SPRING

What is Spring

- A Java platform that provides comprehensive infrastructure support for developing enterprise level Java applications. Primary features are dependency injection (DI) and aspect-oriented programming (AOP)

Spring module vs project

- Modules:
  - Core
    - The *Core and Beans* modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features.
  - Context
    - The *Context* module builds on the solid base provided by the *Core and Beans* modules: it is a means to access objects in a framework-style manner
  - Web(MVC)
    - "Model View Controller"
    - The MVC module abstracts Servlets, most often using a single DispatcherServlet, by taking incoming requests and distributing them to corresponding Controllers which will call their appropriate Service Layer.
  - ORM
    - "Object Relation Mapping"
  - JDBC
    - "Java Database Connectivity"
  - Test
  - AOP
    - "Aspect Oriented Programming"
- Projects:
  - Boot
  - Data
  - Security
  - Cloud

What are some spring modules you've worked with

-DI- (Dependency Injection) a technique whereby one object (or static method) supplies the **dependencies** of another object. A **dependency** is an object that can be used (a service). An **injection** is the passing of a **dependency** to a dependent object (a client) that would use it.

IoC

- Inversion of Control- a design principle in which custom-written portions of a computer program receive the flow of **control** from a generic framework.

What kind of Dependency Injection is supported with Spring

- Setter and constructor injection
- Setter Injection:
    - Calls the Setter method in the class for the property “prop”
    - <Bean name=”setterBean”, class=”com.ex.setterBean”>
        <property-name = “prop” {value||ref} = “reference”
      </bean>
- Constructor Injection:
    - Calls the Constructor method in class setterBean
    - <Bean name=”setterBean”, class=”com.ex.setterBean”>
        <constructor-arg = “constructor” {value||ref} = “reference”
      </bean>

Spring Container

- Will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The **Spring container** uses DI to manage the components that make up an application.

Application Context

- Spring's advanced container. Similar to BeanFactory, it can load bean definitions, wire beans together, and dispense beans upon request. ... This container is defined by org.springframework.**context**.**ApplicationContext**interface.
- Implementations of the app context
    - ClassPathXmlApplicationContext
    - FileSystemXmlApplicationContext
- I18n - internationalization

Bean Factory

- **BeanFactory** is the actual container which instantiates, configures, and manages a number of **beans**. These **beans** typically collaborate with one another, and thus have dependencies between themselves.

Bean wiring vs autowiring

- Bean wiring is manually linking beans via dependency injections with the reference tag
    - <bean></bean>
- autowiring is turning the process over to spring to allow it to automatically wire beans together either byType or byName, beans using autowiring must have the @Autowired annotation

Types of autowiring

- byName - links beans together by the names of the dependencies, and instance variables that the implementation should be injected into

  Public employee (){ public department(){
  @Autowired

Private string name;
Private String address;
@Autowired
Private String department;
Name linked ^
}

Private String department;
name linked ^
}

- byType works the same way but with the types of the dependencies instead of names

@Autowired
AuthorService author; ← Autowires to the “AuthorService” class

Common annotations used with Spring

- https://zeroturnaround.com/rebellabs/spring-framework-annotations-cheat-sheet/
-

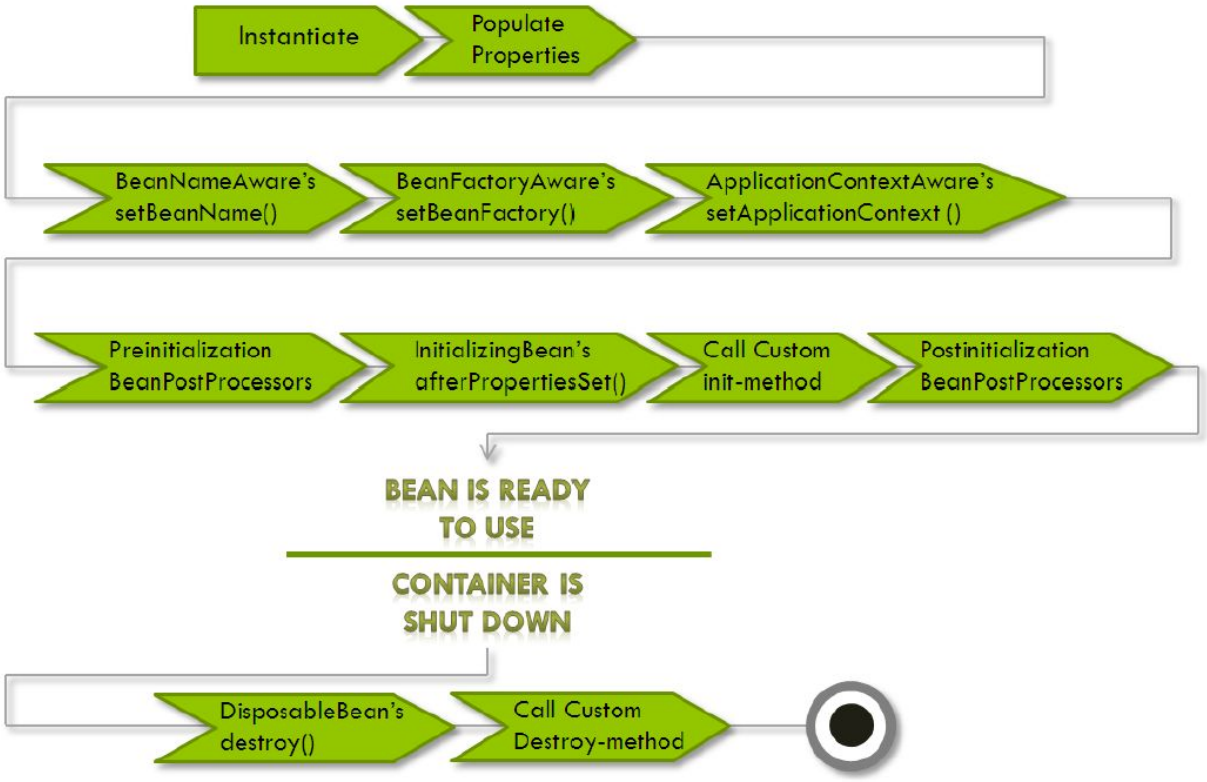
What is a spring bean?

- A class managed by the spring container

Bean scopes

- Singleton - can only have one instance no matter how many references
- Prototype - each reference creates a new instance
- Request - Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
- Session - Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
- GlobalSession - Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.

Spring Bean Lifecycle



XML

- Declare beans
- Inject beans
- Manage custom init and destroy methods

ORM

what is it

- Spring-ORM is an umbrella module that covers many persistence technologies, namely JPA and Hibernate. For each of these technologies, Spring provides integration classes so that each technology can be used following Spring principles of configuration, and smoothly integrates with Spring transaction management.
  For each technology, the configuration basically consists in injecting a DataSource bean into some kind of SessionFactory or EntityManagerFactory etc. bean.

What are contextual sessions

- some form of "contextual" sessions, where a given session is in effect throughout the scope of a given context

Orm vs Hibernate

- Spring ORM isn't an ORM, it **uses** an ORM, such as Hibernate

Important annotations and possible exceptions

- @Transactional-
- @Repository
- @Service

E

DATA

- a high level SpringSource project whose purpose is to unify and ease the access to different kinds of persistence stores, both relational database systems and NoSQL **data** stores

Benefits

- Provides a plethora of common methods most classes use when accessing data saving time and preventing code bloat

How do we start a spring data project

-

Some common JPA interfaces

- JPARepository - gives common crud methods

MVC

- architecture and ready components that can be used to develop flexible and **loosely coupled** web applications
    - each of its components has, or makes use of, little or no knowledge of the definitions of other separate components

What is the Model-View-Controller design pattern

- It is a type of front controller design pattern

What is a controller

- A controller is a component of the Spring MVC which handles HTTP requests

DispatcherServlet

- A component of the Spring MVC module which handles http requests and redirects them to the appropriate controllers.

View Resolver

- enables you to render models in the browser without tying the implementation to a specific view technology.The ViewResolver maps view names to actual views.

How are requests processed with spring mvc

- When sending a request to your application the following happens:
- The request arrives at your server (e.g. Tomcat). Depending on the context path in the url the server decides to which application the request belongs.
- Depending on the url and the servlet mapping in the web.xml file of your application the server knows which servlet should handle the request.
- The request is passed to the servlet filter chain which can modify or reject requests
- The servlet takes control over the request. In case of your Spring application the spring Dispatcherservlet receives the request. Now Spring kicks in
- The request is processed by mvc interceptors preHandle methods
- The request is mapped to a controller based on the url. The corresponding controller method will be called.
- Your controller is processing the request. Many different responses can be returned in controllers (jsp, pdf, json, redirects, etc.). For now i assume you want to render a simple jsp view. Result of the controller are two things: a model and a view. The model is a map that contains the data you want to access later in your view. The view at this stage is most of the time a simple string containing a view name.
- Registered springs mvc interceptors can kick in again using the postHandle method (e.g. for modifying the model)
- The 'view' result of your controller is resolved to a real View using a ViewResolver. Depending on the ViewResolver the result can be jsp page, a tiles view, a thymeleaf template or many other 'Views'. In your case the ViewResolver resolves a view name (e.g. 'myPage') to a jsp file (e.g. /WEB-INF/jsp/myPage.jsp)
- The view is rendered using the model data returned by your controller
- The response with the rendered view will be passed to mvc interceptors again (afterCompletion method)
- The response leaves the dispatcher servlet. Here ends spring land.
- The response passes servlet filters again
- The response is send back to client

@Controller vs @RestController

- **Rest Controller** annotation is the same as **Controller** annotation, but assuming that @ResponseBody is active by default, so all the json are parsed to java objects

@PathVariable

- Annotation which indicates that a method parameter should be bound to a URI template variable.

@RequestMapping

- **RequestMapping** annotation is used to map web requests onto specific handler classes and/or handler methods. @**RequestMapping** can be applied to the controller class as well as methods

@RequestBody/@ResponseBody

- @**RequestBody** and @ResponseBody annotations are used to bind the HTTP request/response body with a domain object in method parameter or return type. Behind the scenes, these annotation uses HTTP Message converters to convert the body of HTTP request/response to domain objects.

ResponseEntity

- **ResponseEntity** is meant to represent the entire HTTP **response**. You can control anything that goes into it: status code, headers, and **body**.

HTTP methods

- Get
- Post
- Put
- Delete
- Head
- Options
- Trace
- Patch
- Connect

Idempotent

- In computing, an idempotent operation is one that has no additional effect if it is called more than once with the same input parameters. For example, removing an item from a set can be considered an idempotent operation on the set.
  (In mathematics, an idempotent operation is one where f(f(x)) = f(x). For example, the abs() function is idempotent because abs(abs(x)) = abs(x) for all x.)
- When multiple Http requests yield the same results as a single request

Safe

- Safe methods are HTTP methods that do not modify resources

Project setup

WEB-INF

ObjectMapper

- provides functionality for reading and writing JSON, either to and from basic POJOs (Plain Old Java Objects), or to and from a general-purpose JSON Tree Model (JsonNode), as well as related functionality for performing conversions

Boot

- A pre-configured set of frameworks/technologies to reduce boiler plate configuration providing you the shortest way to have a Spring web application up and running with smallest line of code/configuration ***out-of-the-box***

Benefits of spring boot

Project setup

@SpringBootApplication

- This is a convenience annotation that Indicates a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning.

Spring boot main method

- (within the main method, Application.run() must be called to start the whole spring framework)

AOP

- aims to increase modularity by allowing the separation of cross-cutting concerns. It does so by adding additional behavior to existing code (an advice) *without* modifying the code itself, instead separately specifying which code is modified via a "pointcut" specification, such as "log all function calls when the function's name begins with 'set'". This allows behaviors that are not central to the business logic (such as logging) to be added to a program without cluttering the code

SOA

- uses "services" as the unit of computer work, to design and implement integrated business applications and mission critical software programs

Service Oriented Architecture

REpresentational State Transfer

- characterized by how they are stateless and separate the concerns of client and server
- code on the client side can be changed at any time without affecting the operation of the server, and the code on the server side can be changed without affecting the operation of the client.
- both the server and the client can understand any message received, even without seeing previous messages. This constraint of statelessness is enforced through the use of *resources*, rather than *commands*

Mock QC ?s

- What is Spring?
- What is IoC?
- What is Dependency Injection?
- What kind of dependency injections are supported by Spring?
- What are the differences between Spring modules and projects
    - List some modules
    - List some projects
- What is a Spring Container?
- What are the 11 steps of the Bean Lifecycle?
- What is application context?
- How do you wire beans?
- What is a View Resolver and what does it do?
- What are the annotations used in Spring?
    - (Split into different Spring aspects: Data, ORM, etc)
- What are the three beans necessary for implementing Spring