Spring Review

* Lightweight Java Framework for developing Enterprise Level Java Applications
* Spring is a modular framework. There is a spring Core module which is the Spring container and then additional modules that can be added like Spring Data.
* There are two ways to configure Spring
  1. Use annotations in your code (newer approach)
  2. Use XML configuration files (older approach)

Configuring Spring with Annotations

* @Component – Put over a class. Signals that this class should be read by Spring Core and turned into a bean for the Spring Container
  + Components have stereotypes which are the more specific versions of a component
    - @Repository – (for your repositories)
    - @Controller- (for your controllers)
    - @Service – (for your services)
    - @Configuration – (for your configuration files)
  + @Configuration – To serve as a definitions list for custom beans
    - Each method in a configuration component should be a bean
    - That method is annotated with @Bean
* @ComponentScan – tells Spring to please scan the referenced packages for components
* @EntityScan – Tells spring to look at this package for entities
* @EnableJpaRepositories- Tells spring to look at this package for the repositories
* @Autowired- Tells spring to search the container for a bean that can fulfill that dependency and inject it

Dependency Injection

* The process by which an object has its dependencies (fields) automatically fulfilled by the framework. (The injector in the framework).
* There are three type of dependency injection
  1. Constructor
     + The object is created with its dependencies fulfilled, dependencies were passes into the constructor of the object
  2. Setter
     + The object is created. The setters on that class will get called to one by one fulfill the dependencies in the object
  3. Interface
     + Spring does not support it

Spring Core (Main Module)

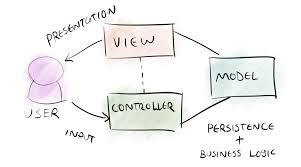
* The main module of Spring
* It is the Spring IoC (inversion of control) Container
  + Newer versions of spring call this the Application Context
  + Older versions of spring call it the Bean Factory
* The purpose of the container is to hold all of your Spring Beans
* Controls various parts of the Spring Bean
  + The Bean lifecycle (creation and destruction of beans)
  + Dependency Injection (Fulfilling the dependencies in beans with other beans)

Spring DATA (Persistence Module)

* Supplementary module to Spring
* Contains and create beans that can be used for saving and persisting information to the database
  + CrudRepository and the ability of Spring DATA to come up with implementation beans of your interfaces
* Uses Hibernate under the hood to perform the persistence operations
* Using Spring DATA
  + You should create a Repository interface for your entity
  + You should annotate with @Component @Repository
  + This interface should extend CrudRepository with appropriate generics
  + You can then autowire this dependency anywhere in Spring
  + You can create abstract methods in the interface that so long as you follow the correct naming conventions will be implemented by Spring DATA

Spring Web aka Spring MVC (WEB/API module)

* MVC (Model view Controller)
  + It is a design pattern not specific to Spring or any particular framework
  + It is one approach to making web applications



* Spring has its own MVC process for handling incoming HTTP request
* 

1. A user sends a request to the application
   1. Spring MVC uses a front controller design pattern
   2. Every request will be sent to the DispatcherServlet
2. HandlerMapping class will read the Request and direct it to the appropriate controller
3. Controller will process the HTTP Request
   1. We have been sending back the model directly back to the client
   2. @ResponseBody on the method we just directly return the model
4. Controller will pair up the model with an html view (A chunk of html)
5. Sends the model and view pair to the dispatcher servlet
6. DispatcherServlet send the pair to the view resolver which will integrate the model information into that piece of html
7. The completed view with the model information integrated into the html will be presented to the client

* You can signal a class as a controller in Spring by putting @Controller
* Each method in an at controller class should be designed to handle an http request
* @RequestMapping(value = “uri pattern”, method = RequestMethod.GET)
* @ResponseBody will say that this method directly returns the model as opposed to a view
  + By Default it is in JSON format
* @RequestBody – automatically parses the body of an incoming request into the parameter object
* @PathVariable – reads in a uri path variable to be put into a parameter
* @RequestParam – reads in a query parameter into the method ?name=smith
* @CrossOrigin – This controller will permit requests from any domain

Spring AOP (Aspect Oriented Programming)

* AOP is a paradigm of programming where we code to an “aspect of the application”
* Pros
  + Avoid code entanglement
    - Avoids making one class reliant on another
  + Avoids boilerplate code
    - Code that must be written and not specific to the logic of the method
* Con
  + You have no idea that an aspect is working on your code
  + Can make applications logic flow very hard to follow
  + Do not overuse aspects
* CCC (Cross Cutting Concern)
  + Any feature/part of the application that cuts across multiple layers of the application
    - Security, Logging
* Aspects
  + Special classes that are designed to address a specific CCC
* Advice
  + An advice is a method in an aspect that is called to address a CCC
* JoinPoint
  + Any location in an application that an advice can be applied
* PointCutExpression
  + A regular expression that will identify what joinpoints the advice method applies to

Spring Actuator (Developer Tools Module)

* Contains a lot of helpful features for developers
* It exposes a whole host of endpoints for you to use to get information about the application
  + Metrics
  + Health
  + Beans
  + Memory usage
  + Shutdown feature

Spring Test (Testing Module)

* Module that contains helpful features for Testing
* Integrates into JUnit
* @Transactional
  + Add to the top of a class
  + @Commit
  + @Rollback
* @MethodOrderer
  + @Order(1)

Spring Boot

* Spring vs Spring Boot
* Spring Boot bills itself as the opinionated version of Spring
* Spring Boot is essentially a starter kit for building a web application
  + Convention over configuration
    - Did not need define a lot of configuration files
    - We had one application.properties where we stored any configuration information
      * We did not need a web.xml
      * We did not need a hibernate.cfg.xml
    - Spring boot will guess much of the configurations that you want for your application
* Embedded version of Tomcat
  + You do not need to make a spring boot application a .war and deploy it to another Tomcat Server
  + You can just run the Spring Boot projects as a straight up Java application.
  + Spring Boots defaults to a .jar file
    - .jar (Java Archive) (100% Java code and related resources)
    - .war (Web Archive) (Java some HTML,JS and related resources)