**Spring framework – Master to beginner**

**Section1**

Spring is a dependency injection framework.

**Section 2**

@Component is used to tell spring to manage instances.

@Autowired is used to tell spring that there is a dependency among the instances of the components it manages.

**Spring terminology:**

**Beans**: the different instances that spring manages are called beans. The different objects managed by spring framework.

**Autowiring**: the process where spring identifies the dependencies, identifies the matches for dependencies, and populates them.

**Dependency injection**: dependency injection is the injection of a method into a complex class or bean.

**Inversion of control**: The control is taken from the class that needs the dependency to the framework to identify the dependency.

**IOC Container**: Generic term to represent anything that is implementing inversion of control.

**Application Context**: In spring frame work, the most important IOC container is the application context.

Application context is the place where all the beans are created and managed. It is the most important part of spring framework. It is where all the core logic of spring framework happens.

**Section 4**

Spring projects can be made using start.spring.io – generate a maven project with java and spring boot version (greater than 2). Dependencies can be added at this step.

Tight coupling: where a method is dependent on an algorithm inside it. Which required that the method be modified every time if there is a change in algorithm. This can be changed by created a separate method for that algorithm and invoking that method inside this method. Using interfaces helps it become more loosely coupled.

We can tell spring what the beans are by using @Component and tell it the dependencies by using @Autowired. We get the application context by running SpringApplciation.run(Main class, args of main class).

We can get beans from the application context by calling applicationContext.getBean().

Go to application.properties and put logging.level.org.springframework = debug to put the spring frame work in debug mode.

There are two types of injection for Autowiring – constructor injection and setter injection. If there are two components for injection, then @primary is used to define which component is the primary one.

If a setter is made instead of a constructor, then the injection is made by setter. Even if there is neither a constructor or a setter, setter injection is done by default. (Earlier recommendation was to use constructor injection for mandatory injections and setter injections for optional dependencies. However, Autowiring automatically takes care of dependencies.)

**Section 5**

In case of conflicts in Autowiring where the interface has multiple components. The easiest way to solve the conflict is to use the name of the required class when initializing the interface. @Primary has higher preference over the name of the initialized variable.

Another option is to use @Qualifier(“”) for specifying the name of the component. This qualifier has to be placed over the @Autowired and the @component class. Auto wiring by name is preferred as it is easier.

Bean Scope: singleton – one instance per spring context

Prototype – new bean whenever requested

Request – one bean per http request

Session – one bean per http session

When we request the application for the same bean multiple times, we get the same instance of the bean. The scope is singleton by default.

To get different instances every time, we can create @Scope (“prototype”) on top of the class. It is singleton by default.

Whenever we are trying to get a bean, but one of its dependencies is a prototype, then we should create a proxy (@Scope (value=”prototype”, proxyMode=ScopedProxyMode.TARGET\_Class)) for the dependency.

Whenever we create a spring boot application, a component scan automatically runs on the current package and the sub packages. Therefore, if there is a component in another package, we will have to add a component scan below @SpringBootApplication - @ComponentScan (“package name”)

Spring automatically manages the life cycle of a component. It is done by the IOC container.

@PostConstruct can be used to call a method which automatically runs after the bean is created.

@PreDestroy can be used to call a method which automatically runs before the bean is destroyed.

CDI tries to define common annotations for injections such as @Inject: for Autowiring

@Named: for component and qualifier

@Singleton: for singleton classes

To use CDI we need to define groupId – javax.inject, artifiactId – javax.inject and version 1 in pom.xml. It is basically an interface to use annotations.

Therefore for @Component I can use @Named and for @Autowiring I can use @Inject.

Spring-boot-starter is the spring boot dependency which enables sprint boot. If we aren’t using spring boot, we can change it to spring-core (in pom.xml) and then introduce spring-context for application context. Spring boot provides @SpringBootApplication to initialize an application context. We would need to use @Configuration to initialize an application Context (using annotationConfigApplicationContext). Spring boot also automatically defines a component scan where @SpringBootApplication is used. Therefore a @ComponentScan needs to be defined as well.

A spring project can be defined using an XML file instead of annotations. Before spring 2.5, everything had to be wired in XML. ApplicaiontContext.xml needs to be created and populated using XML metadata from the internet. We can define beans and their dependencies here.

Eg: <bean id=”” class=””> - for component

<property name=”” ref=”bean name”></property> - for Autowiring

For loading the application context, we can create a ClassPathXmlApplicationContext using ClassPathXmlApplicationContext applicationContext = new ClassPathXmlApplicationContext(“applicationContext.xml”);

**Spring quick guide**

**@SpringBootApplication** – for the main class (provided by spring boot)

**@Configuration** – for the main class(without spring boot)

**@Component** – for searching class (for telling spring that the class is a bean)

**@Primary –** for setting a component as primary

**@Autowired** – for Autowiring (to tell the class that the method or parameter is a dependency)

**@ComponentScan(“com.in28minutes.”)** – for scanning a package (base package by default), Spring boot automatically scans the current package and sub packages)

**@Scope(Value=”prototype”, proxyMode = ScopedProxyMode.target\_class) –** scope for the classes(singleton or prototype) – singleton by default

**@Qualifier(“bubble”) –** for specifying the names of components

**@PostConstruct –** this code runs after a bean is created and the dependencies are defined (must be void)

**@PreDestroy** – Runs just before the bean is destroyed

**CDI – Context dependency injection :** Defines an interface to do things such as

@Inject(@Autowired)

@Named(@Component&@Qualifier)

@Singleton(Defines a scope of singleton)

@Qualifier(Same)

@Scope(setting the scope – singleton or prototype)

@Singleton (Easily define singleton)

Add the following to pom.xml

{<dependency>

<groupId>javax.inject</groupId>

<artifactId>javax.inject</artifactId>

<version>1</version>

</dependency>

}

The functionality does not change whether CDI or normal annotations are used. Spring can be used as the implementation with CDI annotations since CDI is a java EE standard.

**To remove Spring Boot**

Replace the spring boot dependency with these two:

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</dependency>

@SpringBootApplication is provided by spring boot. We can run a spring application without spring boot by replacing this with @Configuration.

SpringApplication is a spring boot class too. So we will have to get an application context using ACAC (AnnotationConfigApplciationContext).

We also need to define a @ComponentScan which was done automatically by spring boot. We define this below @Configuration.