections need to get initialized within a transaction and **within the same Hibernate session**. And that the problem is we're getting this thrown because we are getting it outside of that

But I think it's just a timing issue with the threads and how it's working in the transactional context. Now a very easy way around this, is we are going to utilize a new annotation called transactional. And this is part of the Spring Framework we really haven't gotten into transactions, but by doing this on the application event, now we're going to direct the Spring Framework to **create a transaction around this method**.

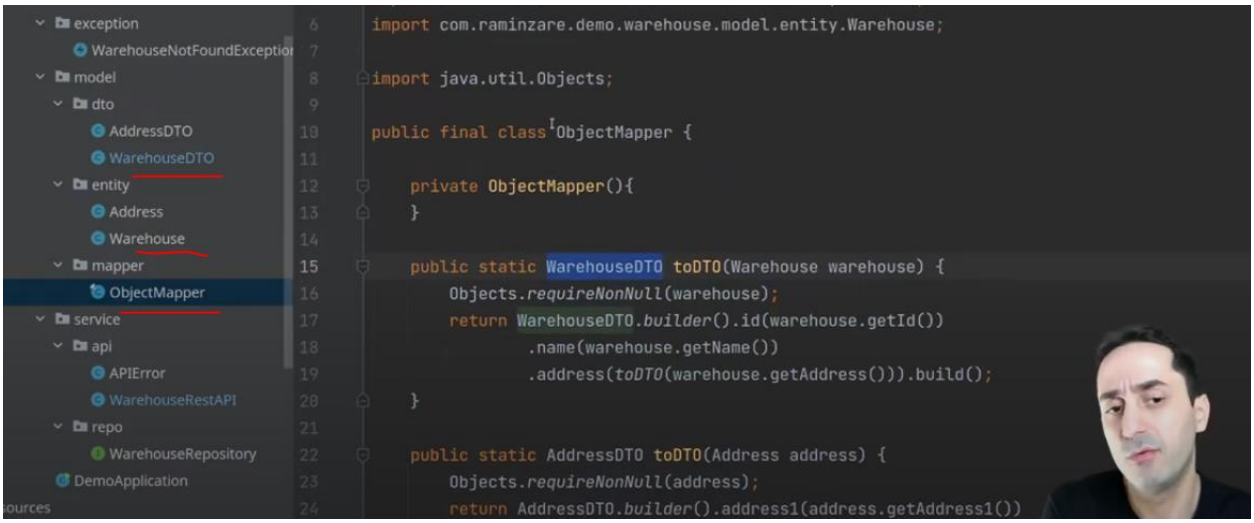
So remember any properties and the many relationships are going to be initialized lazily and that's what's happening with Hibernators.

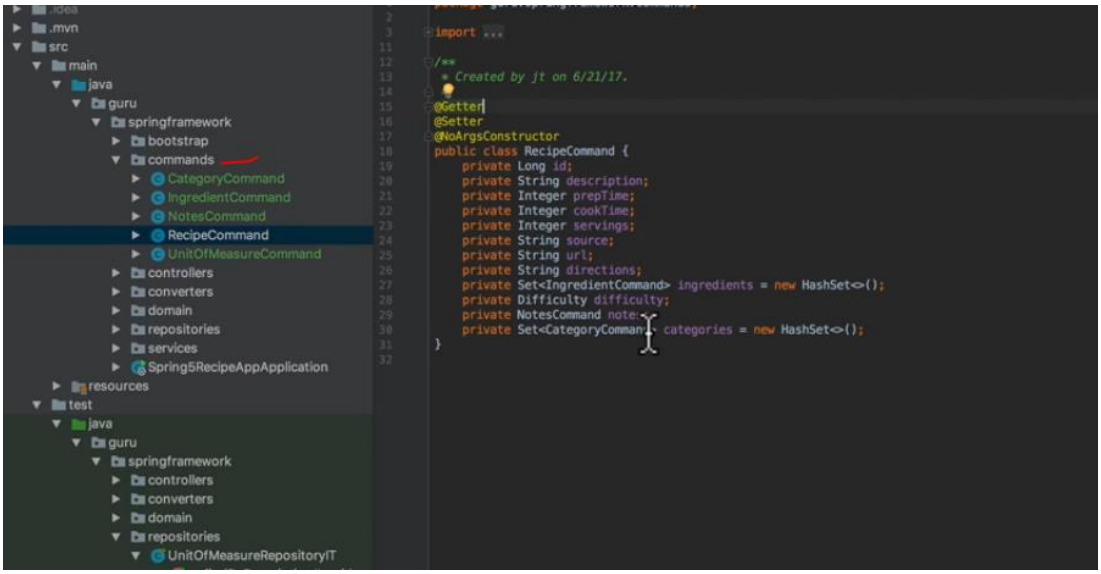
```java
@Override
@Transactional
public void onApplicationEvent(ContextRefreshedEvent event) {
    recipeRepository.saveAll(getRecipes());
}
```

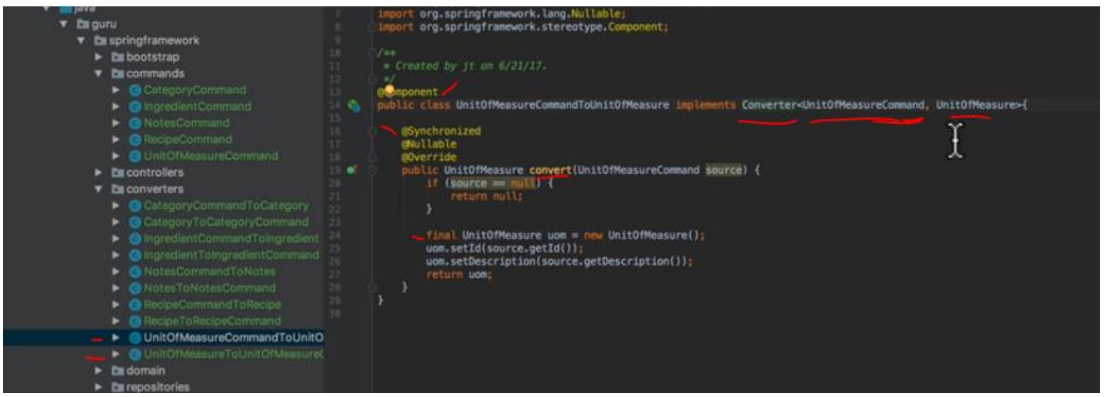
Converts / DTO & Entity

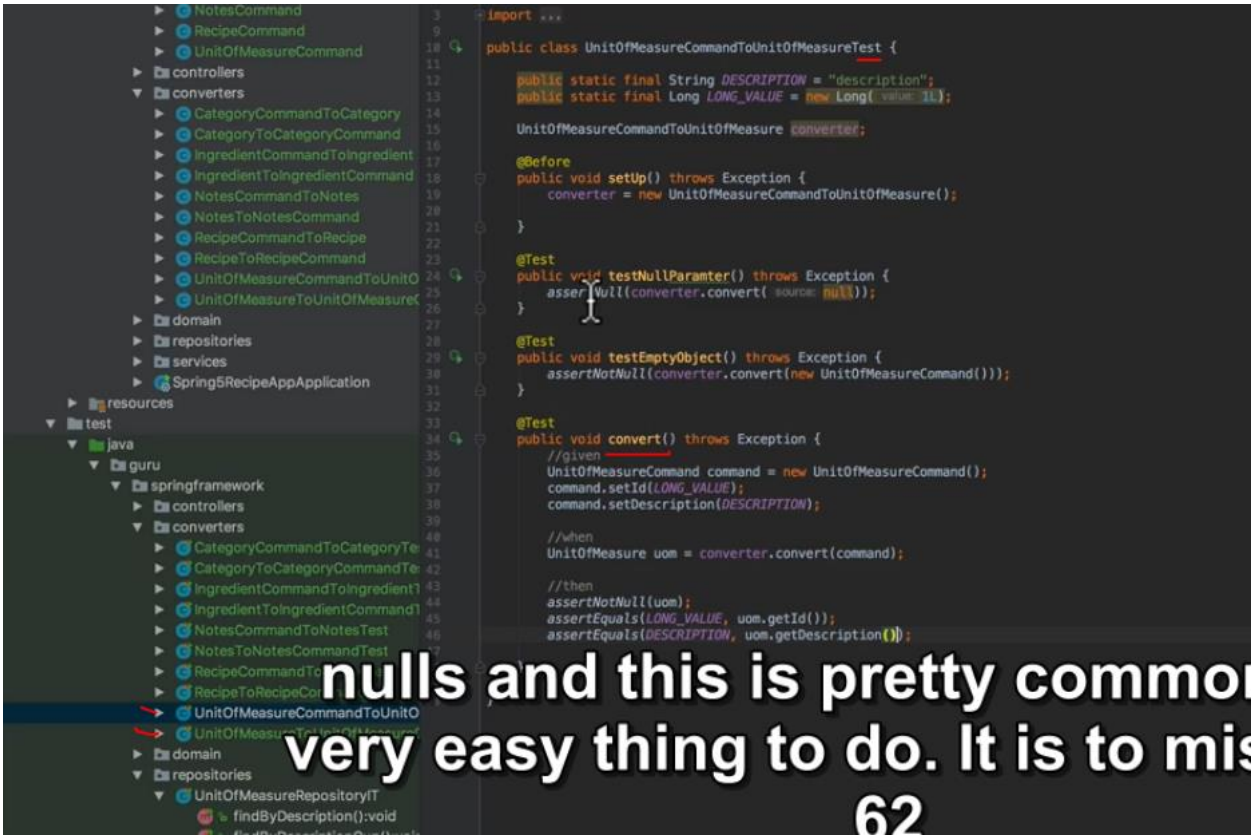
Mapper: any convertor, changing structure like serializing and DE serializing by jackson, stream.map(), a class for converting, ...

ramin






guru
springframework
bootstrap
commands
CategoryCommand
IngredientCommand
NotesCommand
RecipeCommand
UnitOfMeasureCommand
controllers
converters
CategoryCommandToCategory
CategoryToCategoryCommand
IngredientCommandToIngredient
IngredientToIngredientCommand
NotesCommandToNotes
NotesToNotesCommand
RecipeCommandToRecipe
RecipeToRecipeCommand
UnitOfMeasureCommandToUnitO
UnitOfMeasureToUnitOfMeasureC
domain
repositories
import org.springframework.lang.Nullable;
import org.springframework.stereotype.Component;
/**
* Created by jt on 6/21/17.
*/
@Component
public class UnitOfMeasureCommandToUnitOfMeasure implements Converter<UnitOfMeasureCommand, UnitOfMeasure>{
@Synchronized
@Nullable
@Override
public UnitOfMeasure convert(UnitOfMeasureCommand source) {
if (source == null) {
return null;
}
final UnitOfMeasure uom = new UnitOfMeasure();
uom.setId(source.getId());
uom.setDescription(source.getDescription());
return uom;
}
}


NotesCommand
RecipeCommand
UnitOfMeasureCommand
controllers
converters
CategoryCommandToCategory
CategoryToCategoryCommand
IngredientCommandToIngredient
IngredientToIngredientCommand
NotesCommandToNotes
NotesToNotesCommand
RecipeCommandToRecipe
RecipeToRecipeCommand
UnitOfMeasureCommandToUnitO
UnitOfMeasureToUnitOfMeasureC
domain
repositories
services
Spring5RecipeAppApplication
resources
test
java
guru
springframework
controllers
converters
CategoryCommandToCategoryTe
CategoryToCategoryCommandTe
IngredientCommandToIngredient1
IngredientToIngredientCommand1
NotesCommandToNotesTest
NotesToNotesCommandTest
RecipeCommandTo
RecipeToRecipeCo
UnitOfMeasureCommandToUnitO
UnitOfMeasureTo
domain
repositories
UnitOfMeasureRepositoryIT
findByDescription():void
import ...
public class UnitOfMeasureCommandToUnitOfMeasureTest {
public static final String DESCRIPTION = "description";
public static final Long LONG_VALUE = new Long(value: 1L);
UnitOfMeasureCommandToUnitOfMeasure converter;
@Before
public void setUp() throws Exception {
converter = new UnitOfMeasureCommandToUnitOfMeasure();
}
@Test
public void testNullParamter() throws Exception {
assertNull(converter.convert(source: null));
}
@Test
public void testEmptyObject() throws Exception {
assertNotNull(converter.convert(new UnitOfMeasureCommand()));
}
@Test
public void convert() throws Exception {
//given
UnitOfMeasureCommand command = new UnitOfMeasureCommand();
command.setId(LONG_VALUE);
command.setDescription(DESCRIPTION);
//when
UnitOfMeasure uom = converter.convert(command);
//then
assertNotNull(uom);
assertEquals(LONG_VALUE, uom.getId());
assertEquals(DESCRIPTION, uom.getDescription());
nulls and this is pretty commo
very easy thing to do. It is to mis
62

```java
public class RecipeServiceImpl implements RecipeService {

    private final RecipeRepository recipeRepository;
    private final RecipeCommandToRecipe recipeCommandToRecipe;
    private final RecipeToRecipeCommand recipeToRecipeCommand;

    public RecipeServiceImpl(RecipeRepository recipeRepository, RecipeCommandToRecipe recipeCommandToRecipe, RecipeToRecipeCommand recipeToRecipeCommand) {
        this.recipeRepository = recipeRepository;
        this.recipeCommandToRecipe = recipeCommandToRecipe;
        this.recipeToRecipeCommand = recipeToRecipeCommand;
    }

    @Override
    public Set<Recipe> getRecipes() {
        log.debug("I'm in the service");

        Set<Recipe> recipeSet = new HashSet<>();
        recipeRepository.findAll().iterator().forEachRemaining(recipeSet::add);
        return recipeSet;
    }

    @Override
    public Recipe findById(Long l) {

        Optional<Recipe> recipeOptional = recipeRepository.findById(l);

        if (!recipeOptional.isPresent()) {
            throw new RuntimeException("Recipe Not Found!");
        }

        return recipeOptional.get();
    }

    @Override
    @Transactional
    public RecipeCommand saveRecipeCommand(RecipeCommand command) {
        Recipe detachedRecipe = recipeCommandToRecipe.convert(command);

        Recipe savedRecipe = recipeRepository.save(detachedRecipe);
        log.debug("Saved RecipeId:" + savedRecipe.getId());
        return recipeToRecipeCommand.convert(savedRecipe);
```

```java
@Component
public class IngredientToIngredientCommand implements Converter<Ingredient, IngredientCommand> {

    private final UnitOfMeasureToUnitOfMeasureCommand uomConverter;

    public IngredientToIngredientCommand(UnitOfMeasureToUnitOfMeasureCommand uomConverter) {
        this.uomConverter = uomConverter;
    }

    @Synchronized
    @Nullable
    @Override
    public IngredientCommand convert(Ingredient ingredient) {
        if (ingredient == null) {
            return null;
        }

        IngredientCommand ingredientCommand = new IngredientCommand();
        ingredientCommand.setId(ingredient.getId());
        if (ingredient.getRecipe() != null) {
            ingredientCommand.setRecipeId(ingredient.getRecipe().getId());
        }
        ingredientCommand.setAmount(ingredient.getAmount());
        ingredientCommand.setDescription(ingredient.getDescription());
        ingredientCommand.setUom(uomConverter.convert(ingredient.getUom()));
        return ingredientCommand;
    }
}
```

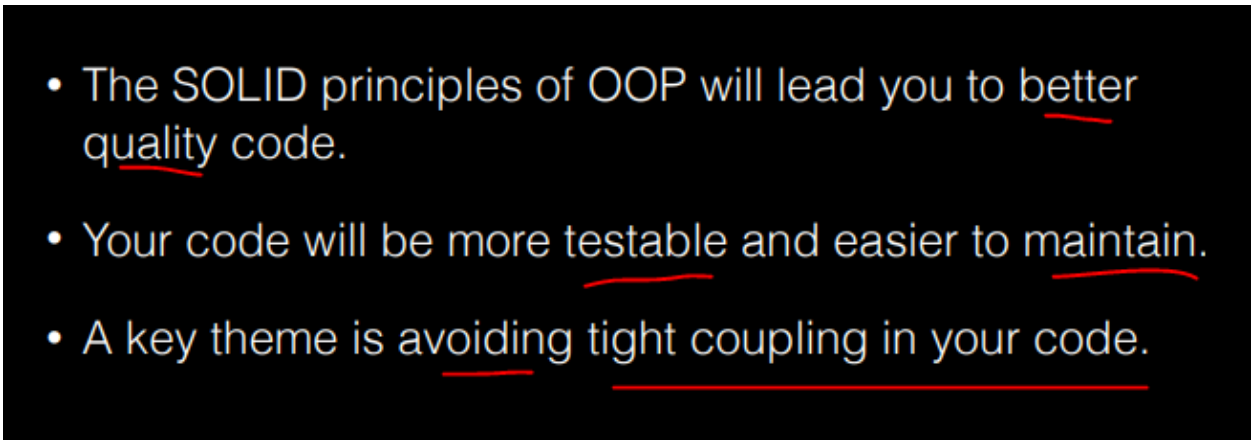
Use:

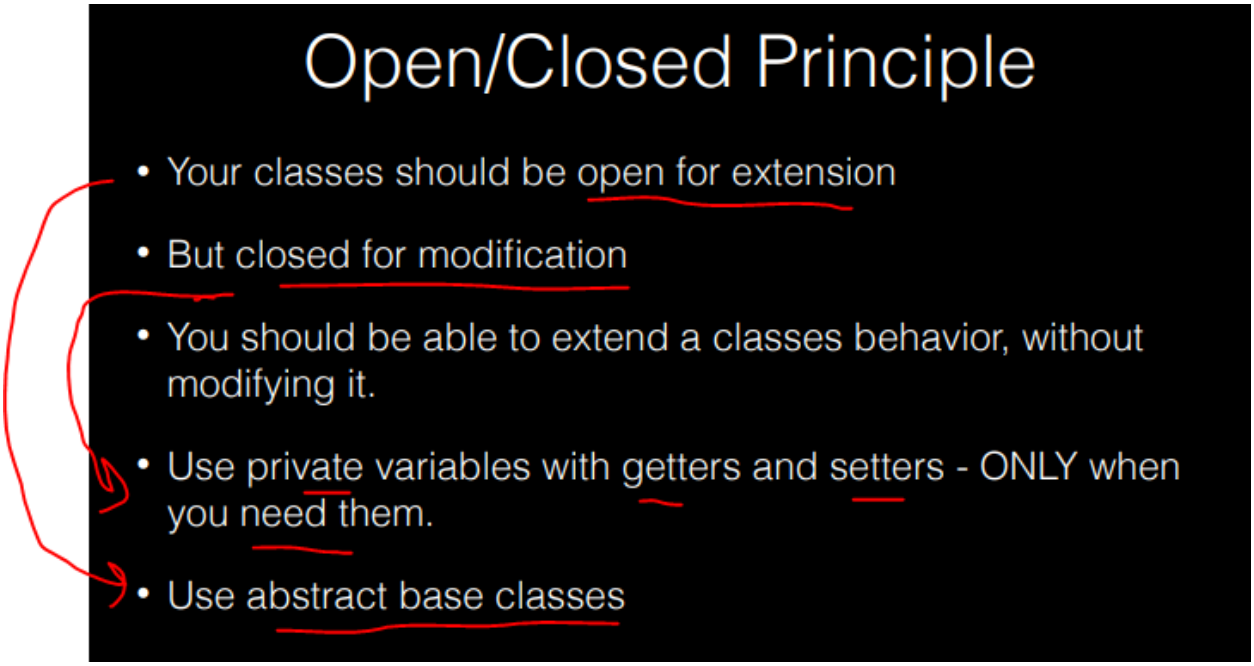
```java
@Override
@Transactional
public IngredientCommand saveIngredientCommand(IngredientCommand command) {
    Optional<Recipe> recipeOptional = recipeRepository.findById(command.getRecipeId());

    if(!recipeOptional.isPresent()){

        //todo toss error if not found!
        log.error("Recipe not found for id: " + command.getRecipeId());
        return new IngredientCommand();
    } else {
        Recipe recipe = recipeOptional.get();

        Optional<Ingredient> ingredientOptional = recipe
                .getIngredients()
                .stream()
                .filter(ingredient -> ingredient.getId().equals(command.getId()))
                .findFirst();

        if(ingredientOptional.isPresent()){
            Ingredient ingredientFound = ingredientOptional.get();
            ingredientFound.setDescription(command.getDescription());
            ingredientFound.setAmount(command.getAmount());
            ingredientFound.setUom(unitOfMeasureRepository
                    .findById(command.getUom().getId())
                    .orElseThrow(() -> new RuntimeException("UOM NOT FOUND"))); //todo address this
        } else {
            //add new Ingredient
            Ingredient ingredient = ingredientCommandToIngredient.convert(command);
            ingredient.setRecipe(recipe);
            recipe.addIngredient(ingredient);
        }
```

----------Spring framework (Spring IOC, ...)

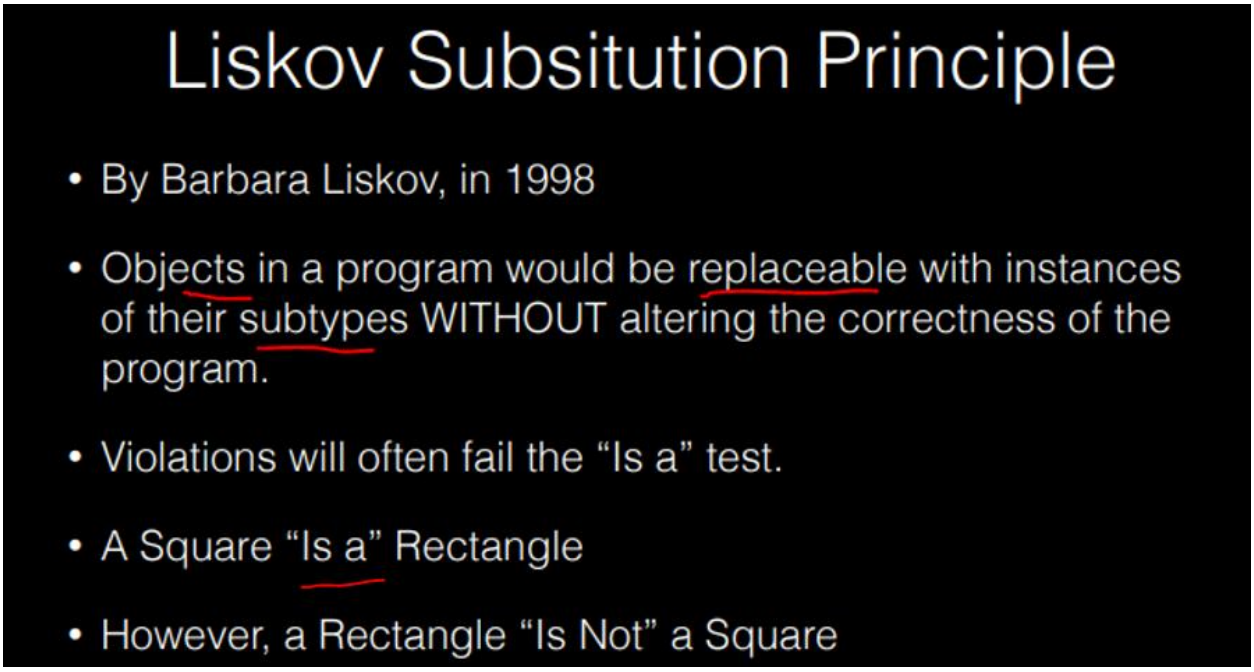
Solid

Solids increases Quality of codes → testable, extendable, reusable





substitution



Interface Segregation Principle

- Make fine grained interfaces that are client specific
- Many client specific interfaces are better than one "general purpose" interface
- Keep your components focused and minimize dependencies between them
- Notice relationship to the Single Responsibility Principle?
  - ie avoid 'god' interfaces

Dependency Inversion Principle

- Abstractions should not depend upon details
- Details should depend upon abstractions
- Important that higher level and lower level objects depend on the same abstract interaction
- This is not the same as Dependency Injection - which is how objects obtain dependent objects

high-level modules should not depend on low-level modules;

low-level modules should depend on high-level modules;

higher level and lower level objects should depend on abstract interaction.

DI pattern – SpringIOC

Based on "**Dependency Inversion**" of "**Solid Principles**":

وقتی کلاس L یک object از کلاس H میخواهد، obj h1 یک dependency است که کلاس L به کلاس H دارد.

inversion of control (IoC) pattern refers to **transferring and controlling** of objects.

Dependency Injection (DI) is a design **pattern** used to implement IoC. Using DI

we move the **creation and binding** of the dependent objects **outside of the class** that depends on them. and provides those objects to a class through different ways (like in SpringIOC: injection an instance of beans).

Spring IoC spring framework (Inversion of Control) **Container** (context) is the core of Spring Framework. It **creates the objects (beans)**, **injects** their dependencies in **runtime**, and **manages life cycle** of those objs.

Design Model Approaches: Strategy / factory / Dependency Injection (**DI**)

Implementation of DI: / Jakarta / Javax inject

SpringIOC یک ابزار است که Spring framework برای پیاده سازی **dependency injection pattern** بصورت **runtime** معرفی کرده. کلا مزیت SpringIOC این است که در runtime تعیین میکنیم dependency ای که میخواهیم inject کنیم از کدام bean باشد و این را از polymorphism جاوا دارد.

SpringIOC: Dependency inject at runtime

کارهایی که SpringIOC میکند:

context ساخته میشود.

در آن از هر component یک bean ساخته و در context قرار میگیرد. (scope: singleton)

هر وقت در runtime کلاسی dependencyای خواست؛ یک instance از آن bean میگیرد و بعنوان dependency، inject میکند.

مدیریت LifeCycle همه ی instanceها.

Instanceای که در زمان inject از SpringIOC میخواهیم بگیریم یک **dependency برای کلاس استفاده کننده** از آن instance محسوب میشود.

SpringBoot، **Initiation configures** دارد که با صدا شدن SpringBootAppliacation.**run ()** اجرا میشود. ولی قبلا مثلا SpringIOC تنظیمات زیادی در سطحServlet Container نیاز داشت که دولوپر باید از اول ایجاد میکرد. حالا یکی از Initiation configures هایی که springboot انجام میدهد برای همین کارهایی است که SpringIOC میکند مثل ساخت contextو beanها و DI.

Every Spring Boot starter has spring IOC.

JRE → ServletContainer Initialization →

SpringbootServletInitialization(ServletContexInitialization) create **IOC ApplicationContext** →

(EE espc. + **DispatcherServlet (mapping)**) →

WebServletContainer (DI-create instance**)**

Defining Components

با اجرای اتوماتیک شدن () SpringBootAppliacation.run، SpringIOC اقدام به ساخت context و beanها میکند؛ SpringIOC را چطور پیدا میکند:

روش های تعریف Component:

- **XML**: در Configuration مشخص است چه beanهای میخواهیم، پس SP آنها را خواهد ساخت و به context میبرد.
- **java based**: در Configuration مشخص است چه beanهای میخواهیم ، پس SP آنها را خواهد ساخت و به context میبرد.

- **Annotation**: sp با کمک **reflection** خواهد فهمید چه beanهایی میخواهیم (توی کل پروژه به دنبال @Component ها خواهد گشت) ، پس SP آنها را خواهد ساخت و به context میبرد.

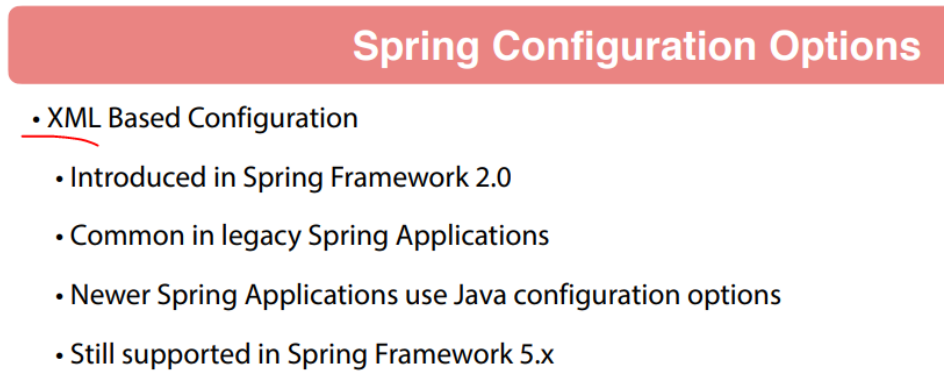
مثلا:

```
@Component
public class Alien {
```

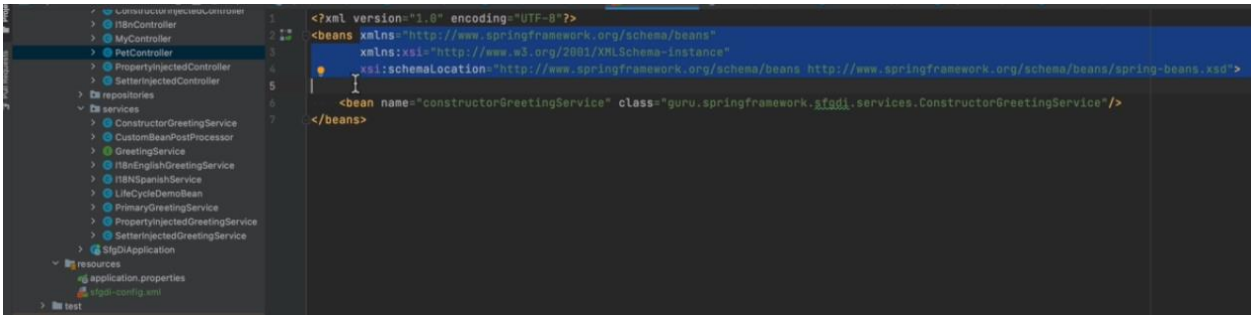
به Spring IOC میگوید: Alien یک Component است و میخواهیم یک bean از آن در Context داشته باشیم.

XML

سرراست این file را میخواند و beanها را میبرد داخل context

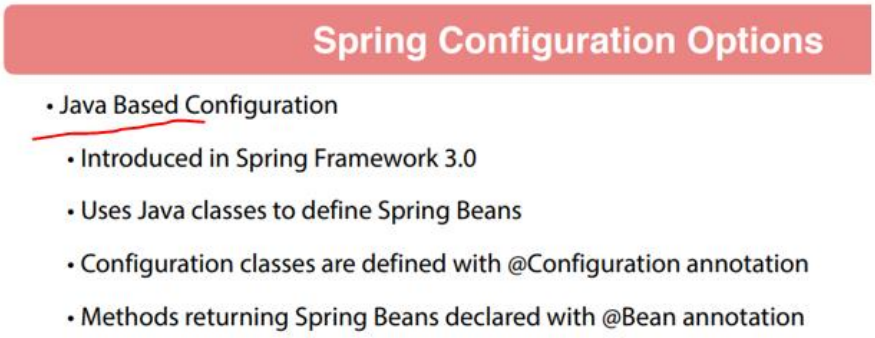



```
*/
@ImportResource("classpath:sfgdi-config.xml")
@Configuration
public class GreetingServiceConfig {
```



Java based (class) / @Bean

سرراست این کلاس را میخواند و beanها را میبرد داخل context



@Been on methods???

اسم کلاس Component را برابر اسم این method قرار میدهیم، تا از آن یک bean بسازد.

روش java based چه زمانی به @ها **ارجحیت** داره؟ زمانی که بخوایم default configuration یک لایبرری third party رو عوض کنیم، مثلا وقتی از jakson یک mapper میخوایم ولی نه طبق روش دیفالتش.

Annotation

توضیحات این روش در کل یادداشت بیان شده

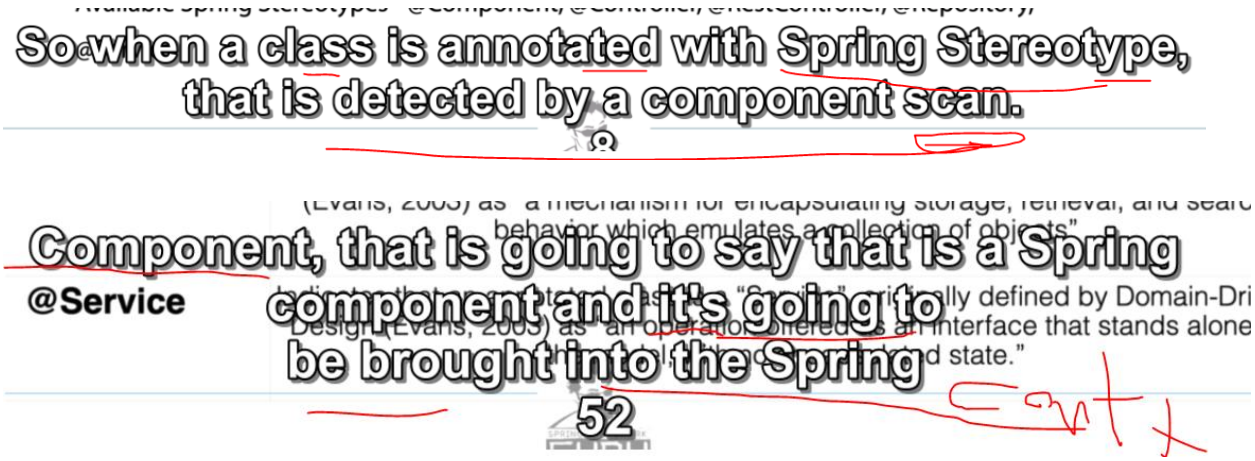
Spring Configuration Options

- Annotation Based Configuration - Annotations introduced in Java 5
  - Introduced in Spring Framework 3
  - Spring beans found via 'Component Scans'
    - A scan of packages for annotated components
  - Spring Beans are found via class level annotations
    - @Controller, @Service, @Component, @Repository
      - More on these later

بالای خود کلاس Component، @Component قرار دهیم.

در این روش Sp component scan از **reflection** استفاده میکند و **load اولیه context** را **کند میکند**.

Component scan:



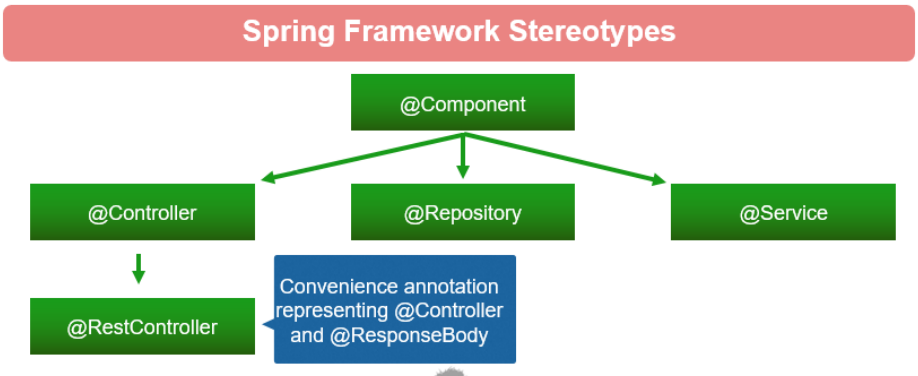
Stereotypes:

- Spring Stereotypes are class level annotations used to define Spring Beans
  - When classes annotated with Spring Stereotypes are detected via the component scan, an instance of the class will be added to the Spring context
- Available Spring Stereotypes - @Component, @Controller, @RestController, @Repository, @Service

Should be in main pack. Otherwise...

- This configuration is Spring Framework specific, NOT Spring Boot
- Spring Boot's auto configuration will tell Spring to perform a component scan of the package of the main class
  - This includes all sub packages of the main class package
- When using Spring Boot, if class is outside of the main class package tree, you must declare the package scan


```
@ComponentScan(basePackages = {"guru.springframework.sfgdi", "com.springframework.pets"})
@SpringBootApplication
public class SfgDiApplication {

    public static void main(String[] args) {
```



Annotation	Description
@Component	Indicates that an annotated class is a “component” and it will be created as a bean
@Controller	Indicates that an annotated class has the role of a Spring MVC “Controller”
@RestController	Convenience Annotation which extends @Controller, and adds @ResponseBody
@Repository	Indicates class is a “Repository”, originally defined by Domain-Driven Design (Evans, 2003) as “a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects”
@Service	Indicates that an annotated class is a “Service”, originally defined by Domain-Driven Design (Evans, 2003) as “an operation offered as an interface that stands alone in the model, with no encapsulated state.”

@Component: I am a bean (to IOC)

@Controller: I am a bean (to IOC)- sp MVC or sp web

Bean’s name

نامی که برای bean ای که از component در context قرار میگیرد.

1- در روش java based:

```
@Bean
I18nEnglishGreetingService i18nService(){
    return new I18nEnglishGreetingService();
}
```

```
@Bean("i18nService")
```

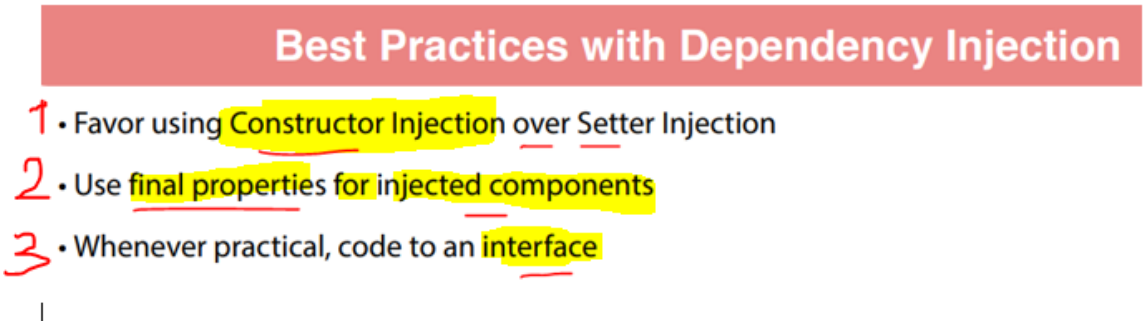
2- در روش @

```
@Component
public class Alien {
```

```
@Component("loplop")
```

```
@Service
public class PrimaryGreetingService implements GreetingService {
```

*****Best practice Dependency Injection in SpringIOC:**



Components are types of an Interface

componentها بهتر است **فرزندهایی از <u>یک نوع Interface</u>** باشند، بجای کلاس سخت.

<u>فایده</u>: همه ی فرزندها componentهای جداگانه هستند، برایشان bean هایی با nameهای متفاوت ساخته میشود. و همه interface را impl کرده اند. این دست ما را باز میگذارد که با استفاده از قابلیت **polymorphism** در **<u>runtime</u>**؛ <u>با توجه به profile یا qualifier یا primary</u>؛ تصمیم بگیریم **instance** از روی **چه beanای** inject شود و **در refای از interface پدر قرار گیرد**. کلا مزیت SpringIOC این است که در runtime تعیین میکنیم dependency ای که میخواهیم inject کنیم از کدام bean باشد و این را از polymorphism جاوا دارد.

```java
@Primary
@Service
public class PrimaryGreetingService implements GreetingService {

    @Override
    public String sayGreeting() {
        return "Hello World - From the PRIMARY Bean";
    }
}
```

Injecting: private final prop of **super type** (interface) **+ constructor** for initialize

By constructor (preferred)- implicitly

```
private final GreetingService greetingService;

public MyController(GreetingService greetingService) {
    this.greetingService = greetingService;
}
```

در runtime وقتی میخواهیم injection کنیم **دراین ref از interfaceپدر که بنفش است**، میتوانیم هر یک از beanهای انواع فرزندانش را inject کنیم.

وقتی final private تعیین میشود، ref قابل تغییر نیست.

در روش constructor، **dependency** (instanceای که از beanمد نظرمان SpringIOC برایمان میسازد) **زمانی inject میشود** که؛ داریم **یک instance از خود کلاس inject کننده میسازیم.** (با constructor)

پس این کار best practice نیست:

```
@Component
public class Laptop {
```

```
@Component
public class Alien {
    @Autowired
    private Laptop laptop;
```

Concrete Classes vs Interfaces

- DI can be done with Concrete Classes or with Interfaces
- Generally DI with Concrete Classes should be avoided
- DI via Interfaces is highly preferred
  - Allows runtime to decide implementation to inject
  - Follows Interface Segregation Principle of SOLID
  - Also, makes your code more testable

Is it good practice to use concrete classes for dependency injection?

No. You should use interfaces, which will allow the runtime environment to determine the implementation to inject.

Types of dependency Injection in IOC

برای scope (تعداد instanceها) فرقی نمیکند از چه روشی injection (constructor یا @Autowired) را انجام داده باشیم.

handy Inject instance from Context

SpringBoot، Initiation configurations دارد که با صدا شدن **SpringBootAppliacation.run ()** اجرا میشود.
ولی قبلا مثلا SpringIOC تنظیمات زیادی در سطحServlet Container نیاز داشت که دولوپر باید از اول ایجاد میکرد.
حالا یکی از Initiation configurationsهایی که springboot انجام میدهد (SpringBootAppliacation.run ()) برای همین SpringIOC هست. یعنی **ساخت Application context و bean ها در آن. این متد اتوماتیک کال میشود نه توسط Developer.**

SpringApplication. run ()

return Spring Application context with all beans.

```
@SpringBootApplication
public class SfgDiApplication {

    public static void main(String[] args) {
        ApplicationContext ctx = SpringApplication.run(SfgDiApplication.class, args);
```

developer به Spring IOC میگوید میخواهیم یک instance از bean بگیریم، برایم inject کن:

Get instance by ctx.**getBean("Class"):**

Get a bean **handy** from ctx:

برای استفاده از هر bean، با این دستور میتوانیم یک instance از آنرا inject کنیم:

```
Alien a = context.getBean(Alien.class);
```

```java
@Controller
public class MyController {

    public String sayHello(){
        System.out.println("Hello World!!!");

        return "Hi Folks!";
    }
}
```

```java
@SpringBootApplication
public class SfgDiApplication {

    public static void main(String[] args) {
        ApplicationContext ctx = SpringApplication.run(SfgDiApplication.class, args);

        MyController myController = (MyController) ctx.getBean( s: "myController");

        String greeting = myController.sayHello();

        System.out.println(greeting);
    }

}
```

در پروژه های واقعی inject controller را app server درخواست میکند

سه روش پیشرفته تر هم دارد:

Explicitly- @Autowired on property

By Class properties (@Autowired (using **reflection**)) - explicitly

- Public (bad)
- Private (Evil) ???

```java
@Qualifier("propertyInjectedGreetingService")
@Autowired
public GreetingService greetingService;
```

Can you use Dependency Injection against a private property in Spring?

Yes, Spring will support this, and inject the dependency using reflection at runtime. However, this is considered a VERY bad practice.

Explicitly- @Autowired on setterMethods (debate)

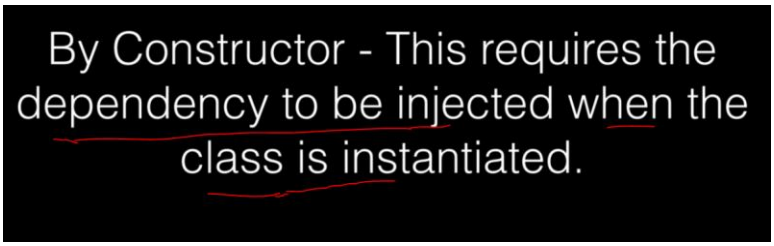
```
private GreetingService greetingService;

@Qualifier("setterInjectedGreetingService")
@Autowired
public void setGreetingService(GreetingService greetingService) {
    this.greetingService = greetingService;
```

Implicitly- Constructor + final prop + in ref of Interface (preferred**)**

توضیحات در قسمت Best practice in SpringIOC داده شد.

```
private final GreetingService greetingService;

public MyController(GreetingService greetingService) {
    this.greetingService = greetingService;
}
```

```
    public ConstructorInjectedController(@Qualifier("constructorGreetingService")
GreetingService greetingService) {
        this.greetingService = greetingService;
    }
```



Sample with spring: https://github.com/springframeworkguru/sfg-di/tree/spring-di/src/main/java/guru/springframework/sfgdi

@Qualifier

کلا مزیت SpringIOC این است که در runtime تعیین میکنیم dependency ای که میخواهیم inject کنیم از کدام bean باشد و این را از polymorphism جاوا دارد.

beanها بر اساس نامی که هنگام تعریف Component تعیین شده، درContext قرار گرفته اند.

حالا در زمان injection با qualifier تعیین میکنیم، از کدام bean یک instance میخواهیم.

bean:

```
@Component("loplop")
public class Laptop {
```

injection:

```
public class Alien {
    @Autowired
    @Qualifier("loplop")
    private Laptop laptop;
```

کاربرد اصلی qualifier خودش رو در best practice نشان میدهد.(beanهایی از یک نوع interface)

```
    public ConstructorInjectedController(@Qualifier("constructorGreetingService")
GreetingService greetingService) {
        this.greetingService = greetingService;
    }
```

@Primary bean

یک راه این بود که با @Qualifier تعیین کنیم که dependencyای میخواهیم چیست (از کدام bean برایمان instance بسازد) و **inject** کند، راه دیگر @primary است. **@primary** وقتی که **دو bean کاندید inject** شدن هستند، **اولویت** را مشخص میکند.

```
private final GreetingService greetingService;

public MyController(GreetingService greetingService) {
    this.greetingService = greetingService;
}
```

چون qualifier نزدیم، الان IOC نمیداند از کدامیک از beanهایی که از این نوع interface ساخته شده، instance بسازد و inject کند، پس exception میخورد؛ راه حل:

اولویت qualifier بالاتر از primary است. اگر inject کننده Qualifier نکرد از کدام bean، instance میخواهد.

اگر inject کننده صراحتا نام beanای که میخواهد را **Q کند، تعریف primary bean اهمیت ندارد.**

تعیین کردن primary bean :

```
@Primary
@Service
public class PrimaryGreetingService implements GreetingService {

    @Override
    public String sayGreeting() {
        return "Hello World - From the PRIMARY Bean";
    }
}
```

با روش java based :

```
@Primary
@Bean
PrimaryGreetingService primaryGreetingService(){
    return new PrimaryGreetingService();
}
```

@Profiles

It allows us to have **Different run time Environment**

فقط وقتیکه آن profile، active است، از آن Componentهایی که در آن profile هستند bean ساخته میشود و در context قرار میگیرد.

قرار است، این componentها هر کدام در یک Environment جداگانه استفاده شوند.

بدین وسیله براحتی در زمان runtime با تغییر profile،been ای که میخواهیم ساخته شود را تغییر میدهیم.

قرار دادن یک bean در یک profile خاص:

```
@Profile("EN")
@Bean
I18nEnglishGreetingService i18nService(){
    return new I18nEnglishGreetingService();
}
```

Componentای که برایش profile تعیین نمیشود، در همه profileها هست و برایش bean ساخته میشود.

وقتی یک interface را چند component، impl کرده باشند، هر کدام را میتوان در profileای جداگانه قرار داد.

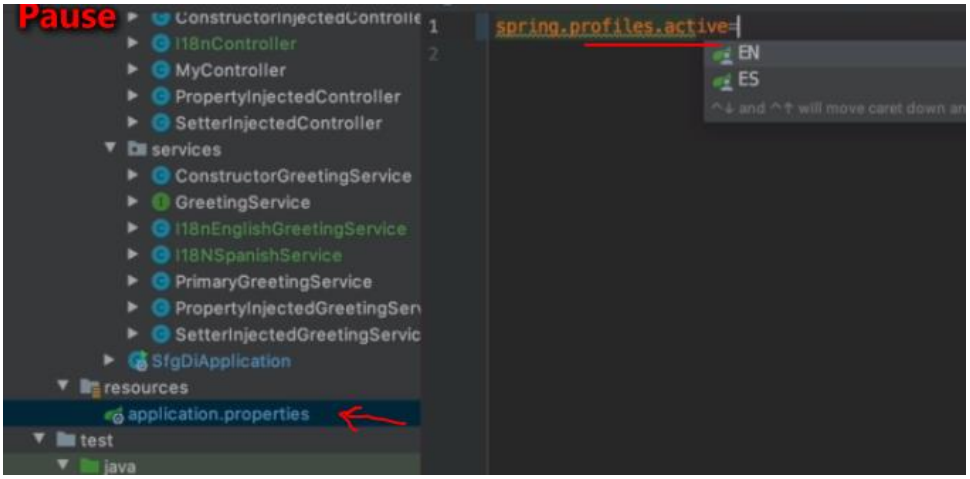
(حتی میتوانند نام های یکسان هم بگیرند، هر کدام در Env خود فعال است)

(هر دو هم میتونن @primary بگیرند، هر کدام در Env خود فعال است)

Active Profile

با تعیین کردن active profile برای application مشخص میکنیم، از بین beanهای که profile برایشان تعیین شده، کدامشان قابل bean شدن و inject شدن هستند.

Componentای که برایش profile تعیین نمیشود، در همه profileها هست و برایش bean ساخته میشود.



اینکار جز bootstrap بودن appهای مبتنی بر Springboot است.

```java
@Profile("ES")
@Service("i18nService")
public class I18NSpanishService implements GreetingService {
    @Override
    public String sayGreeting() {
        return "Hola Mundo - ES";
    }
}
```

```java
@Profile("EN")
@Service("i18nService")
public class I18nEnglishGreetingService implements GreetingService {
    @Override
    public String sayGreeting() {
        return "Hello World - EN";
    }
}
```

```java
@Controller
public class I18nController {

    private final GreetingService greetingService;

    public I18nController(@Qualifier("i18nService") GreetingService greetingService) {
        this.greetingService = greetingService;
    }

    public String sayHello(){
        return greetingService.sayGreeting();
    }
}
```

وقتی qualifier تعیین نکنیم؛ از بین یک bean بدون profile ولی primary، و یک bean که در active profile است، اونی که primary است inject میشود.

Default Profile

اولویت <u>active profile</u> بالاتر از default profile است.

وقتی در active profile هیچ profileای active اعلام نشده باشد، componentای که "default" باشد، active محسوب میشود و از آن bean ساخته میشود:

```java
@Profile({"ES", "default"})
@Bean("i18nService")
I18NSpanishService i18NSpanishService(){
    return new I18NSpanishService();
}

@Profile("EN")
@Bean
I18nEnglishGreetingService i18nService(){
    return new I18nEnglishGreetingService();
}
```

```java
@Profile({"ES", "default"})
@Service("i18nService")
public class I18NSpanishService implements GreetingService {
    @Override
    public String sayGreeting() {
        return "Hola Mundo - ES";
    }
}
```

```java
 */
@Service
@Profile({"default", "map"})
public class OwnerMapService extends AbstractMapService<Owner, Long> implements OwnerService {
```

```java
 */
@Service
@Profile("springdatajpa")
public class SpecialitySDJpaService implements SpecialtyService {

    private final SpecialtyRepository specialtyRepository;
```

resources > application.properties

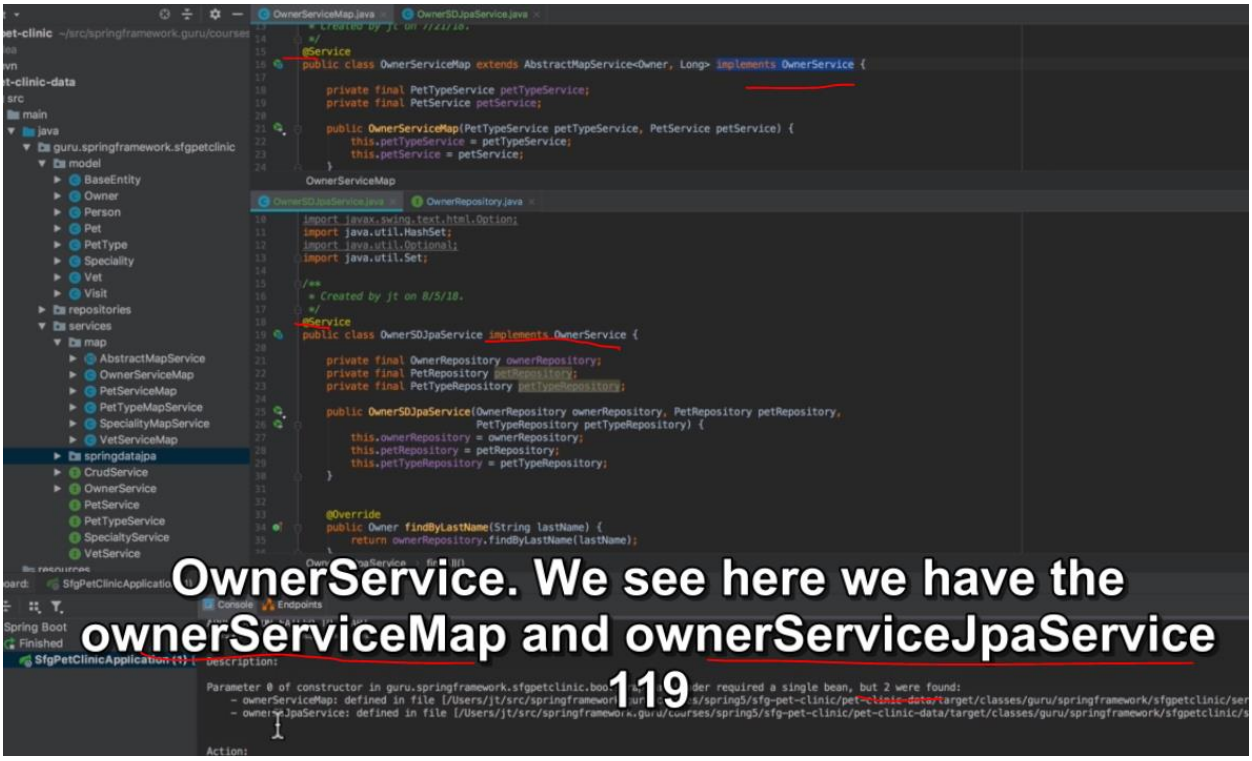
MapService.jav | application.properties | SpecialitySDJpaService

```
1  spring.banner.image.location=vizsla.jpg
2
3  # Internationalization
4  spring.messages.basename=messages/messages
5
6  spring.profiles.active=springdatajpa
```

sample

```
@Service
public class OwnerServiceMap extends AbstractMapService<Owner, Long> implements OwnerService {

    private final PetTypeService petTypeService;
    private final PetService petService;

    public OwnerServiceMap(PetTypeService petTypeService, PetService petService) {
        this.petTypeService = petTypeService;
        this.petService = petService;
    }

    @Override
    public Set<Owner> findAll() { return super.findAll(); }

    @Override
    public Owner findById(Long id) { return super.findById(id); }

    @Override
    public Owner save(Owner object) {

        if(object != null){
            if (object.getPets() != null) {
                object.getPets().forEach(pet -> {
                    if (pet.getPetType() != null){
                        if(pet.getPetType().getId() == null){
                            pet.setPetType(petTypeService.save(pet.getPetType()));
```

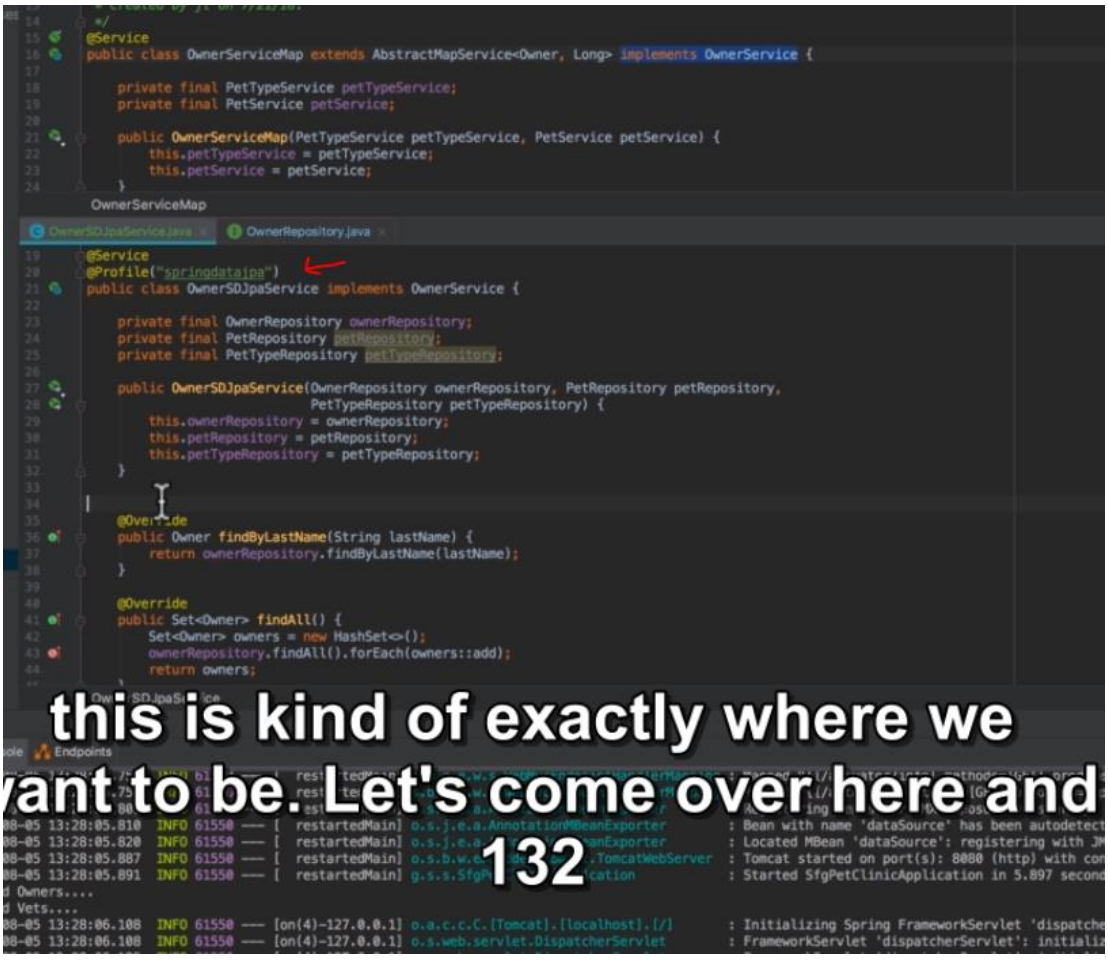
Two component are candidate for injection: Exception



Solution 1: use different @qualifiers injection

Solution 2: set @primary component

Solution 3: different @profiles



factory

Spring IOC برای ساخت (bean) میتواند بصورت **default** برخورد کند یا از ما **factory** بگیرد.

```
@Bean
PetServiceFactory petServiceFactory(){
    return new PetServiceFactory();
}

@Profile({"dog", "default"})
@Bean("petService")
PetService dogPetService(PetServiceFactory petServiceFactory){
    return petServiceFactory.getPetService( petType: "dog");
}
```

new

```
public class PetServiceFactory {

    public PetService getPetService(String petType){
        switch (petType){
            case "dog":
                return new DogPetService();
            case "cat":
                return new CatPetService();
            default:
                return new DogPetService();
        }
    }
}
```

```
public class DogPetService implements PetService {
    public String getPetType() { return "Dogs are the best!"; }
}
```

استفاده یک bean از bean دیگر

```java
@Bean
EnglishGreetingRepository englishGreetingRepository(){
    return new EnglishGreetingRepositoryImpl();
}

@Profile("EN")
@Bean
I18nEnglishGreetingService i18nService(EnglishGreetingRepository englishGreetingRepository){
    return new I18nEnglishGreetingService(englishGreetingRepository);
}
```

Scope

تنظیم **scope**: برای inject کردنهای یک dependency از هر bean؛ چه تعداد instance ساخته شود.

Declaring Bean Scope

- No declaration needed for singleton scope.
- In Java configuration use @Scope annotation
- In XML configuration scope is an XML attribute of the 'bean' tag
- 99% of the time singleton scope is fine!

Spring Bean Scopes

- **Singleton** - (default) Only one instance of the bean is created in the IoC container.
- **Prototype** - A new instance is created each time the bean is requested.
- **Request** - A single instance per http request. Only valid in the context of a web-aware Spring ApplicationContext.
- **Session** - A single instance per http session. Only valid in the context of a web-aware Spring ApplicationContext.

برای scope (تعداد instanceها) فرقی نمیکند از چه روشی injection (constructor یا @Autowired) را انجام داده باشیم.

ما معمولا service ها رو component در نظر میگیریم، و فقط یک instance داشتن از آن اکی هست (default: singleton)

آیا کار درستی هست entityها را هم با SpringIOC، inject کنیم؟؟؟

کلا تفاوت بین http req و Session ؟؟؟ در حالت های سوم به بعد instance از روی چی ساخته میشود؟؟؟

اگر برای یک component نوع scope را singleton تعریف کنیم، در initiation execute ، از هر component یک bean میسازد. و هر بار که بعنوان dependency درخواست شد همان inject میشود.

بقیه انواع در scope در initiation execute، bean ساخته نمیشود.

injection در Composition:

```
@Component
public class Laptop {
```

```
@Component
public class Alien {
    @Autowired
    private Laptop laptop;
```

****

اگر scope برای Alien، singleton باشد و برای Laptop، prototype باشد، و bean مربوط alien گرفته شود: یکبار موقع ساخته شدن context ، یک obj برای Alien ساخته شده که برای آن، یک بار Laptop ساخته شده، پس وقتی bean مربوط به Alien حتی اگر 100 بار گرفته شود همان یک obj اول ارایه میشود که برای آن، laptop هم فقط یک bean ساخته شده است. اگر جداگانه از Laptop بخواهیم bean بگیریم در آن زمان objهای جدید برای laptopخواهد ساخت.

اگر scope برای Alien، prototype باشد و برای Laptop، singleton باشد، و دوبار alien گرفته شود، فقط یکبار در زمان ایجاد context، bean مربوط به laptop ساخته شده است و برای هربار ساخته شدن alien جدید همان استفاده خواهد شد.

اگر هردو prototype باشند، برای هر بار خواسته شدن bean، یک obj جدید ساخته میشود از هر کدام.

اگر هردو singleton باشند، فقط یکبار در زمان ایجاد context، bean مربوط به هر دو ساخته شده است و برای هربار صدا زده شدن bean مربوط alien از همان objهای اولی استفاده خواهد شد.

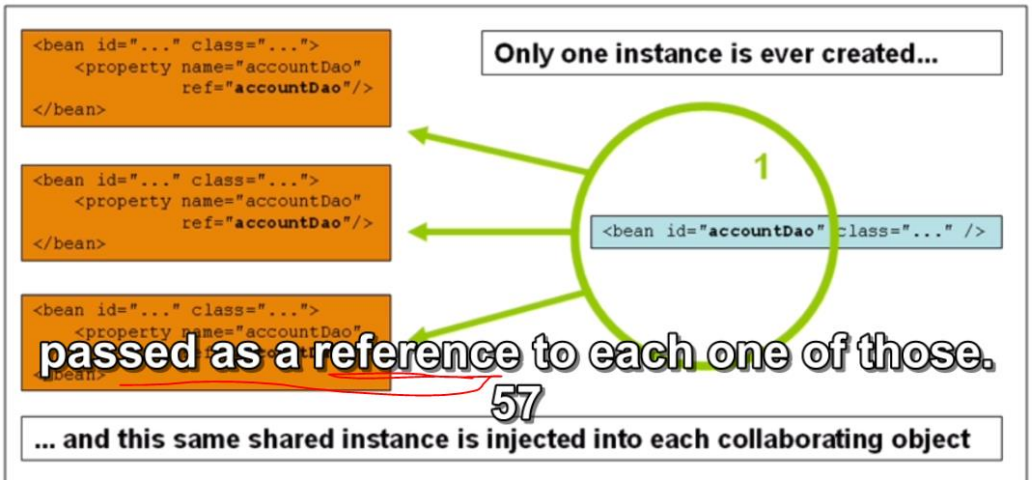
Spring Bean Scopes

- **Global-Session** - A single instance per global session. Typically only used in a Portlet context. Only valid in the context of a web-aware Spring ApplicationContext.
- **Application** - bean is scoped to the lifecycle of a ServletContext. Only valid in the context of a web aware.
- **Websocket** - Scopes a single bean definition to the lifecycle of a WebSocket. Only valid in the context of a web-aware Spring ApplicationContext.

Spring Bean Scopes

- **Custom Scope** - Spring Scopes are extensible, and you can define your own scope by implementing Spring's "Scope" interface.
- See Spring's Java docs for details.
- You cannot override the built in Singleton and Prototype Scopes

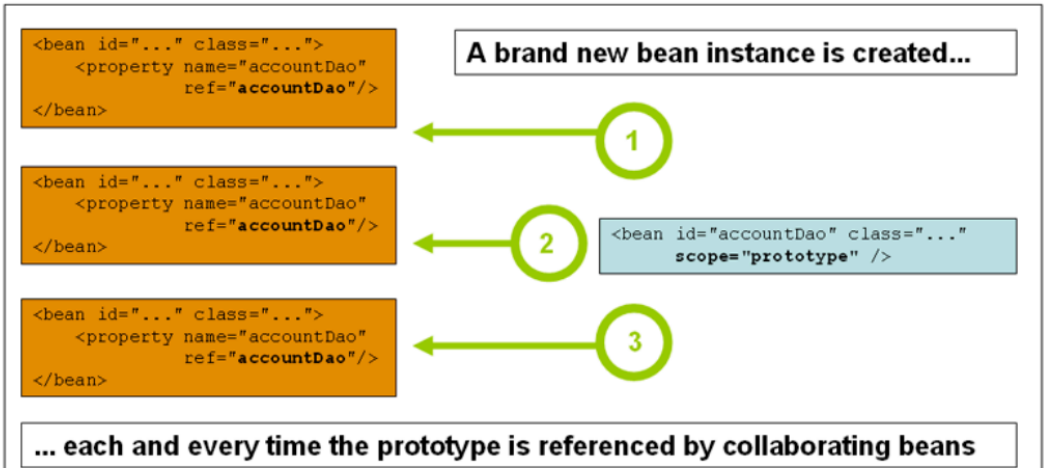
■ **singleton**

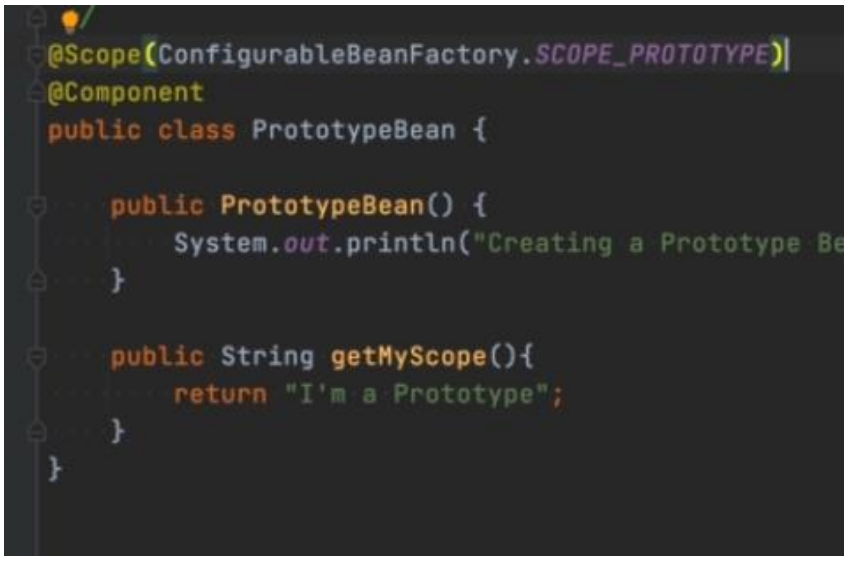



```
SingletonBean singletonBean1 = ctx.getBean(SingletonBean.class);
SingletonBean singletonBean2 = ctx.getBean(SingletonBean.class);
```

- **prototype**

Prototype Scope





1 and 2 : desktop

1- Singleton (default)

از هر bean ، یک Object داریم

2- Prototype

برای هر getBean() (درخواست injection حتی از سمت web servlet) ، یک Object جداگانه میسازد

3- Request

برای هر httpRequest یک bean میسازد

4- Session

برای هر httSession یک bean میسازد

5- Application context

برای هر spring Container Life cycle یک bean از هر نوع میسازد

6- WebSocket

برای هر web socket خاص یک bean از هر نوع میسازد

Properties

changes are easy by springboot apps.

- What can change?
  - usernames, passwords, urls, API keys, paths, queue names, etc

Ways:

Spring Framework: **Datasource.properties** - @Configuration - @Bean

Spring Framework with spring boot: **Datasource.properties** - @**ConfigurationPropeties**-@Bean

Spring Boot: **Application.properties or Application.yaml**

Spring boot: **Environment profiles** with .pro & .yaml

Environment vars

Vm option, Command line with run jar

هر دو روش آخر، به properties **اولویت** دارند

بین دوتای آخر، اولویت بالاتر با **CML** است :

با این روش میتوانیم با تغییر active profile در runtime ، obj را برای inject تغییر دهیم.

مزیت yaml:

در yml میتونیم اطلاعات profile رو داخل خود app.yaml بیاریم و یک فایل داشته باشیم

Env var for **sensitive** values

Setting External Properties

- How do you set external properties?
  - Command Line Arguments
  - SPRING_APPLICATION_JSON - JSON object via command line or environment variable
  - JNDI
  - OS Environment variables
  - Property files (property file or YAML variants)

Property Hierarchy

- Review Section 24 - Externalized Configuration of Spring Boot for specifics
- Properties can be overridden depending on how they are defined
  - Gives you a lot of flexibility for deployments
- Lowest are properties defined in JAR or war properties or YAML files
- Next are external property files to JAR via file system
- Higher are profile specific properties files (in jar, then external)
- OS Environment variables

- Java system properties
- JNDI
- SPRING_APPLICATION_JSON
- Command Line Arguments
- Test Properties (for testing)

John's Pragmatic Guide

- Favor using application.properties or application.yml in packaged JAR (or WAR)
- Use profile specific properties or YAML files for profile specific properties
- For deployments, override properties that change with environment variables
  - Typically 70-80% of values do not change, only override what is needed
  - Environment variables offer a secure way of setting sensitive values such as passwords

Spring Cloud Config

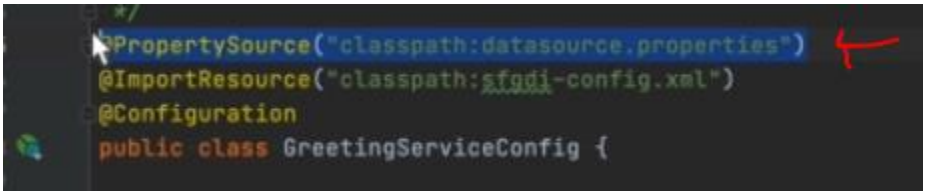
- Spring Cloud Config allows Spring applications to obtain configuration from a configuration server
- Application starts up and obtains configuration values from a configuration server
- Bootstrap process runs before normal Spring startup
- Spring Cloud Config is out of scope of this course
  - Covered in depth in my Spring Boot Microservices course

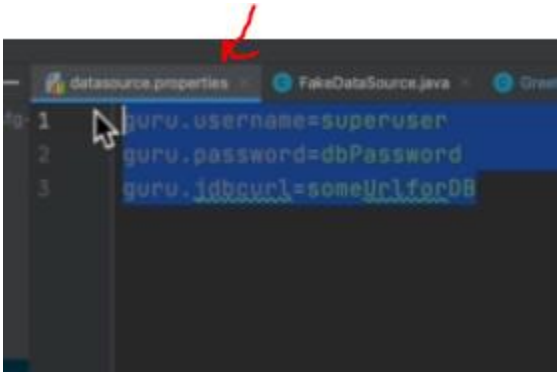
Env var for sensitive values

از اولویت پایین به بالا که میتواند override کند مطرح میکنیم:

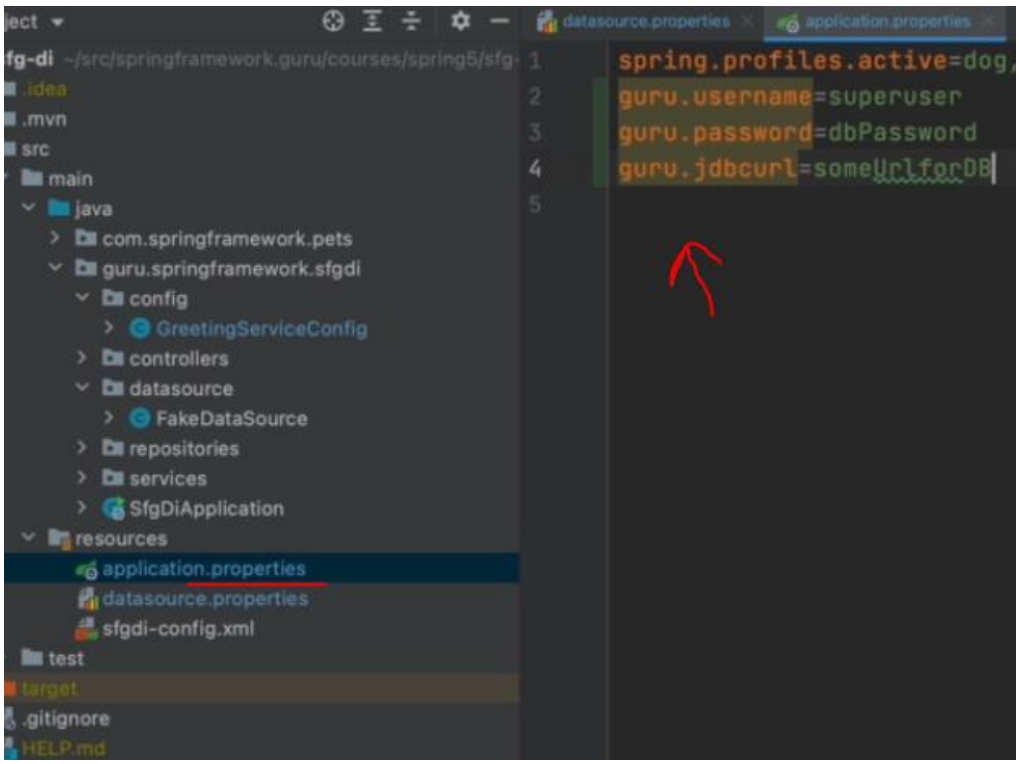
***Spring Boot: Application.properties or Application.yaml**

حذف نسبت به حالت بدون sp boot:

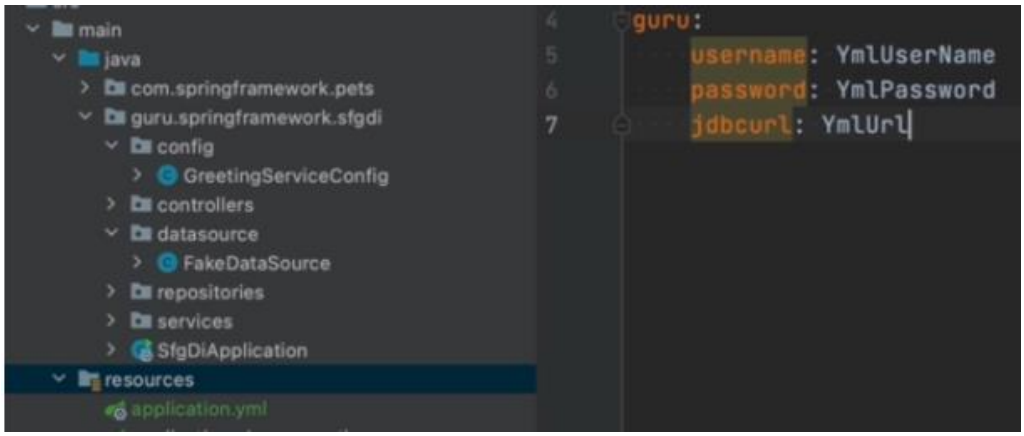




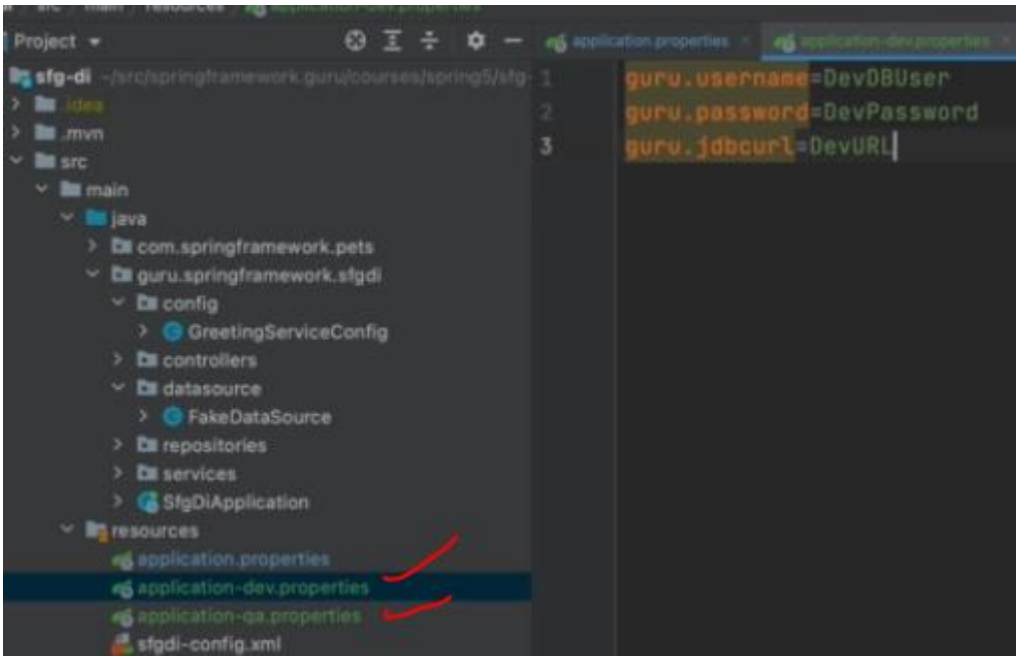
Now:

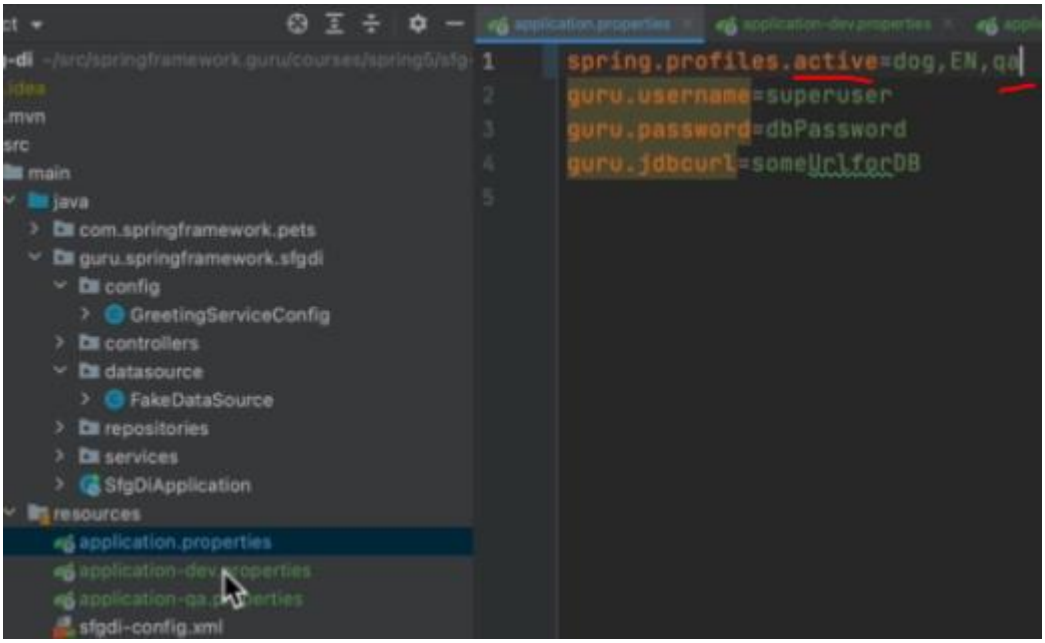


همین کارها با yaml:



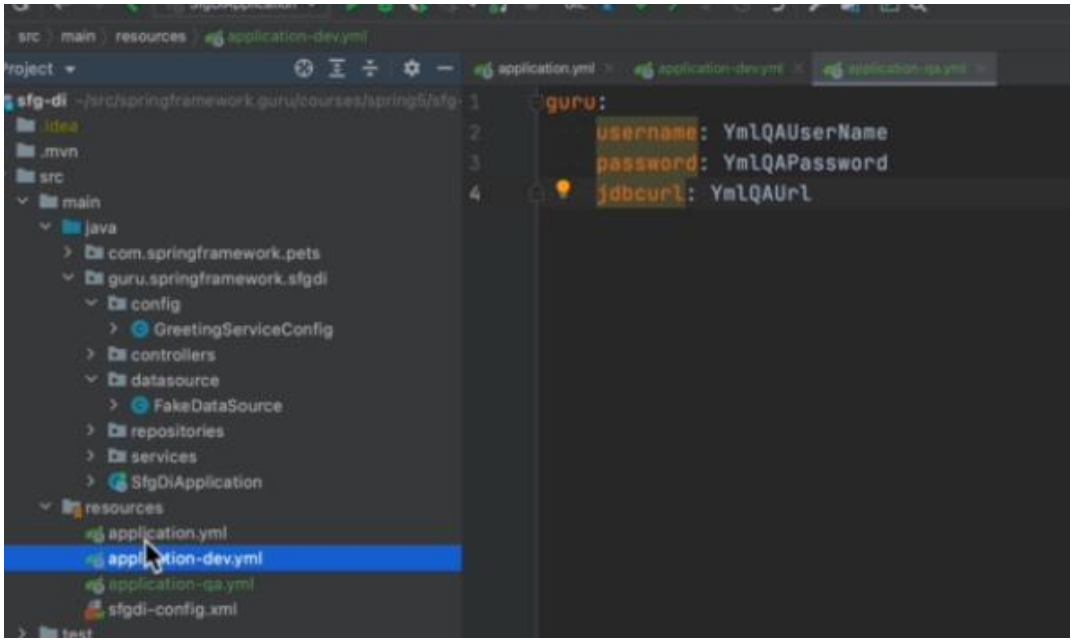
***Spring boot: Environment profiles with .pro & .yaml**

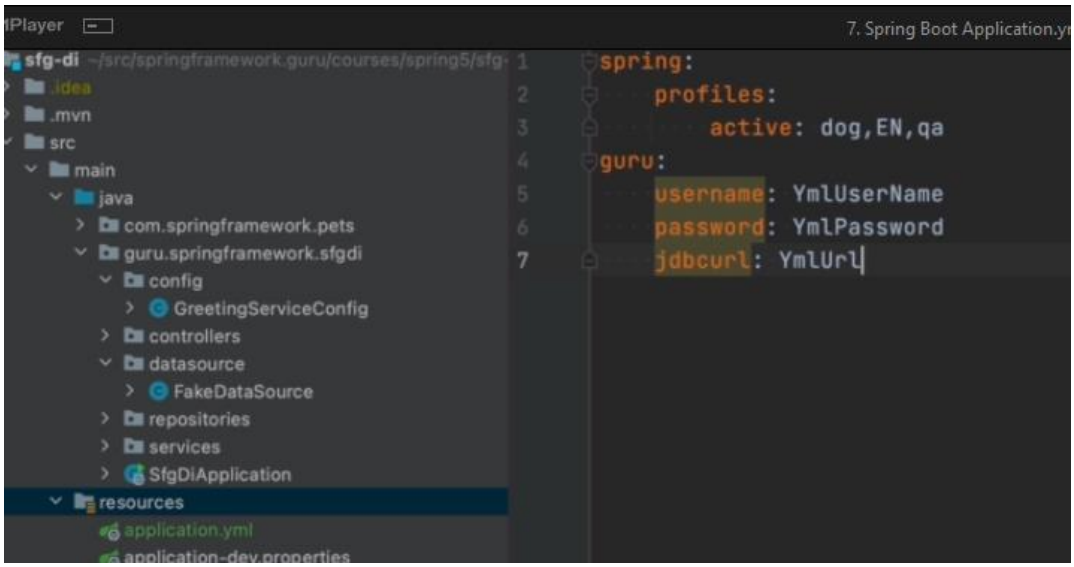




اگر تعیین نمیکردیم میشد حالت قبلی، از روی خود app.pro میخوند.

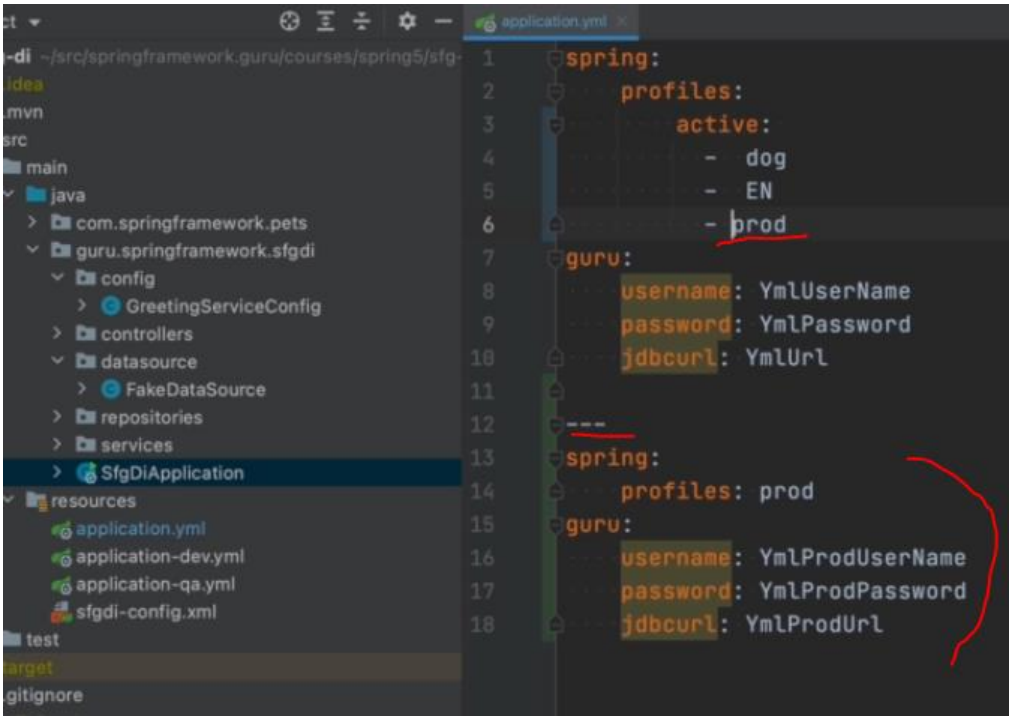
همین کارها با yaml:





مزیت yaml:

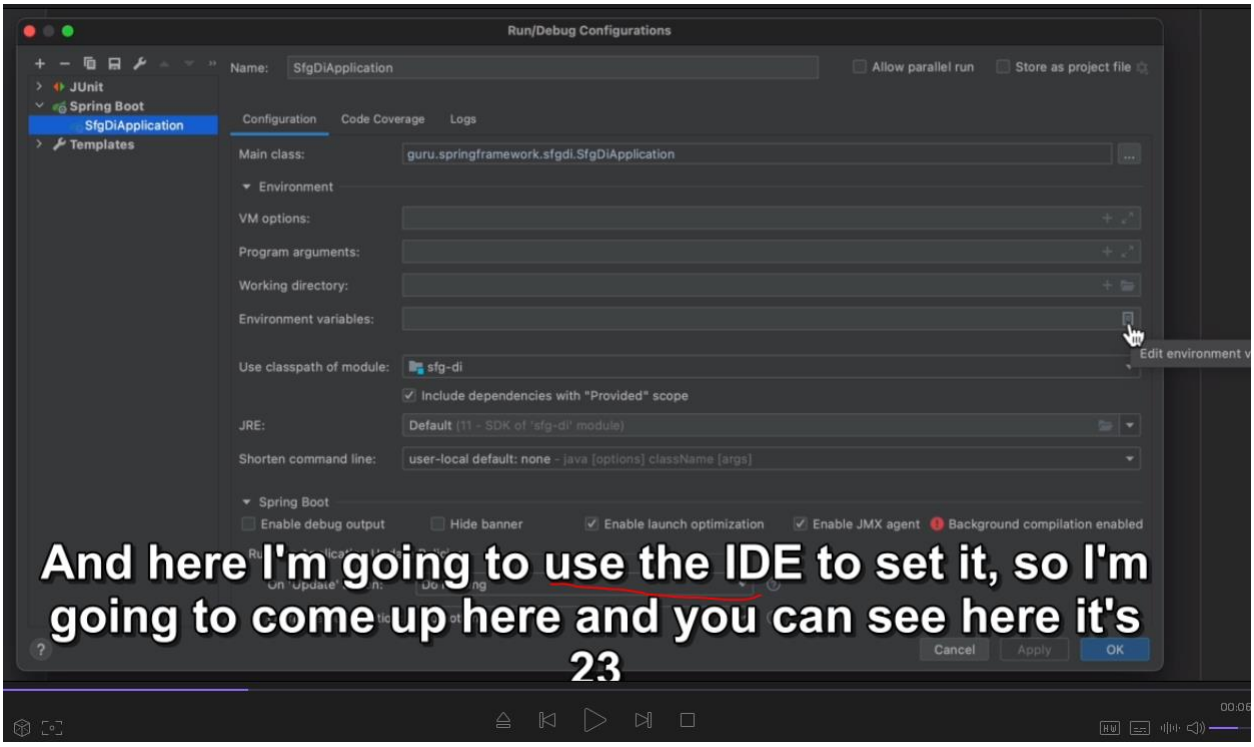
در yml میتونیم اطلاعات profile رو داخل خود app.yaml بیاریم و یک فایل داشته باشیم

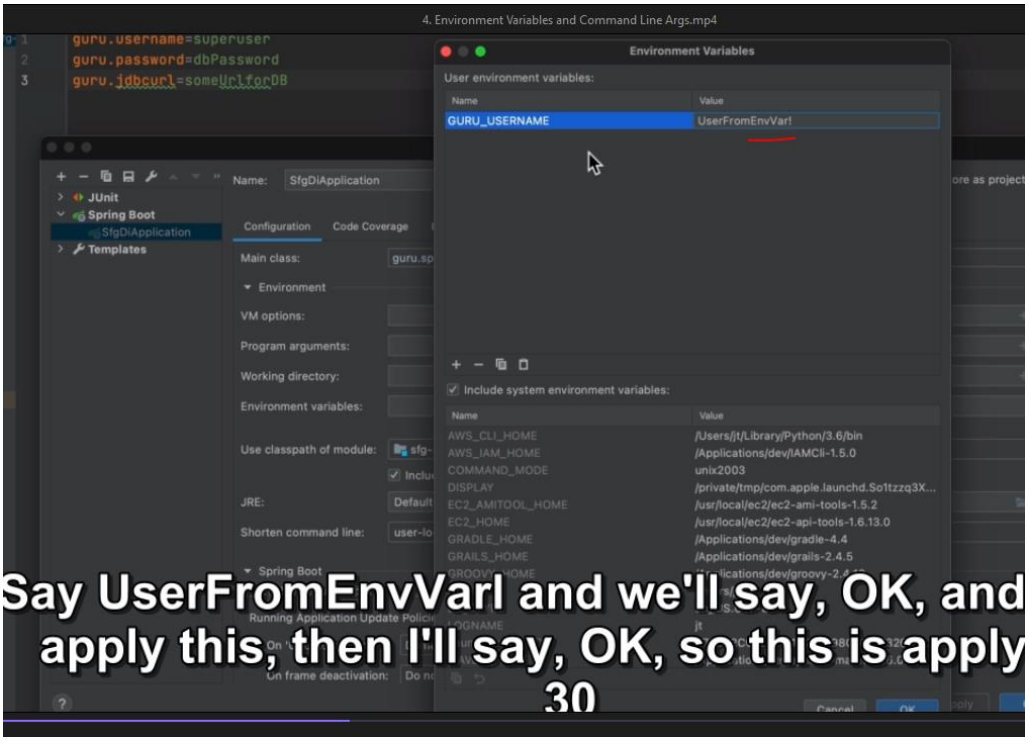


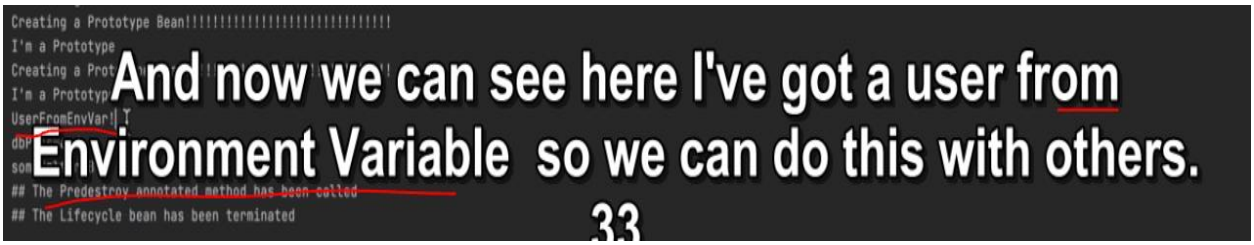
***Environment vars**

environment variables to override the properties.

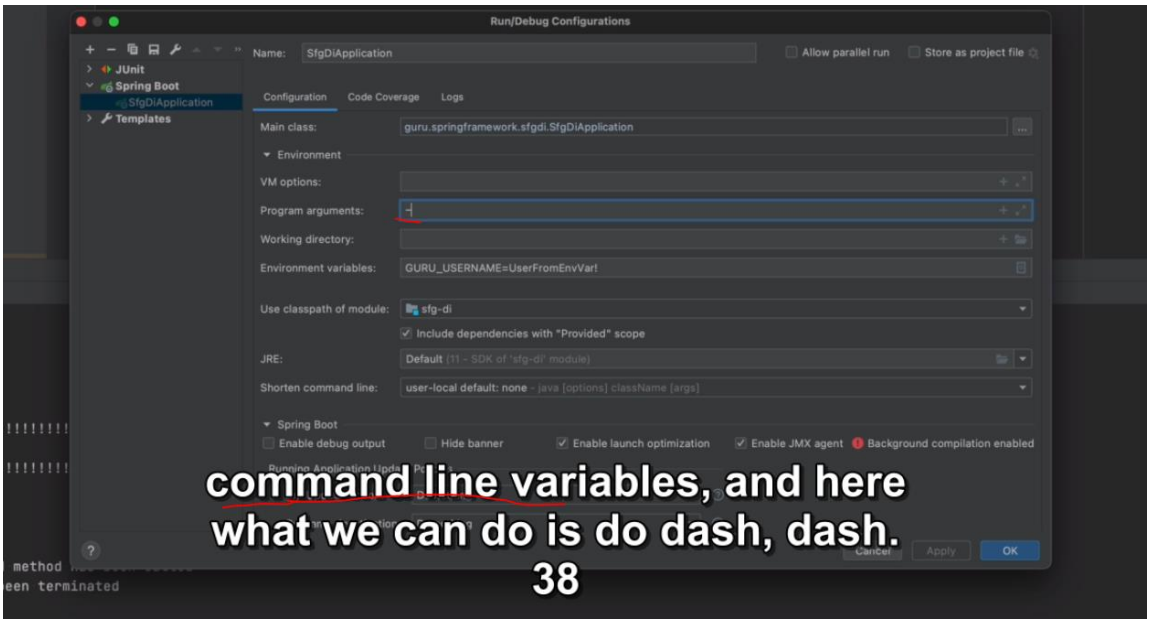
Env var for sensitive values


Run/Debug Configurations
Name: SfgDiApplication
Allow parallel run
Store as project file
JUnit
Spring Boot
SfgDiApplication
Templates
Configuration
Code Coverage
Logs
Main class: guru.springframework.sfgdi.SfgDiApplication
Environment
VM options:
Program arguments:
Working directory:
Environment variables:
Use classpath of module: sfg-di
Include dependencies with "Provided" scope
JRE: Default (11 - SDK of 'sfg-di' module)
Shorten command line: user-local default: none - java [options] className [args]
Spring Boot
Enable debug output
Hide banner
Enable launch optimization
Enable JMX agent
Background compilation enabled
Cancel
Apply
OK
And here I'm going to use the IDE to set it, so I'm going to come up here and you can see here it's 23


4. Environment Variables and Command Line Args.mp4
guru.username=superuser
guru.password=dbPassword
guru.jdbcurl=someUrlforDB
Environment Variables
User environment variables:
Name
Value
GURU_USERNAME
UserFromEnvVar!
Include system environment variables:
AWS_CLI_HOME
/Users/jt/Library/Python/3.6/bin
AWS_IAM_HOME
/Applications/dev/IAMCli-1.5.0
COMMAND_MODE
unix2003
DISPLAY
EC2_AMITOOL_HOME
/usr/local/ec2/ec2-ami-tools-1.5.2
EC2_HOME
/usr/local/ec2/ec2-api-tools-1.6.13.0
GRADLE_HOME
/Applications/dev/gradle-4.4
GRAILS_HOME
/Applications/dev/grails-2.4.5
Say UserFromEnvVar! and we'll say, OK, and apply this, then I'll say, OK, so this is apply 30

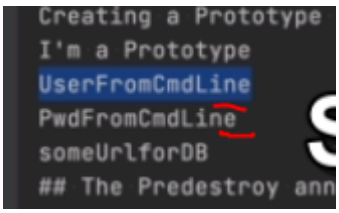


***VM Option, Command line with run jar**



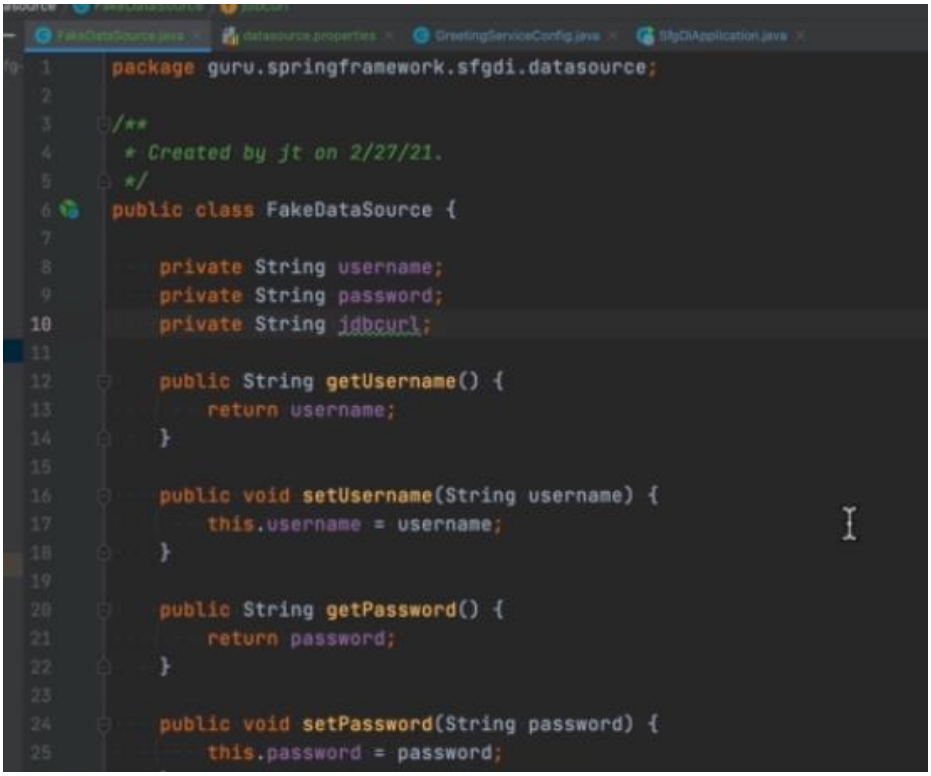
هر دو روش آخر، به properties اولویت دارند

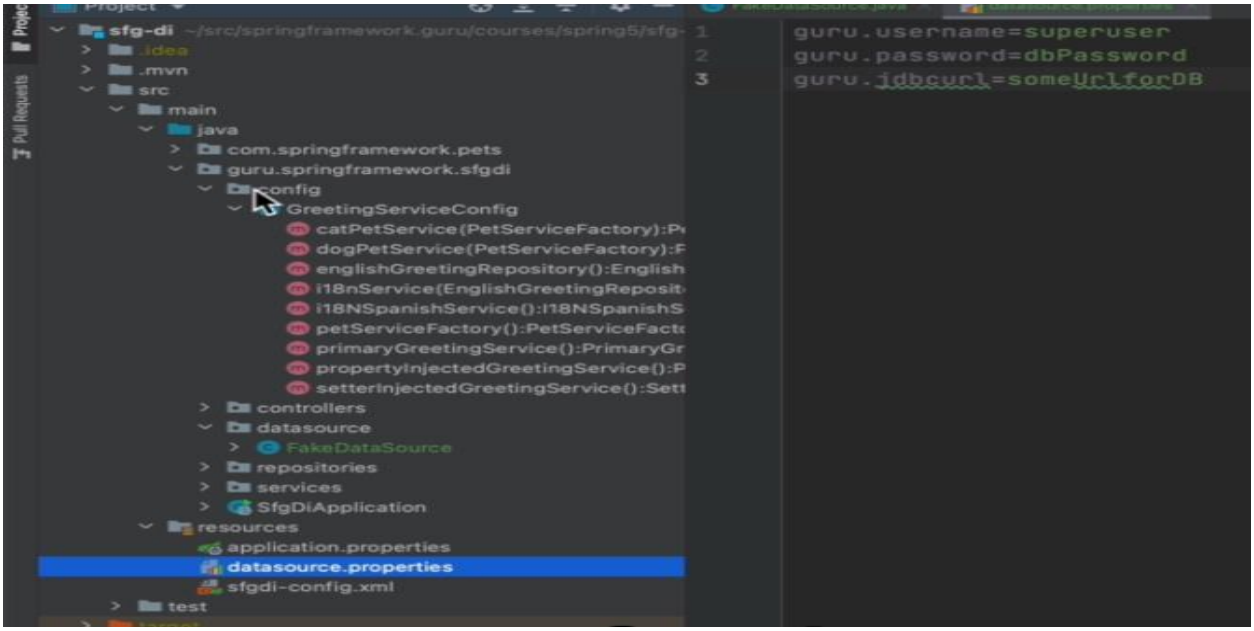
بین دوتای آخر، اولویت بالاتر با CML است :

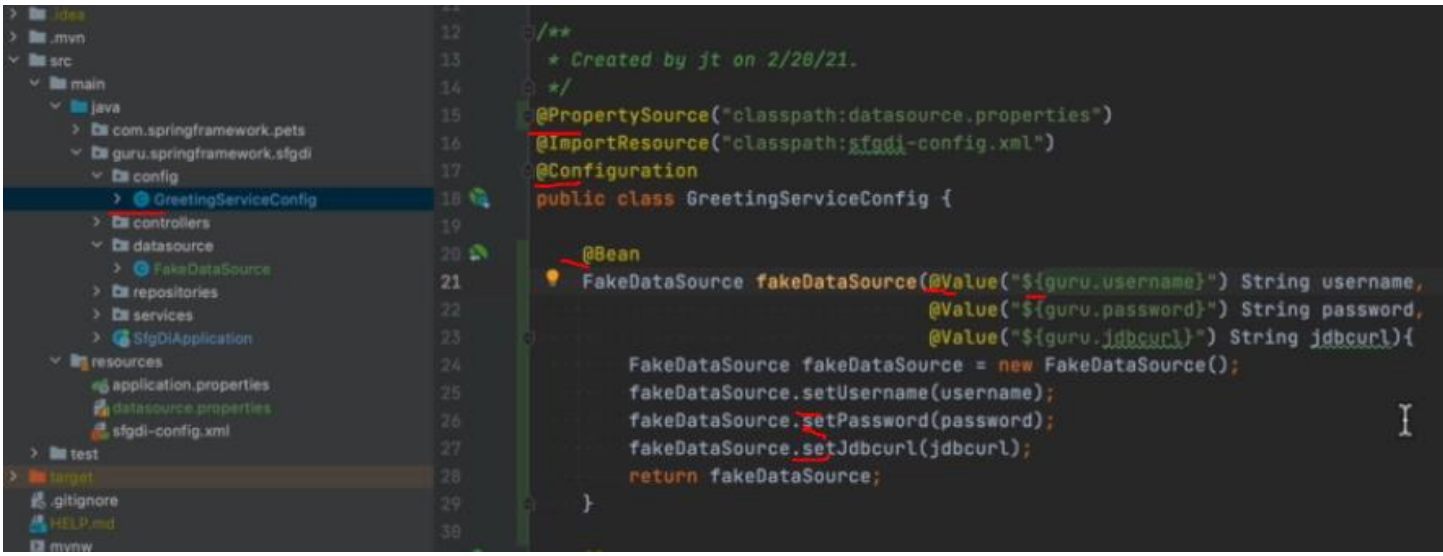


Spring Framework: Datasource.properties - @Configuration - @Bean

یکی از امکانات spring frameworkاست، الان بهترش رو همراه با spring boot داده.

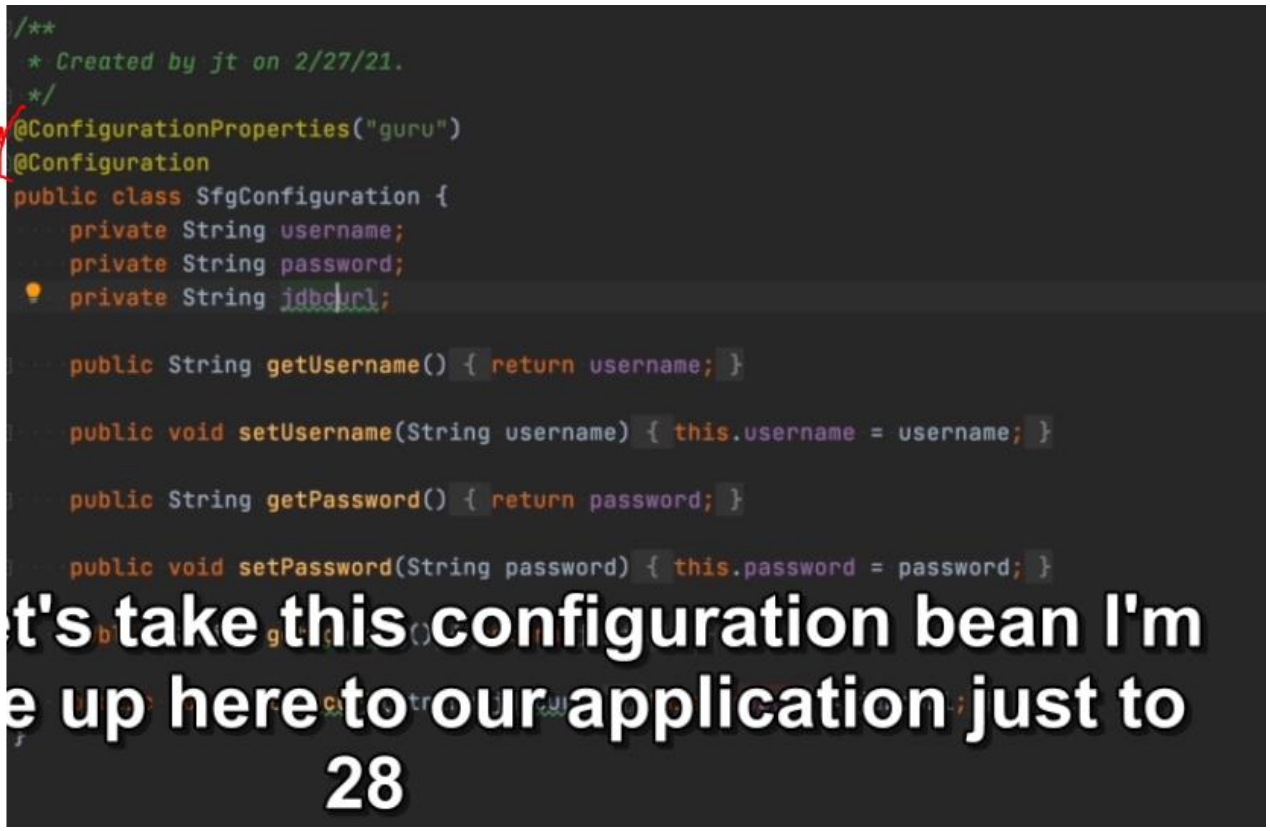

FakeDataSource.java
datasource.properties
GreetingServiceConfig.java
SfgDiApplication.java
package guru.springframework.sfgdi.datasource;
/**
* Created by jt on 2/27/21.
*/
public class FakeDataSource {
private String username;
private String password;
private String jdbcurl;
public String getUsername() {
return username;
}
public void setUsername(String username) {
this.username = username;
}
public String getPassword() {
return password;
}
public void setPassword(String password) {
this.password = password;


sfg-di ~/src/springframework.guru/courses/spring5/sfg-
.idea
.mvn
src
main
java
com.springframework.pets
guru.springframework.sfgdi
config
GreetingServiceConfig
catPetService(PetServiceFactory):P
dogPetService(PetServiceFactory):F
englishGreetingRepository():English
i18nService(EnglishGreetingReposit
i18NSpanishService():I18NSpanishS
petServiceFactory():PetServiceFact
primaryGreetingService():PrimaryGr
propertyInjectedGreetingService():P
setterInjectedGreetingService():Sett
controllers
datasource
FakeDataSource
repositories
services
SfgDiApplication
resources
application.properties
datasource.properties
sfgdi-config.xml
test
guru.username=superuser
guru.password=dbPassword
guru.jdbcurl=someUrlforDB

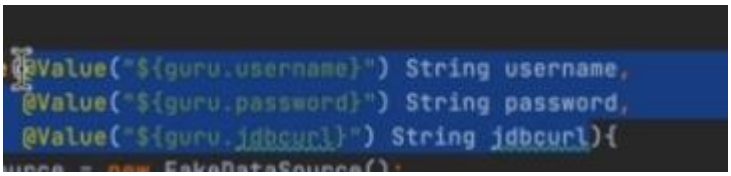


Spring Framework with spring boot: Datasource.properties - @ConfigurationPropeties - @Bean

خود POJO میشود Conf@ . دیگر نیازی به ساخت bean در @Configuration نیست.



خودش میره متغیرهای Guru رو از yaml فایل میخونه. دیگه به این setها نیاز نداره:



```java
@ImportResource("classpath:sfgdi-config.xml")
@Configuration
public class GreetingServiceConfig {

    @Bean
    FakeDataSource fakeDataSource(SfgConfiguration sfgConfiguration){
        FakeDataSource fakeDataSource = new FakeDataSource();
        fakeDataSource.setUsername(sfgConfiguration.getUsername());
        fakeDataSource.setPassword(sfgConfiguration.getPassword());
        fakeDataSource.setJdbcurl(sfgConfiguration.getJdbcurl());
        return fakeDataSource;
    }
```

میتونیم بهترش هم کنیم، immutable باشه:

```java
@ConstructorBinding
@ConfigurationProperties("guru")
public class SfgConstructorConfig {
    private final String username;
    private final String password;
    private final String jdbcurl;

    public SfgConstructorConfig(String username, String password, String jdbcurl) {
        this.username = username;
        this.password = password;
        this.jdbcurl = jdbcurl;
    }

    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public String getJdbcurl() {
        return jdbcurl;
    }
}
```

و برای اینکه در در بقیه beanها قابل استفاده باشه:


@EnableConfigurationProperties(SfgConstructorConfig.class)
@ImportResource("classpath:sfgdi-config.xml")
@Configuration
public class GreetingServiceConfig {
@Bean
FakeDataSource fakeDataSource(SfgConstructorConfig sfgConstructorConfig){
FakeDataSource fakeDataSource = new FakeDataSource();
fakeDataSource.setUsername(sfgConstructorConfig.getUsername());
fakeDataSource.setPassword(sfgConstructorConfig.getPassword());
fakeDataSource.setJdbcurl(sfgConstructorConfig.getJdbcurl());
return fakeDataSource;
}

Life Cycle

Spring Bean Life Cycle Demo .14 / Spring Bean Life Cycle .13

What two annotations can be used to access the Spring Bean lifecycle?	@PostConstruct and @PreDestroy
What are the two callback interfaces you can implement to tap into the bean lifecycle?	InitializingBean and DisposableBean

----------Spring MVC

Rest architect, we use it Over http protocol

RPC protocol, impl: soup, grpc, rmi

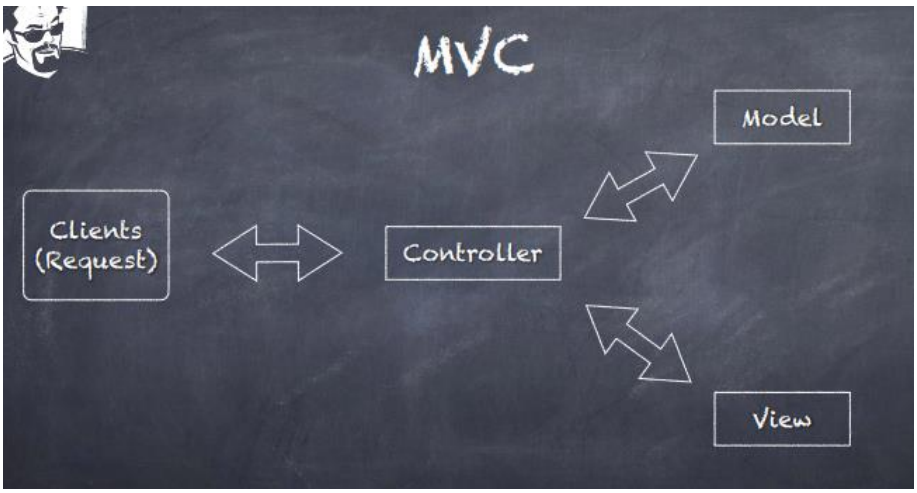
از مزیت های springboot web یا spring mvc

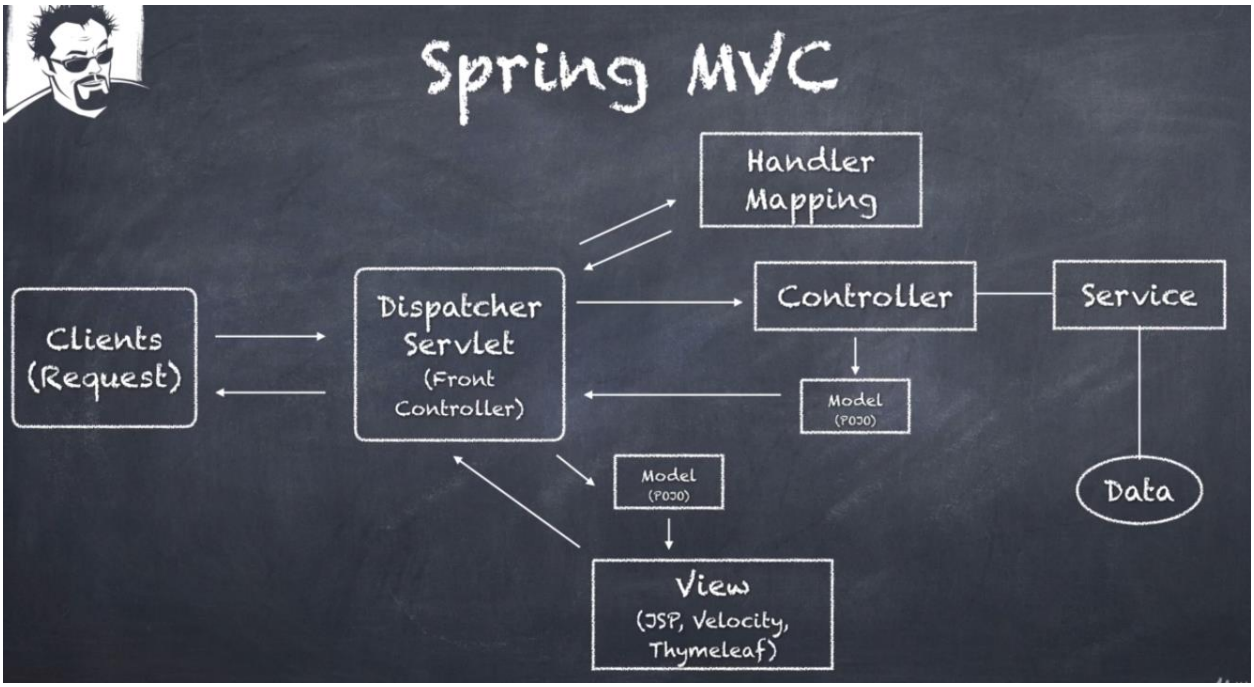
استفاده از servletMapping

استفاده از restful: قابیلت HATOSAS در rest باعث میشود کاملا داینامیک باشیم و چیزی در ui هارد کد نشده باشد.

استفاده از HTTP protocol (verbs)

jackson serialization





dispatcher servlet، درخواست inject از bean یک @controller را دارد.

(احتمالا handler mapping با @RequestMapping تعامل میکند.)

با توجه به نوع scope، یک instance از آن را تحویل میگیرد.

سپس دولوپر در controller مربوطه، طبق مواردی که در DI گفته شد با روش constructor ، یک instance از یک@bean (@Service @Repo) که از یک component ساخته شده و در context قرار گرفته را inject میکند.

Spring MVC: Develop applications using web (spring framework, RESTful, Rest)

SpringBoot Web: (<Starter-web>)

Spring MVC (spring framework, RESTful, Rest)

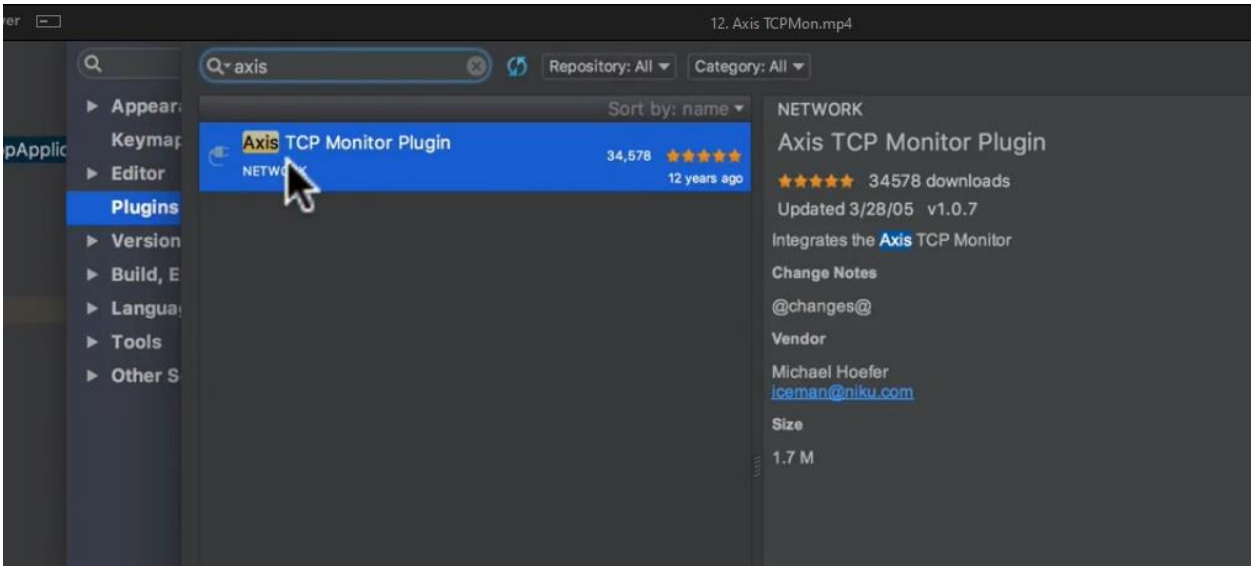
@**Controller** (type of a component)

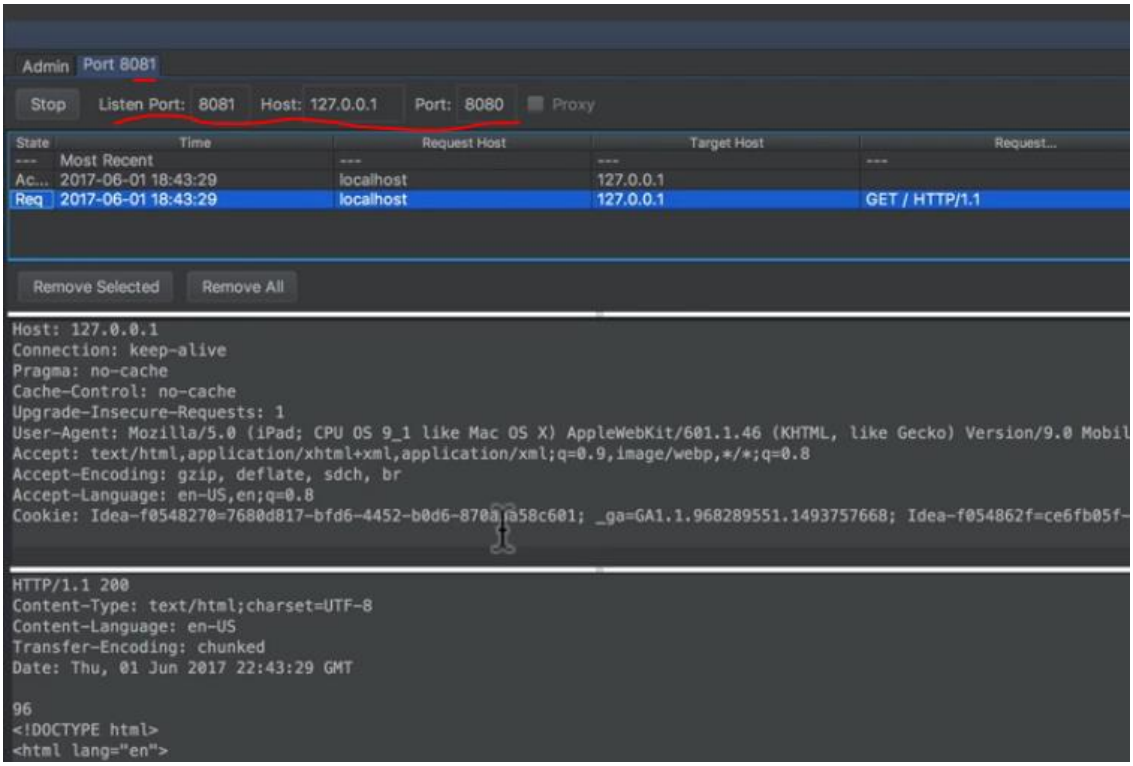
Initiation configurations & changes are easy (embedded Tomcat)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

HTTP tools

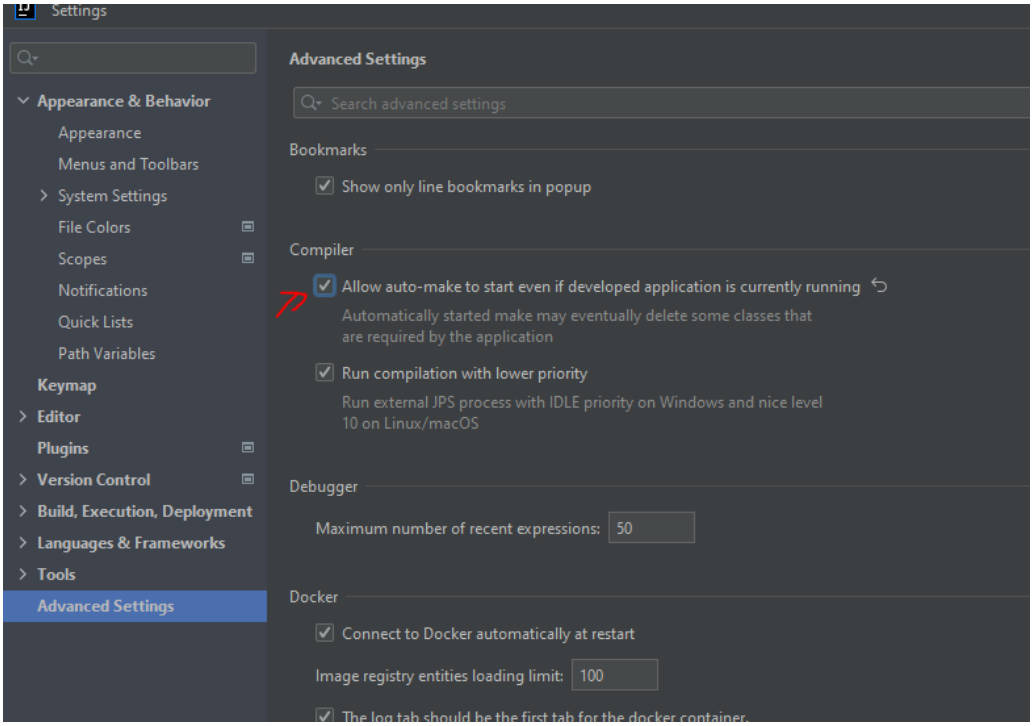
Axis monitor

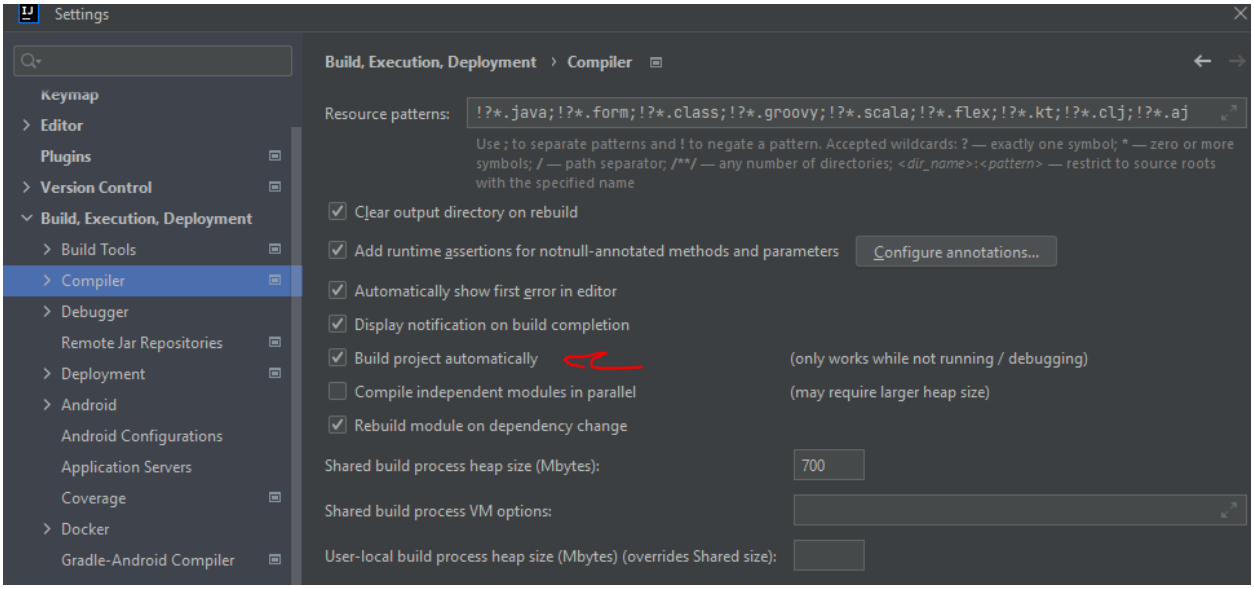




Development tools

اتوماتیک rebuildمیشه میره میشینه تو run


Settings
Advanced Settings
Search advanced settings
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Advanced Settings
Bookmarks
Show only line bookmarks in popup
Compiler
Allow auto-make to start even if developed application is currently running
Automatically started make may eventually delete some classes that are required by the application
Run compilation with lower priority
Run external JPS process with IDLE priority on Windows and nice level 10 on Linux/macOS
Debugger
Maximum number of recent expressions: 50
Docker
Connect to Docker automatically at restart
Image registry entities loading limit: 100


Settings
Build, Execution, Deployment › Compiler
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Build Tools
Compiler
Debugger
Remote Jar Repositories
Deployment
Android
Android Configurations
Application Servers
Coverage
Docker
Gradle-Android Compiler
Resource patterns: !?*.java;!?*.form;!?*.class;!?*.groovy;!?*.scala;!?*.flex;!?*.kt;!?*.clj;!?*.aj
Use ; to separate patterns and ! to negate a pattern. Accepted wildcards: ? — exactly one symbol; * — zero or more symbols; / — path separator; /**/ — any number of directories; <dir_name>:<pattern> — restrict to source roots with the specified name
Clear output directory on rebuild
Add runtime assertions for notnull-annotated methods and parameters
Configure annotations...
Automatically show first error in editor
Display notification on build completion
Build project automatically
(only works while not running / debugging)
Compile independent modules in parallel
(may require larger heap size)
Rebuild module on dependency change
Shared build process heap size (Mbytes): 700
Shared build process VM options:
User-local build process heap size (Mbytes) (overrides Shared size):

Request

HTTP Request

1- **Request Line**

HTTP **Method**

Request URI (**protocol**, **path**, **queries**, ...)

HTTP **Version**

2- **Request Headers**

Host, User agent, Accept, content type, accepted languages, cookies, etc

3- **Request Body (Optional)**

Typically, with methods like POST or PUT, such as form data, JSON, XML.

کلاینت:

بقیه قسمت های req توسط browser یا postman و .. ساخته میشود، این فقط Uri از req line است:

URI: http://localhost:8080/home

سرور:

```
@Controller
public class HomeController {

    @RequestMapping("home")
    public void home(){
        System.out.println("HomeController.home");
    }
}
```

It does sout.

9 request Methods

METHOD	Request Body	Response Body	Safe	Idempotent	Cachable
GET	No	Yes	Yes	Yes	Yes
HEAD	No	No	Yes	Yes	Yes
POST	Yes	Yes	No	No	Yes
PUT	Yes	Yes	No	Yes	No
DELETE	No	Yes	No	Yes	No
CONNECT	Yes	Yes	No	No	No
OPTIONS	Optional	Yes	Yes	Yes	No
TRACE	No	Yes	Yes	Yes	No
PATCH	Yes	Yes	No	No	Yes

- GET - is a request for a resource (html file, javascript file, image, etc)
- GET - is used when you visit a website.
- HEAD - is like GET, but only asks for meta information without the body.

- POST - is used to post data to the server.
- Typical use case for POST is to post form data to the server (like a checkout form)
- PUT - is a request for the enclosed entity be stored at the supplied URI. If the entity exists, it is expected to be updated.
- POST is a create request
- PUT is a create OR update request

- DELETE - Is a request to delete the specified resource
- TRACE - Will echo the received request. Can be used to see if request was altered by intermediate servers
- OPTIONS - Returns the HTTP methods supported by the server for the specified URL

TRACE: درخواست دریافت شده را تکرار می کند. می توان از آن برای مشاهده اینکه آیا درخواست توسط سرورهای میانی تغییر داده شده است استفاده کرد.

- CONNECT - Converts the request to a transparent TCP/IP tunnel, typically for HTTPS through an unencrypted HTTP proxy
- PATCH - Applies partial modifications to the specified resource

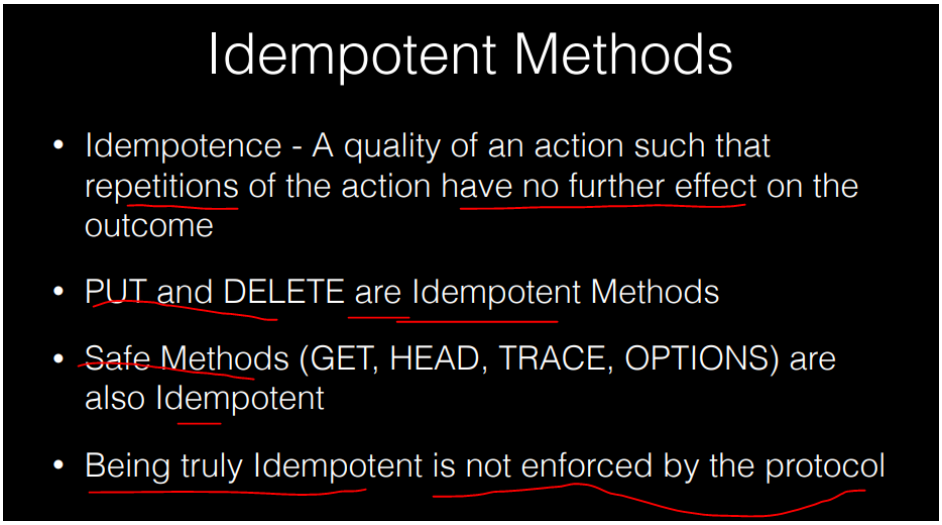
4 Safe methods

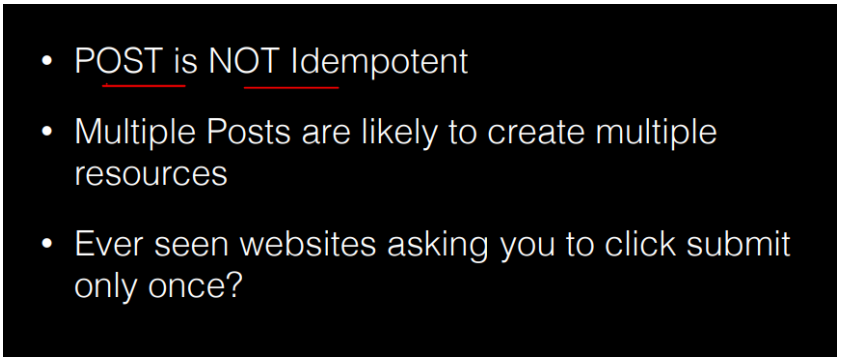
Safe Methods

- Safe Methods are considered safe to use because they only fetch information and do not cause changes on the server
- The Safe Methods are: GET, HEAD, OPTIONS, and TRACE

3 Non-Idempotent

Post / connect/ patch





متدهای سرور باید Idempotent نوشته شوند.

نوع **potentialمتدهای HTTP** و **<u>شیوه کد زدن ما</u>** در نوشتن سرویس تاثیر گذار است. **<u>یعنی</u>** میتونیم post را طوری بنویسیم که idem باشد و put و Del را طوری بنویسیم که idem نباشند.

Idempotent:

با فاخوانی مکرر یک متد ، نتایج و effectهای **یکسان** بگیریم . دکمه های آسانسور Idempotent هستند

اگر DELETE را اینطوری بنویسیم دیگر Idempotent نیست:

```
DELETE /item/last
```

در مورد بالا، فراخوانی عملیات N بار N منبع را حذف می‌کند - بنابراین DELETE در این مورد Idempotent نوشته نشده .

راه حل :در این مورد، یک پیشنهاد خوب ممکن است تغییر API فوق به POST باشد – زیر POST ، Idempotent نیست.

Non-Idempotent: make same results multiple times in server

با فاخوانی مکرر یک متد ، نتایج و effectهای **جدید** بگیریم

Post / connect/ patch ، Idempotent نیستند

وقتی یک درخواست POST را N بار فراخوانی می کنیم، Nمنبع جدید خواهیم داشت. بنابراین، side effect ایجاد میکند

روش یک :

if they consistently produce the same result when called with the **same data ID**

روش دو:

پاسخ‌های POST قابل کش هستند: زمانی که server **هدرهای Cache-Control و Expires** را تنظیم کند، چون معمولا درخواستهای تکرای مشکل ساز، پشت سر هم در فاصله زمانی کوتاه صادر میشوند.

روش سه:

زمانی که server **هدر ETags** را تنظیم کند،

Using ETags

Request

```
GET /users/123
```

Response

```
HTTP/1.1 200 Ok
ETag: "a915ecb02a9136f8cfc0c2c5b2129c4b"

{
    "name": "John Doe",
    "email": "john@doe.com"
}
```

ETags **are generated by the server** based on the current resource representation

```
PATCH /users/123
If-Match: "a915ecb02a9136f8cfc0c2c5b2129c4b"

{
    "name": "John Smith"
}
```

We use the *If-Match* condition to tell the server only to execute the request if the ETag matches.

Updating the resource leads to an updated ETag on the server side

So, if the request is accidentally sent **twice**, the server **rejects** the second request **because the ETag no longer matches**

Obviously ETags can only be used if the resource **already exists**

روش چهار:

Using a separate idempotency key

the **client generates a key** and adds it to the request using a custom header (e.g. *Idempotency-Key*).

```
POST /users
Idempotency-Key: 1063ef6e-267b-48fc-b874-dcf1e861a49d

{
    "name": "John Doe",
    "email": "john@doe.com"
}
```

Now the server can persist the idempotency key and reject any further requests using the same key.

two questions:

requests that have **not been completed** successfully, Should the idempotency key be saved by the server in these cases?

What to return **if the server retrieves** a request with an already known idempotency key.

Personally I tend to save the idempotency key only if the **request finished successfully**

In the second case I would return **HTTP 409** (conflict) to indicate that a request with the given idempotency key has already been executed.

https://www.javacodegeeks.com/2021/06/making-post-and-patch-requests-idempotent.html

------- Data in the URI Request

1- Query Parameters

http://example.com/resource?param1=value1&param2=value2.

2- Path Parameters:

http://example.com/resource/{id}.

typically used in RESTful APIs

HttpServletRequest / .getParameter

http://localhost:8080/home**?name=Mehrad**

```
@Controller
public class HomeController{

    @RequestMapping("home")
    public String home(HttpServletRequest httpServletRequest){
        String name= httpServletRequest.getParameter("name");
        System.out.println("hi "+ name);
        return "home";
    }
}
```

@RequestParam

http://localhost:8080/home**?name=Mehrad**

```
@Controller
public class HomeController {
    @RequestMapping("home")
```

```
    public String home(@RequestParam("param1") String name){
        System.out.println("hi "+ name);
        return "home";
    }
}
```

***@PathVariable/ get & del

http://localhost:8080/alien/**111**

```
@RequestMapping("alien/{aid}")
@ResponseBody
public Optional<Alien> getOneAlien (@PathVariable ("aid") int
aidx){
    return alienRepository.findById(aidx);
}
```

تا اینجا ما دیتا را از url میگرفتیم.

------- Data in the Body Request

1- Form Data: as key-value pairs within the body

2- JSON

3- XML

4- Binary Data: such as images, files

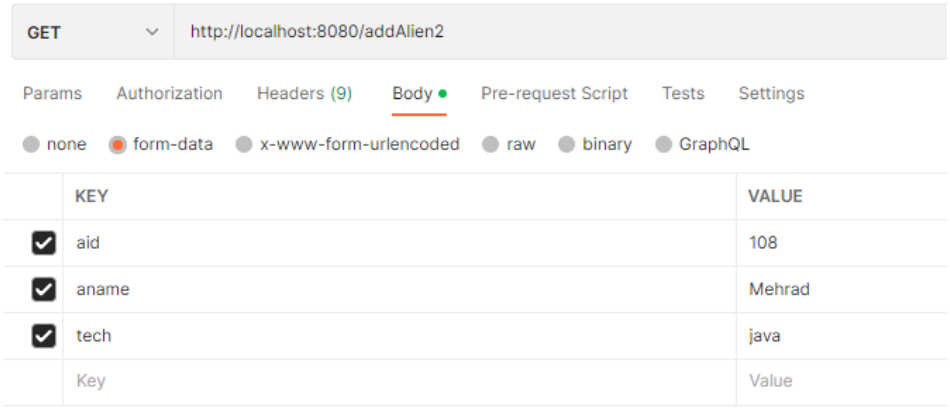
روش های استخراج دیتا از HTTP Req **Body**:

***Entity /** form-data

دیتای مورد نیاز سرویس بصورت form-data، در body ی req می آید.

http://localhost:8080/alien**?aid=100&aname=Sareh&tech=java**

```
    @RequestMapping("home")
    public ModelAndView home(Alien alien){
        System.out.println("hi "+ alien);
        return "home";
    }
```



ModelAttribute / data binding to a web view

`@ModelAttribute("employee") Employee employee` در پارامتر req: آمده از **دیتا های form-data**

ارسال داده به view: **Bindig** بین **ModelMap** با (`employee`)bean

we can use @ModelAttribute either as a **method parameter** or at the **method level**

@ModelAttribute is an annotation that binds a **method parameter** or **method return value** to a named model attribute, and then **exposes it to a web view**.

- **method level:**

add one or more global model attributes

Such methods support the same argument types as @RequestMapping methods, but they can't be **mapped directly to requests**.

a method that adds an attribute named msg to all models defined in the controller class.

addAttributes () method. Its purpose is to add values in the Model that'll be identified globally. That is, every request to every controller method will return a default value as a response.

addAttributes () method will be the very first to run, prior to the rest of the @RequestMapping methods.

@ModelAttribute methods are invoked before the controller methods annotated with @RequestMapping are invoked

- **As a Method Argument**

It binds the form data with a bean

In **Spring MVC**, we refer to this as **data binding**, a common mechanism that saves us from having to parse each form field individually

The controller annotated with @RequestMapping can have custom class argument(s) annotated with @ModelAttribute.

It's also important that we annotate the respective class as @ControllerAdvice.

```
<form:form method="POST" action="/spring-mvc-basics/addEmployee"
  modelAttribute="employee">
    <form:label path="name">Name</form:label>
    <form:input path="name" />

    <form:label path="id">Id</form:label>
    <form:input path="id" />

    <input type="submit" value="Submit" />
</form:form>
```

```
@Controller
@ControllerAdvice
public class EmployeeController {

    private Map<Long, Employee> employeeMap = new HashMap<>();

    @RequestMapping(value = "/addEmployee", method = RequestMethod.POST)
    public String submit(
      @ModelAttribute("employee") Employee employee,
      BindingResult result, ModelMap model) {
        if (result.hasErrors()) {
            return "error";
        }
        model.addAttribute("name", employee.getName());
        model.addAttribute("id", employee.getId());

        employeeMap.put(employee.getId(), employee);

        return "employeeView";
    }

    @ModelAttribute
    public void addAttributes(Model model) {
        model.addAttribute("msg", "Welcome to the Netherlands!");
    }
}
```

```
<h3>${msg}</h3>
Name : ${name}
ID : ${id}
```

After the submission is complete, and without any errors, the user expects to see the previously submitted data displayed on another screen.

```
@XmlRootElement
public class Employee {

    private long id;
    private String name;

    public Employee(long id, String name) {
        this.id = id;
        this.name = name;
    }

    // standard getters and setters removed
}
```

https://www.baeldung.com/spring-mvc-and-the-modelattribute-annotation#:~:text=%40ModelAttribute%20is%20an%20annotation%20that,submitted%20from%20a%20company's%20employee.

***Restful / @RequestBody

Restful: وقتی داده ها در req و res بصورت **row-data** در body تبادل میشوند. نوعی serialization در spring Web.

در **rest** ، model در req و res جابجا نمیشود، row-type جابجا میشود. دیتای مورد نیاز سرویس بصورت row-data (json, xml)، در body ی req می آید و res هم بصورت row-data در Res-body داده میشود.

عملیات read(**deserialize**) برای @RequestBody و write(**serialize**) در @ResponseBody

در دل **Spring MVC** بصورت **automatic** انجام میشود.

In rest-full SpringBoot uses a Jackson.***ObjectMapper*** instance to serialize **responses** and deserialize **requests**.

انوتیشن های **@RequestBody** و **@ResponseBody** ؛ rest را در Spring-webمهیا میکنند.

the **Type** we annotate with the ***@RequestBody*** annotation **must correspond** to the **JSON** sent from our client-side controller

```
public Alien addAAlien (@RequestBody Alien alien){ ..}
```

طبق تنظیمات دیفالت Jackson در spring boot برای reserializing؛ اگر برای فیلدی دیتا نیامد خطا نمیخورد.

Request در فیلدهای **header** خود اعلام میکند:

فیلد **content-type** میگه من به چه فرمتی دارم در req داده میدم. (دیفالت Consume: json)

فیلد **Accept** میگه هنگام فرستادن response به من، با این فرمت **produce** کن. (دیفالت Consume: json)

Service در فیلدهای **@RequestMapping** اعلام میکند:

فیلد **consume** میگه سرویس داره به چه فرمتی از req داده میگیره. (دیفالت Consume: json)

فیلد **Produces** میگه سرویس داره به چه فرمتی در res داده produce میکند. (دیفالت Consume: json)

Response در فیلد **header** خود اعلام میکند:

فیلد **content-type** میگه من به چه فرمتی دارم در res داده میدم. (دیفالت Consume: json)

دیفالت SpringBoot-web برای row-type، **json** است

مثال:

```
@PostMapping("/alien")
@ResponseBody
public Alien addAAlien(@RequestBody Alien alien){
    alienRepository.save(alien);
    return alien;
}

      = (because json is default)


@PostMapping("/alien", consume={"application/json"})
@ResponseBody
public Alien addAAlien(@RequestBody Alien alien){
    alienRepository.save(alien);
    return alien;
}
```

در @RequestMapping فیلد **consume** میگه سرویس داره بصورت json داده میگیره.

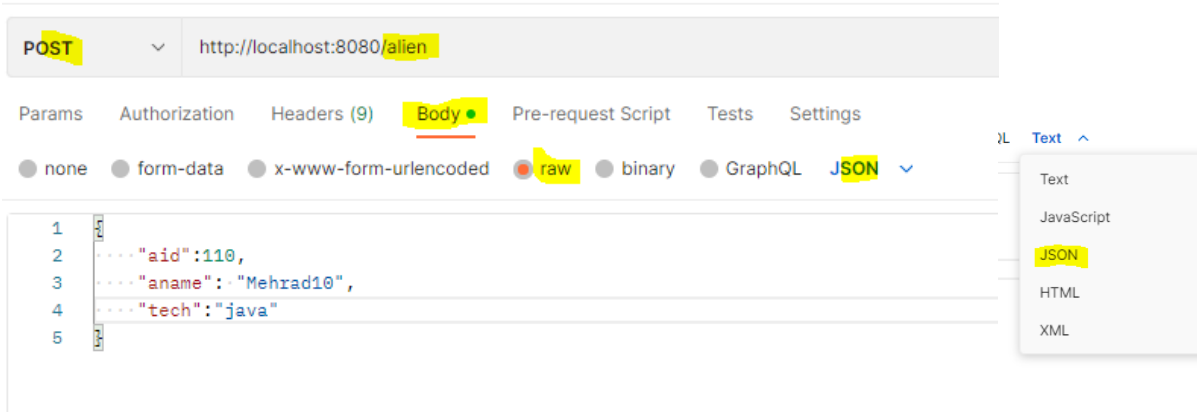
در @RequestMapping فیلد **Produces** میگه سرویس داره در حالت default بصورت json داده produce میکند.

در HTTP request Header فیلد **content-type** داره میگه من به چه فرمتی دارم داده میدم.

در HTTP request Header فیلد **Accept** داره میگه هنگام فرستادن response به من، با این فرمت **produce** کن.

```
curl -i \
-H "Accept: application/json" \
-H "Content-Type:application/json" \
-X POST --data
  '{"username": "johnny", "password": "password"}' "https://localhost:8080/.../request"
```

پس request میره داخل سرویسی که این دو شرایط را مچ داشته باشد.



XML:

(consume کردن xml در spring-web پیچیدگی دارد) ؟؟؟

https://stackoverflow.com/questions/70138672/how-to-force-spring-boots-requestbody-to-accept-xml

خواندن دیتا از header یک request ؟؟؟

@RequestMapping - methods

dispatcher servlet، درخواست inject از bean یک @controller را دارد.

(احتمالا handler mapping با @RequestMapping تعامل میکند.)

```
@RequestMapping({"", "/", "index", "index.html"})
```

@RequestMapping روی کلاس سرویس کنترلر یا سرویس متد:

```java
@Controller
public class OwnerController {

    @RequestMapping({"/owners", "/owners/index", "/owners/index.html"})
    public String listOwners(){
        return "owners/index";
    }
}
```

```java
*/
@RequestMapping("/owners")
@Controller
public class OwnerController {

    @RequestMapping({"", "/", "/index", "/index.html"})
    public String listOwners(){
        return "owners/index";
    }
}
```

تفاوتی که متدهای Del و get و .. ایجاد میکنند، میبینیم که آدرس برای get و del یکیه ولی متدی که توش میفته متفاوت میشه.

https://www.youtube.com/watch?v=50P3eGwGnPo

Shortcuts

Long Way	Short Way
@RequestMapping(method = RequestMethod.GET)	@GetMapping
@RequestMapping(method = RequestMethod.POST)	@PostMapping
@RequestMapping(method = RequestMethod.PUT)	@PutMapping
@RequestMapping(method = RequestMethod.DELETE)	@DeleteMapping
@RequestMapping(method = RequestMethod.PATCH)	@PatchMapping

تفاوت نسبت به حالتی که @RequestMapping را استفاده میکردیم و متد رو مثل parameter مشخص میکردیم:

Just for **readability**

@PostMapping

http Post method/ adding a object

```
@PostMapping("/alien")
@ResponseBody // it returns json
public Alien addAlien(Alien alien){ // it gets FormData
    alienRepository.save(alien);
    return alien;
}
```

داریم rest کار میکنیم (@ReponseBody) (همین مثال رو برای بحث Jpa repos داشتیم با restful)

@DeleteMapping

```
@DeleteMapping("/alien/{aid}")
public Alien deleteAlien(@PathVariable int aid){
    Alien alien= alienRepository.findById(aid).orElse(null);
    alienRepository.delete(alien);
    return alien; ???
}
```

این روش درست نیست:

Entity must not be null!

روش درست:

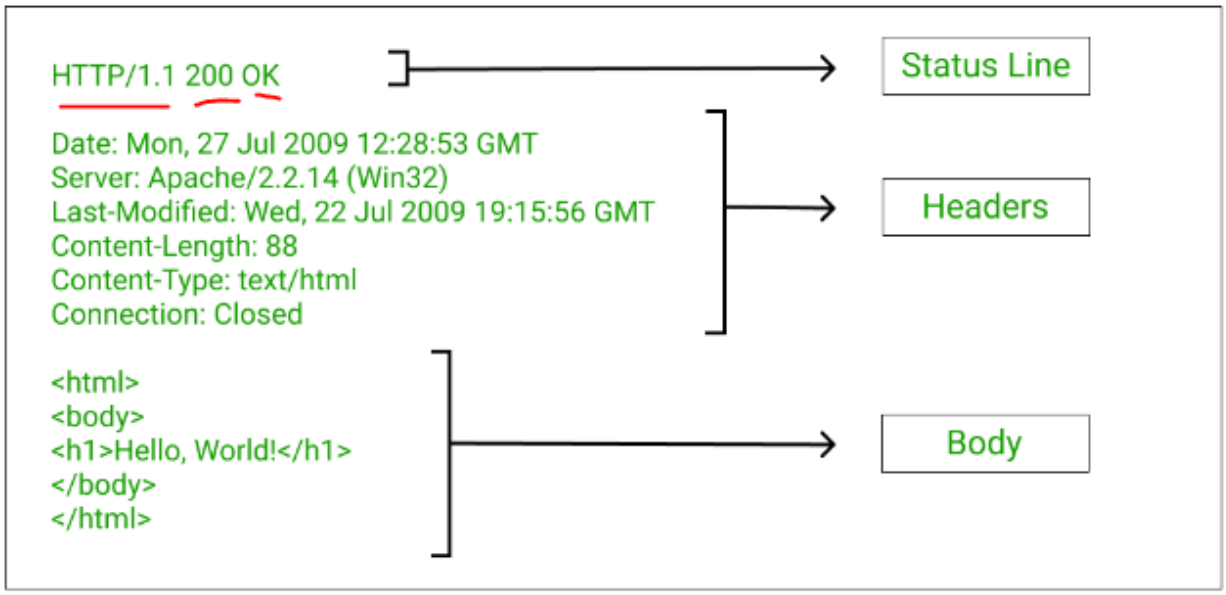
```
@DeleteMapping("/alien/{aid}")
public String deleteAlien(@PathVariable int aid){
    Alien alien= alienRepository.findById(aid).orElse(null);
    alienRepository.delete(alien);
    return "deleted";
}
```

@PutMapping

```
@PutMapping(value = "/alien", consumes = {"application/json"})
@ResponseBody  // it returns json
public Alien addOrUpdateAlien(@RequestBody Alien alien){ // it
gets json
    alienRepository.save(alien);
    return alien;
}
```

Response

HTTP Response



1- **Status Line**

   HTTP **Version**

   Status **Code**

   Status **Message**

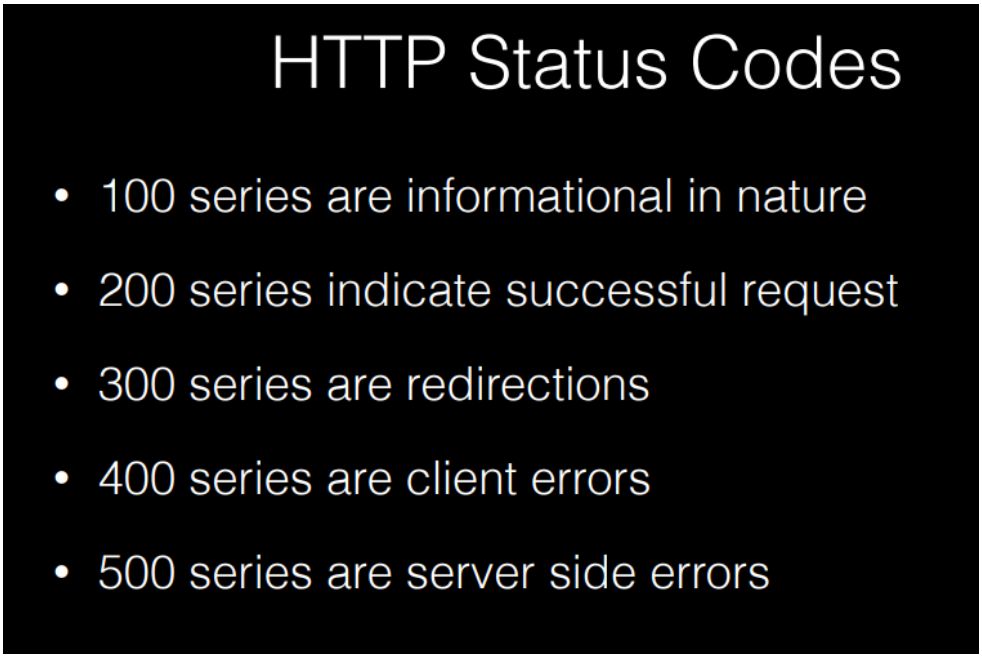
2- Response Headers

   content type & length, caching directives, server information, cookies

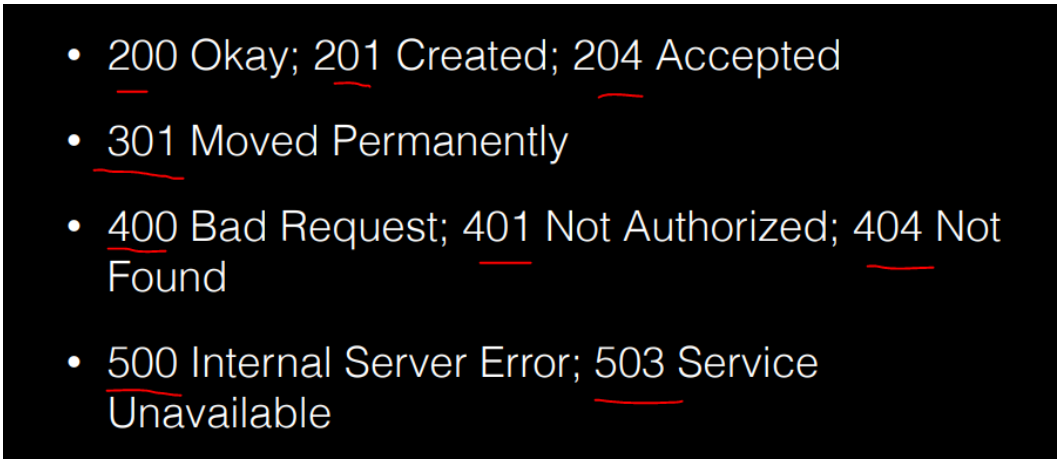
3- Response Body

   HTML, JSON, XML, images, or any other data

http status codes



***



روش های ارسال data به سمت client در Response

*HttpSession of httpServletRequest

دریافت و ارسال data روی session

```
@Controller
public class HomeController {

    @RequestMapping("home")
    public String home(HttpServletRequest httpServletRequest){
        HttpSession session= httpServletRequest.getSession();
        String name= httpServletRequest.getParameter("name");
        System.out.println("hi "+ name);
        session.setAttribute("name",name);
        return "home";
    }
}
```

```
<body>
        I'm ${name} / I'm into learning
</body>
```

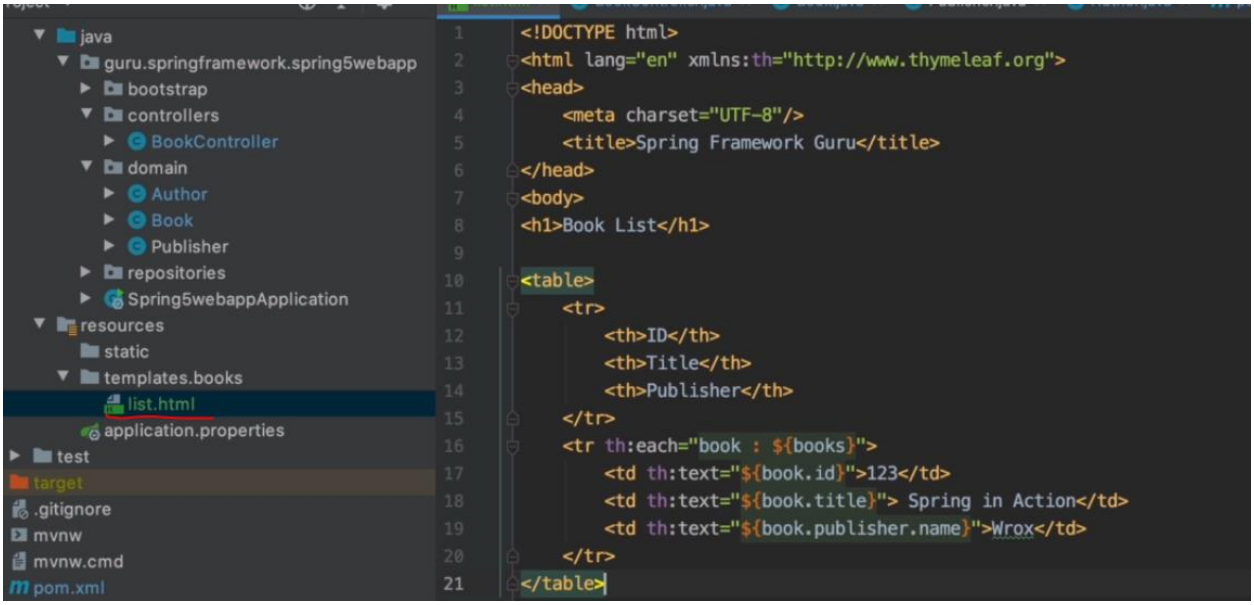
بهتر:

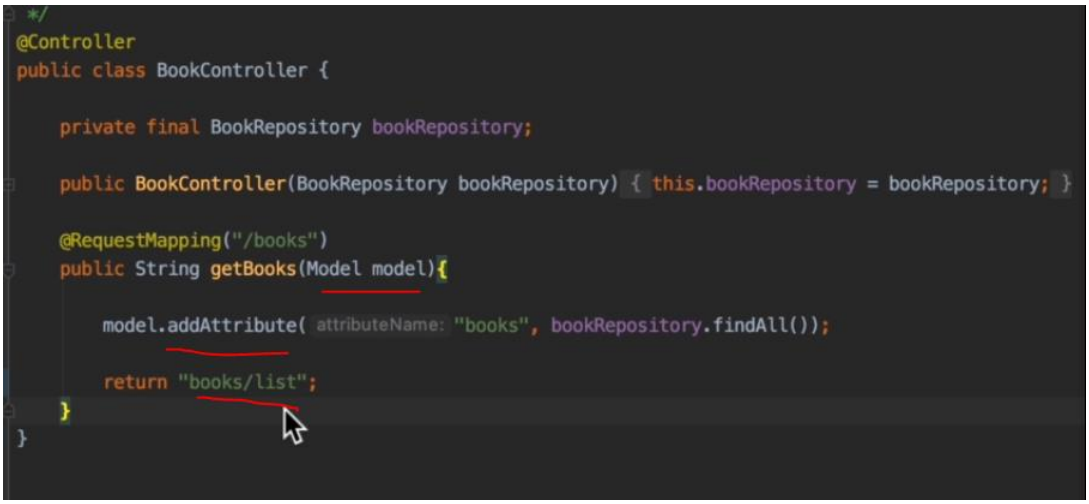
```
@Controller
public class HomeController {

    @RequestMapping("home")
    public String home(String name, HttpSession session ){
//        HttpSession session= httpServletRequest.getSession();
//        String name= httpServletRequest.getParameter("name");
        System.out.println("hi "+ name);
        session.setAttribute("name",name);
        return "home";
    }
```

Model

ارسال data روی Model به سمت client

With Thymeleaf





ModelAndView

ارسال data روی ModelAndView به سمت client

With Jsp

```
@Controller
public class HomeController {
    @RequestMapping("home")
    public ModelAndView home(){
        ModelAndView modelAndView= new ModelAndView();
        modelAndView.addObject("name",”Mehrad or json/object”);
        modelAndView.setViewName("home");
        return modelAndView;
    }
}
```

Sample of: ModelAndView & parameter Entity (form-data)

```
@Controller
public class HomeController {
    @RequestMapping("home")
    public ModelAndView home(Alien alien ){
        ModelAndView modelAndView = new ModelAndView();
        modelAndView.addObject("object",alien);
        modelAndView.setViewName("home");
        return modelAndView;
    }
}
```

```
<body>
        I'm ${object.aname}, ${object.aid} , ${object.lang}  /
I'm into learning
</body>
```

؟؟؟ تا اینجا روش های گفته شده بصورت restful سرویس ارایه میدهند، بخاطرهمین **view name** را به client میفرستند. حالا میخواهیم rest ارایه دهیم، یعنی بجای view page name ، **data**(json) برگردانند:

***Restful / @ResponseBody or @RestController

the returned object is automatically **serialized** into JSON

انوتیشن های **@RequestBody** و **@ResponseBody** ؛ rest را در Spring-webمهیا میکنند.

@ResponseBody on a method

@RestController instead of @Controller on Class= **all method as @ResponseBody**

طبق تنظیمات دیفالت Jackson در spring boot برای serialization؛ فیلدهای null هم ارسال میشوند.

We know Restful before:

```
@RequestMapping("home ")
public String home() {
    return "home.jsp"; // it returns webpage home.jsp
}
```

Now Rest:

```
@RequestMapping("home")
@ResponseBody
    public String home(){
        return "home.jsp"; // it returns this String as a json
    }
```

```
@RequestMapping("/alien")
@ResponseBody
public Alien addAlien(Alien alien){
    return alien; // it returns json
}
```

```
@RequestMapping("/addAlien1")
@ResponseBody
public String addAlien1(Alien alien) {
    return alien.toString(); // it returns ResponseBody
}
```

Example for **@RestController**:

```
@RestController
public class HomeController {

@RequestMapping("/alien")
public Alien addAlien(Alien alien){
    return alien; // it returns json
} }
```

Req1:

```
curl -i \
-H "Accept: application/json" \
-H "Content-Type:application/json" \
-X POST --data
  '{"username": "johnny", "password": "password"}'
"https://localhost:8080/.../content"
```

پس request میره داخل سرویسی که این دو شرایط را داشته باشد:

Let's now add a new endpoint that sends a JSON response:

```
@PostMapping(value = "/content", produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public ResponseTransfer postResponseJsonContent(
  @RequestBody LoginForm loginForm) {
    return new ResponseTransfer("JSON Content!");
}
```

```
public class ResponseTransfer {
    private String text;

    // standard getters/setters
}
```

```
HTTP/1.1 200
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 20 Feb 2020 19:43:06 GMT

{"text":"JSON Content!"}
```

Response1:

Req2:

```
curl -i \
-H "Accept: application/xml" \
-H "Content-Type:application/json" \
-X POST --data
  '{"username": "johnny", "password": "password"}'
"https://localhost:8080/.../content"
```

Req با json درخواست میدهد و با xml، res میخواهد.

Service:

```
@PostMapping(value = "/content", produces = MediaType.APPLICATION_XML_VALUE)
@ResponseBody
public ResponseTransfer postResponseXmlContent(
  @RequestBody LoginForm loginForm) {
    return new ResponseTransfer("XML Content!");
}
```

Response2:

```
HTTP/1.1 200
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Thu, 20 Feb 2020 19:43:19 GMT

<ResponseTransfer><text>XML Content!</text></ResponseTransfer>
```

در spring-web برخلاف consume کردن xml که براش پیچیدگی داشت، produce کردن آسان است.

ResponseEntity Class

برای وقتیکه بخواهیم یک **response کامل** ارسال کنیم.

اگر فقط میخواستیم body را مهیا کنیم، همان @ResponseBody Rest کفایت میکرد.

نیاز نیست حتما @RestController باشد.

Spring takes care of the rest.

ResponseEntity **represents the whole HTTP response: status code, headers, and body**/ fully configure the HTTP response.

return it from the endpoint;

ResponseEntity is a generic type. we can use any type as the response body

```
@GetMapping("/hello")
ResponseEntity<String> hello() {
    return new ResponseEntity<>("Hello World!", HttpStatus.OK);
}
```

```
return new ResponseEntity<>(
  "Your age is " + calculateAge(yearOfBirth),
  HttpStatus.OK);
```

```
@GetMapping("/customHeader")
ResponseEntity<String> customHeader() {
    HttpHeaders headers = new HttpHeaders();
    headers.add("Custom-Header", "foo");

    return new ResponseEntity<>(
      "Custom header set", headers, HttpStatus.OK);
}
```

```
@GetMapping("/hello")
ResponseEntity<String> hello() {
    return ResponseEntity.ok("Hello World!");
}
```

```
return ResponseEntity.ok(HttpStatus.OK);
```

```
BodyBuilder accepted();
BodyBuilder badRequest();
BodyBuilder created(java.net.URI location);
HeadersBuilder<?> noContent();
HeadersBuilder<?> notFound();
BodyBuilder ok();
```

```
@GetMapping("/customHeader")
ResponseEntity<String> customHeader() {
    return ResponseEntity.ok()
        .header("Custom-Header", "foo")
        .body("Custom header set");
}
```

body should be the last call

jackson in SpringBootWeb-restful

In rest-full SpringBoot uses a Jackson.***ObjectMapper*** instance to serialize **responses** and deserialize **requests**.

وقتی rest هستیم (کاری که @RequsetBody و @ResponseBody میکنند) یعنی model در req و res جابجا نمیشود، row-type جابجا میشود

دیفالت SpringBoot برای row-type، json است

عمل write و read در دل Spring MVC انجام میشود.

تنظیمات serialize و deserialize در Rest Spring-mvc میتواند :

- تنظیمات ابتدایی **در Spring-mvc** میتواند در **@RequestMapping** انجام شود (در حد تغییر به xml)

  مثلا **Consume:** تنظیمات اینکه سرویس چه type ای را بعنوان دیتای ورودی میگیرد و req طبق آن درخواست بفرستد.

- برخی از تغییرات با **Configure کردن jackson** قابل انجام است.

  بعضی تغییرات را خود **SpringBoot** بصورت **default** در jackson کانفیگ کرده **مثلا** در هنگام serialization **اگر برای فیلدی مقدار نیامد خطا نخوریم:**

  By default, the Spring Boot configuration will disable the following:

  - *MapperFeature.DEFAULT_VIEW_INCLUSION*
  - *DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES*
  - *SerializationFeature.WRITE_DATES_AS_TIMESTAMPS*

  خودمان **تغییرات بیشتر** را هم میتوانیم Configuring کنیم **مثلا** در هنگام deserialization، **nullها نروند و فرمت تاریخ millisecond نباشد.**

- برخی از تغییرات با Configure کردن قابل انجام نیست و **باید Custom Serializer با jackson** بنویسیم. اگر jsonای که میآید مچ نیست با سرویس ما و یا jsonای که ما میخواهیم بفرستیم به third party مچ نیست با سرویسش باید Custom Serializer با jackson بنویسیم.

راه دیگر: convert نوع هم میتواند انجام بگیرد و نوع جدید serialize شود. میتوانیم obj ای که میخواهیم با client رد و بدل کنیم را **convert** کنیم و converted obj را برای serialize/deserialize به spring-mvc rest بسپاریم. responseClass بنویسیم.

```
public class Coffee {

    private String name;
    private String brand;
    private LocalDateTime date;

   //getters and setters
}
```

```
@GetMapping("/coffee")
public Coffee getCoffee(
        @RequestParam(required = false) String brand,
        @RequestParam(required = false) String name) {
    return new Coffee()
      .setBrand(brand)
      .setDate(FIXED_DATE)
      .setName(name);
}
```

If:

GET http://lolcahost:8080/coffee?brand=Lavazza

*بخاطر تنظیمات دیفالتی که SpringBoot روی Jackson گذاشته، با وجود اینکه json رسیده منطبق بر obj ای که سرویس انتظار دارد نیست، **Deserialize خطا نمیخورد.***

*و **Response** ای که به کلاینت میدهد این است:*

```
{
  "name": null,
  "brand": "Lavazza",
  "date": "2020-11-16T10:21:35.974"
}
```

But

We would like to exclude *null* values and to have a custom date format (dd-MM-yyyy HH:mm). response:

```
{
  "brand": "Lavazza",
  "date": "04-11-2020 10:34"
}
```

پس باید تنظیمات بیشتری به SpringBoot بدهیم:

When using Spring Boot, we have the option to customize the default *ObjectMapper* or to override it. We'll cover both options in the next sections.

***Configuring the ObjectMapper

Date Serialize in SpringBoot & Dont serialize null value:

روش اول Customizing :

application properties:

```
spring.jackson.<category_name>.<feature_name>=true,false
```

Dont serialize null value:

```
spring.jackson.default-property-inclusion=always, non_null, non_absent, non_default, non_empty
```

At this point, we'll obtain this result:

```
{
  "brand": "Lavazza",
  "date": "2020-11-16T10:35:34.593"
}
```

Date Serialize in SpringBoot

```
@Configuration
@PropertySource("classpath:coffee.properties")
public class CoffeeRegisterModuleConfig {

    @Bean
    public Module javaTimeModule() {
        JavaTimeModule module = new JavaTimeModule();
        module.addSerializer(LOCAL_DATETIME_SERIALIZER);
        return module;
    }
}
```

```
spring.jackson.serialization.write-dates-as-timestamps=false
```

```
{
  "brand": "Lavazza",
  "date": "16-11-2020 10:43"
}
```

روش دوم Customizing :

Jackson2ObjectMapperBuilderCustomizer

create configuration beans

```
@Bean
public Jackson2ObjectMapperBuilderCustomizer jsonCustomizer() {
    return builder -> builder.serializationInclusion(JsonInclude.Include.NON_NULL)
      .serializers(LOCAL_DATETIME_SERIALIZER);
}
```

The configuration beans are applied in a specific order, which we can control using the *@Order* annotation. This elegant approach is suitable if we want to configure the *ObjectMapper* from different configurations or modules.

***Custom: Overriding the ObjectMapper

Date Serialize in SpringBoot & Dont serialize null value:

If we want to have full control over the configuration, **there are several options that will disable the auto-configuration and allow only our custom configuration to be applied.**

روش اول Overriding:

The simplest way to override the default configuration is to define an *ObjectMapper* bean and to mark it as *@Primary*.

```
@Bean
@Primary
public ObjectMapper objectMapper() {
    JavaTimeModule module = new JavaTimeModule();
    module.addSerializer(LOCAL_DATETIME_SERIALIZER);
    return new ObjectMapper()
      .setSerializationInclusion(JsonInclude.Include.NON_NULL)
      .registerModule(module);
}
```

روش دوم Overriding:

jackson2ObjectMapperBuilder

```
@Bean
public Jackson2ObjectMapperBuilder jackson2ObjectMapperBuilder() {
    return new Jackson2ObjectMapperBuilder().serializers(LOCAL_DATETIME_SERIALIZER)
      .serializationInclusion(JsonInclude.Include.NON_NULL);
}
```

روش سوم Overriding:

MappingJackson2HttpMessageConverter

```
@Bean
public MappingJackson2HttpMessageConverter mappingJackson2HttpMessageConverter() {
    Jackson2ObjectMapperBuilder builder = new
Jackson2ObjectMapperBuilder().serializers(LOCAL_DATETIME_SERIALIZER)
      .serializationInclusion(JsonInclude.Include.NON_NULL);
    return new MappingJackson2HttpMessageConverter(builder.build());
}
```

It will configure two options by default:

- disable MapperFeature.DEFAULT_VIEW_INCLUSION
- disable DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES

Exception handling

https://www.baeldung.com/cs/http-status-codes

https://www.baeldung.com/exception-handling-for-rest-with-spring

https://medium.com/thefreshwrites/exception-handling-spring-boot-rest-api-c2656b575fee

brief

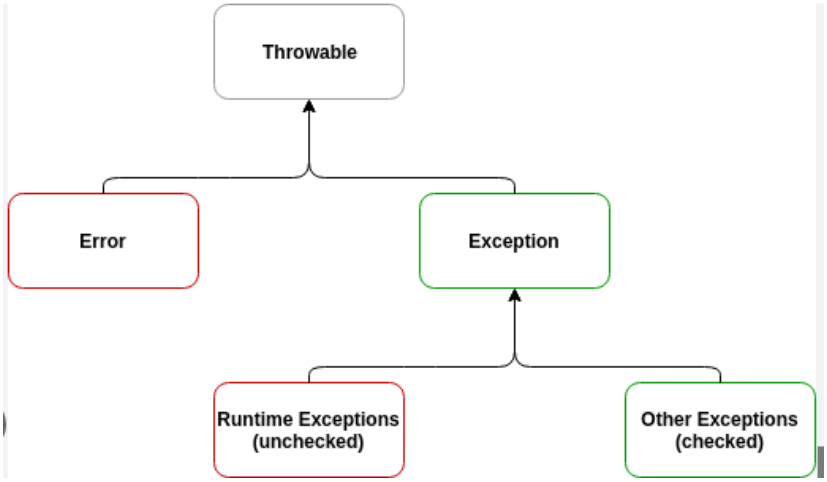
An exception is an event (is an **object** which **is thrown** at **runtime**) that **disrupts** the

normal flow of the program.

exception handling is a **mechanism** that **keeps** the normal flow of the program.

our logic (unchecked ex.) or **java libs** (checked and unchecked ex.): **throw new Exception()**

بطور خلاصه، یجایی (من یا جاوا) **throw Exc**. زدیم. این یا همانجا باید **catch** شود یا **Throws Exc**. اعلام شود.



Checked exceptions are checked at **compile time** and must be handled by the programmer

such as: **File**NotFoundException, **SQL**Exception, **IO**Exception

Unchecked exceptions (runtime ex.) **runtime**

such as: **Null**PointerException, **Array**IndexOutOfBound Exception, Arithmatic, our logic

if is not handled: **500 res. Code;** we handled: **40x, 50x**

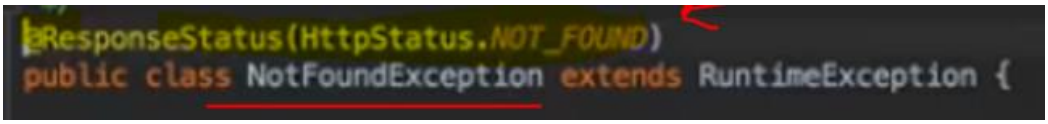
هندل نکرده بودیم برای همه ی exceptionها 500 برمیگرداند ، ولی تعدادی 40x بودند و یا چه نوع 50xای مهم است.

NotFoundExcep./

■

Extend RunTimeExcep. Field: msg

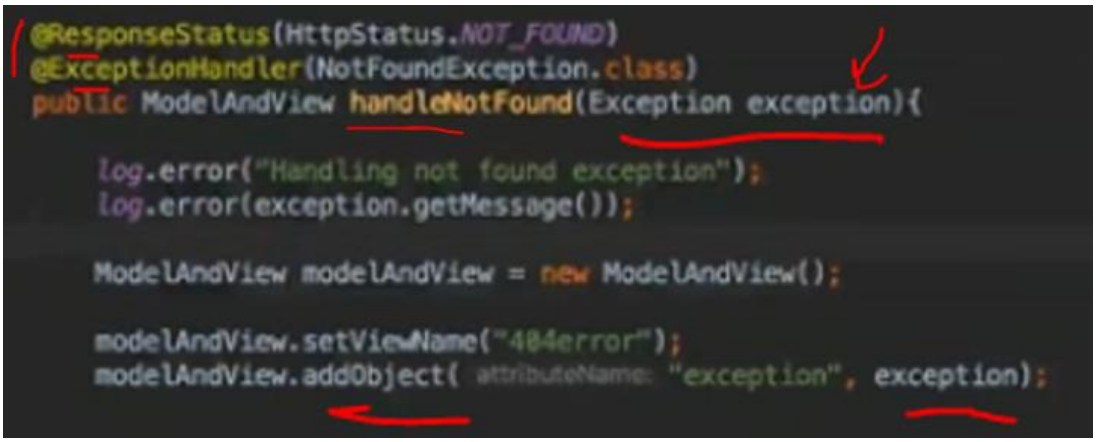
تعریف کلاس exception: + با @ هر .excep ای ، response Code دقیق برمیگرونه



Or Spring web **creates** this subException Class (like NumberFormatedException)

■

تعریف متد هندلر در controller:



یک Controller میسازیم مخصوص handling همه ی subExceptionها:

```java
package guru.springframework.controllers;

import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.servlet.ModelAndView;

/**
 * Created by jt on 7/14/17.
 */
@Slf4j
@ControllerAdvice
public class ControllerExceptionHandler {

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(NumberFormatException.class)
    public ModelAndView handleNumberFormat(Exception exception){

        log.error("Handling Number Format Exception");
        log.error(exception.getMessage());

        ModelAndView modelAndView = new ModelAndView();

        modelAndView.setViewName("400error");
        modelAndView.addObject( attributeName: "exception", exception);

        return modelAndView;
    }
}
```

■

استفاده: throw new myException(msg)

```java
@Override
public Recipe findById(Long l) {

    Optional<Recipe> recipeOptional = recipeRepository.findById(l);

    if (!recipeOptional.isPresent()) {
        throw new NotFoundException("Recipe Not Found. For Id value: " + l.toString() );
    }

    return recipeOptional.get();
}
```

بجای رخ دادن Excep. کلی، یک excep. دقیق نوشتیم و در زمان throw، msg دقیق بهش پاس دادیم

@ResponseStatus به excep. ای که نوشته بودیم دادیم، که در زمان throw این excep. یک status code دقیق به client بدهد. (این excep. ، map شد با یک ResponseStatusCode خاص)

در controller با @ExceptionHandler کارهای handling آن excep. را نوشتیم: بجای توقف اجرای برنامه برای آن req کاربر، فلان page را نشان اش بده.

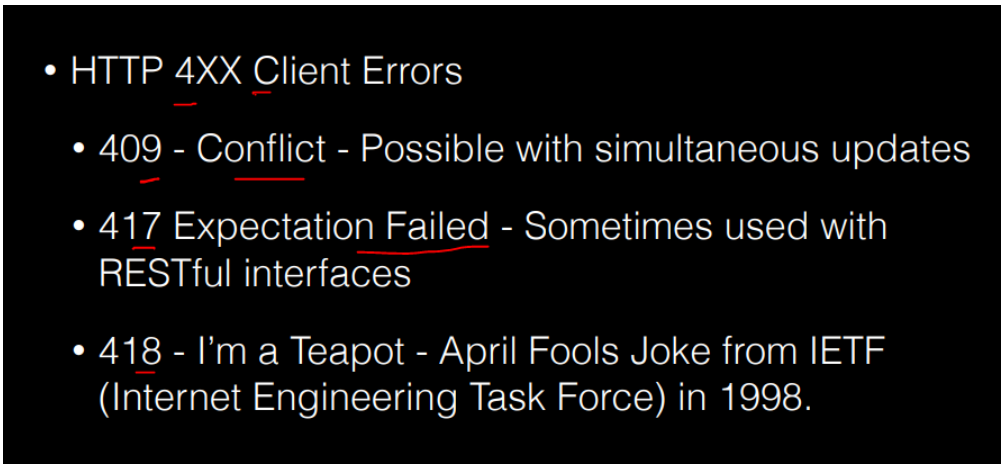
برای اینکه این handler رو برای همه ی controllerهامون یکجا داشته: @ControllerAdvice

Exception Handling in Spring MVC

HTTP Status Codes

- HTTP 5XX Server Error
  - HTTP 500 - Internal Sever Error
    - Generally, any unhandled exception
  - Other 500 errors are generally not used with Spring MVC

- HTTP 4XX Client Errors - Generally Checked Exceptions
  - 400 Bad Request - Cannot process due to client error
  - 401 Unauthorized - Authentication required
  - 404 Not Found - Resource Not Found
  - 405 Method Not Allowed - HTTP method not allowed



417 like: Property missing

تا الانکه هیچ کاری نکردیم؛ وقتی data از DB نمیاد، exception دیفالت برمیگرده و 500 به کاربر داده میشه در صورتی که 404 درستش هست.

Creating Related Exception Class

یک کلاس .Exce میسازیم:

```
//@ResponseStatus(HttpStatus.NOT_FOUND)
public class NotFoundException extends RuntimeException {

    public NotFoundException() {
        super();
    }

    public NotFoundException(String message) {
        super(message);
    }

    public NotFoundException(String message, Throwable cause) {
        super(message, cause);
    }

}
```

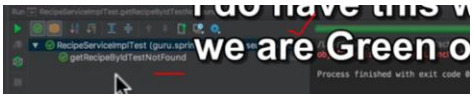
In service:

```
if (!recipeOptional.isPresent()) {
    //throw new RuntimeException("Recipe Not Found!");
    throw new NotFoundException("Recipe Not Found");
}
```

testServiceImpl:

```
@Test(expected = NotFoundException.class)
public void getRecipeByIdTestNotFound() throws Exception {

    Optional<Recipe> recipeOptional = Optional.empty();

    when(recipeRepository.findById(anyLong())).thenReturn(recipeOptional);

    Recipe recipeReturned = recipeService.findById( l: 1L);

    //should go boom
}
```

Test passed:



Test controller:

```
@Test
public void testGetRecipeNotFound() throws Exception {

    Recipe recipe = new Recipe();
    recipe.setId(1L);

    when(recipeService.findById(anyLong())).thenThrow(NotFoundException.class);

    mockMvc.perform(get( urlTemplate: "/recipe/1/show"))
            .andExpect(status().isNotFound());
}
```

Not passed:

بجای .Excep کلی؛ Exception ای که ما ساختیم رخ میدهد، ولی status code هنوز 500 برمیگردد.

(الان exception را service نمیسازد، بلکه تست میسازد و بهش msg نمیدهد.)

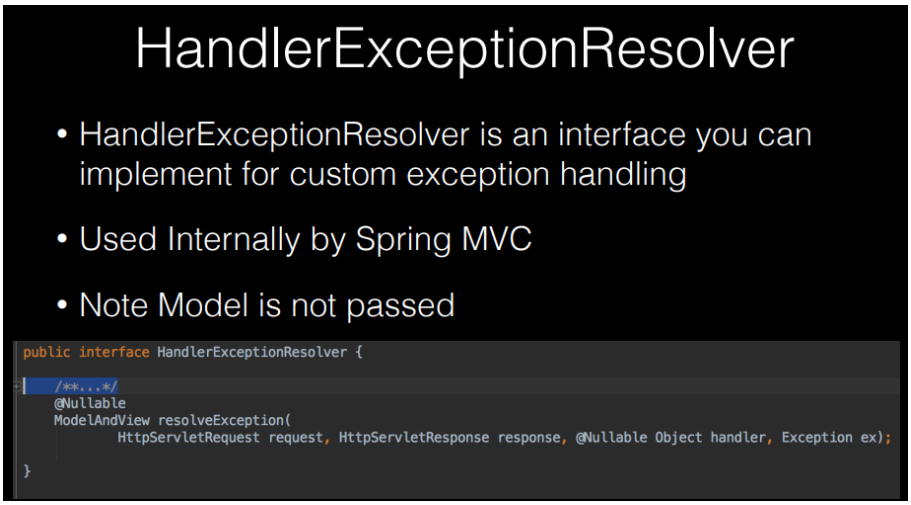
from **url**: Show to user

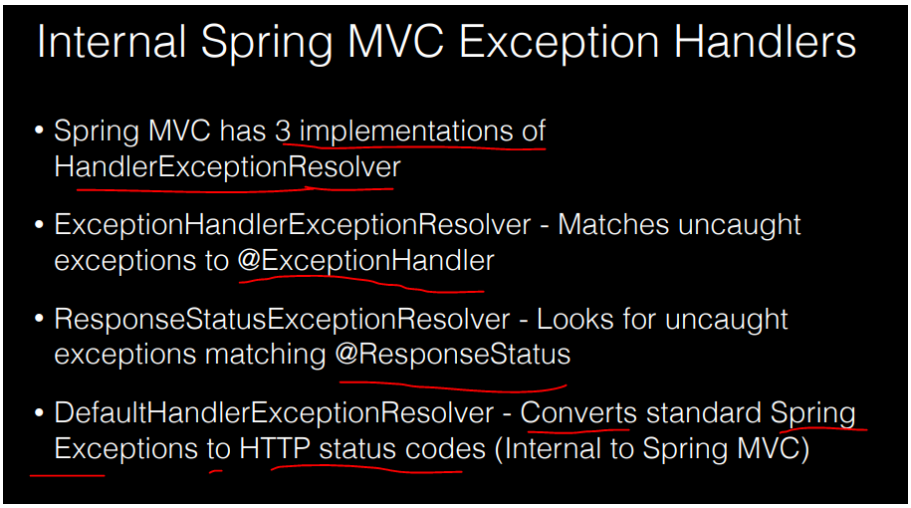
تا الان کاری که کردیم:

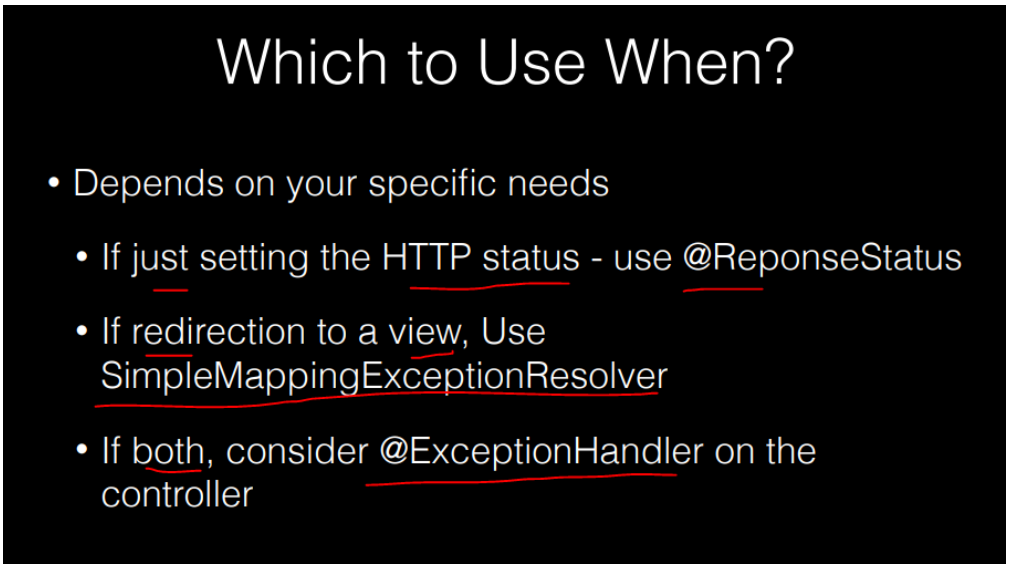
فقط یک Excep. Class درست کردیم که Exception کلی نگیریم و مشخص باشه چه نوع کلاس exceptionای خوردیم و msg دقیق از exception میگیریم. فقط دیگه excep. کلی نمیخوریم. **ولی هنوز برای handle اش کدی وارد نکردیم** (چون unchecked است جاوا اجازه میدهد). و**response status به کاربر ندادیم**، میخوایم درستش کنیم: exception دقیق برگرده و به کاربر 404 برگردونه.

I HandlerExceptionResolver:

custom exc. Handling with spring MVC





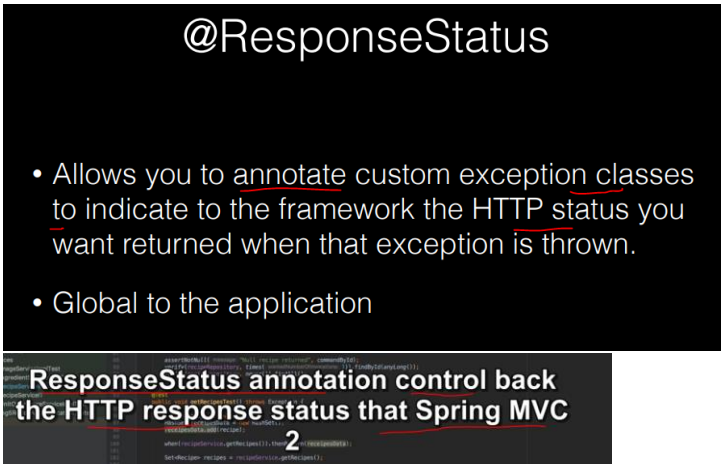


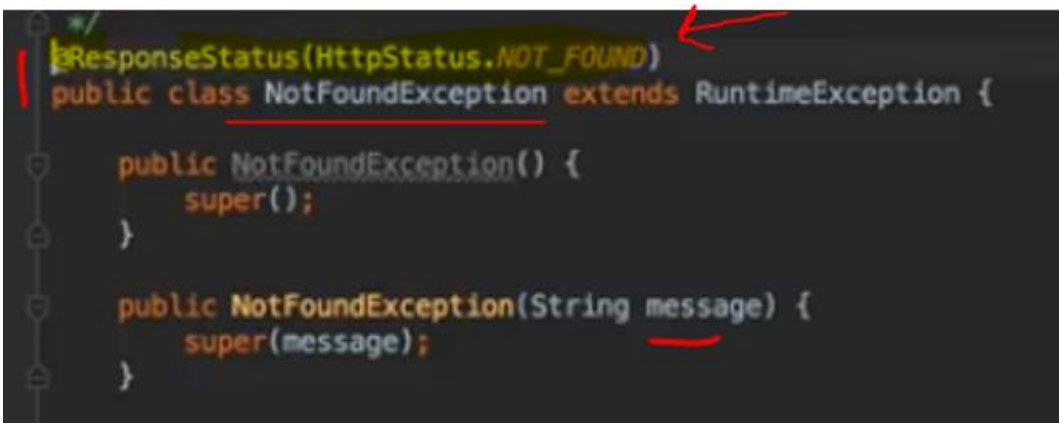
@ResponseStatus

تا الان کاری که کردیم:

فقط یک Excep. Class درست کردیم که مشخص باشه چرا exception خوردیم و msg به exception میدهیم. فقط دیگه excep. کلی نمیخوریم. **ولی هنوز برای handle اش کدی وارد نکردیم. وresponse status به کاربر ندادیم**، میخوایم درستش کنیم: exception دقیق برگرده و به کاربر 404 برگردونه.

یک کلاس Exc را به یک http status، map میکند.





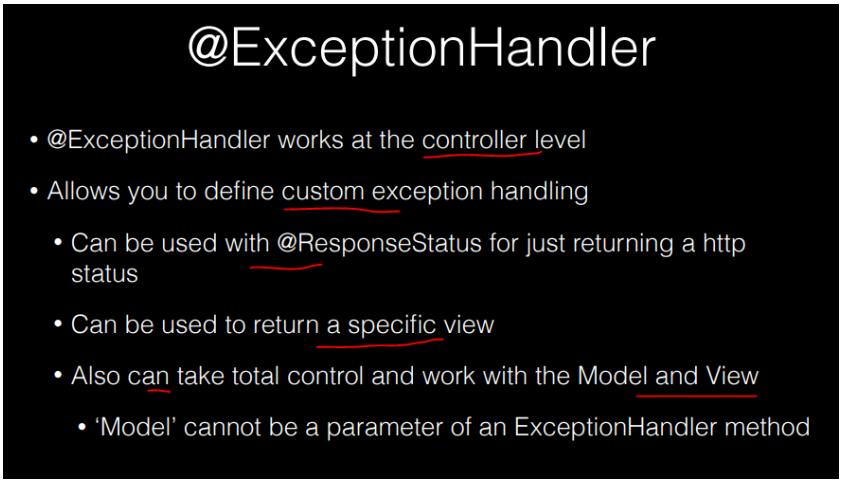
controller Test is **passed** now (and **correct status code** sent to user):

با کمک **@ResponseStatus** ، **ResponseCode مرتبط** به Excp **not found** که درست کرده تخصیص داده شد.
براش resp. status مپ کردیم.

@ExceptionHandler for handling (page & msg)

ولی **هنوز catch برای این exc نوشته نشده** --> در لاگ برنامه مشخص است که exception رخ میدهد همان page دیفالت به کاربر داده میشود (با msg ای که موقع ساخت exc. گرفته با responseCodeای که بهش دادیم). ولی اگر **میخواهیم page مخصوص این exception را داشته باشیم** و **msg را هم کنترل کنیم** و ... باید handler خودش را بنویسیم.

این کاری است که **@ExceptionHandler** انجام میدهد: الان هندل ما این است که page 404 درست کنیم و به کاربر بدهیم. و مرحله ی بعد msg.



NotFoundException: تغییری نمیکنه

```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class NotFoundException extends RuntimeException {

    public NotFoundException() {
        super();
    }

    public NotFoundException(String message) {
        super(message);
    }
```

Exc. Handling (catch) is done i**n controller**:

```java
@ResponseStatus(HttpStatus.NOT_FOUND)
@ExceptionHandler(NotFoundException.class)
public ModelAndView handleNotFound(){

    log.error("Handling not found exception");

    ModelAndView modelAndView = new ModelAndView();

    modelAndView.setViewName("404error");

    return modelAndView;
}
```

کاری که در **handling** کردیم این بود که یک **model view –page** ، **Return** کردیم.

Test Controller:

```java
@Test
public void testGetRecipeNotFound() throws Exception {

    when(recipeService.findById(anyLong())).thenThrow(NotFoundException.class);

    mockMvc.perform(get( urlTemplate: "/recipe/1/show"))
            .andExpect(status().isNotFound())
            .andExpect(view().name( expectedViewName: "404error"));
}
```

```
404error.html
README.md | RecipeController.java | 404error.html | index.html | RecipeControllerTest.java | NotFoundExceptio
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>404 Not Found Error</title>

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
          integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin=
          th:href="@{/webjars/bootstrap/3.3.7-1/css/bootstrap.min.css}">

    <script src="/webjars/jquery/1.11.1/jquery.min.js"></script>

    <!-- Latest compiled and minified JavaScript -->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
            integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
            crossorigin="anonymous" th:src="@{/webjars/bootstrap/3.3.7-1/js/bootstrap.min.js}"></script>
</head>
<body>
<div class="container-fluid" style="...">
    <div class="row">
        <div class="col-md-6 col-md-offset-3">
            <h1>404 Not Found</h1>
        </div>
    </div>
</div>

</body>
</html>
```

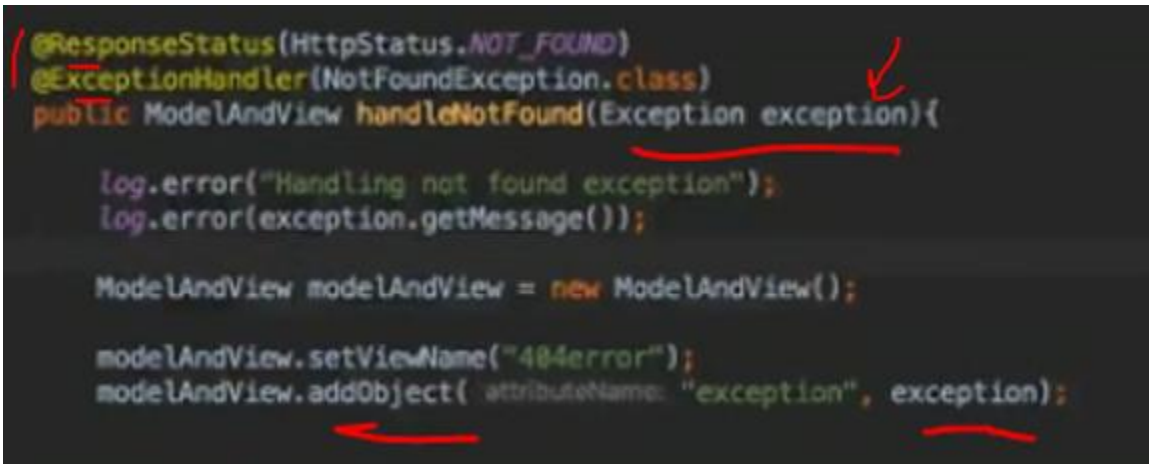
Test Passed:

```
2017-07-13 16:33:59.287  INFO 25662 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Fr...rvlet 'dispatcherServlet': initializatio
2017-07-13 16:33:59.30...  INFO 25662 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Fr...rvlet 'dispatcherServlet': initializatio
2017-07-13 16:33:59.358 ERROR 25662 --- [nio-8080-exec-1] g.s.controllers.RecipeController         : Handling not found exception
2017-07-13 16:33:59.361  WARN 25662 --- [nio-8080-exec-1] .m.m.a.ExceptionHandlerExceptionResolver : Resolved exception caused by Handler execution: gur
```

چون براش handler یا catch نوشتیم، در لاگ دیگه exception رخ نمیدهد و interrupt نمیکند ، بلکه چیزی که ما در handleنوشتیم انجام میشود.

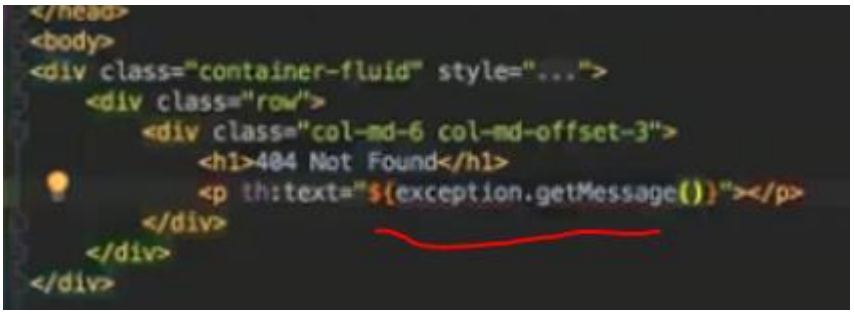
تا الان هر کسی new Exception میکند میتواندmsg پاس بدهد چون constructorاشmsg میگیرد،ولی handler جدیدی که ما نوشتیم ورودی msg هنوز ندارد، میخواهیم براش بذاریم:

handler:

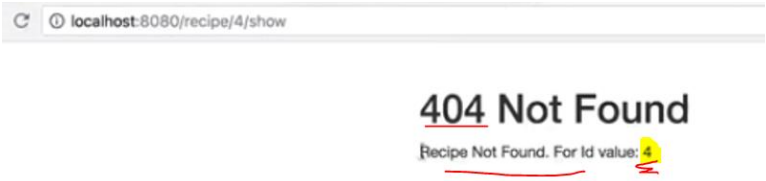


ServiceImpl:

Error Page:



در نتیجه با تست کلاینت:



@Handling MVCspring's Exceptions

حالا میخواهیم یکی از .excepهای که **خود springMVC ، new و throw میکند** را ما handle کنیم که دیگه دیفالت 500 نباشه. Page و code و تغییر msg.

الان این نوع exception وقتی throw (توسط خود Spring MVC) میشود:

روند اجرای برنامه متوقف میشود (handle نمیشود)

ResponseStatusCode صحیحی به client داده نمیشود.

Default Error Page به کابر نشان داده میشود.

فرق های زیادی بین Exception داخلی spring Web و زیرکلاسهایی که ما مینویسیم مثل بخش قبل وجود ندارد، جفتشون رو ما باید هندل کنیم، **فقط برای نوع قبلی باید زیرکلاس Exception رو خودمون بنویسیم**.

وقتی .excep را spring throw کند ؛ **msg** را **خود spring ساخته**، طبیعی است چون spring عمل validate را انجام داده و exception را new کرده و msg بهش داده. ما میتونیم در handler آنرا تغییر دهیم.

میدانیم تا وقتی exception ها رو (چه آنهایی که خود java یا spring mvc از Exception زیرکلاس ساخته یا اونهایی که ما میسازیم) هندل نکنیم (handler=catch)، همه اونها یک error برای سرور ایجاد میکنند و به client، 500 فرستاده میشود.

Controller:

```java
@Slf4j
@Controller
public class RecipeController {

    private final RecipeService recipeService;

    public RecipeController(RecipeService recipeService) { this.recipeService = recipeService; }

    @GetMapping("/recipe/{id}/show")
    public String showById(@PathVariable String id, Model model){

        model.addAttribute( s: "recipe", recipeService.findById(new Long(id)));

        return "recipe/show";
    }
```

Page:

Stop running req:

از کجا باید بفهمیم چه exceptionای را باید handle کنیم؟ از متن .excep رخ داده شده بالا:

NumberFormatedException

Handler براش مینویسیم (برو به فلان page با فلان msg)

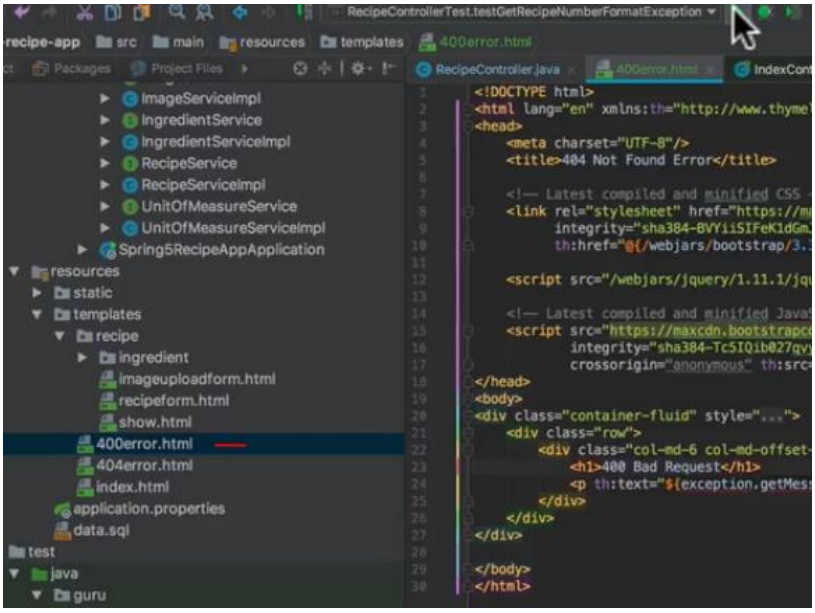
و

وصلش میکنیم به bad.req 400 ResponceStatusCode

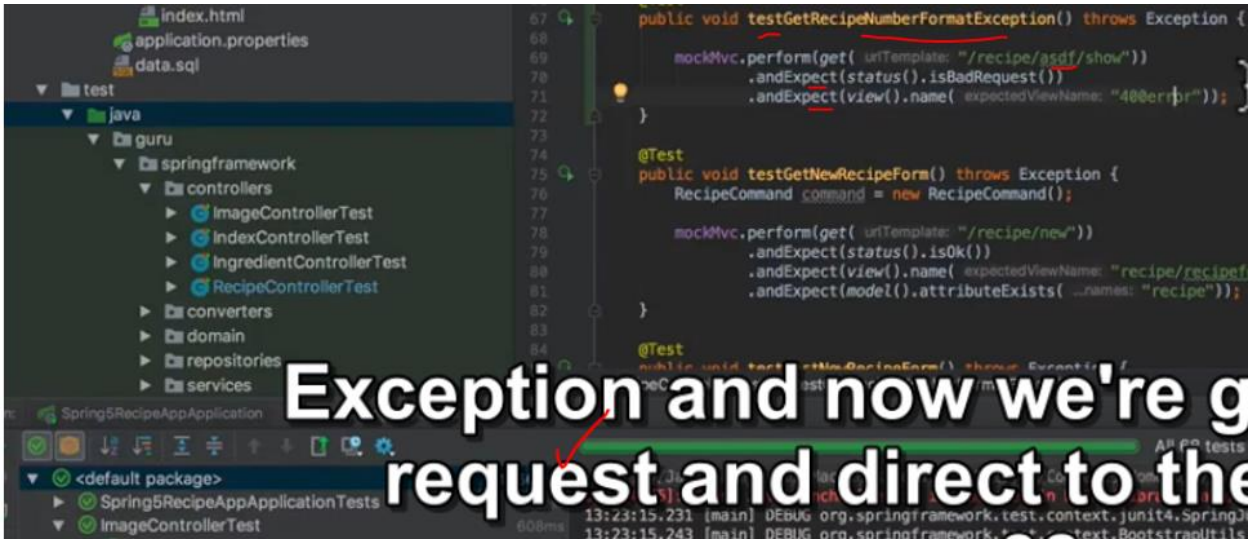
```java
@ResponseStatus(HttpStatus.BAD_REQUEST)
@ExceptionHandler(NumberFormatException.class)
public ModelAndView handleNumberFormat(Exception exception){

    log.error("Handling Number Format Exception");
    log.error(exception.getMessage());

    ModelAndView modelAndView = new ModelAndView();

    modelAndView.setViewName("400error");
    modelAndView.addObject( attributeName: "exception", exception);

    return modelAndView;
}
```

Msg را هم میتونستیم تغییر بدیم.

Bad req. page:

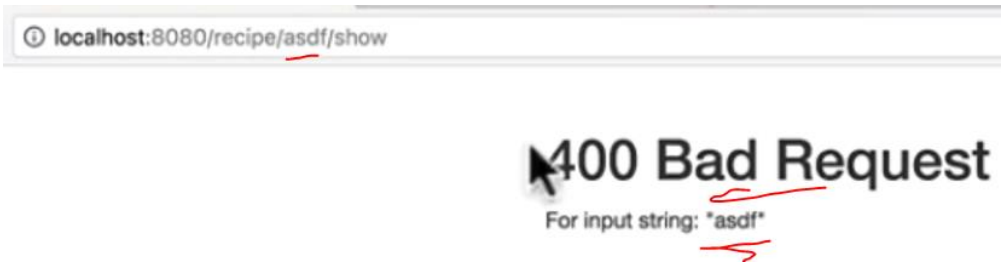


Test controller: Passed



چون این دسته از exceptionها رو خود spring ، exception رو new و throw میکنه نیازی نیست در تست ما اینکار هارو بکنیم.

Ui:



@ControllerAdvice: global ExceptionHandler

برای اینکه handler رو برای همه ی controllerهامون یکجا داشته باشیم و نخواهیم controller شلوغ بشه با نوشته شدن handlerها در اون، یک Controller میسازیم مخصوص handling:

```java
package guru.springframework.controllers;

import lombok.extern.slf4j.Slf4j;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.servlet.ModelAndView;

/**
 * Created by jt on 7/14/17.
 */
@Slf4j
@ControllerAdvice
public class ControllerExceptionHandler {

    @ResponseStatus(HttpStatus.BAD_REQUEST)
    @ExceptionHandler(NumberFormatException.class)
    public ModelAndView handleNumberFormat(Exception exception){

        log.error("Handling Number Format Exception");
        log.error(exception.getMessage());

        ModelAndView modelAndView = new ModelAndView();

        modelAndView.setViewName("400error");
        modelAndView.addObject( attributeName: "exception", exception);

        return modelAndView;
    }
}
```

Test*: در هنگام ساخت MockMVC میتوانیم ControllerAdvice را هم مشخص کنیم:

```java
6 usages
@Mock
RecipeService recipeService;

2 usages
RecipeController controller;

8 usages
MockMvc mockMvc;

John Thompson
@Before
public void setUp() throws Exception {
    MockitoAnnotations.initMocks( testClass: this);

    controller = new RecipeController(recipeService);
    mockMvc = MockMvcBuilders.standaloneSetup(controller)
            .setControllerAdvice(new ControllerExceptionHandler())
            .build();
}
```

```java
@Test
public void testGetRecipe() throws Exception {

    Recipe recipe = new Recipe();
    recipe.setId(1L);

    when(recipeService.findById(anyLong())).thenReturn(recipe);

    mockMvc.perform(get( urlTemplate: "/recipe/1/show"))
            .andExpect(status().isOk())
            .andExpect(view().name( expectedViewName: "recipe/show"))
            .andExpect(model().attributeExists( ...names: "recipe"));
}
```

با این روش تست نیازی نیست کل context spring بالا بیاید. فقط قسمتهایی که در این تست نیاز است. unit test است.

Data validation

برای اینکه این @ های روی فیلدهای Entity در ورودی های سرویسهای controller چک شوند، از **@Valid قبل از نام Entity** استفاده میکنیم.

میتوانیم exception های مرتبطی تعریف و همینجا در controller، throw کنیم.

JSR 380 Validator

JSR 380 - Bean Validation 2.0

- Approved August 2017
- Added to Spring Framework 5.0 RC2
- Available in Spring Boot 2.0.0 +
- Uses Hibernate Validator 6.0 + (Implementation of Bean Validation 2.0)
- Primary goal of Bean Validation 2.0 is Java 8 language features
- Added ~11 new built in validation annotations
- Remainder of presentation will focus on Bean Validation 2.0

Standard validator

Standard Validators

@AssertFalse	@Min
@AssertTrue	@NotNull
@DecimalMax	@Null
@DecimalMin	@Past
@Digits	@Pattern
@Future	@Size
@Max	

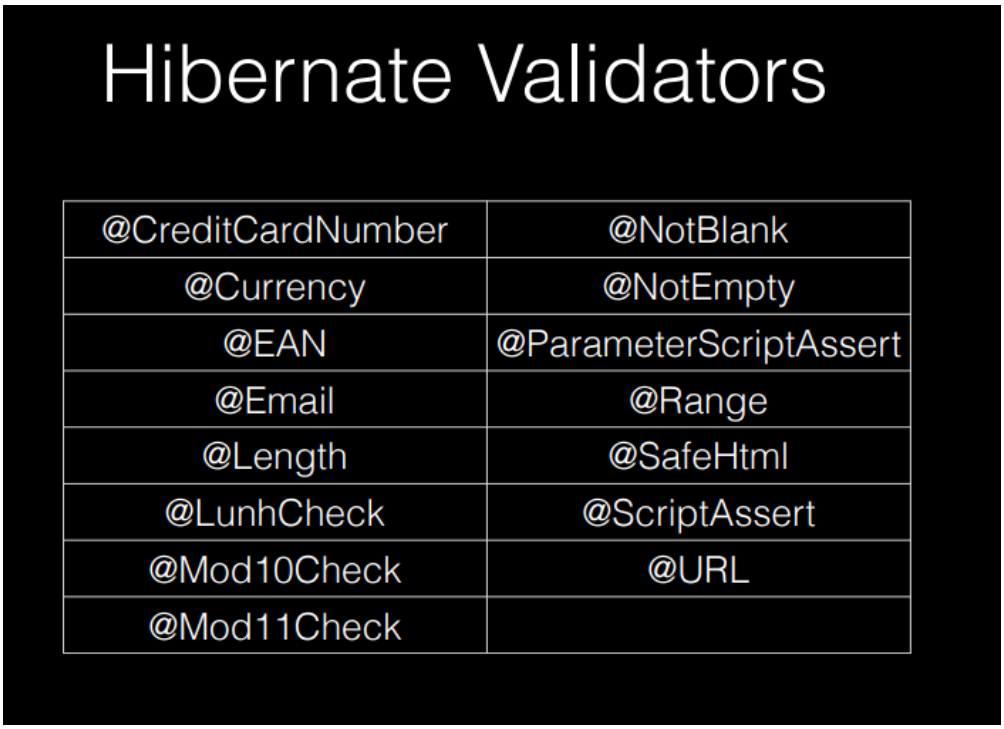
- **@DecimalMin** - Value is larger
- **@DecimalMax** - Value is less than
- **@Negative** - Value is less than zero. Zero invalid.
- **@NegativeOrZero** - Value is zero or less than zero
- **@Positive** - Value is greater than zero. Zero invalid.
- **@PositiveOrZero** - Value is zero or greater than zero.
- **@Size** - checks if string or collection is between a min and max

These are pretty self explanatory

- **@NotEmpty** - Checks if value is null or empty (whitespace characters or empty collection)
- **@NonBlank** - Checks string is not null or not whitespace characters
- **@Email** - Checks if string value is an email address

- **@Digits** - check for integer digits and fraction digits
- **@Past** - Checks if date is in past
- **@PastOrPresent** - Checks if date is in past or present
- **@Future** - Checks if date is in future
- **@FutureOrPresent** - Checks if date is present or in future
- **@Pattern** - checks against RegEx pattern

Hibernate Validator



- **@ScriptAssert** - Class level annotation, checks class against script
- **@CreditCardNumber** - Verifies value is a credit card number
- **@Currency** - Valid currency amount
- **@DurationMax** - Duration less than given value
- **@DurationMin** - Duration greater than given value
- **@EAN** - Valid EAN barcode
- **@ISBN** - Valid ISBN value

Never had a need to use it by

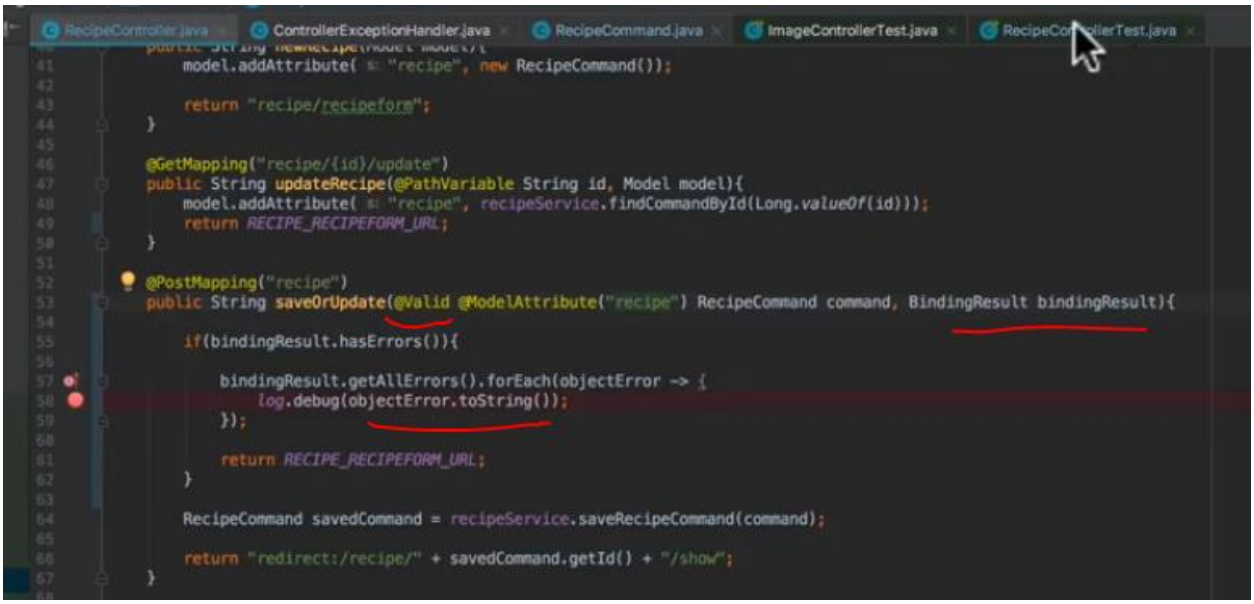
- **@Length** - String length between given min and max
- **@CodePointLength** - Validates that code point length of the annotated character sequence is between min and max included.
- **@LuhnCheck** - Luhn check sum
- **@Mod10Check** - Mod 10 check sum
- **@Mod11Check** - Mod 11 check sum
- **@Range** - checks if number is between given min and max (inclusive)
- **@SafeHtml** - Checks for safe HTML
- **@UniqueElements** - Checks if collection has unique elements
- **@Url** - checks for valid URL

Hiberate Validator Country Constraints

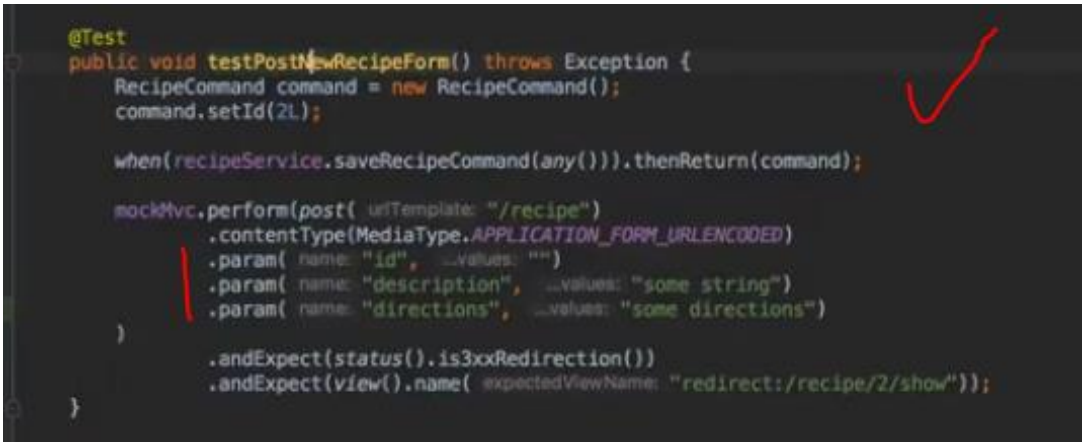
- **@CNPJ** - Brazilian Corporate Tax Payer Registry Number
- **@CPF** - Brazilian Individual Taxpayer Registry Number
- **@TituloEleitoral** - Brazilian voter ID
- **@NIP** - Polish VAR ID
- **@PESEL** - Polish National Validation Number
- **@REGON** - Polish Taxpayer ID

برای اینکه این @ های روی فیلدهای Entity در ورودی های سرویسهای controller چک شوند، از **Valid@ قبل از نام Entity** استفاده میکنیم.

میتوانیم exception های مرتبطی تعریف و همینجا در controller، throw کنیم.

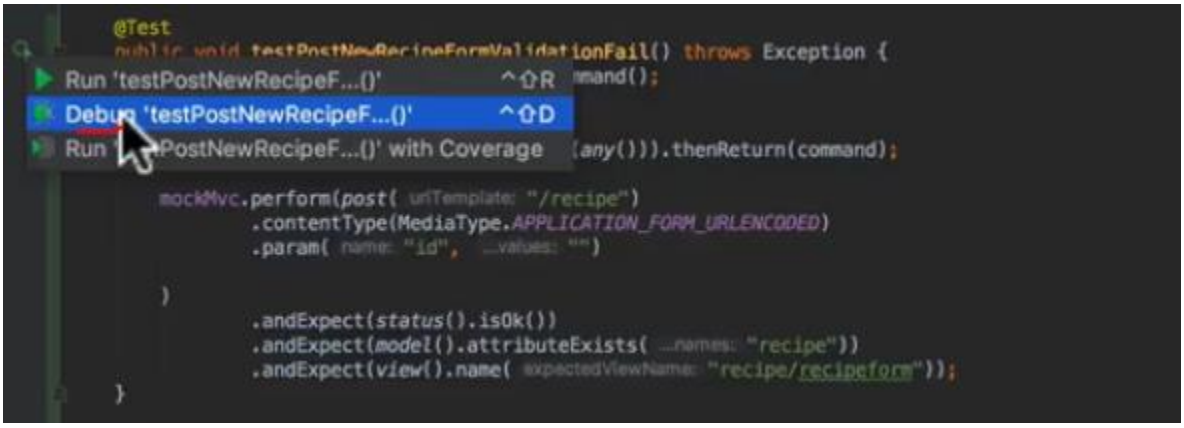


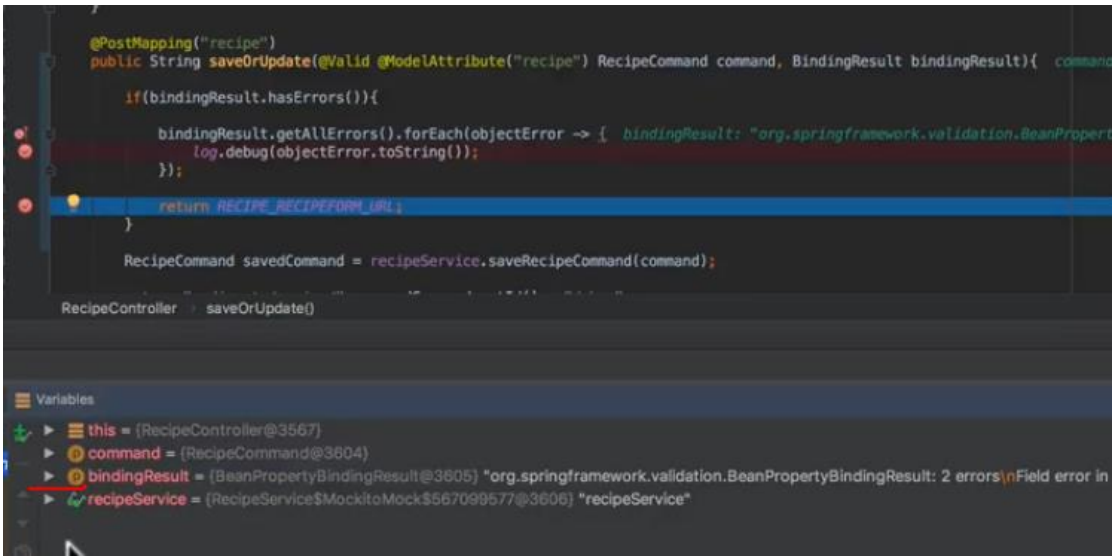
*Test:



Check log debug:

@Bindingresultنتایج validation را نگه میدارد، میتوانیم درمحیط debug آنها را log کنیم.





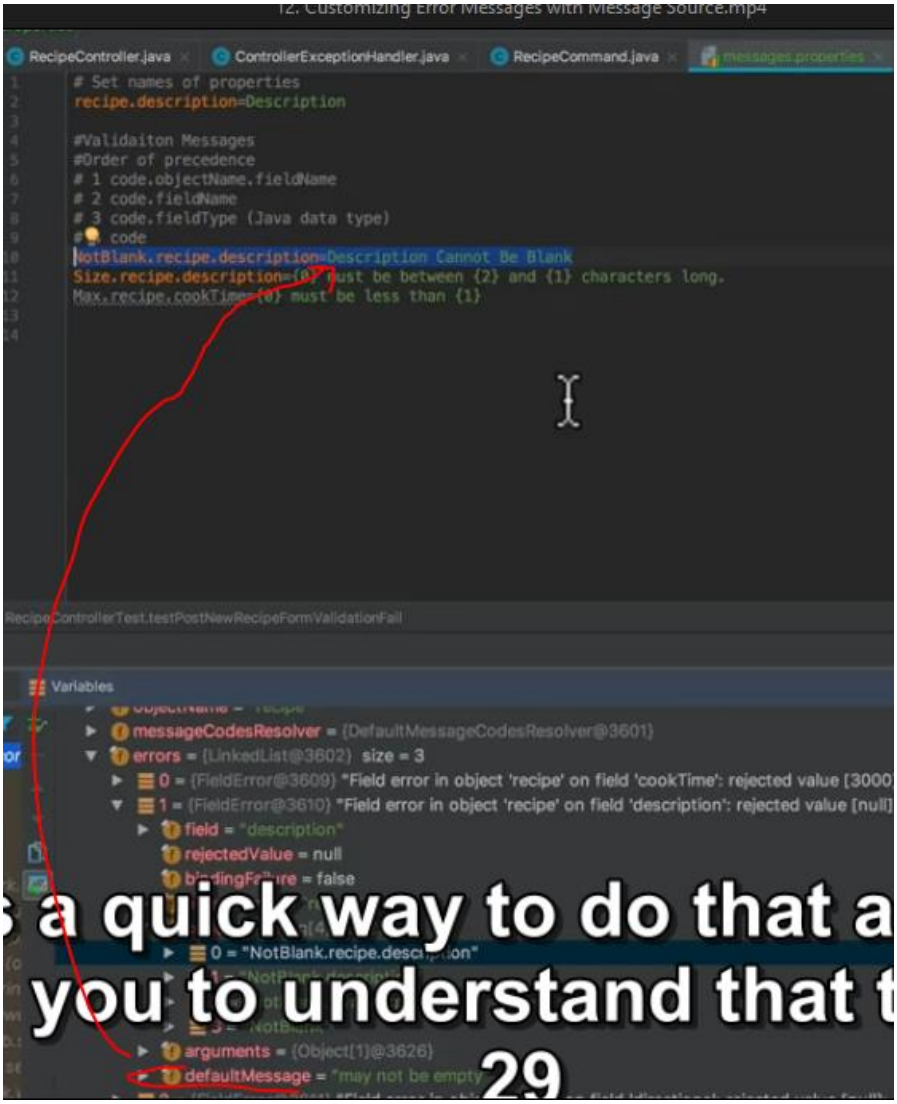
BindingResult از Client فیدبک میگیرد:

```
import org.springframework.validation.BindingResult;
```

```java
@GetMapping
public String processFindForm(Owner owner, BindingResult result, Model model){
    // allow parameterless GET request for /owners to return all records
    if (owner.getLastName() == null) {
        owner.setLastName(""); // empty string signifies broadest possible search
    }

    // find owners by last name
    List<Owner> all_owner = ownerService.findAllByLastNameLike("%"+ owner.getLastName() + "%");

    if (all_owner.isEmpty()) {
        // no owners found
        result.rejectValue( s: "lastName",  s1: "notFound",  s2: "not found");
        return "owners/findOwners";
    } else if (all_owner.size() == 1) {
        // 1 owner found
        owner = all_owner.get(0);
        return "redirect:/owners/" + owner.getId();
    } else {
        // multiple owners found
        model.addAttribute( s: "selections", all_owner);
        return "owners/ownersList";
    }
```


Custom msg

میتوانیم متن خطای اختصاصی برای هر مورد در هر کلاس تعیین کنیم.

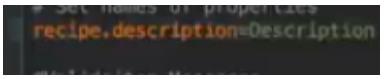

12. Customizing Error Messages with Message Source.mp4
RecipeController.java
ControllerExceptionHandler.java
RecipeCommand.java
messages.properties
# Set names of properties
recipe.description=Description
#Validaiton Messages
#Order of precedence
# 1 code.objectName.fieldName
# 2 code.fieldName
# 3 code.fieldType (Java data type)
# 4 code
NotBlank.recipe.description=Description Cannot Be Blank
Size.recipe.description={0} must be between {2} and {1} characters long.
Max.recipe.cookTime={0} must be less than {1}
RecipeControllerTest.testPostNewRecipeFormValidationFail
Variables
messageCodesResolver = {DefaultMessageCodesResolver@3601}
errors = {LinkedList@3602} size = 3
0 = {FieldError@3609} "Field error in object 'recipe' on field 'cookTime': rejected value [3000
1 = {FieldError@3610} "Field error in object 'recipe' on field 'description': rejected value [null]
field = "description"
rejectedValue = null
bindingFailure = false
0 = "NotBlank.recipe.description"
arguments = {Object[1]@3626}
defaultMessage = "may not be empty"
a quick way to do that a
you to understand that t
29

So:


Edit Recipe Information
Recipe Description:
Description Cannot Be Blank
Description must be between 3 and 255 characters long.
Categories:
Cat 1

I18n

Browser automatically set some of HTTP headers



Locale Detection

- Default behavior is to use Accept-Language header
- Can be configured to use system, a cookie, or a custom parameter.
  - Custom Parameter is useful to allow user to select language. **that in,**

98

Locale Resolvers

- AcceptHeaderLocaleResolver is the Spring Boot Default
- Optionally, can use FixedLocaleResolver
  - Uses the locale of the JVM
- Available: CookieLocaleResolver, SessionLocaleResolver **to use a FixedLocaleResolver,**

115

استفاده ی کمی دارند

Changing Locale

- Browsers are typically tied to the Locale of the operating system
- Locale changing plugins are available
- Spring MVC provides as LocaleChangeInterceptor to allow you to configure a custom parameter to use to change the locale.

And, I'm guessing that anything by Microsoft

Resource Bundles

- Resource bundles (aka messages.properties) are selected on highest match order.
- First selected will be on language region
  - ie en-US would match messages_en_US.properties

- If no exact match is found, just the language code is used.
  - en-GB would match messages_en_GB.properties
  - OR if no file found, would match messages_en.properties
  - Finally would match messages.properties

Web client

JSP

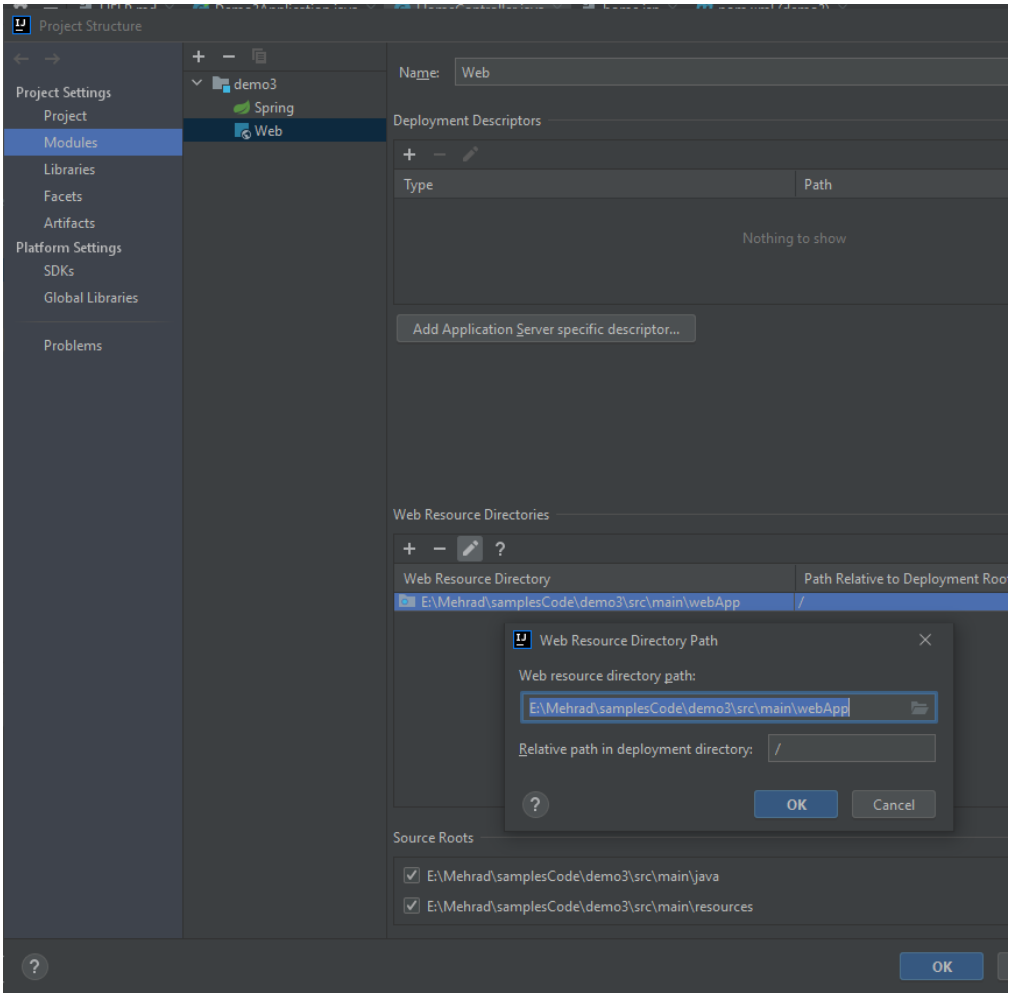
```
@Controller
public class HomeController {

    @RequestMapping("home")
    public String home(){
        return "home.jsp";
    }
}
```

It returns the home.jsp page. (the page will be **downloaded** by browser)

(Spring boot doesn't support JSP by default)

```
<dependency>
    <groupId>org.apache.tomcat</groupId>
    <artifactId>tomcat-jasper</artifactId>
    <version>9.0.68</version> <!-- it's embeded tomcat version--
>
</dependency>
```

Now spring boot knows **how to convert JSP to Servlet**



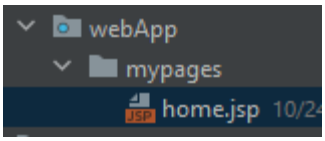
"home.jsp" is in webApp

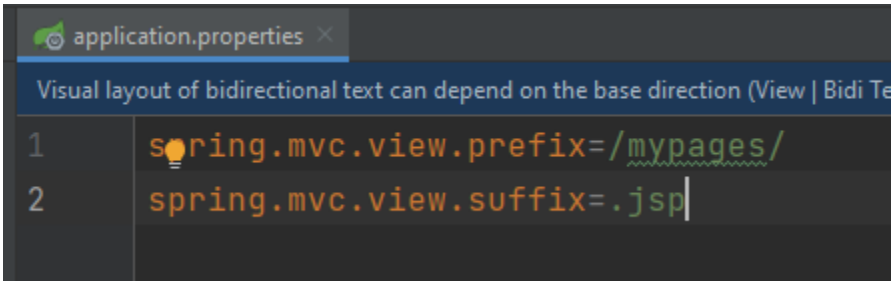
```
@Controller
public class HomeController {

    @RequestMapping("home")
    public String home(){
        return "home.jsp";
    }
}
```

Now It returns the home.jsp page body

Prefix/ suffix

If we change webpages folder:





So:

```
@Controller
public class HomeController {

    @RequestMapping("home")
    public String home(){
        return "home";
    }
}
```

Template engine thymeleaf

هر تغیری میدیم رو page میبینیم. برای اجرا نیازمند servlet نیست. مثل html.

- Thymeleaf is a Java template engine producing XML, XHTML, and HTML5.
- Thymeleaf is a replacement for JSPs (Java Server Pages)
- Thymeleaf is a 'Natural' Template Engine.
- Is not tied to web environment. (ie can be used for producing HTML for emails)
- Thymeleaf is not a web framework

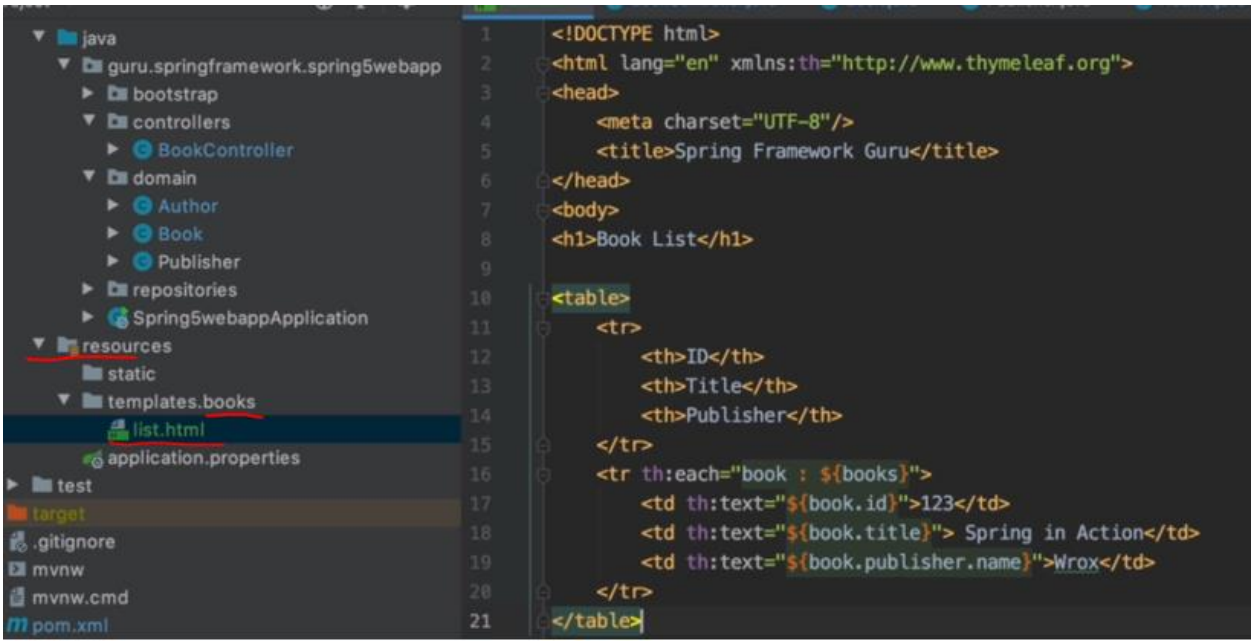
Thymeleaf vs JSP

- Thymeleaf templates are valid HTML documents you can view in the browser.
- JSP files are not valid HTML, and look awful in the browser
- The natural templating ability allows you to perform rapid development, without the need to run a container to parse the template/JSP to view the product in a browser.
  - Speeds development time

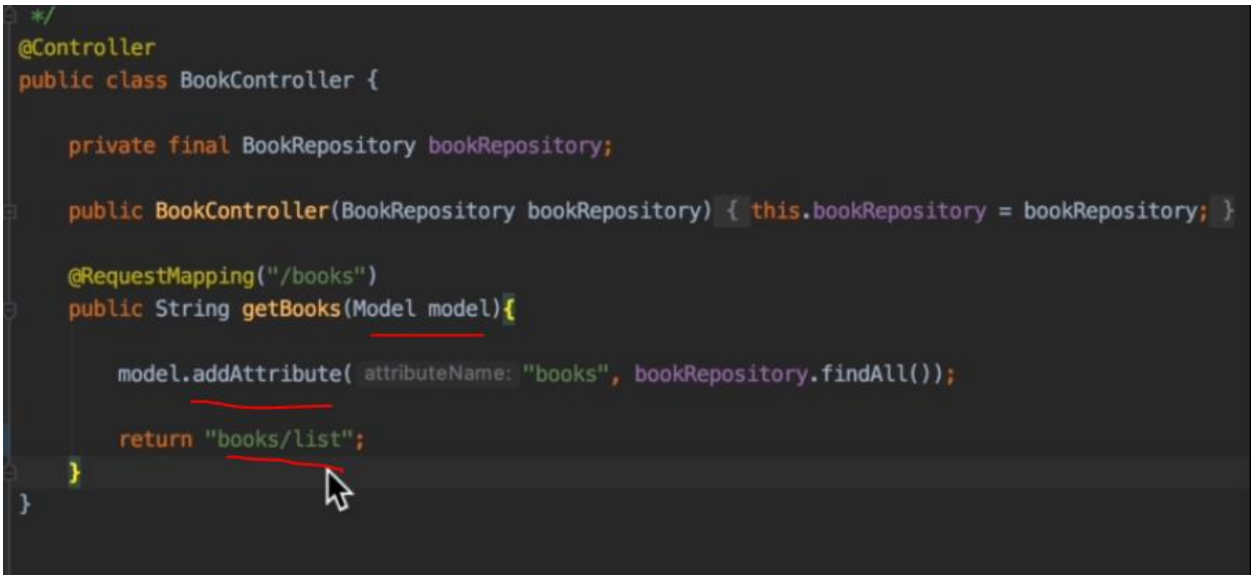
thymeleaf با starter به spring boot project به راحتی اضافه میشود:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

در خود Resources که maven میسازد قرار میگیرد و مثل JSP نیاز به webApp ندارد.


java
guru.springframework.spring5webapp
bootstrap
controllers
BookController
domain
Author
Book
Publisher
repositories
Spring5webappApplication
resources
static
templates.books
list.html
application.properties
test
target
.gitignore
mvnw
mvnw.cmd
pom.xml
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8"/>
<title>Spring Framework Guru</title>
</head>
<body>
<h1>Book List</h1>
<table>
<tr>
<th>ID</th>
<th>Title</th>
<th>Publisher</th>
</tr>
<tr th:each="book : ${books}">
<td th:text="${book.id}">123</td>
<td th:text="${book.title}"> Spring in Action</td>
<td th:text="${book.publisher.name}">Wrox</td>
</tr>
</table>

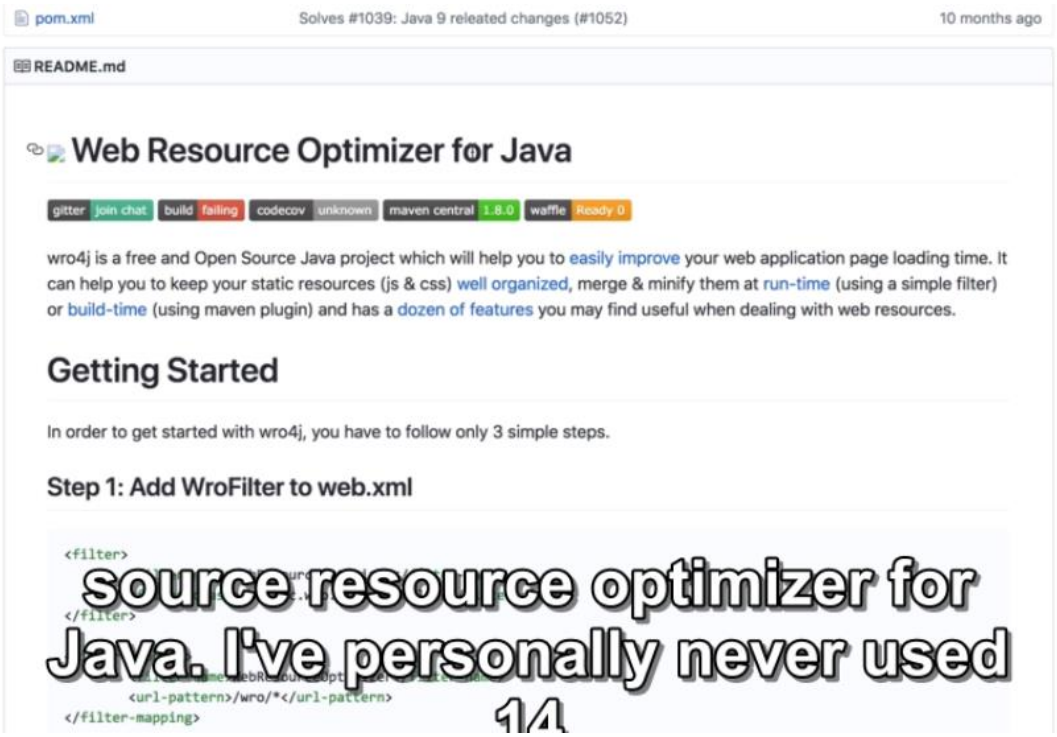
Model:


*/
@Controller
public class BookController {
private final BookRepository bookRepository;
public BookController(BookRepository bookRepository) { this.bookRepository = bookRepository; }
@RequestMapping("/books")
public String getBooks(Model model){
model.addAttribute(attributeName: "books", bookRepository.findAll());
return "books/list";
}
}

Web Resource Optimizer

17. Spring Pet Clinic - Implement Web Resource Optimizer for Java

چسباندن ui به spring boot



Web Data Binder

@InitBinder (very old! And good)

Having WebDataBinder **inject into Controller** and control it! (Sending a disallowed field)

```
import org.springframework.web.bind.annotation.*;
```

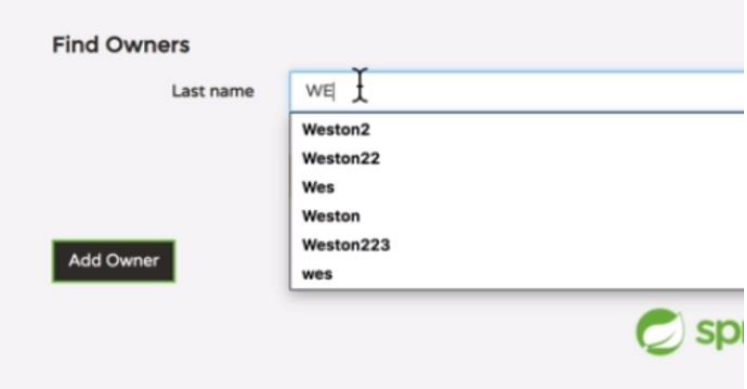
اینکه برای کل Annotation@های Spring MVC هست، یکیش این InitBinder@

برای client مهیا میکند:

```
import org.springframework.web.bind.WebDataBinder;
```

```
John Thompson
@InitBinder
public void setAllowedFields(WebDataBinder dataBinder) { dataBinder.setDisallowedFields("id"); }
```

```
<h2>Find Owners</h2>

<form th:object="${owner}" th:action="@{/owners}" method="get"
  class="form-horizontal" id="search-owner-form">
  <div class="form-group">
    <div class="control-group" id="lastNameGroup">
      <label class="col-sm-2 control-label">Last name </label>
      <div class="col-sm-10">
        <input class="form-control" th:field="*{lastName}" size="30"
          maxlength="80" /> <span class="help-inline"><div
            th:if="${#fields.hasAnyErrors()}">
            <p th:each="err : ${#fields.allErrors()}" th:text="${err}">Error</p>
          </div></span>
      </div>
    </div>
  </div>
```



WebDataBinder و BindingResult هیچ ربطی به هم ندارند.

Webjar

Popular web component

```
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>
    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>bootstrap</artifactId>
        <version>3.3.7-1</version>
    </dependency>
    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>jquery</artifactId>
        <version>3.2.1</version>
    </dependency>
    <dependency>
        <groupId>org.
```

e want to do is

index.html

Recipe.java | RecipeBootstrap.java | spring5-recipe-app | index.html

html head link

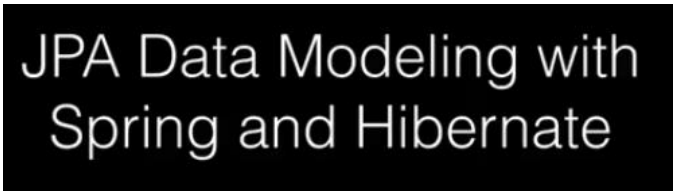
```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>Recipe Home</title>

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
          integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous"
          th:href="@{/webjars/bootstrap/3.3.7-1/css/bootstrap.min.css}">

    <script src="/webjars/jquery/1.11.1/jquery.min.js"></script>

    <!-- Latest compiled and minified JavaScript -->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
            integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
            crossorigin="anonymous" th:src="@{/webjars/bootstrap/3.3.7-1/js/bootstrap.min.js}"></script>
</head>
<body>

<div class="container-fluid" style="...">
    <div class="row">
        <div class="col-md-6 col-md-offset-3">
            <div class="panel panel-primary">
```

```
<html lang="en">
<head>
    <meta charset="UTF-8"/>
    <title>Recipe Home</title>

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="/webjars/bootstrap/3.3.7-1/css/bootstrap.min.css"
          integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">

    <script src="/webjars/jquery/1.11.1/jquery.min.js"></script>

    <!-- Latest compiled and minified JavaScript -->
    <script src="/webjars/bootstrap/3.3.7-1/js/bootstrap.min.js"
            integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
            crossorigin="anonymous"></script>
</head>
<body>

<div class="container-fluid" style="margin-top: 20px">
    <div class="row">
        <div class="col-md-6 col-md-offset-3">
```

----------JPA-Hibernate

Hibernate: بیشتر کارهای مرتبط به ساختار **DB** را در app انجام میدهد. مثل DDL و DML، @Entity

JPA: بیشتر کارهای مرتبط به کار روی **data** را در app انجام میدهد. Prepared queries



Entity

Entity

Create table:

```
@Entity
public class Alien {

    @Id
    private int aid;
```

@ID

@GeneratedValue (Strategy=)

Table: use from table to get **max +1**

Sequence: for that table, we need seq. **generator**

IDENTITY: **DB provides** us primary key; it depends on DB type.

Auto: **hibernate** determines **one of above** based on underlying DB

Default: JPA provides

اگر بخواهیم خود jpa تولید id را به عهده بگیرد:

```
@Id
@GeneratedValue
private int aid;
```

Base Entity

```
@MappedSuperclass
public class BaseEntity implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }
}
```

It can contain time of insert and update ...

@LOB

JPA برای **String** تعداد 256کاراکتر در DB در نظر میگیرد.

برای اینکه بیشتر بشه در **String** نوشت و در DB ذخیره کرد، نوعش در DB باید **CLOBS** باشه:

```
@Lob
private String recipeNotes;
```

و برای **عکس** فیلد **ByteArray** در حالت معمولی:

```
private Byte[] image;
```

و فیلد **BLOBS** در DB برای عکس های بزرگ :

```
@Lob
private Byte[] image;
```

MultipartFile

.20Persisting images to Database

```
/**
 * Created by jt on 7/3/17.
 */
@Controller
public class ImageController {

    private final ImageService imageService;
    private final RecipeService recipeService;

    public ImageController(ImageService imageService, RecipeService recipeService) {
        this.imageService = imageService;
        this.recipeService = recipeService;
    }

    @GetMapping("recipe/{id}/image")
    public String showUploadForm(@PathVariable String id, Model model){
        model.addAttribute( s "recipe", recipeService.findCommandById(Long.valueOf(id)));

        return "recipe/imageuploadform";
    }

    @PostMapping("recipe/{id}/image")
    public String handleImagePost(@PathVariable String id, @RequestParam("imagefile") MultipartFile file){

        imageService.saveImageFile(Long.valueOf(id), file);

        return "redirect:/recipe/" + id + "/show";
    }
}
```

```java
    @Override
    @Transactional
    public void saveImageFile(Long recipeId, MultipartFile file) {

        try {
            Recipe recipe = recipeRepository.findById(recipeId).get();

            Byte[] byteObjects = new Byte[file.getBytes().length];

            int i = 0;

            for (byte b : file.getBytes()){
                byteObjects[i++] = b;
            }

            recipe.setImage(byteObjects);

            recipeRepository.save(recipe);
        } catch (IOException e) {
            log.error("Error occurred", e);

            e.printStackTrace();
        }


    }
}
```

JPA Repository

```java
@Indexed
public interface Repository<T, ID> {

}
```

```java
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {

    /**
     * Saves a given entity. Use the returned instance for further operat
     * entity instance completely.
```

```java
*/
@NoRepositoryBean
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {

    /**
     * Returns all entities sorted by the given options.
     *
     * @param sort
     * @return all entities sorted by the given options
     */
    Iterable<T> findAll(Sort sort);

    /**
     * Returns a {@link Page} of entities meeting the paging restriction provided in the {@code Pageable} object.
     *
     * @param pageable
     * @return a page of entities
     */
    Page<T> findAll(Pageable pageable);
}
```

```java
*/
@NoRepositoryBean
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T> {

    /*
     * (non-Javadoc)
     * @see org.springframework.data.repository.CrudRepository#findAll()
     */
    List<T> findAll();

    /*
     * (non-Javadoc)
     * @see org.springframework.data.repository.PagingAndSortingRepository#findAll(org.springframework.data.domain.Sort)
     */
```

```java
 * Created by jt on 6/13/17.
 */
public interface RecipeRepository extends Crud {
}
```

CrudRepository<T, ID> (org.springframework.data.repository)
CrudMethodMetadata (org.springframework.data.jpa.repository.su…
CrudMethods (org.springframework.data.repository.core)
ReactiveCrudRepository<T, ID> (org.springframework.data.reposi…
DefaultCrudMethods (org.springframework.data.repository.core.s…
RxJava1CrudRepository<T, ID> (org.springframework.data.reposit…
RxJava2CrudRepository<T, ID> (org.springframework.data.reposit…
Press ^Space to see non-imported classes >>

reflection and Java generics. So I'm

کلاسی را برای impl repo در نظر گرفته، در حالی که متدها خودش بدنه داشت و جواب میداد. فکر میکنم تنها دلیلش اینه که مشخص گنه محتوای خروجی متدها که Optional هست داخلش چیه **تکلیفش معلوم بشه** و به لایه ی بالاتر بدهد:

اینجا یک **کلاس repo** تعریف شده، در بهضی پروژه ها همین کار را **در دل متدهای کلاس service** میکنند.

```
    @Override
    public Set<PetType> findAll() {
        Set<PetType> petTypes = new HashSet<>();=
        petTypeRepository.findAll().forEach(petTypes::add);
        return petTypes;
    }

    @Override
    public PetType findById(Long aLong) {
        return petTypeRepository.findById(aLong).orElse( other: null);
    }

    @Override
    public PetType save(PetType object) {
        return petTypeRepository.save(object);
    }

    @Override
    public void delete(PetType object) {
        petTypeRepository.delete(object);
    }

    @Override
    public void deleteById(Long aLong) {
        petTypeRepository.deleteById();
    }
}
```

with Service:

```
public interface RecipeService {

    Set<Recipe> getRecipes();
}
```

```java
@Service
public class RecipeServiceImpl implements RecipeService {

    private final RecipeRepository recipeRepository;

    public RecipeServiceImpl(RecipeRepository recipeRepository) {
        this.recipeRepository = recipeRepository;
    }

    @Override
    public Set<Recipe> getRecipes() {
        Set<Recipe> recipeSet = new HashSet<>();
        recipeRepository.findAll().iterator().forEachRemaining(recipeSet::add);
        return recipeSet;
    }
}
```

```java
@Controller
public class IndexController {

    private final RecipeService recipeService;

    public IndexController(RecipeService recipeService) {
        this.recipeService = recipeService;
    }

    @RequestMapping({"", "/", "/index"})
    public String getIndexPage(Model model) {

        model.addAttribute( attributeName: "recipes", recipeService.getRecipes());

        return "index";
    }
}
```

You could write :

```
Foo foo = repository.findById(id)
                    .orElse(new Foo());
```

or get a `null` default value if it makes sense (same behavior as before the API change) :

```
Foo foo = repository.findById(id)
                    .orElse(null);
```

2. Suppose that if the entity is found you want to return it, else you want to throw an exception.

You could write :

```
return repository.findById(id)
        .orElseThrow(() -> new EntityNotFoundException(id));
```

You could write :

```
Optional<Foo> fooOptional = fooRepository.findById(id);
if (fooOptional.isPresent()) {
    Foo foo = fooOptional.get();
    // processing with foo ...
} else {
    // alternative processing....
}
```

Simple samples

Sample one: addAlien

Model:

```java
@Entity
public class Alien {

    @Id
    private int aid;

    private String aname;
```

Repo:

```java
public interface AlienRepository extends
CrudRepository<Alien,Integer> {

}
```

Controller:

```java
@Controller
public class AlienController {

    @Autowired
    AlienRepository alienRepository;

    @RequestMapping("/addAlien")
    public String addAlien(Alien alien){
        alienRepository.save(alien);
       return "home.jsp";
    }
}
```

ui:

```
<form action="addAlien">
    <input type="text" name="aid">
    <input type="text" name="aname">
    <input type="submit"> <\<br>
</form>
```

http://localhost:8080/addAlien?aid=101&aname=Mehrad

Sample two: getAlien

home.jsp:

```
<form action="getAlien">
    <input type="text" name="aid"> <br>
    <input type="submit"><br>
</form>
```

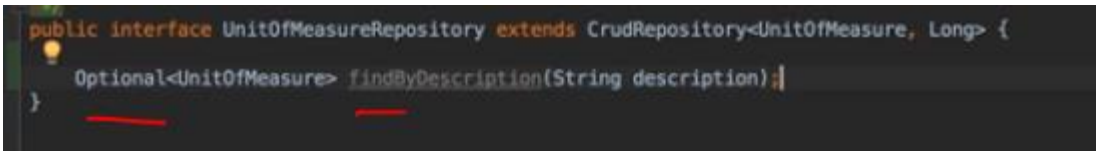
Controller:

```
@RequestMapping("/getAlien")
public ModelAndView getAlien(@RequestParam("aid") int aid){
    Alien alien = alienRepository.findById(aid).orElse(new
Alien());
    ModelAndView modelAndView = new
ModelAndView("showAlien.jsp");
    modelAndView.addObject(alien);
    return modelAndView;
}
```

showAlien.jsp:

```
<body>
        ${alien}
</body>
```

Query method



JPA پیاده سازی کویری متدهارو انجام میده و hiber دیتا رو میاره و در Optional قرار میده

```
public interface AlienRepository extends
CrudRepository<Alien,Integer> {
    List<Alien> findByTech(String tech);
    List<Alien> findByAidGreaterThan(int aid);
}
```

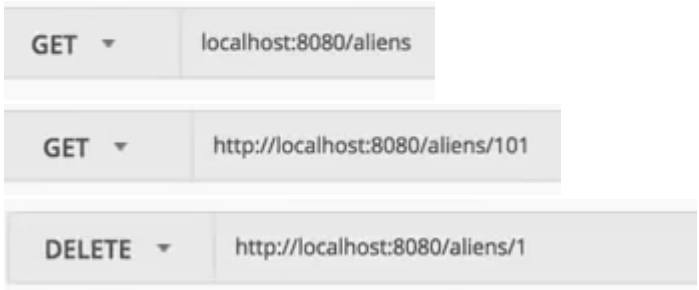
My own Query

```
@Query("from Alien where tech=?1 order by aname")
List<Alien> findByMyQuery(String tech);
```

@RepositoryRestResource

SpringBoot (embedded tomcat, SP Framework) + **JPA** + restful, ...
but We don't need SpringBoot-Web (@controller, …)


```
@RepositoryRestResource(collectionResourceRel="aliens",path="aliens")
public interface AlienRepo extends JpaRepository<Alien, Integer>
{

}
```



POST http://localhost:8080/aliens

```
{
    "aname": "Priya",
    "tech": "ML",
    "_links": {
        "self": {
            "href": "http://localhost:8080/aliens/106"
        },
        "alien": {
            "href": "http://localhost:8080/aliens/106"
        }
    }
}
```

PUT http://localhost:8080/aliens/103

Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
{
    "aname": "Pravin",
    "tech": "AI",
    "_links": {
        "self": {
            "href": "http://localhost:8080/aliens/103"
        },
        "alien": {
            "href": "http://localhost:8080/aliens/103"
        }
    }
}
```

چکار کنیم لینک هارو نیاره؟؟؟

Relations

Owning side (in DB) / @mappedBy= foreign_field

<u>نگه دارنده foreign key یا دیتای relation</u> در DB است.

In **Owner** side**: DB** has **foreign_field**

In **Owned** (non-owner side) side**: Class** tells @mappedBy= which of my fields

برای MTM ، owner هیچکدام از Entityها نیستند، چون دیتای mapping در DB، درهیچکدام از آن دو table مربوط به entityها قرار نمیگیرد و در یک table جدید قرار میگیرد.

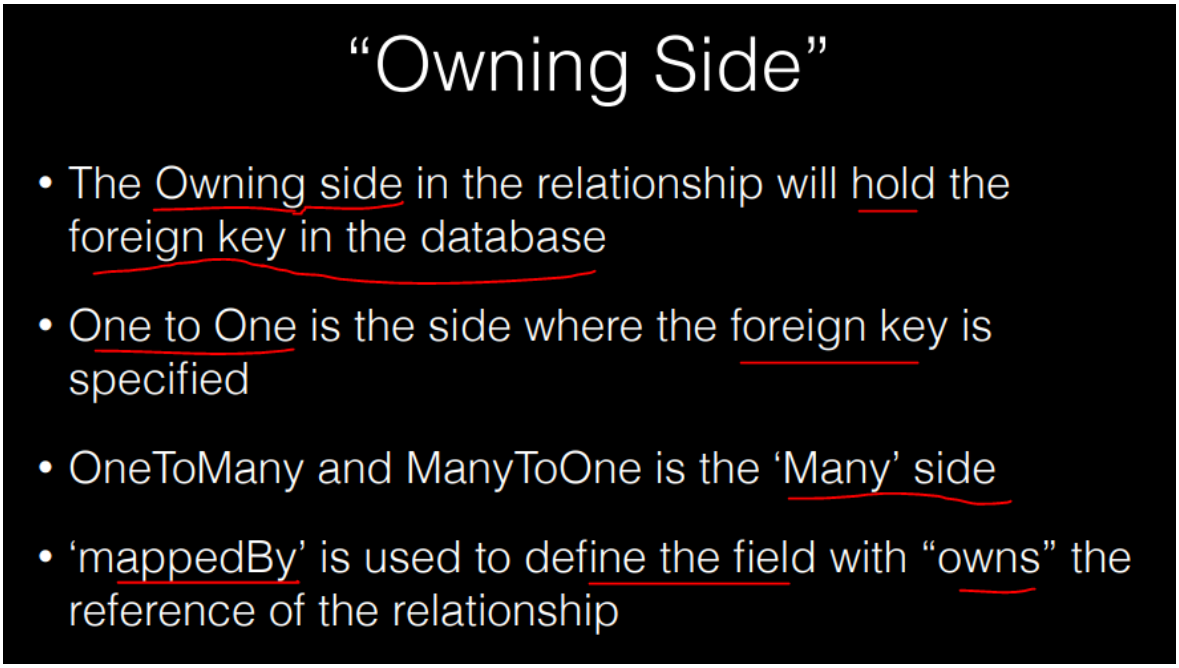
<u>OTO:</u>

Sideای که **mappedBy میزند، owned** است و دیگیری owner است.

<u>M</u>TO:

حواسمان باشد درارتباط های MTO ما در کلاس سمت owned لیست داریم ولی در DB لیستی نگه داشته نمیشود! در رم در زمان runtime این لیست ها پر میشوند. پس طبیعی است که سمت @OneToMany که دارای لیست است ولی در DB آنرا ندارد موجودی ضعیف تر باشد و owner همیشه @ManyToOne باشد.

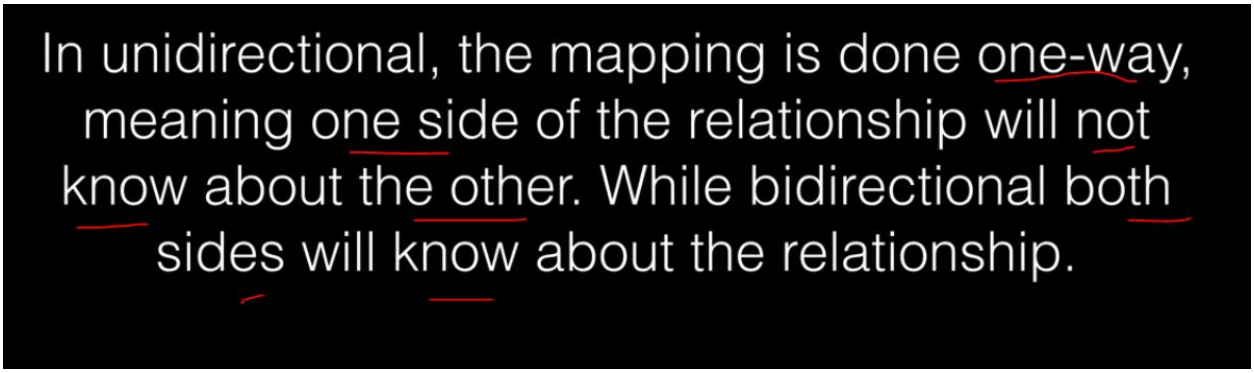
فیلد



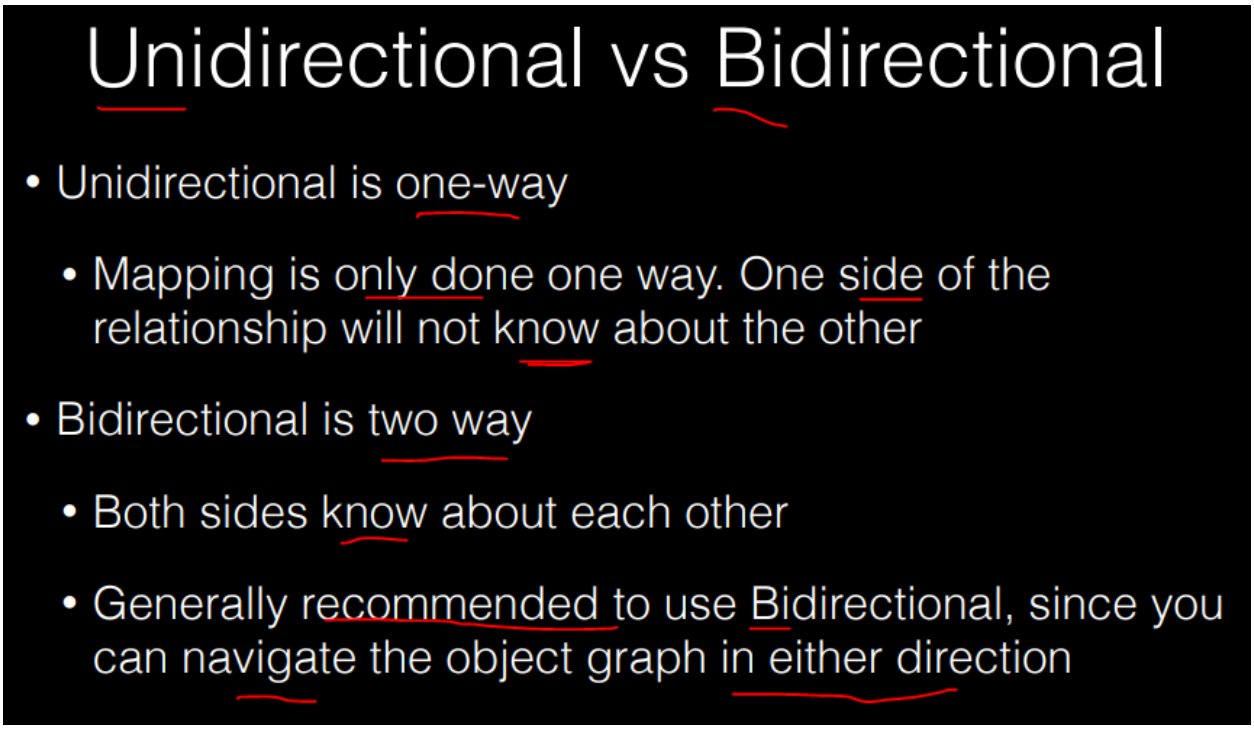
Directional(knowing, .get()**)/ @ for mapping**

Direction یعنی **knowing** داشتن یک entity از relation که با گذاشتن هر نوع از **@mapping** ایجاد میشود.

وقتی bi-directional هستیم، به معنای این است هر دو طرف میتوانند get کنند و relation را ببینند.



We can make Bidirectional mapping: for example, MTO and OTM at the same time



Fetch type (jpa) / def: owner= Eager

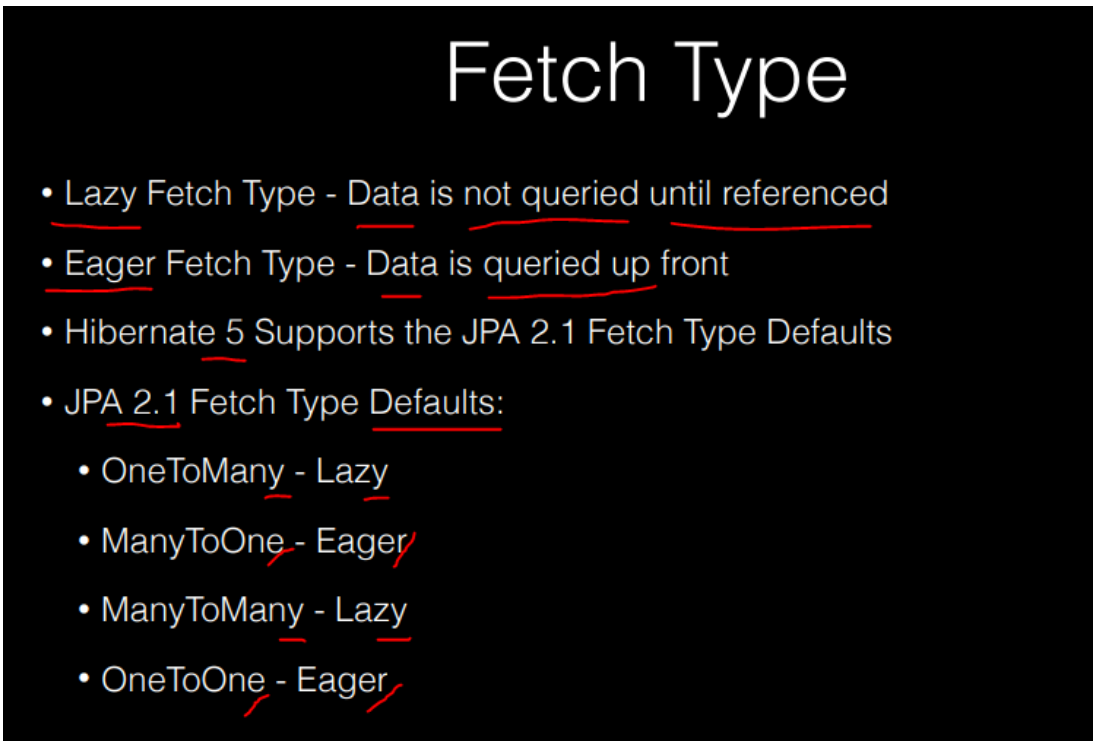
مقدار **ديفالت**:

برای entityهایی که **owner** هستند FT برابر **Eager** قرار داده شده است.(دیتای relation را هم فچ میکند)

و برای entityهایی که owner نیستند FT برابر Lazy قرار داده شده است.

چون برای entityای که owner نیست، اطلاعات در DB سمتش نیست، پس fetch کردنشان بار اضافی دارد.

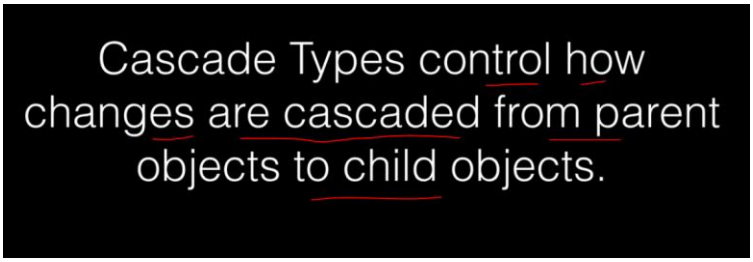
مقدار دیفالت قابل تغییر است



Cascade types (parent) (jpa)/

How changes are cascaded from parent to child.

رخداد **تغییر یک entity** در entity سمت دیگر رابطه تاثیر بگذارد یا نه

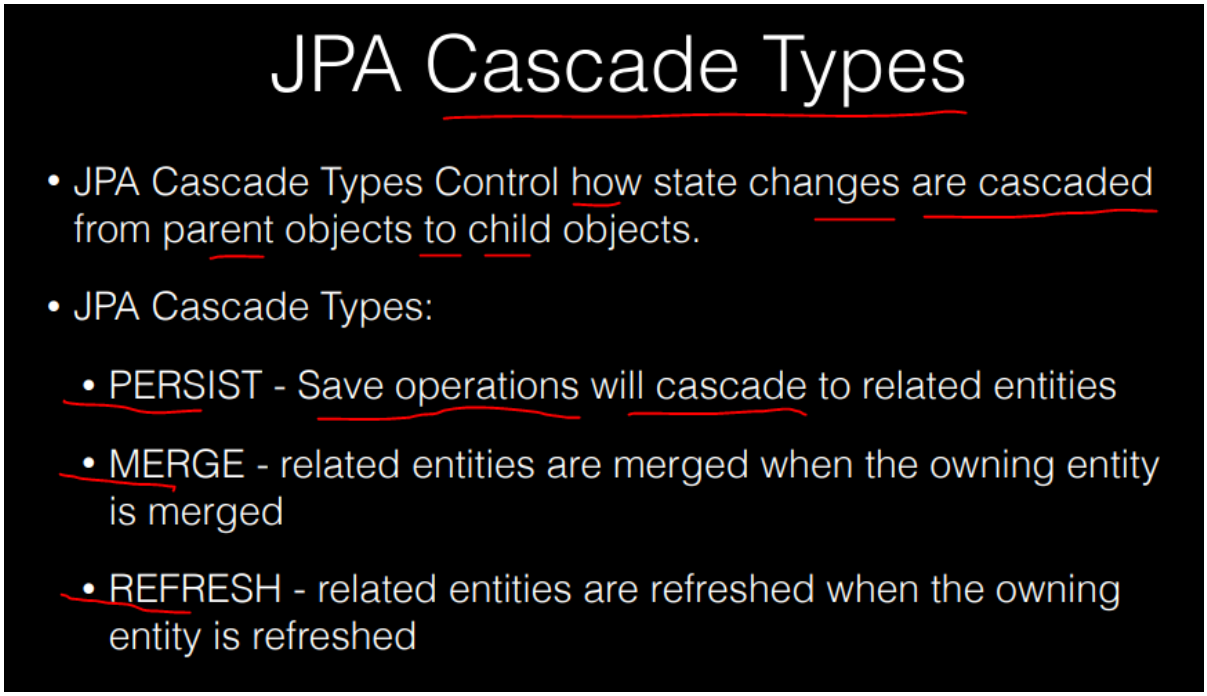


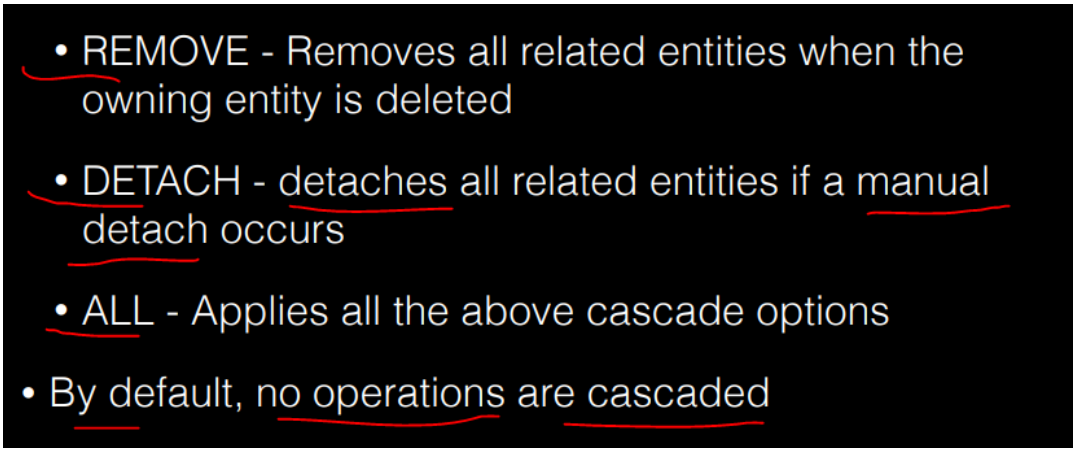
JPA بصورت default، cascade به **هیچکدام از entity**های یک relation داده نمیشود. یعنی همه ی اتفاقاتی که در اثر تغییر یک entity برای entity دیگر رخ میده باید دستی هندل شود.

Parent بودن در یک relation در گرو **cascade_type@**است.

براساس data model **طراح** تصمیم میگیرد.

مسیله مهم **اینه قدرت دست owner است** و چون دیتا را در DBاش دارد میتواند تغییرات را owned انتقال دهد، owned نمیتواند به تنهایی owner را تحت تاثیر قرار دهد.





Detach: No association with a hibernate session

یعنی ممکن است یک entity که owner نیست، parent باشد.؟ **بله**

پس میتوانیم cascade را برای non-owner side بگذاریم و آنرا parent قرار دهیم **ولی گاهی کار نمیکند**:

- **بستگی** به این دارد **چه نوع متدی** را برای propagate تغییرات در نظر گرفته ایم و در **چه mapping**ای

- اگر ارتباط **یک به چند** باشد **persist کردن با ملاحظاتی(owner را هم با کد مستقیم تغییر دهیم)** کار خواهد کرد:

مثلاً وقتی که می‌خواهیم یک رکورد به سمت one اضافه کنیم و همراهش لیستی از سمت many هم وارد کنیم: مثال بخش @One to Many & @ManyToOne

یا **remove :**

https://www.baeldung.com/spring-jpa-unidirectional-one-to-many-and-cascading-delete

- اگر ارتباط یک به یک باشد از طرف owned به owner ؛ **remove** و **persist** هیچ کدام کار نخواهند کرد:

https://coderanch.com/t/681648/databases/JPA-persist-cascading-related-entities

https://stackoverflow.com/questions/1534599/a-question-about-jpa-cascade-and-removing-entity

وقتی یک Entity قرار است بعنوان **child** با cascade، crud شود، **نیازی نیست** برایش جداگانه **repository** بنویسیم، مثلا نیازی نیست برای هر دو طرف I Repository (save, ...) بنویسیم و entity دیگر رو پاس بدیم. بلکه یک طرف حتما parent هست و cascade میکند اونیکی رو. **پس در setter از parent میشه entity دیگر رو set کرد** و نیاز به save در entity دوم نیست.

entityای که **owner نیست** (یعنی دیتای mapping را درDB ندارد) ، بهتر است (برای زیاد نشدن بار DB):

Fetch type را در حالت default نگه داریم (**lazy**)

Parent نباشد، یعنی Cascade بهش ندیم، مخصوصا اگر **fetch=lazy نگهشان میداریم.**

مشکل lazy با detach چیه؟

اگر دریک relation، owned را lazy نگه داریم ولی cascade آن را detach بگذاریم (غیر owner، parent شد) یعنی داریم مسؤلش میکنیم برو دیتاهای mapping رو از بین ببر. این بدبخت که حتی اجازه get رو با fetch type بهش ندادیم (lazy) چطوری بره دیتای mapping رو پاک کنه! پس با هم نمیسازند. ولی وقتی به آن entity که علارقم owner نبودن بهش fetch-type=eager دادیم، حالا میتونیم بهش نقش parent از نوع detach=cascade بگیرد.

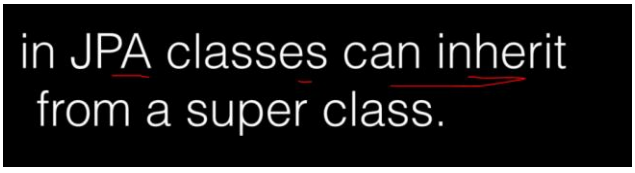
میتواند unidirectional هم باشد:

https://stackoverflow.com/questions/68565412/hibernate-onetoone-unidirectional-mapping-cascade-delete

Inheritance – embedded type (jpa & hibernate)

Relational **DB does not support inheritance**

But:



Embeddable Types

- JPA / Hibernate support embeddable types
- These are used to define a common set of properties
- For example, an order might have a billing address, and a shipping address
- An embeddable type could be used for the address properties

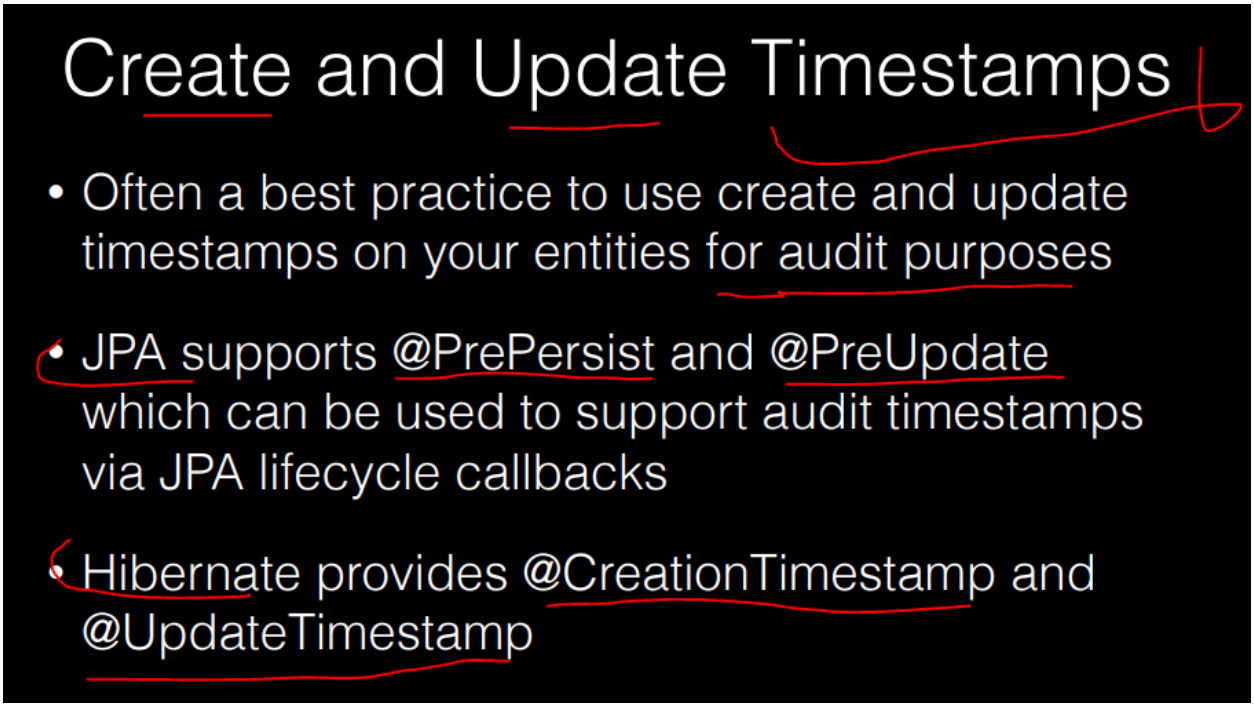
Inheritance

- MappedSuperclass - Entities inherit from a super class. A database table IS NOT created for the super class
- Single Table - (Hibernate Default) - One Table is used for all subclasses
- Joined Table - Base class and subclasses have their own tables. Fetching subclass entities require a join to the parent table
- Table Per Class - Each subclass has its own table

فرق آخری با اولی؟؟؟

Single Table can lead to a lot of unused database columns.

Timestamp (jpa & hibernate)



DDL-auto (spring to hibernate)

Hibernate: کارهای Data به سمت DB. مثل DDL و DML

JPA: کارهای Data به سمت App. مثل ساخت modelهای entityها (@ها)

DDL is used to define database structures such as tables and indexes. While DML is used with data operations such as inserts and updates

The ddl-auto property controls what if any DDL operations Hibernate will perform on startup.

Hibernate DDL Auto

- DDL = Data Definition Language
- DML = Data Manipulation Language
- Hibernate property is set by the Spring property spring.jpa.hibernate.ddl-auto
- Options are: none, validate, update, create, create-drop
- Spring Boot will use create-drop for embedded databases (hsql, h2, derby) or none

None: hibernate در زمان startup هیچ کاری نمیکند

Validate کردن: اگر مشکلی در DDL یا با JPA model ها در تضاد باشد، عملیات startup به مشکل میخورد

Update: اتوماتیک DDL (DB schema) را update میکند. در محیط production با احتیاط استفاده شود چون اگر مشکلی در تغییرات JPA model باشد DB آپدیت میشود و همه چی خراب میشه. (یوزرهای محیط production نباید قابلیت DLL داشته باشند.)

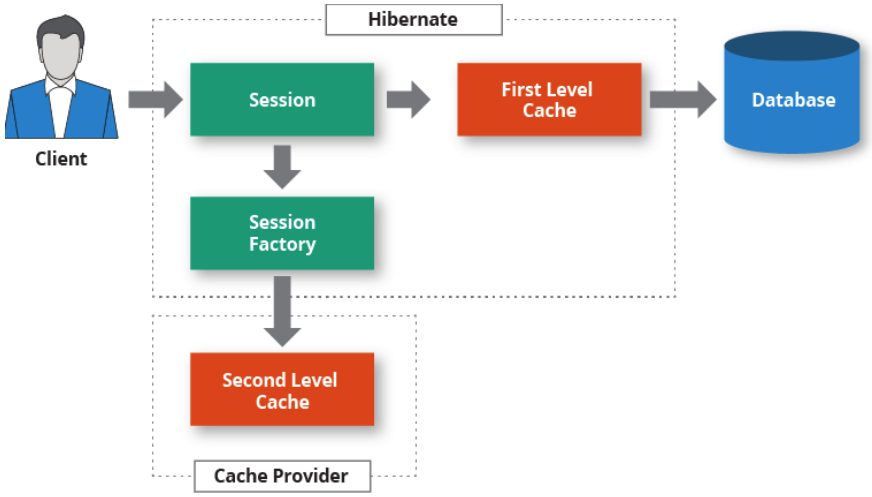
Create: initialize DB

Create-drop: initialize DB و وقتی DB خاموش شد همه چی drop میشه.

Sp boot دیفالت DB embedded H2 و Derby را create-drop قرار داده است.

Sp boot دیفالت را **برای DBهای غیر embedded**را **none** قرار داده است.

Cashing in hiber



:First level cache

ما هر بار که یک کوئری توی دیتابیس میزنیم نتیجش میاد **توی کش لایه ی یک** میشه

اگر **همون آبجکت** باز خواسته شد؛ **سشن نمیره** سمت دیتابیس اونو بیاره، از توی کش لایه یک میخونه. سرعت زیاد میشه

Default ، active است.

چندتا مشکل داره:

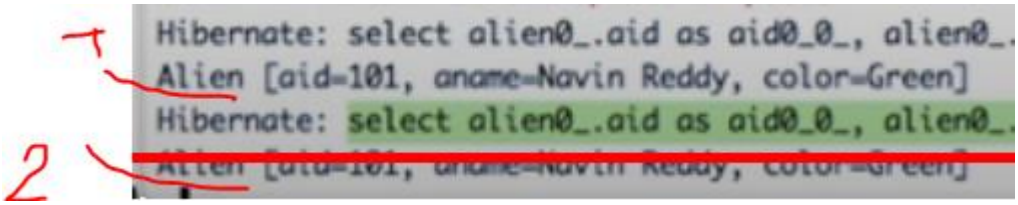
- **هر session، cache جداگانه خودش را دارد، یعنی فقط objectهایی که خودش** transaction کرده میتواند cache کند و بهشون دسترسی داره، برای درخواست مجدد اون objectها با همون session دیگه transaction جدید ایجاد نمیکنه.

این دیتا برای همون user در همان session، cache شده، هر user، میتواند تعداد بیشتری session داشته باشد.

User, app session, hiber session, new transaction,

- اگر objای که cache کرده توسط session دیگه ای **عوض شد**، نمیفهمه تا دوباره بره بگیرتش

```
SessionFactory sf = con.buildSessionFactory(reg);
Session session1 = sf.openSession();
session1.beginTransaction();

a = (Alien) session1.get(Alien.class, 101);
System.out.println(a);

session1.getTransaction().commit();
session1.close();

Session session2 = sf.openSession();
session2.beginTransaction();

a = (Alien) session2.get(Alien.class, 101);
System.out.println(a);

session2.getTransaction().commit();
session2.close();
```



:Second level cache

SLC آبجکت هایی که میخواهیم رو از FLC میگیرد و کش میکنیم و **بین همه سشن ها** به اشتراک میگذارد

به محض اینکه یکی ازین سشن ها تغییر کرد از FLC میخواهد تا آپدیتش کند

Default ، active نیست.

Second providers like: Redis و Ehcache, swan, OS,

Configures: Poem.xnl cfg.xml, @cashable, @cache(strategy)

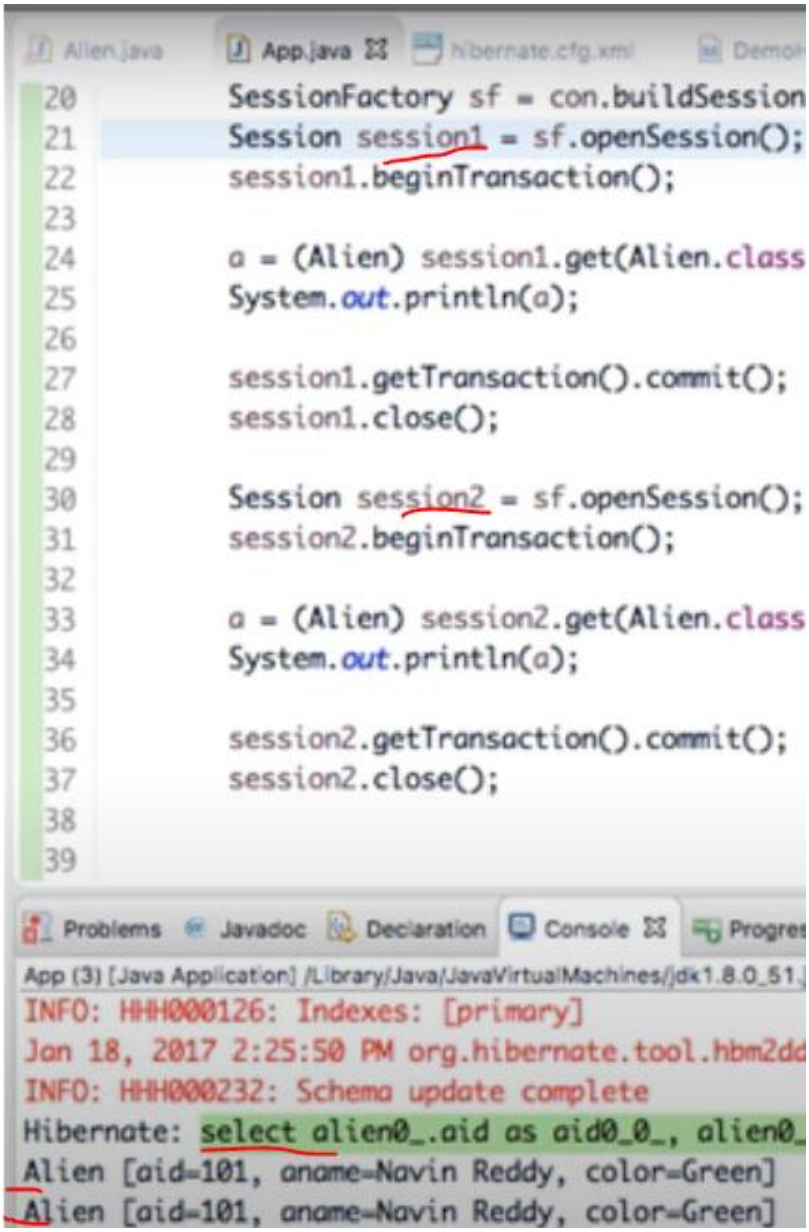
Redis cache را در فایل ردیس توضیح دادیم. ولی نه بعنوان hiber cache . چطور میتوانیم از redis بعنوان hiber cache استفاده کنیم؟؟؟

```xml
  <dependency>
      <groupId>net.sf.ehcache</groupId>
      <artifactId>ehcache</artifactId>
      <version>2.10.3</version>
  </dependency>
  <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-ehcache</artifactId>
      <version>5.2.6.Final</version>
  </dependency>
</dependencies>
```

en.java | App.java | hibernate.cfg.xml | DemoHib/pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3307/neon</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="show_sql">true</property>
        <property name="hibernate.cache.use_second_level_cache">true</property>
        <property name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>
    </session-factory>
</hibernate-configuration>
```

```java
@Entity
@Table(name="alien_table")
@Cacheable
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY)
public class Alien
{
    @Id
    private int aid;
    private String aname;
```



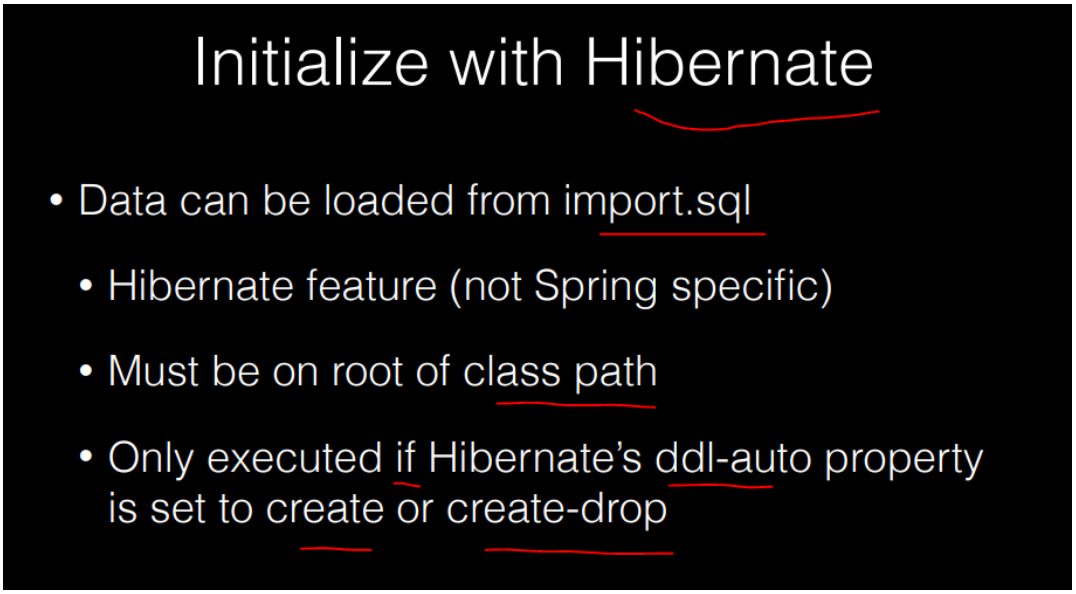
https://www.youtube.com/watch?v=TCHm1h7rBmo&list=PLsyeobzWxl7qBZtsEvp_n2A7sJs2MpF3r&index=19&pp=iAQB

https://www.youtube.com/watch?v=G6Y5wDF6h5Q

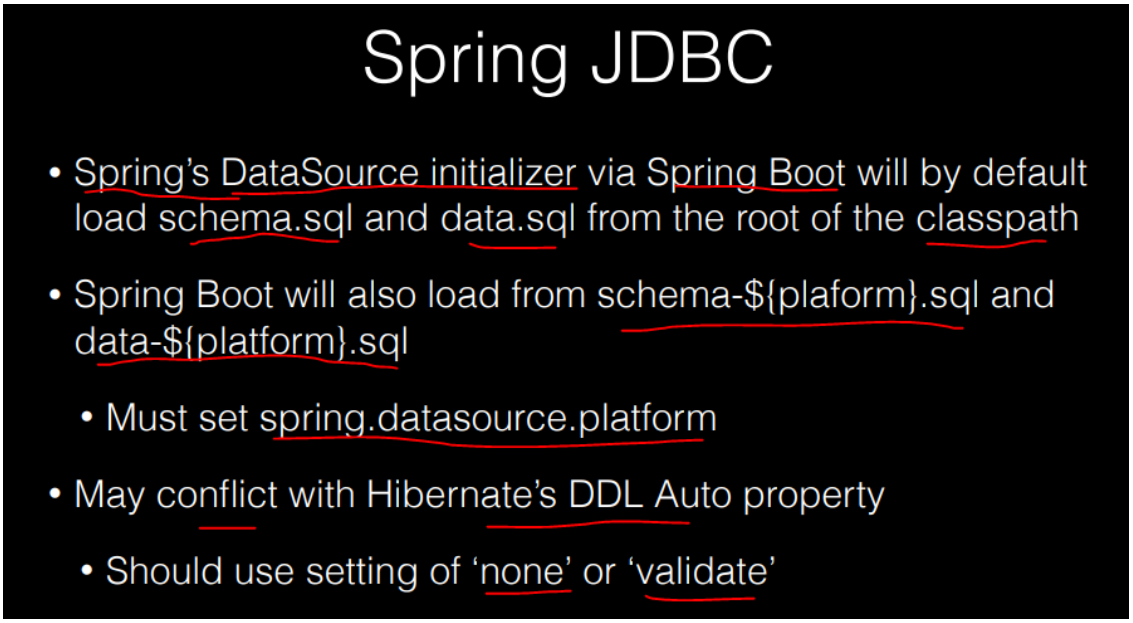
SQL Data loading in DB

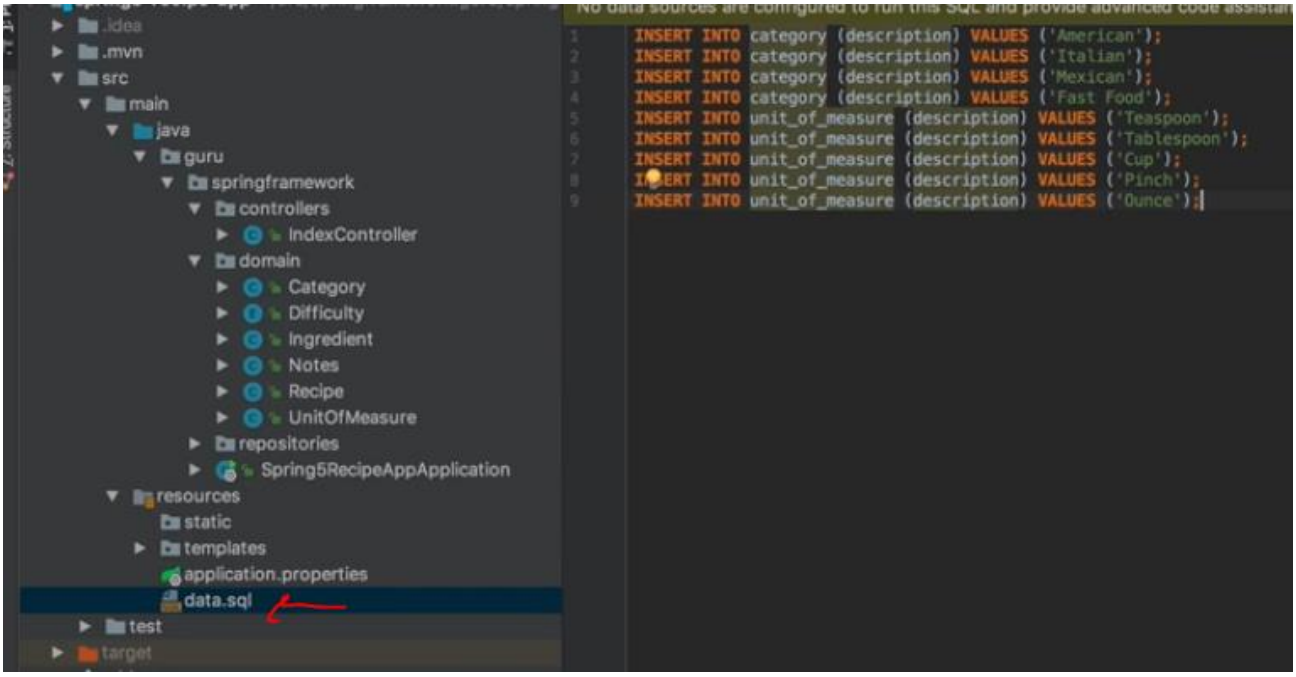
این دستورات میتواند شمال DDL و DML باشد.

به روش hibernate:



به روش Spring: این روش توصیه میشود.




.idea
.mvn
src
main
java
guru
springframework
controllers
IndexController
domain
Category
Difficulty
Ingredient
Notes
Recipe
UnitOfMeasure
repositories
Spring5RecipeAppApplication
resources
static
templates
application.properties
data.sql
test
target
INSERT INTO category (description) VALUES ('American');
INSERT INTO category (description) VALUES ('Italian');
INSERT INTO category (description) VALUES ('Mexican');
INSERT INTO category (description) VALUES ('Fast Food');
INSERT INTO unit_of_measure (description) VALUES ('Teaspoon');
INSERT INTO unit_of_measure (description) VALUES ('Tablespoon');
INSERT INTO unit_of_measure (description) VALUES ('Cup');
INSERT INTO unit_of_measure (description) VALUES ('Pinch');
INSERT INTO unit_of_measure (description) VALUES ('Ounce');

H2

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

App.pro

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:mehrad
```

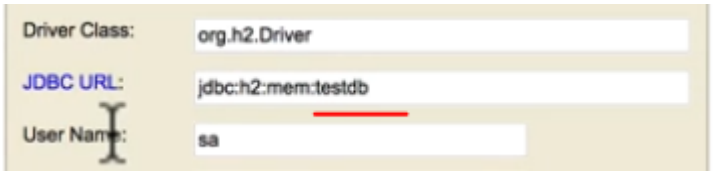
```
1 spring.h2.console.enabled=true
2 spring.datasource.platform=h2
3 spring.datasource.url=jdbc:h2:mem:navin
```

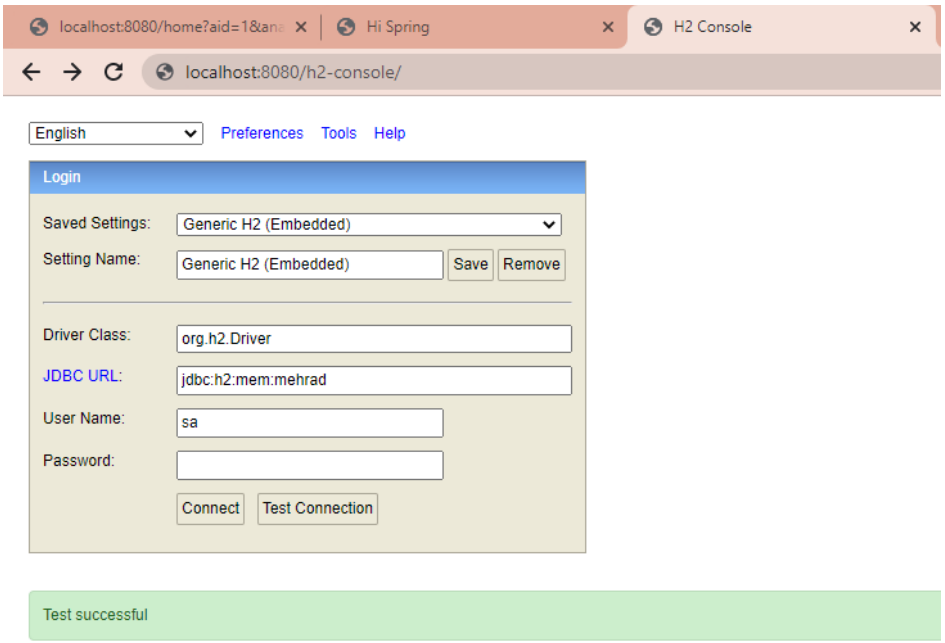
Just it:

```
spring.h2.console.enabled=true
```

Run → http://localhost:8080/h2-console/

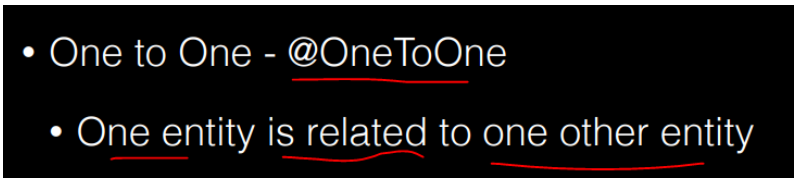
Default JDBC URL when use sp boot for H2



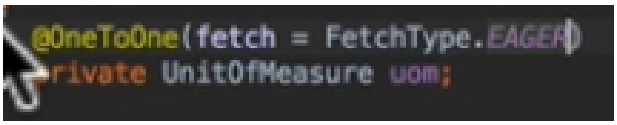

localhost:8080/home?aid=1&an
Hi Spring
H2 Console
localhost:8080/h2-console/
English
Preferences
Tools
Help
Login
Saved Settings:
Generic H2 (Embedded)
Setting Name:
Generic H2 (Embedded)
Save
Remove
Driver Class:
org.h2.Driver
JDBC URL:
jdbc:h2:mem:mehrad
User Name:
sa
Password:
Connect
Test Connection
Test successful

Mappings

@One to One / زن و شوهر متمدن



وقتی که میخواهیم یک table بزرگ **normalize** شود به دو table کوچکتر



نوع uni-directional :

این entity ذاتا **owner** ک هست، بصورت دیفالت **eager** هم هست، میتواند parent هم در نظر گرفته شود.

اگر owner را parent تعریف کنیم: در **setter**اش: entity دومی رو میگیرد و درtableاش save میکند (کلید خارجی را set میکند.)

Eager را بهتر است برای خوانایی کد بنویسیم.

طرف entity دیگر OTO نمیدهیم، اون طرف در DB اصلا فیلد این entity رو ندارد و در سطح app هم خبر ندارد که در relation با entity دیگری قرار دارد.

bi-directional:

پس هر دو از وجود هم باخبر هستند و میتوانند با **get** و **dot** هم را بگیرند.

هر دو طرف **owner** هستند، **eager** هستند

اگر parent تعریف نکنیم: برای هر دو I repo و setter جداگانه مینویسیم.

اگر **parent تعریف کنیم**: اونی که **parent** است میتواند در **setter**اش: entity دومی رو بگیره تا آنرا درtableاش save کند (کلید خارجی خودش را set کند) و خودش را بدهد به آن تا او هم save کند در کلید خارجی خودش.

recipe که Parent است، در setter خود برای هر دو entity، کلیدهای خارجی را ثبت میکند. (فعلا در ram هستیم)

```
@Data
@Entity
public class Recipe {

    @OneToOne(cascade = CascadeType.ALL)
    private Notes notes;

public void setNotes(Notes notes) {
    if (notes != null) {
        this.notes = notes;
        notes.setRecipe(this);
    }
}
```

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

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToOne
    private Recipe recipe;
```

استفاده از setter:

```
final Recipe recipe = new Recipe();
recipe.setId(source.getId());
recipe.setCookTime(source.getCookTime());
recipe.setPrepTime(source.getPrepTime());
recipe.setDescription(source.getDescription());
recipe.setDifficulty(source.getDifficulty());
recipe.setDirections(source.getDirections());
recipe.setServings(source.getServings());
recipe.setSource(source.getSource());
recipe.setUrl(source.getUrl());
recipe.setNotes(notesConverter.convert(source.getNotes()));
```

obj Parent وقتی از I repo استفاده میکند ، در DB: **هم خودش save میشود، هم فرزند را save میکند.**

به همین خاطر میگوییم نیاز نیست فرزند I repo داشته باشد، setter و repo پدر کفایت میکند.

@One to Many & @ManyToOne / پدر و فرزندها

پدر لیستی از فرزندهایش دارد. یک entity از نوع (پدر)، با لیستی از entityهای نوع دیگر (فرزندها) در ارتباط است.

@OneToMany **فقط میتواند bi-directional** باشد: هر دو از وجود رابطه آگاه هستند.

- One to Many - @OneToMany
  - One entity is related to many entities (List, Set, Map, SortedSet, SortedMap)

@OneToMany: (پدر)

(فرزندها) در کلاس (پدر) add میشوند.

باید یک many ای باشد که در DB اطلاعات را نگه دارد، بنابراین: (بایدی نیست، uni هم میتواند باشد مثالش پایینتر هست)

@ManyToOne: (فرزندها)

میدانیم که many اطلاعات relation را در tableاش دارد، **Owner است.**

owner، بصورت دیفالت **eager** است.

چون bi-directional هستیم هر دو طرف میتوانند get و set کنند و از هم آگاه باشند.

Owner: (فرزندها)

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

    new *
    public Ingredient(String description, BigDecimal amount, UnitOfMeasure uom) {
        this.description = description;
        this.amount = amount;
        this.uom = uom;
    }

    @ManyToOne
    private Recipe recipe;
```

```
ingredient.getRecipe().getId()
```

روش یک: many (فرزندها) اگر چه owner است ، ولی خودش حتی I repo جداگانه نمیخواهد.

در این مثال **بخاطر consept**، **(پدر)** را اگرچه **owned** است، **parent** تعریف کرده ایم cascade=all

Setter (فرزندها)، در داخل متد add (پدر) استفاده میشود:

```
ingredient.setRecipe(this);
```

Setter توسط Lombok هست. ولی یک **متد add** هم دارد که عملیات ساخت relation در (پدر)انجام میدهد.

```
@Data
@Entity
public class Recipe {

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

    30 usages   new *
    public Recipe addIngredient(Ingredient ingredient){
        ingredient.setRecipe(this);
        this.ingredients.add(ingredient); // add is a method from Set Collection
        return this;
    }
```

```
tacosRecipe.addIngredient(new Ingredient( description: "Ancho Chili Powder", new BigDecimal( val: 2), tableSpo
tacosRecipe.addIngredient(new Ingredient( description: "Dried Oregano", new BigDecimal( val: 1), teapoonUom));
```

+ Recipe

تا اینجا objهای (پدر)و (فرزندها) را در ram داریم.

در نهایت با **I repo(پدر) هر دو obj در relation ، save میشوند**. Cascade=all

حتی برای اضافه شدن یک فرزند جدید به لیست فرزندهای یک پدر که از قبل هست، باز هم به سراغ repo پدر میاییم، ولی احتمالا هزینه اش بالا باشد، پس برای اینکار روش پایین فکر کنم بهتر است:

روش دو:

اگر برای Entity Child یک Repository جداگانه نوشتیم (میتوانیم ننویسیم و saveاش با parentاش باشد)، در هنگام save این Entity در Repository **باید چک شود** که اگر parent، null بود exception برگرداند:

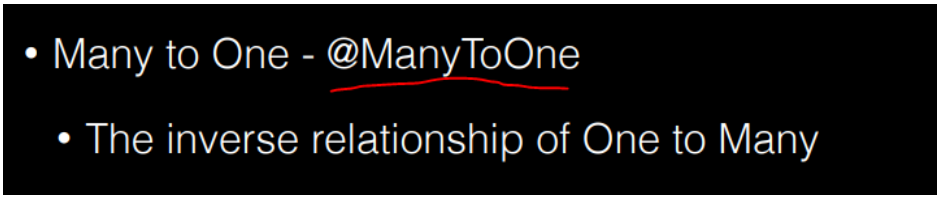
```
@Override
public Visit save(Visit visit) {

    if(visit.getPet() == null || visit.getPet().getOwner() == null || visit.getPet().getId() == null
        || visit.getPet().getOwner().getId() == null){
        throw new RuntimeException("Invalid Visit");
    }

    return super.save(visit);
}
```

یک مثال برای استفاده unidirectional از OTO !!! و OrphanRemoval

https://www.baeldung.com/spring-jpa-unidirectional-one-to-many-and-cascading-delete

@Many to One / Pet & Type (uni,bi) - inverse OTM

این Entity، در هر صورت owner است.



سه نوع کاربرد دارد:

- **bi-directional** ، **Inverse** یک **@OneTOMany** ، مثل رابطه (پدر) و (فرزندها).
  که many (فرزندها) میشد owner و parent؛ در بخش OneTOMany توضیح داده شد.

- **Uni-directional** ، **@ManyToOne تنهایی** بیاید (مثل **Pet** (many) و (**type**(one)

  سمت many@، برای آگاهی کافی است. **owner** هم است یعنی در DB ذخیره میشود.

یک relation، وقتی uni-directional است، یعنی **parent ای وجود نخواهد داشت**(cascade نمیشود).

```java
@Entity
@Table(name = "pets")
public class Pet extends BaseEntity{

    1 usage
    @ManyToOne
    @JoinColumn(name = "type_id")
    private PetType petType;
```

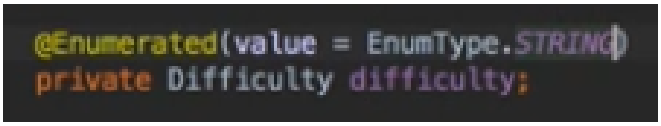
```java
Pet mikesPet = new Pet();
mikesPet.setPetType(savedDogPetType);
```

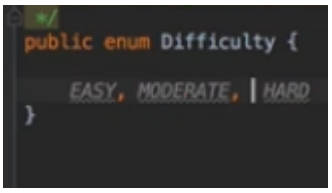
اگر برای این side که owner هم هست cascade تعریف کنیم فکر کنم کار owned save را هم انجام میدهد.

قسمت One رابطه، نیازی به دانش اینکه چه لیستی از entityهای Many با آن در ارتباط است ندارد پس **حتی فیلدی از entity دیگر و @map نمیگیرد**؛ و Setterای برای set کردن entity دیگر نخواهد داشت.

```java
@Table(name = "types")
public class PetType extends BaseEntity {

    @Builder
    public PetType(Long id, String name) {
        super(id);
        this.name = name;
    }

    @Column(name = "name")
    private String name;


    @Override
    public String toString() {
        return name;
    }
}
```

این نوع @ManyToOne میتواند با @Enumerated جایگزین شود:





- **bi-directional** ، **@ManyToOne** که entityای که **many** است: هم **owner** است هم **parent**

در این مثال بر خلاف قسمت:

پدر و فرزندها / @One to Many & @ManyToOne

Consept طوری است که parent سمت many قرار میگیرد.

همون رابطه ی type و pet به نحوی که اگر pet جدیدی آمد که type های قبلی بهش نمیخورد، همزمان با pet save ، type جدید هم save شود.

؟؟؟ مثال رو از نت دربیار

Many to Many / 2

- Many to Many - @ManyToMany
  - Many entities are related to many entities
  - Each has a List or Set reference to the other
  - A join table is used to define the relationships

وقتی از دو طرف oneToMany باشد.

وقتی دو طرف @MTM میذاریم بدون mappedBy، دوتا table واسط میسازد. با mappedBy میگوییم این فیلدی که روش @ میزنیم با چه فیلدی از side دیگر قرار است join بخورد.

(mappedBy در OTO و MTM معانی متفاوتی دارد)

میتواند uni-directional و Bi-directional باشد.

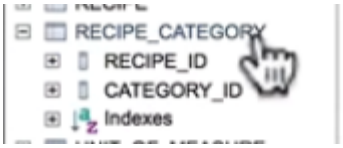
- **Bi-directional: Product and category**

```java
@Data
@Entity
public class Recipe {

    @ManyToMany
    @JoinTable(name = "recipe_category",
            joinColumns = @JoinColumn(name = "recipe_id"),
            inverseJoinColumns = @JoinColumn(name = "category_id"))
    private Set<Category> categories = new HashSet<>();
```

```java
@Data
@EqualsAndHashCode(exclude = {"recipes"})
@Entity
public class Category {

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

هر کدام از Entityها جداگانه setter و getter خود را دارند برای entiry دیگر.

```java
tacosRecipe.getCategories().add(americanCategory);
tacosRecipe.getCategories().add(mexicanCategory);
```



- **uni-directional: <u>Vet & Speciality</u>**

```java
@Entity
@Table(name = "vets")
public class Vet extends Person {

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "vet_specialties", joinColumns = @JoinColumn(name = "vet_id"),
            inverseJoinColumns = @JoinColumn(name = "speciality_id"))
    private Set<Speciality> specialities = new HashSet<>();
}
```

```java
@Entity
@Table(name = "specialties")
public class Speciality extends BaseEntity {

    @Column(name = "description")
    private String description;

}
```

Speciality نیازی به داشتن اطلاعات Vet ها ندارد.

```
Vet vet1 = new Vet();
vet1.setFirstName("Sam");
vet1.setLastName("Axe");
vet1.getSpecialities().add(savedRadiology);

vetService.save(vet1);
```

Lombok

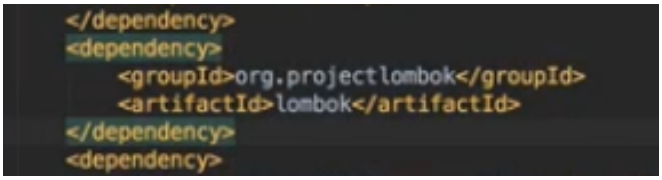
Installation

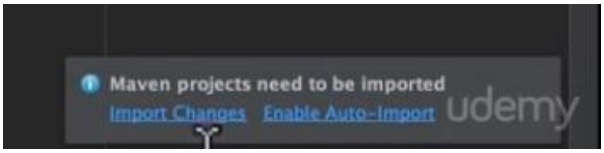
How Lombok Works

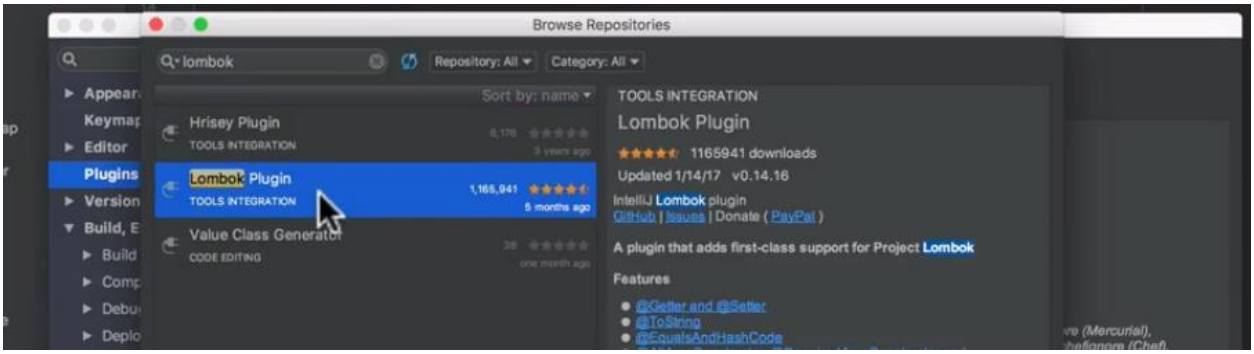
- Hooks in via the Annotation processor API
- The AST (raw source code) is passed to Lombok for code generation before the Java complier continues
- Thus, produces properly compiled Java code in conjunction with the Java compiler
- Under the 'target/classes' you can view the complied class files
- IntelliJ will decompile to show you the source code

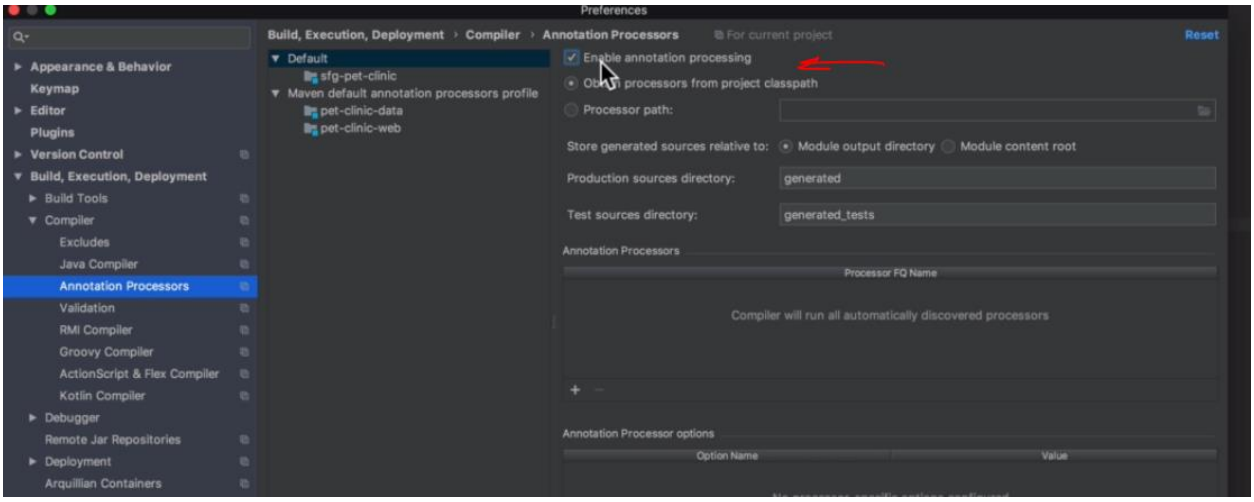
Project Lombok and IDEs

- Since complied code is change, and source files are not, IDE's can get confused by this.
- More of an issue for IDEs several years old.
- Modern IDEs such as IntelliJ, Eclipse (and offshoots), Netbeans support Project Lombok
- Plugin Installation may be necessary
- IntelliJ
  - Verify you've enabled 'annotation' processing under compiler settings









Facilities

Project Lombok Features

- Features as of **Version 1.18.8** - Released on May 7th, 2019
  - Check official documentation for new features
- **val** - local variables declared final
- **var** - mutable loval variables
- **@NonNull** - Null check, will throw NPE if parameter is null
- **@Cleanup** - Will call close() on resource in finally block

- **@Getter** - Creates getter methods for all properties
- **@Setter** - Creates setter for all non-final properties
- **@ToString**
  - Generates String of classname, and each field separated by commas
  - Optional parameter to include field names
  - Optional parameter to include call to the super toString method

- **@EqualsAndHashCode**
  - Generates implementations of equals(Object other) and hashCode()
  - By default will use all non-static, non-transient properties
  - Can optionally exclude specific properties

- **@NoArgsConstructor**
  - Generates no args constructor
  - Will cause compiler error if there are final fields
  - Can optionally force, which will initialize final fields with 0 / false / null
- **@RequiredArgsContructor**
  - Generates a constructor for all fields that are final or marked @NonNull
  - Constructor will throw a NullPointerException if any @NonNull fields are null

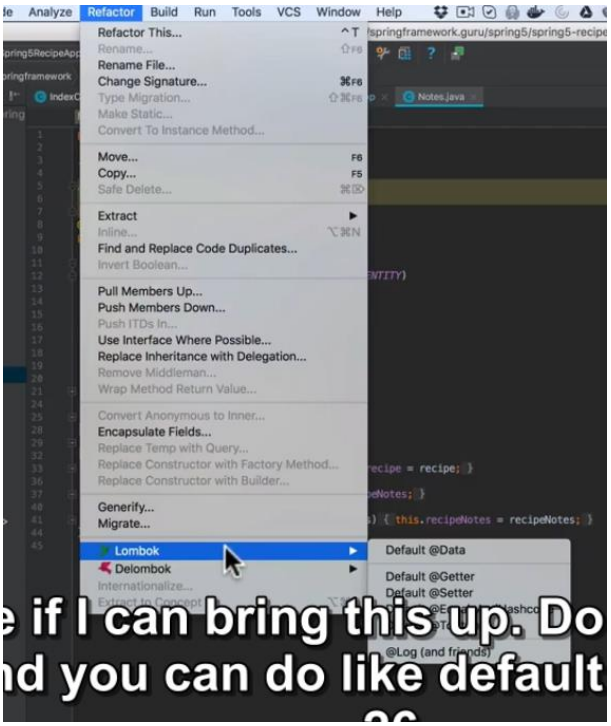
- **@AllArgsConstructor**
  - Generates a constructor for all properties of class
  - Any **@NotNull** properties will have null checks
- **@Data**
  - Generates typical boilerplate code for POJOs
  - Combines - **@Getter**, **@Setter**, **@ToString**, **@EqualsAndHashCode**, **@RequiredArgsConstructor**
  - No constructor is generated if constructors have been explicitly declared

- **@Value**
  - The immutable variant of @Data
  - All fields are made private and final by default
- **@NonNull**
  - Set on parameter of method or constructor and a NullPointerException will be thrown if parameter is null

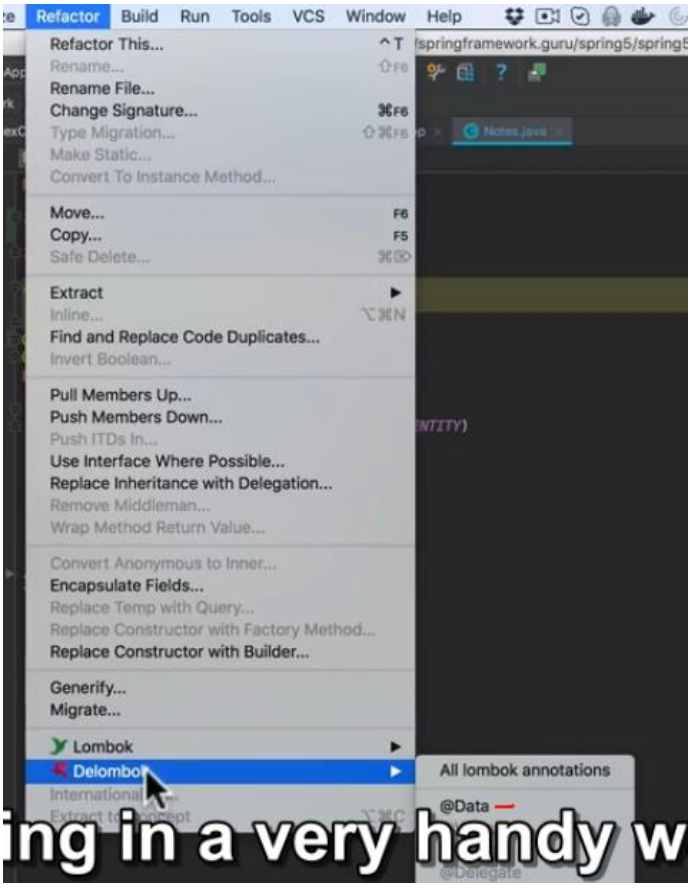
- **@Getter(lazy = true)** - for expensive getters
  - Will calculate value first time and cache
  - Additional gets will read from cache
- **@Log -** Creates a Java util logger - *Java util loggers are awful*
- **@Slf4j** - Creates a SLF4J logger.
- Recommended - SLF4J is a generic logging facade
- Spring Boot's default logger is LogBack

Refactoring

Modify(refactor) your present class:



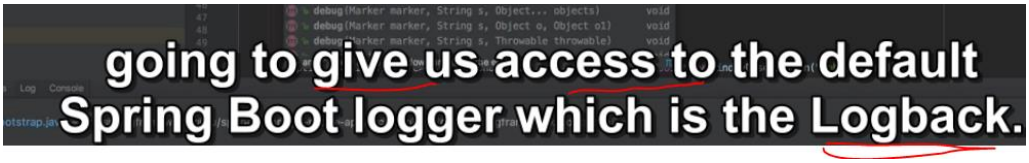
Look the impl from Lombok:



Slf4j logger

```java
@Slf4j
@Service
public class RecipeServiceImpl implements RecipeService {

    private final RecipeRepository recipeRepository;

    public RecipeServiceImpl(RecipeRepository recipeRepository) { this.recipeRepository = recipeRepository; }

    @Override
    public Set<Recipe> getRecipes() {
        log.debug("I'm in the service");

        Set<Recipe> recipeSet = new HashSet<>();
        recipeRepository.findAll().iterator().forEachRemaining(recipeSet::add);
        return recipeSet;
    }
}
```

???



endless loop

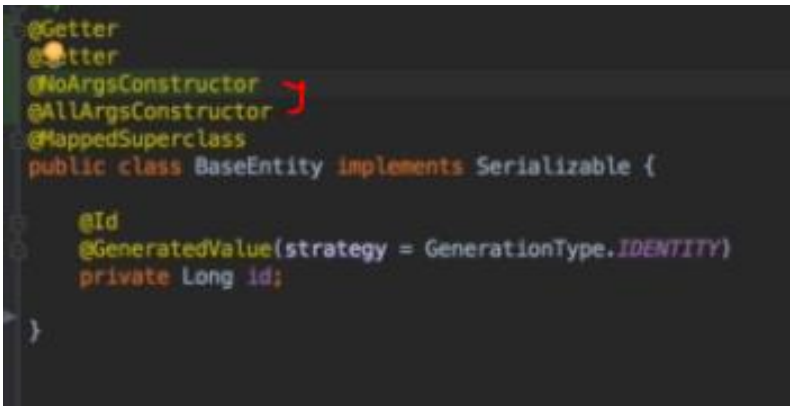
When we have **bi-directional relationship**, if we use **@Data**:

Creating **hash-code** have problem (**endless loop**), because of circular reference

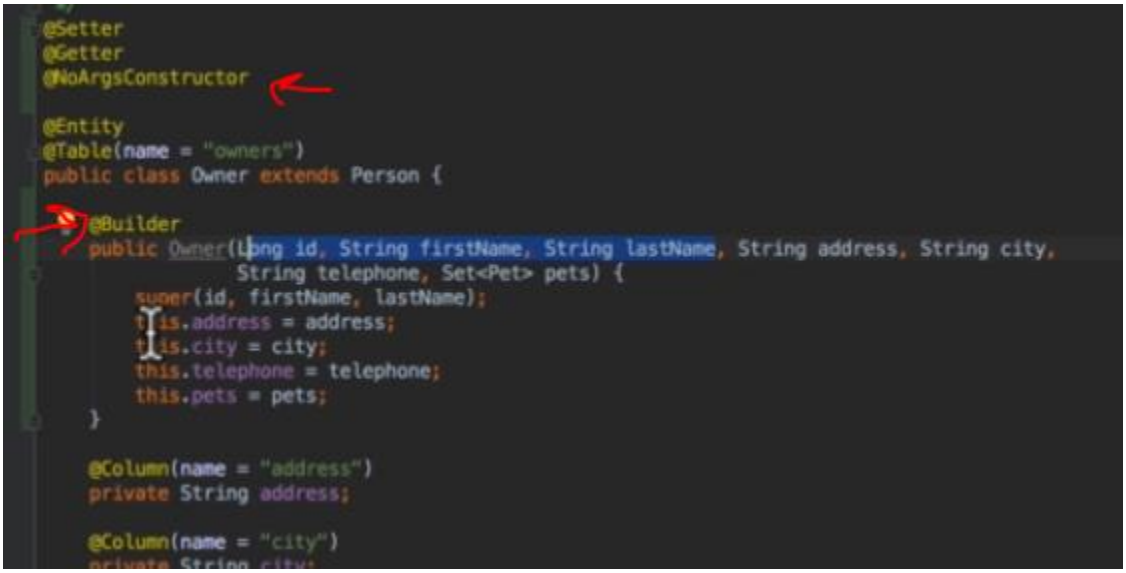
So we **should exclude some properties** from creating hashCode in **at least in one side** of relationship:

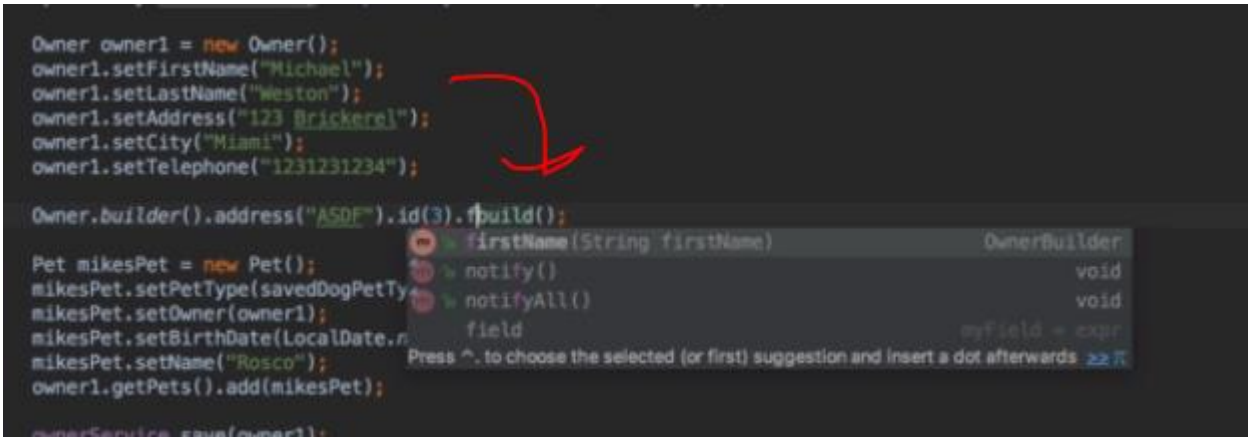
```java
@Data
@EqualsAndHashCode(exclude = {"recipes"})
@Entity
public class Category {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String description;

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

}
```

@Builder


@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@MappedSuperclass
public class BaseEntity implements Serializable {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
}


@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@MappedSuperclass
public class Person extends BaseEntity {
public Person(Long id, String firstName, String lastName) {
super(id);
this.lastName = lastName;
}
@Column(name = "first_name")
private String firstName;
@Column(name = "last_name")
private String lastName;
}

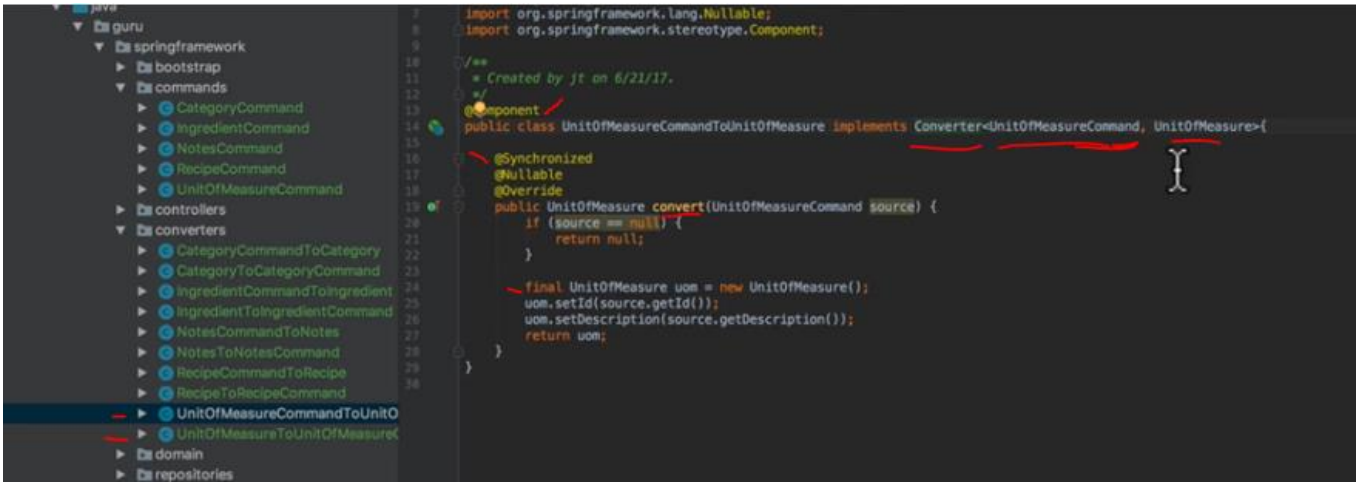

@Setter
@Getter
@NoArgsConstructor
@Entity
@Table(name = "owners")
public class Owner extends Person {
@Builder
public Owner(Long id, String firstName, String lastName, String address, String city,
String telephone, Set<Pet> pets) {
super(id, firstName, lastName);
this.address = address;
this.city = city;
this.telephone = telephone;
this.pets = pets;
}
@Column(name = "address")
private String address;
@Column(name = "city")



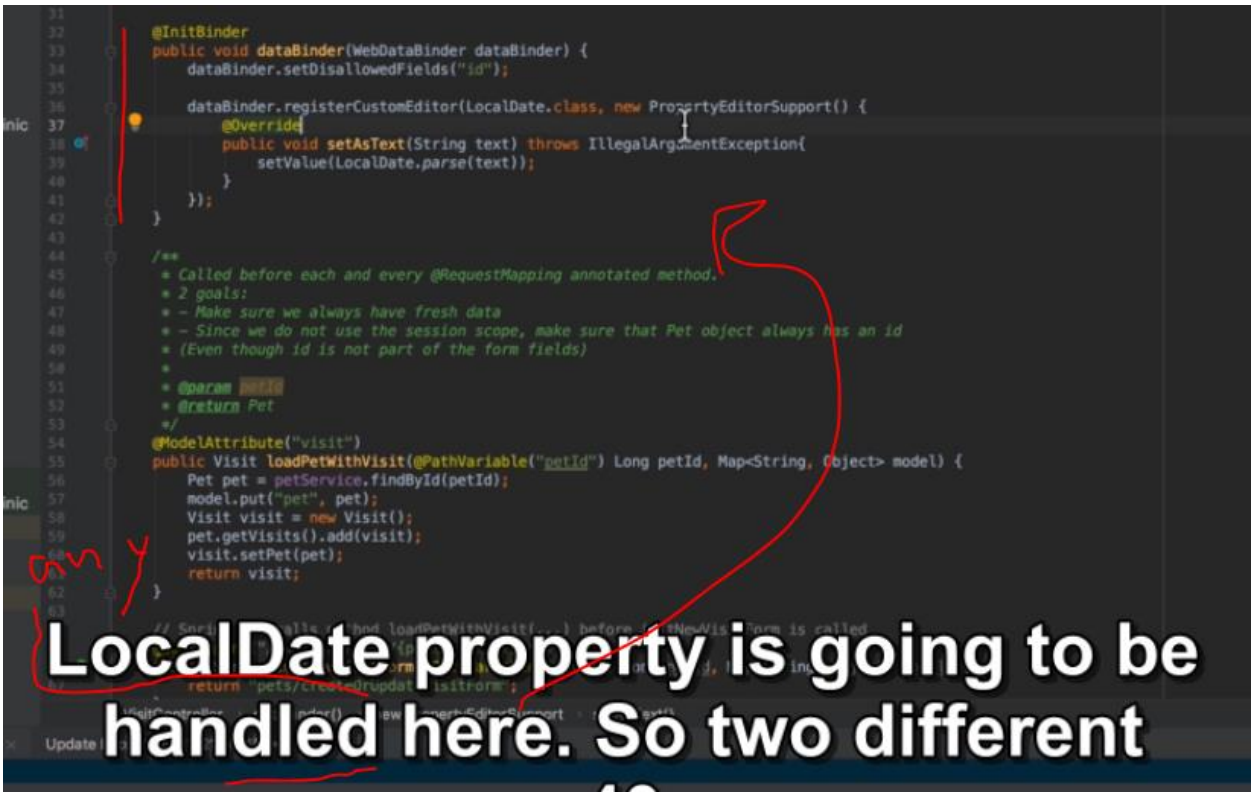
Lombok's synchronizes

Spring doesn't guarantee thread safety.

Lombok's synchronizes and thread safe methods. So we can use them in a multi-thread app.



LocalDate



Spring EventPublisher

In springboot/ based on AOP

```
@SpringBootApplication
public class ObserverApplication {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Obse
        Subject bean = context.getBean(Subject.class);
        bean.notifyAllShutdown();
    }
}

class ShutDownEvent{

}

@Component
class Subject{

    @Autowired
    ApplicationEventPublisher applicationEventPublisher;

    public void notifyAllShutdown(){
        applicationEventPublisher.publishEvent(new ShutDownEvent());
    }
}
```

```java
@Component
class Observer{

    @EventListener
    public void tellMeWhenSystemIsShuttingDown(ShutDownEvent shutDownEvent){
        System.out.println("System Shutting down 1");
    }

    @EventListener
    public void update(ShutDownEvent shutDownEvent){
        System.out.println("System Shutting down 2");
    }
}
```

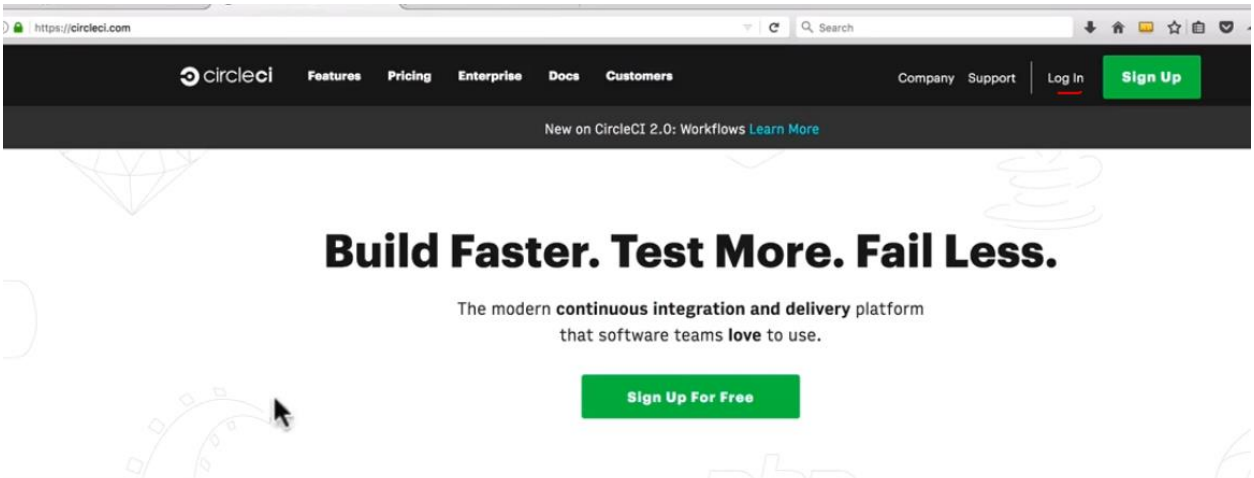
CircleCI

Stateless system ؟؟؟

24. Spring Pet Clinic –

Con. IT CircleCI

12. Continuous Integration Testing with Circle CI

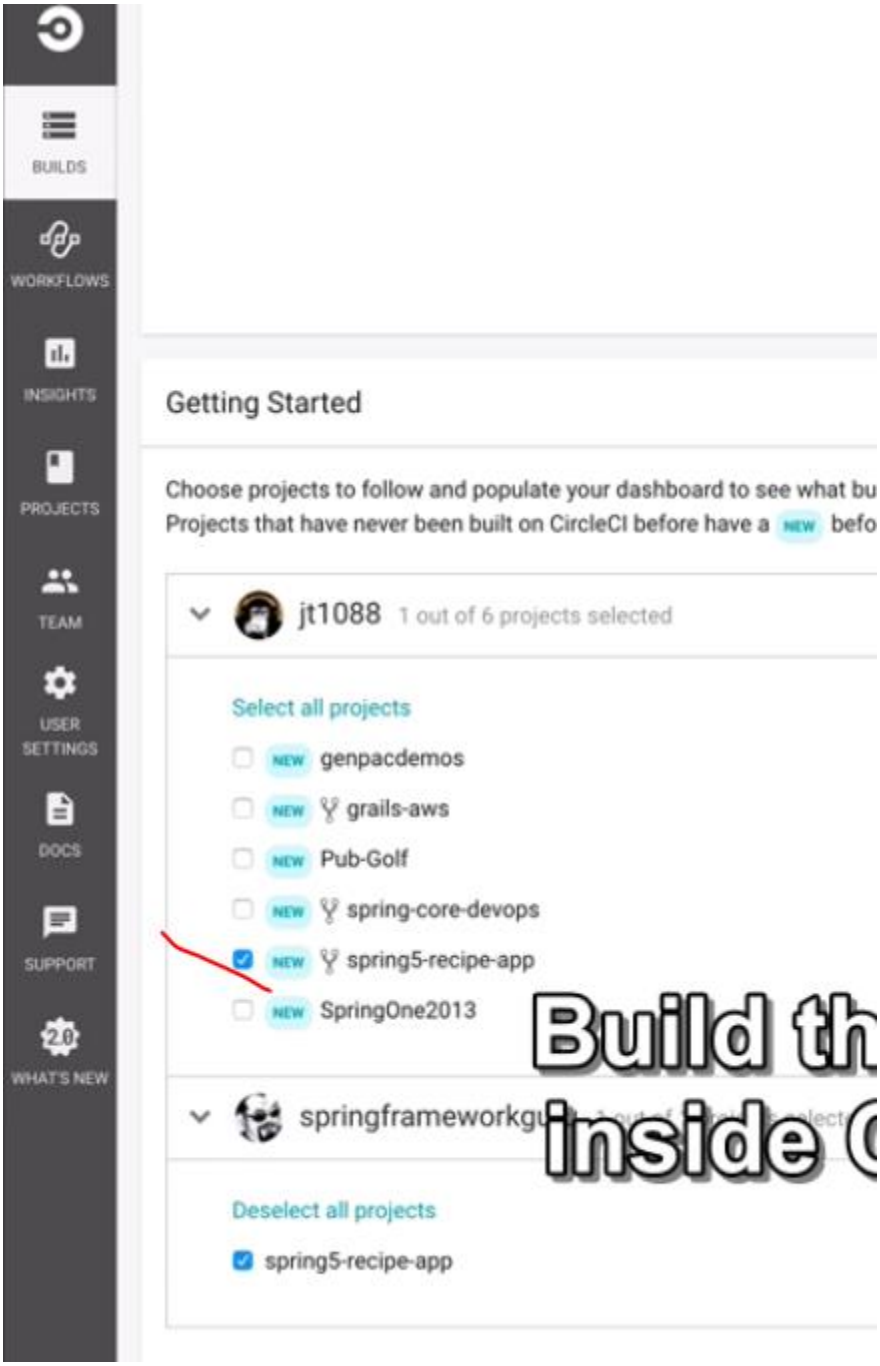


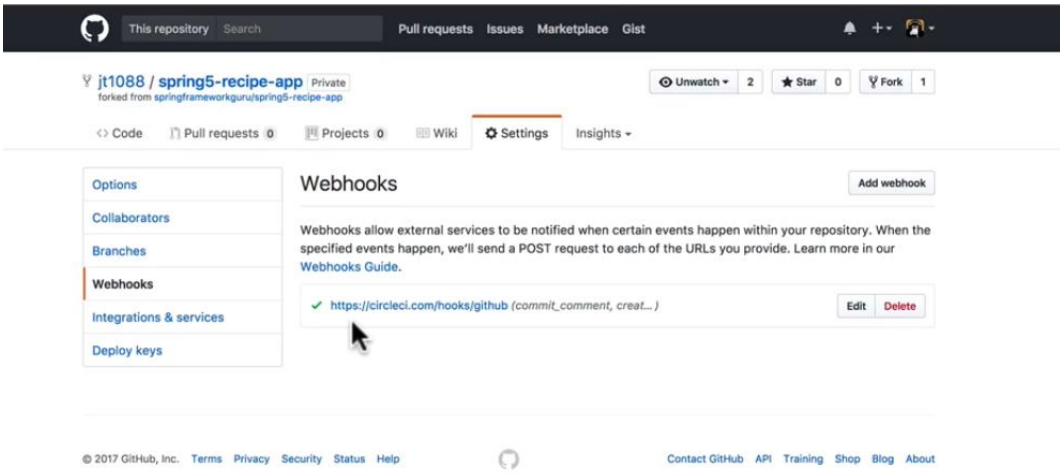
Sign In To CircleCI

You'll be taken to GitHub, Bitbucket, or Google to authenticate so you can start shipping f

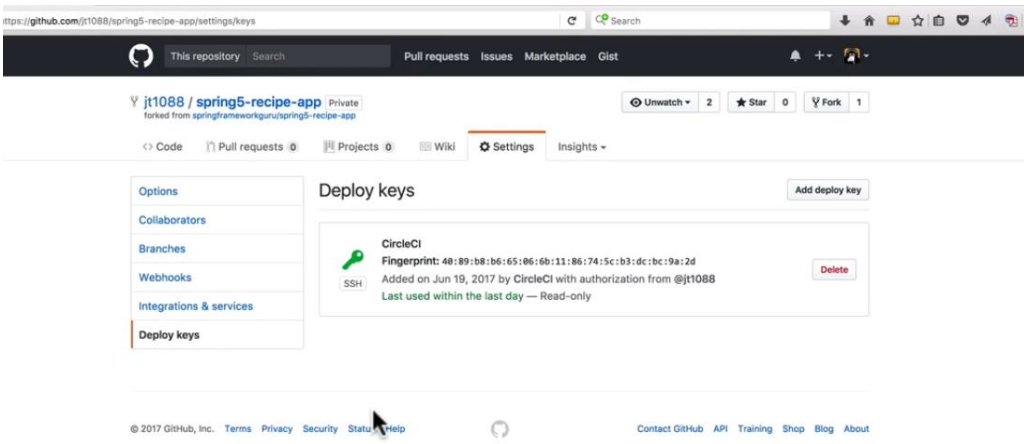
Looking to create a new account?

By clicking any sign in button above, you are agreeing to our Terms of Service and Privacy Policy.


BUILDS
WORKFLOWS
INSIGHTS
PROJECTS
TEAM
USER SETTINGS
DOCS
SUPPORT
WHAT'S NEW
Getting Started
Choose projects to follow and populate your dashboard to see what bu
Projects that have never been built on CircleCI before have a NEW befo
jt1088 1 out of 6 projects selected
Select all projects
NEW genpacdemos
NEW grails-aws
NEW Pub-Golf
NEW spring-core-devops
NEW spring5-recipe-app
NEW SpringOne2013
Build th
inside
Deselect all projects
spring5-recipe-app



that CircleCI automatically configured a webhook for you. So that's how it's 92



Deploy keys. So CircleCI also created its own SSH key back to GitHub. So these 95

19. Spring Pet Clinic - CI with CircleCI

20. Spring Pet Clinic - CircleCI Build Badge

Versions

6.0

Spring Framework

Release	Released	Latest
6.0	5 months ago (16 Nov 2022)	**6.0.8** (13 Apr 2023)

Spring Boot 3.0.6 available now

Hibernate is free software that is distributed under the GNU Lesser Genera
2.1.

...

Hibernate (framework)

Developer(s)	Red Hat
Initial release	23 May 2001
Stable release	6.1.4.Final / October 5, 2022

v1. 18.26 (Feb 3rd, 2023)

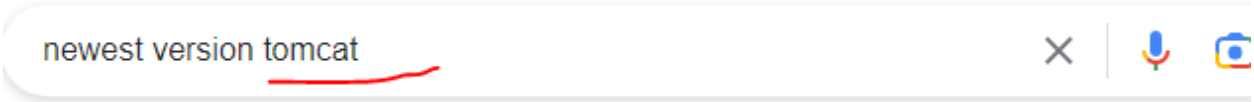
Project Lombok
https://projectlombok.org › changelog

About 918,000,000 results (0.49 seconds)

JUnit 5

JUnit 5 is the latest generation of JUnit. It includes focussing on Java 8 and above with the facility of allowing various types of testing for developers on JVM. The latest release of the JUnit 5 generation is 5.7. 1 which was released in February 2021.

Apache Maven 3.9.1 is the latest release all users.



About 29,800,000 results (0.41 seconds)

Releases

Series	Declared stable	Latest release date
10.0	**2021-02-02**	2022-10-10
10.1	2022-09-26	2023-02-24
11.0	(alpha)	2023-02-23