Spring Review

* Lightweight Java Framework for developing Enterprise level Java Applications
* Spring is a modular framework. Spring Core is the main module with the IoC container.
  + Can add additional modules as needed
* Two ways to configure
  + Annotations (Newer)
  + XML files (Older)

IoC Container

* Inversion of Control Container
  + It is a box that holds a bunch of Spring Beans
  + The IoC container manages these beans
    - How the beans are created?
      * Spring Bean lifecycle
    - How many of each bean?
    - How are its dependencies satisfied
      * Performs dependency injection into Spring Beans
  + Old Spring
    - IoC Container was called the Bean Factory
  + Current Spring
    - IoC Container is the ApplicationContext
      * ApplicationContext has more functionality

Dependency Injection

* The process by which an object has its dependencies (fields) automatically fulfilled by the framework (Usually done by a class called an Injector)
* Three types of dependency injection
  + Constructor
    - The object is created with its dependencies passed in through the constructor
  + Setter
    - The object is created. Then the setters are used to set the dependencies
  + Interface
    - Spring does not support

Configuring Spring IoC container with Annotations

* @Component
  + Put over a class
  + Says that is class should be turned into a Bean in the IoC container
  + Stereotypes are more specific Components
    - @Controller
    - @Repository
    - @Service
    - @Congfiguration
      * A class that allows you to define additional beans via @Bean above the method signature
* @ComponentScan
  + Tells spring to scan referenced packages for components to build the IoC container
* @EntityScan
  + Tells spring to look at this package for entities
* @EnableJpaRespositoroes
  + Tells spring to look at this package for repositories
* @Autowired
  + Tells spring to inject this field with a bean that satisfies this dependency

Spring CORE Module

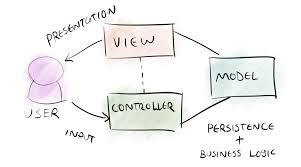
* Main Module of spring
* You must have this module to make a spring application
* It contains the IoC container

Spring DATA

* The persistence module for Spring
* Allows you to interact with a database to perform CRUD operations
* Using Spring Data
  + You create a Repository Interface for your entity
  + @Repository
  + Extends CrudRepository<Entitit,PrimarKeyType>
  + You can then autowire this dependency throughout your application
  + You can write abstract methods in the interface
    - If you follow Spring DATA naming conventions Spring data will automatically create the implementation

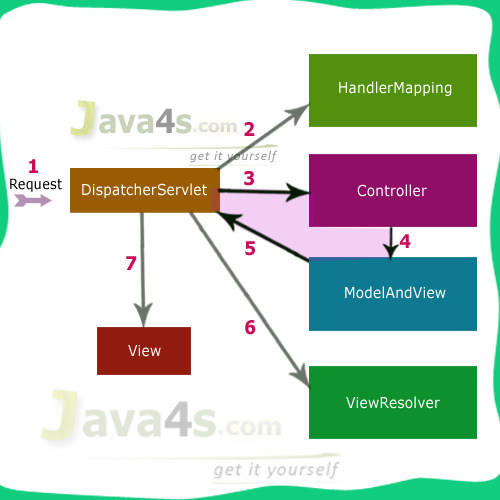
Spring WEB aka Spring MVC

* API module
* Allows you to create controllers that handle incoming requests
  + @Controller
    - @RestController puts @ResponseBody on top of each method
  + @RequestMapping(value =”path”, method = get)
    - @Get(value = ‘path’)
    - @Post(value = ‘path’)
  + @PathVariable
  + @RequsestParam
  + @RequestBody



* MVC design
  + Users are presented with a **view**
    - Web Page to interact with
  + They then perform some action which sends information to a **controller**
  + That controller uses a service(does some sort of logic) and gets a **model**
    - A model is some sort of object (For us it is usually an entity)
  + This model is used to update the **view**
  + Then the cycle repeats itself
* Example
  + I am on the login page (VIEW)
  + Put in my information and press submit (Sending information to interact with the CONTROLLER)
  + The controller gets a model (A user object MODEL)
  + That MODEL is sent back to the frontend where it updates my (VIEW)
  + My VIEW is the home page loaded with information from my MODEL
* Dispatcher Servlet
  + A servlet is a Java Class than can handle HTTP requests and give back HTTP responses
    - Javalin and Spring Abstract away you having to directly create these classes
  + Front-controller design pattern
    - All HTTP requests to a Spring application get sent to a single DispatcherServlet Class
      * Technically you could create an application with a bunch of servlets

Completely server side rendering Spring MVC



1. HTTP Request is sent to the DispatcherServlet
2. The DispatcherServlet looks at the HandlerMapping to see what controller and what handler method in that controller to send the HTTP request to
3. Then the request is sent to the controller
   1. We have been directly sending back the model at this point
4. Controller gets the model and pairs with an html view
5. This pair is returned to the DispatcherServlet
6. Pair is sent to the ViewResolver that combines the two into a finished html page
7. Send back the completed HTML page

Spring AOP

* Aspect Oriented Programming
  + Paradigm of Programming where we code to certain features or ‘aspects of our application’
* Cross Cutting Concern
  + A facet of your application that needs to be addressed across multiple layers
    - Examples, Security or logging
* Aspect
  + A class that contains code that addresses CCC
* Advice
  + Advice are the methods in an Aspect class that ‘advise’ another part of your application
    - @Before
    - @After
    - @Around
    - @AfterThrowing
    - @AfterReturning
* JoinPoints
  + Any part of your application that can be advised
    - In Spring this is limited to methods executing
* PointCutExpression
  + A regular expression that selects certain JoinPoints to be advised
* Pros
  + Decouples different aspects of your application
  + Make its really easy to refactor
* Cons
  + Slight performance hit
  + Can be hard to debug if you have too many aspects because you cannot see an aspect working on a piece of code

Spring Actuator Module

* Developer tools module
* Comes with helpful API endpoints for monitoring and controlling your application
* Usually not included in a final build of a Spring Application

Spring TEST

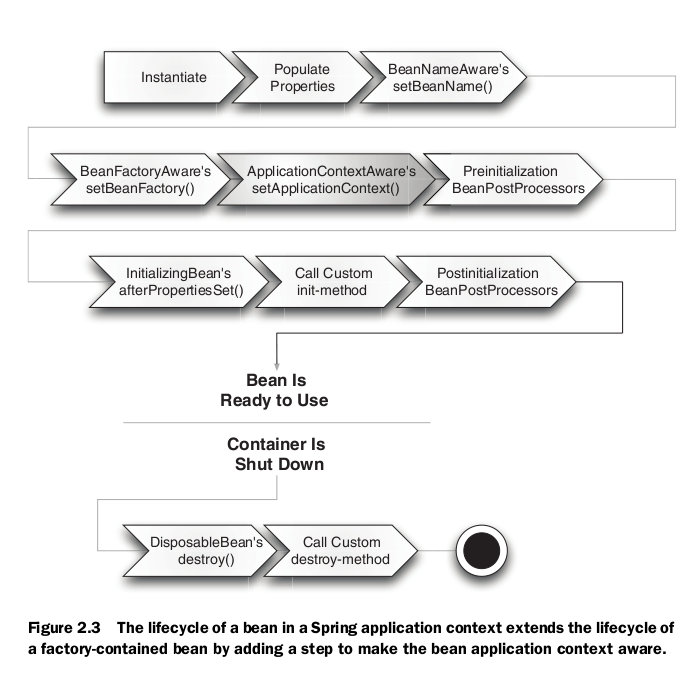
* Module with helpful features for testing
* Integrates in JUnit
* @SpringBootTest
* @Transactional
  + @Commit
  + @Rollback
* @MethodOrderer
  + @Order(1)
* Integrates with Mockito

Spring Bean Scopes

* Singleton beans
  + Only one instance of this bean in the container
    - All components, services, controllers, repositories are automatically singletons in Spring
* Prototype beans
  + 0 – many instances of this bean in the container
* Request beans
  + 1 instance per HTTP request
* Session beans
  + 1 instance per HTTP session
    - Session management in Spring
* Portlet bean
  + Deals with portlets

Spring Bean Lifecycle

* Spring Bean Lifecycle is how the IoC container creates and destroys beans



* A bean in created in the container it
* Its meta-information like the name of the bean is registered in the container
* Fields are injected
* You can optionally run initialization code for your beans
* Your beans are ready to use
* When you shutdown the container (end the application)
* Spring will destroy your beans
  + Can release any memory blocks or close resources