Getting Started with Java Spring Framework

You can build a variety of applications using Java, Spring and Spring Boot:  
-Web  
-REST API  
-Full Stack   
-Microservices  
  
---------------------------------------------------------------------------  
WHY IS COUPLING IMPORTANT?  
  
\*Coupling: How much work is involved in changing something?  
Examples: An engine is tightly coupled to a car, a wheel is loosely coupled to a car.  
We want loose coupling as much as possible, that means, functional changes with as less code changes as possible.  
  
---------------------------------------------------------------------------  
-Beans: the things that are managed by Spring  
-You can use the name or the type of the Bean in order to retrieve it in context  
  
-Spring Container: Manages Spring beans & their lifecycle  
-Bean Factory: Basic Spring Container  
-Application Context: Advanced Spring Container with enterprise-specific features  
-Easy to use in web apps  
-Easy internationalization  
-Easy integration with Spring AOP  
------------------------------------------------------------------------------  
  
EXPLORING JAVA BEAN vs POJO vs SPRING BEAN  
  
\*Java Bean: Classes adhering to 3 constraints:  
-Have public default constructors  
-Allow access to their properties using getter and setter methods  
-Implement java.io.Serializable  
  
\*POJO: Plain old java object:  
-No constraints  
-Any Java Object is a POJO  
  
\*Spring Bean: Any java object that is managed by Spring  
-Spring uses IOC Container(Bean Factory or Application Context) to manage these objects.

Using Spring Framework to create and manage your Java Objects

@Primary-Indicates that a bean should be given preference when multiple candidates are qualified to autowire a single-valued dependency. If exactly one primary bean exists among the candidates, it will be the autowired value.   
  
@Qualifier-A specific bean should be autowired(the name of the bean can be used as qualifier)  
  
---------------------------------------------------------------------------------------------------------------------  
\*Exploring Spring-Dependency Injection Types  
-Constructor-based: Dependencies are set by creating the Bean using its Constructor  
-Setter-based: Dependencies are set by calling setter methods on your beans  
-Field: No setter or constructor, dependency is injected using reflection.  
  
As soon as you put @Autowired in a field, Spring will do field injection.  
  
When you use Constructor-based injection the @Autowired annotation is not mandatory, the injection will be done even without it.  
  
WHICH ONE SHOULD YOU USE?  
-Spring team recommends Constructor-based injection as dependencies are automatically set when an object is created.  
----------------------------------------------------------------------------------------------------------------------  
\*Spring Framework-Important Terminology  
-@Component: An instance of class will be managed by Spring framework  
-Dependency: GameRunner needs GamingConsole impl! (GamingConsole Impl ex:MarioGame, is a dependency of GameRunner)  
-Component Scan: How does Spring Framework find component classes?(Is scans packages)  
-Dependency Injection: Identify beans, their dependencies and wire them together(provides IOC-Inversion of Control)  
-Autowiring: Process of wiring in dependencies for a Spring Bean  
--------------------------------------------------------------------------------------------------------------------  
@Component VS @Bean  
\*For @Component  
-Can be used on any Java class  
-Very easy. Just add an annotation  
-Autowiring: YES-Field, Setter or Constructor Injection  
-Spring Framework creates the beans  
-It is recommended for instantiating Beans for your own Application Code:@Component  
  
\*For @Bean  
-Typically used on methods in Spring Configuration classes  
-You write all the code  
-Autowiring: YES-method call or method parameters  
-You write bean creation code  
-Recommended for a)Custom Business Logic, b)Instantiating beans for 3rd party libraries  
---------------------------------------------------------------------------------------------------------------------  
  
Why do we have a lot of Dependencies?  
  
In GameRunner Hello World App, we have very few classes but real world apps are much more complex:  
-Multiple layers(Web, Business, Data etc)  
-Each layer is dependent on the layer below it  
  
With Spring Framework:   
-Instead of focusing on objects, their dependencies and wiring, you can focus on the business logic of your application  
- Spring Framework manages the lifecycle of objects  
-Mark components using annotations: @Component  
-Mark dependencies using @Autowired  
-Allow Spring Framework to do its magic

Exploring Spring Framework Advanced Features

\*Exploring Lazy initialization of Spring Beans  
-Default initialization for Spring Beans: Eager  
-Eager initialization is recommended: Errors in the configuration are discovered immediately at application startup  
-However, you can configure beans to be lazy initialized using Lazy annotation: Not recommended and not frequently used.  
-Lazy annotation   
-Can be used almost everywhere @Component and @Bean are used  
-Lazy-resolution proxy will be injected instead of actual dependency  
-Can be used on Configuration (@Configuration) class: All @Bean methods within the @Configuration will be lazily initialized  
---------------------------------------------------------------------------------------------------------------------------------------  
\*Comparing Lazy Initialization vs Eager Initialization  
  
For LAZY:  
-Initialization time: Bean initialized when it is first made use of in the application   
-Default: NOT Default  
-Code Snippet: @Lazy OR @Lazy(value=true)  
-What happens if there are errors in initializing: Errors will result in runtime exceptions  
-Usage: Rarely used  
-Memory Consumption: Less (until bean is initialized)  
-Recommended Scenario: Beans very rarely used in your app  
  
  
For EAGER:  
-Initialization time: Bean initialized at startup of the application  
-Default: Default  
-Code Snippet: @Lazy(value=false) OR (Absence of @Lazy)  
-What happens if there are errors in initializing: Errors will prevent application from starting up  
-Usage: Very frequently used  
-Memory Consumption: All beans are initialized at startup  
-Recommended Scenario: Most of your beans  
-------------------------------------------------------------------------------------------------------------------------------------  
\*Spring Bean Scopes  
  
Spring Beans are defined to be used in a specific scope:  
-Singleton: One object instance per Spring IoC container  
-Prototype-Possibly many object instances per Spring IoC container  
  
-Scopes applicable ONLY for web-aware Spring ApplicationContext  
-Request: One object instance per single HTTP request  
-Session: One object instance per user HTTP Session  
-Application: One object instance per web application runtime  
-Websocket: One object instance per WebSocket instance  
------------------------------------------------------------------------------------------------------------------------------------  
Prototype vs Singleton Bean Scope  
  
For Prototype:  
-Instances: Possibly Many per Spring IOC Container  
-Beans: New bean instance created every time the bean is referred to  
-Default: NOT Default  
-Code Snippet: @Scope(value=ConfigurableBeanFactory.SCOPE\_PROTOTYPE)  
-Usage: Rarely used  
-Recommended Scenario: Statefull beans  
  
  
For Singleton:  
-Instances: One per Spring IOC Container  
-Beans: Same bean instance reused  
-Default: Default  
-Code Snippet: @Scope(value=ConfigurableBeanFactory.SCOPE\_SINGLETON) OR Default  
-Usage: Very frequently used  
-Recommended Scenario: Stateless beans  
--------------------------------------------------------------------------------------------------------------------------  
  
Spring Stereotype Annotations- @Component & more...  
  
@Component-Generic annotation applicable for any class  
-Base for all Spring Stereotype Annotations  
-Specializations of @Component:  
-@Service: Indicates that an annotated class has business logic  
-@Controller: Indicates that an annotated class is a "Controller" (e.g. a web controller) and is used to define controllers in your web applications and REST API  
-@Repository: Indicates that an annotated class is used to retrieve and/or manipulate data in a database  
-------------------------------------------------------------------------------------------------------------------------  
  
Quick Review of Important Spring Concepts  
  
-Dependency Injection: Identify beans, their dependencies and wire them together (provides IOC: Inversion of Control)  
  
-Constr. injection: Dependencies are set by creating the Bean using its Constructor  
  
-Setter injection: Dependencies are set by calling setter methods on your beans  
  
-Field injection: No setter or constructor. Dependency is injected using reflection.  
  
-IOC Container: Spring IOC Context that manages Spring Beans and their lifecycle  
  
-Bean Factory: Basic Spring IOC Container  
  
-Application Context: Advanced Spring IOC Container with enterprise-specific features, Easy to use in web applications with internationalization features and good integration with Spring AOP  
  
-Spring Beans: Objects managed by Spring  
  
-Auto-wiring: Process of wiring in dependencies for a Spring Bean  
------------------------------------------------------------------------------------------------------------------  
  
Spring Big Picture-Framework and Modules  
  
\*Spring Framework contains multiple Spring Modules  
-Fundamental Features: Core (IOC Container, Dependency Injection, Auto Wiring, ...)  
-Web: Spring MVC etc (Web Applications, REST API)  
-Web Reactive: Spring WebFlux etc  
-Data Access: JDBC, JPA etc  
-Integration: JMS etc  
-Testing: Mock Objects, Spring MVC Test etc  
  
\*Why is Spring Framework divided into Modules?  
-Each app can choose modules they want to make use of  
-They do not need to make use of everything in Spring framework  
--------------------------------------------------------------------------------------------------------------  
  
Spring Big Picture- Spring Projects  
  
\*Application architectures evolve continuously  
-Web>REST API>Microservices>Cloud>...  
  
\*Spring evolves through Spring Projects:  
-First Project: Spring Framework  
-Spring Security: Secure your web application or REST API or microservice  
-Spring Data: Integrate the same way with different types of databases: NoSQL and Relational  
-Spring Integration: Address challenges with integration with other applications  
-Spring Boot: Popular framework to build microservices  
-Spring Cloud: Build cloud native applications

Getting Started with Spring Boot

\*What's the most important goal of Spring Boot?  
-Help you build PRODUCTION-READY apps Quickly  
->Build QUICKLY:  
-Spring Initializer  
-SB Starter Projects  
-SB Auto Config  
-SB DevTools  
->Be PRODUCTION-READY  
-Logging  
-Different Config for Different Environments (Profiles, ConfigProperties)  
-Monitoring(SB Acutator)  
----------------------------------------------------------------------------------------------------------------------  
\*Exploring SB Starter Projects  
  
->I need a lot of frameworks to build app features:  
-Build a REST API: I need Spring, Spring MVC, Tomcat, JSON conversion...  
-Write Unit Tests: I need Spring Test, JUnit, Mockito, ...  
  
->How can I group them and make it easy to build apps?  
-Starters: Convenient dependency descriptors for diff. features  
  
->SB provides variety of starter projects:  
-Web Apps & REST API: SB Starter Web(spring-webmvc, spring-web...)  
-Unit Tests: SB Starter Test  
-Talk to database using JPA: SB starter data JPA  
-Talk to database using JDBC: SB starter JDBC  
-Secure your web app or REST API-SB starter Security  
----------------------------------------------------------------------------------------------------------------------  
\*Managing App. Configuration using Profiles  
  
->Apps have different environments: Dev, QA, Stage, Prod, ...  
  
->Different environments need different configuration:  
-Different Databases  
-Different Web Services  
  
->How can you provide different configuration for different environments?  
-Profiles: Environment specific configuration  
  
Basically if we have prod, dev ... we can create [application.properties](http://application.properties/) files for each environment and write specific configs for them, whereas the original [application.properties](http://application.properties/) file is the default one, so when there is no config specified on the specific app.prop files, the configs on the default file will be taken into account.   
We can create a Config class to handle complex configs  
-----------------------------------------------------------------------------------------------------------------------  
\*Monitor Applications using Spring Boot Actuator  
  
->Monitor and manage your application in your production  
  
->Provides a number of endpoints:  
-beans: Complete list of Spring beans in your app  
-health: App health information  
-metrics: App metrics  
-mappings: Details around Request Mappings  
-----------------------------------------------------------------------------------------------------------------------  
\*Understanding Spring Boot vs Spring MVC vs Spring  
  
->Spring Framework: Dependency Injection  
-@Component, @Autowired, @ComponentScan ...  
-Just Dependency Injection is NOT sufficient, you need other frameworks to build apps. Spring Modules and Spring Projects: Extend Spring Eco System, Provide good integration with other frameworks(Hibernate/JPA, JUNIT & Mockito for Unit Testing)  
  
-> Spring MVC (Spring Module): Simplify building web apps and REST API  
-Building web apps with Struts was very complex   
-@Controller, @RestController, @RequestMapping  
  
->Spring Boot(Spring Project): Build Production-Ready apps Quickly  
-Starter Projects: Make it easy to build variety of apps  
-Embedded Server: No need for separate apps servers!  
-Logging and Error Handling  
-Profiles and ConfigurationProperties

Getting Started with JPA and Hibernate with Spring and Spring Boot

\*Launching up H2 Console and creating Course Tables  
-When working with H2, the URL to connect is dynamically created each time we run the application. This makes things very difficult and complicated, so in order to save time and repetition, we configure a static connection on app.prop file:   
spring.h2.console.enabled=true   
spring.datasource.url=jdbc:h2:mem:testdb  
  
-To create tables in H2, we create a .sql file in src/main/resources and we write the sql there.  
  
-JPA is different from JDBC because instead of writing long complicated queries to interact with the DB table, with JPA we map the class (entity) with the DB table directly  
----------------------------------------------------------------------------------------------------------------------  
• JDBC to Spring JDBC to JPA to Spring Data JPA  
  
->JDBC  
-Write a lot of SQL queries  
-Write a lot of Java code  
  
->Spring JDBC  
-Write a lot of SQL queries  
-Write lesser Java code  
  
->JPA  
-Do NOT worry about queries  
-Just Map Entities to tables  
  
->Spring Data JPA  
-Let's make JPA even more simple  
-I will take care of everything  
--------------------------------------------------------------------------------------------------------------------------  
\*Hibernate vs JPA  
  
->JPA defines the specification. It is an API  
-How do you define entities?  
-How do you map attributes?  
-Who manages the entities?  
  
->Hibernate is one of the popular implementations of JPA   
  
->Using Hibernate directly would result in a lock in to Hibernate  
-There are other JPA implementations (Toplink for exp)

Build Java Web App with Spring Framework, Spring Boot and Hibernate

\*Building your first web app can be complex:  
  
-Web App concepts(Browser, HTML, CSS, Request, Response, Form, Session, Authentication)  
  
-Spring MVC (Dispatcher, Servlet, View Resolvers, Model, View, Controller, Validations...)  
  
-Spring Boot(Starters, Auto Configuration...)  
  
-Frameworks/Tools(JSP, JSTL, JPS, Bootstrap, Spring Security, MySQL, H2)  
-------------------------------------------------------------------------------------------------------------------------  
- JSP(JavaServer Pages) serves as view and is one option we can use as a templating mechanism for our HTML pages.   
--------------------------------------------------------------------------------------------------------------------------  
\*Spring MVC Front Controller - Dispatcher Servlet  
  
->A: Receives HTTP Request  
->B: Processes HTTP Request  
-B1: Identifies correct controller method (Based on request URL)  
-B2: Executes Controller Method (Returns Model and View Name)  
-B3: Identifies correct View (Using ViewResolver)  
-B4: Executes view  
->C: Returns HTTP Response  
------------------------------------------------------------------------------------------------------------------------  
\*Session vs Request Scopes  
  
->All requests from browser are handled by our web app deployed on a server  
  
->Request Scope: Active for a single request ONLY:  
-Once the response is sent back, the request attributes will be removed from memory  
-These cannot be used for future requests  
-Recommended for most use cases  
  
->Session Scope: Details stored across multiple requests  
-Be careful about what you store in session (Takes additional memory as all details are stored on server)  
  
-------------------------------------------------------------------------------------------------------------------------  
\*Spring Boot Auto Configuration Magic-Data JPA  
  
->Spring Boot Auto Configuration does some magic:  
-Initialize JPA and Spring Data JPA frameworks  
-Launch an in memory database (H2)  
-Setup connection from App to in-memory database  
-Launch a few scripts at startup (example: data.sql)  
  
->Remember-H2 is in memory database  
-Does NOT persist data  
-Great for learning  
-BUT NOT so great for production  
  
--------------------------------------------------------------------------------------------------------------------------

Creating a Java REST API with Spring Boot, Spring Framework and Hibernate

• Building REST API with SB - Goals  
  
  
->WHY SB?  
-You can build REST API WITHOUT SB  
-What is the need for SB?  
  
->HOW to build a great REST API?  
-Identifying Resources(/users,/users/{id}/posts)  
-Identifying actions (GET, POST, PUT, DELETE,...)  
-Defining Request and Response structures  
-Using appropriate Response Status (200, 404, 500,...)  
-Understanding REST API Best Practices  
=>Thinking from the perspective of your consumer  
=>Validation, Internationalization, Exception Handling, HATEOAS, Versioning, Documentation, Content Negotiation...  
------------------------------------------------------------------------------------------------------------------  
  
• Building REST API with SB- Approach  
  
1: Build 3 Simple Hello World REST API  
- Understand the magic of SB  
- Understand fundamentals of building REST API with SB  
>@RestController, @RequestMapping, @PathVariable, JSON conversion  
  
2: Build a REST API for a Social Media App  
-Design and Build a Great REST API  
>Choosing the right URI for resources  
>Choosing the right request method for actions  
>Designing Request and Response structures  
>Implementing Security, Validation and Exception Handling  
-Build Advanced REST API Features  
>Internationalization, HATEOAS, Versioning, Documentation...  
  
3: Connect your REST API to a DB  
- Fundamentals of JPA and Hibernate  
- Use H2 and MySQL as DB  
--------------------------------------------------------------------------------------------------------------------

Docker

\*Why is Docker Popular?

-Standardized Application Packaging

-Same packaging for all types of applications (Java, Python or JS)

-Multi Platform Support

-Local Machine, Data Center, Cloud

-Isolation

-Containers have isolation from one another

*docker container run -d -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE*

Docker image is downloaded from Docker Registry

-https://hub.docker.com/r/in28min/hello-world-nodejs

-Image is a set of bytes

-Container: Running image

-in28min/hello-world-nodejs: Repository Name

-0.0.1.RELEASE: Tag (or version)

-> -p hostPort:containerPort: Maps internal docker port (container port) to a port on the host (host port)

-By default, Docker uses its own internal network called bridge network

-We are mapping a host port so that users can access your application

-> -d Detached Mode (Don’t tie up the terminal)

Understanding Docker Terminology

-Docker Image: A package representing specific version of your application

-Docker Registry: A place to store your docker images

-Docker Hub: A registry to host Docker images

-Docker Repository: Docker images for a specific app

-Docker Container: Runtime instance of a docker image

-Dockerfile: File with instructions to create a Docker image

What is Aspect Oriented Programming

-A layered approach is typically used to build apps:

->Web Layer: View logic for web apps OR JSON conversion for REST API

->Business Layer: Business Logic

-> Data Layer: Persistence Logic

-Each layer has different responsibilities, however, there are a few common aspects that apply to all layers

->Security

->Performance

->Logging

-These common aspects are called Cross Cutting Concerns

-Aspect Oriented Programming can be used to implement Cross Cutting Concerns

Aspect Oriented Programming-Important Terminology

\*Compile Time

-Advice: What code to execute? (Example: Logging, Authentication)

-Pointcut: Expression that identifies method calls to be intercepted (Example: execution)

-Aspect: A combination of (advice and pointcut)

-Weaver: Weaver is the framework that implements AOP (AspectJ or Spring AOP)

\*Runtime

-Join Point: When pointcut condition is true, the advice is executed. A specific execution instance of an advice is called a Join Point.

Spring Security

\*Understanding Security Fundamentals

In any system:

-You have resources: A REST API, A Web App, A Database, A resource in the cloud, …

-You have identities: Identities need to access to resources and perform actions (execute a REST API call, read/modify data in a db)

-Key Questions: How to identify users? How to configure resources they can access & actions that are allowed?

-Authentication (is it the right user?)

->UserId/password (What do you remember?)

->Biometrics (What do you possess?)

-Authorization (do they have the right access?)

->User XYZ can only read data

-> User ABC can read and update data

\*Understanding Important Security Principles

-A chain is only as strong as its WEAKEST link

->Small security flaw makes an app with robust architecture vulnerable

-6 Principles Of Building Secure Systems

1-> Trust nothing: validate every request, validate piece of data or information that comes into the system

2-> Assign Least Privileges: start the design of the system with security requirements in mind, have a clear picture of the user roles and accesses, assign minimum possible privileges at all levels

3-> Have Complete Mediation: apply a well-implemented security filter. Test the role and access of each user.

4-> Have Defense In Depth: multiple levels of security

5-> Have economy of mechanism: security architecture should be simple, simple systems are easier to protect

6-> Ensure openness of design: easier to identify and fix security flaws, opposite of the misplaced idea of “Security Through Obscurity”

\*Understanding Spring Security Authentication

-Authentication is done as part of the Spring Security Filter Chain!

-AuthenticationManager: Responsible for authentication (Can interact with multiple authentication providers)

-AuthenticationProvider: Perform specific authentication type

-UserDetailsService: Core interface to load user data

-> How is authentication result stored?

-SecurityContextHolder-> SecurityContext -> Authentication > Granted Authority

\*Getting Started with OAuth

-How can you give an application access to files present on your google drive?

->You don’t want to provide your credentials because it is not secure

-OAuth: Industry-standard protocol for authorization

->Also supports authentication now

-Lets say you want to provide access to your Google Drive files to the Todo management app

->Important concepts:

\*Resource owner: You

\*Client application: Todo management application

\*Resource server: Contains the resources that are being accessed – Google Drive

\*Authorization server: Google OAuth Server