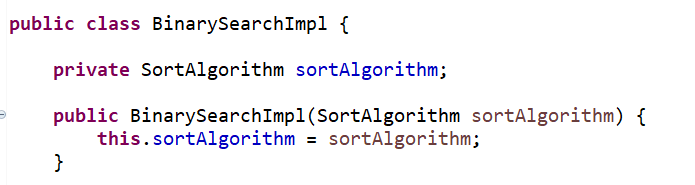
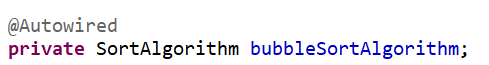
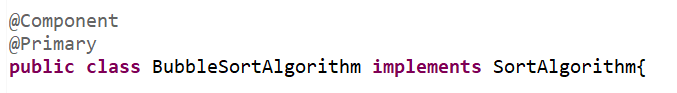
1. Introduction to Spring Framework
   1. Spring is a dependency injection framework.
   2. Spring framework will come under business layer.
   3. Dependency Injection
      1. Using any class object in other class in classed dependency.
      2. Inject dependency using loose coupling instead of tight coupling.
      3. Loose coupling – instead of create object inside class, inject using constructor or setter method.
   4. Spring frameworks create objects and populate dependencies.
   5. Using @Component annotation use to tell spring that this class need to be managed. And inject all dependencies.
   6. Using @Autowired annotation use to tell spring that which dependencies need to be injected.
   7. Few terminologies which are frequently associated with Spring framework.
      1. Beans - Beans are different objects that are managed by this Spring framework.
      2. Autowiring - The process where spring identifies the dependences, identifies the matches for the dependencies and populates them.
      3. Dependency Injection – injecting beans using constructor or setter method
      4. Inversion of control – provide bean managing control to IOC container
      5. IOC Container – handle life cycle of beans
      6. Application Context – used to get beans
2. Spring Framework in depth
   1. Implement Loose coupling



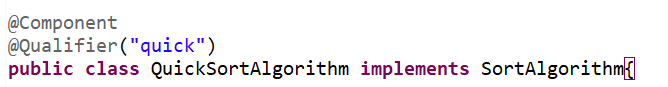
* 1. Manage dependencies by Spring framework
     1. Step 1: Where to search for a bean?
        1. Use @ComponentScan annotation at class level
     2. Step 2: What are the beans?
        1. Use @Component annotation at class level to register class as bean
     3. Step 3: What are the dependencies of a bean?
        1. Use @Autowired annotation at field level
     4. Step 4: How to get bean object?
        1. Using Application context, we can get bean object.
     5. Step 5: If we have multiple type of dependency then which one to inject?
        1. Use @Primary annotation or use bean name as object name.
     6. Step 6: How many ways spring inject dependency in bean?
        1. Constructor – create constructor with dependency as argument
        2. Setter Method – add setter method for dependency
        3. No Constructor or Setter Method – same as setter method without setter method (Popular way to do dependency injection)
  2. Multiple beans available for Auto wiring
     1. By name: use object name as Bean name.



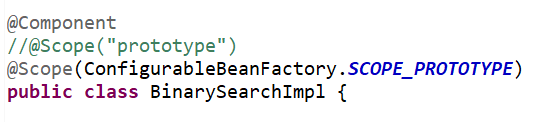
* + 1. @Primary annotation: it has higher priority than By Name auto wiring



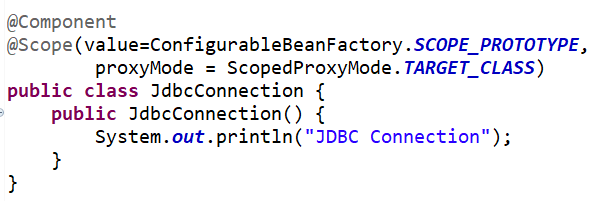
* + 1. @Qualifier annotation: use at class level in bean and use at field level @Autowired annotation.



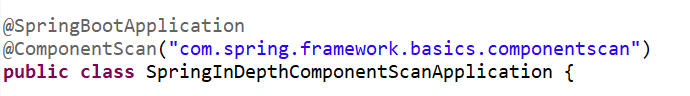
* 1. Scope of a Bean
     1. Different types of scope
        1. Singleton – One instance per Spring Context. (Default)
        2. Prototype – New bean whenever request
        3. Request – One bean per HTTP request, used in web application
        4. Session – One bean per HTTP session, used in web application
     2. We can change bean scope using @Scope annotation with scope type



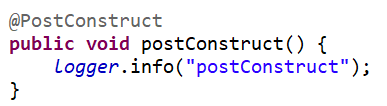
* + 1. We can change dependency scope using @Scope annotation with proxy mode else it uses singleton scope always.



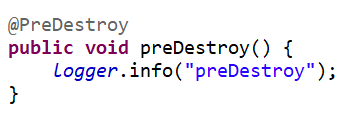
* 1. Using component scan to scan beans
     1. @ComponentScan scans for beans in current package and its sub package. So, if some beans are outer the current package then it will not pick those beans and throw error.



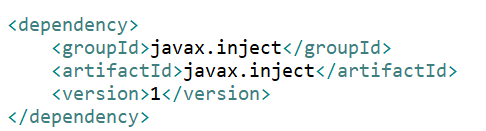
* 1. Lifecycle of a bean - @PostConstruct and @PreDestroy
     1. The entire life cycle of this bean is maintained by Spring IoC container.
     2. @PostConstruct
        1. If we want to perform operation after construction of a bean, then we can use @PostConstruct annotation at method level.



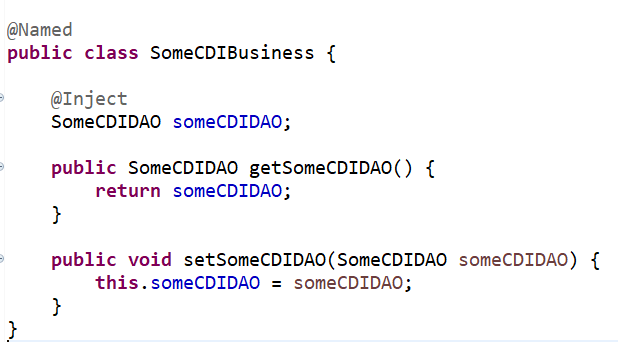
* + 1. @PreDestory
       1. If we want to perform operation before destroying of a bean.



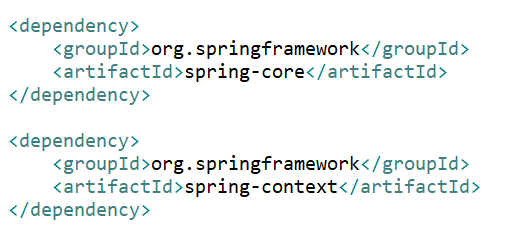
* 1. Container and Dependency Injection (CDI)- @Named and @Inject
     1. CDI
        1. TDA is an interface defining how to do Dependency Injection. It says, if I want to auto-wire something in, then you'd need to use @Inject. If you'd want to do @Named, that means you are defining a component.
        2. Java EE Dependency Injection Standard (JSR-330)
        3. Spring framework is actual implementation CDI.
        4. CDI supports most annotation
           1. @Inject (@Autowired)
           2. @Named (@Component & @Qualifier)
           3. @Singleton (Define a scope of singleton)
     2. Use CDI in spring application
        1. Add dependency in pom xml file



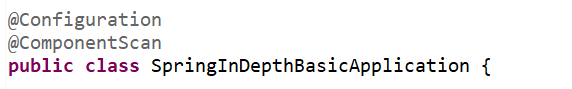
* + - 1. Use @Named to create bean and @Inject to auto wire dependency.



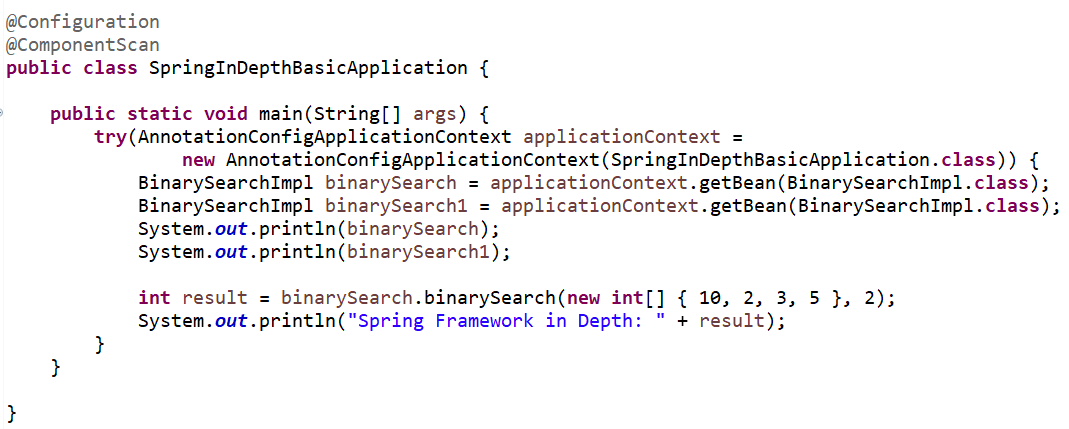
* 1. Create Spring framework project using annotations
     1. Add below dependency in pom xml



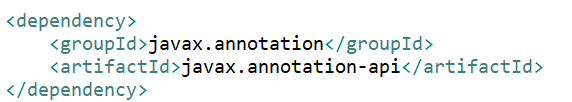
* + 1. Use @Configuration and @ComponentScan to define starting class and where to search for beans



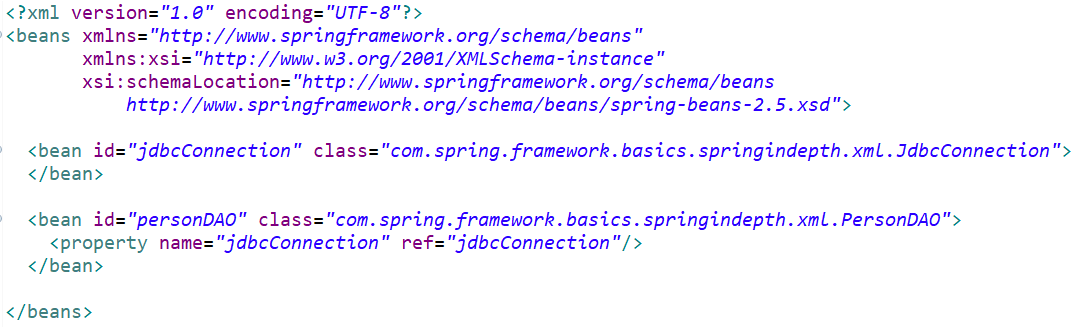
* + 1. Use AnnotationConfigApplicationContext class to create Application Context object with annotation and get beans from this object.



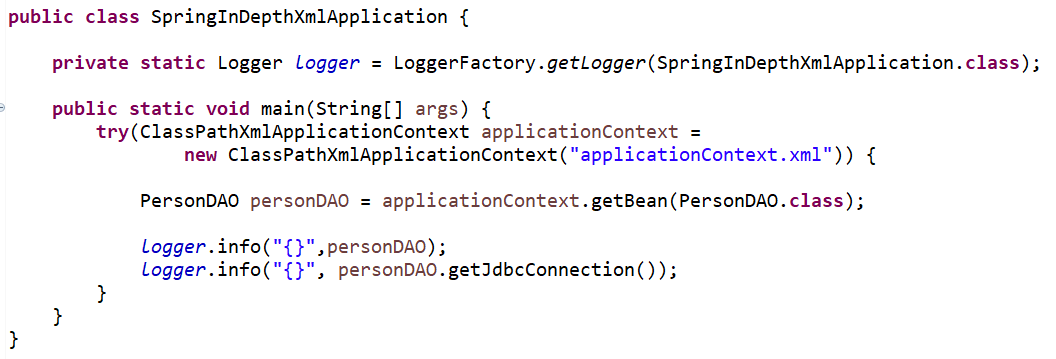
* + 1. For beans use @Component, @Autowiring and @Scope annotation
    2. For @PostConstruct and @PreDestory add below dependency



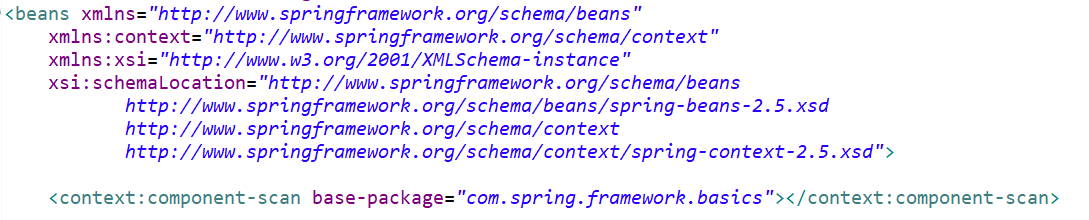
* 1. Defining Spring Application Context using xml
     1. <https://docs.spring.io/spring-framework/docs/2.5.x/reference/beans.html>
     2. Create applicationContext.xml file under src/main/resources/



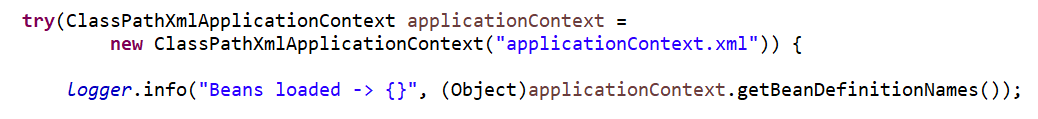
* + 1. Creating Application Context object using ClassPathXmlApplicationContext and get beans



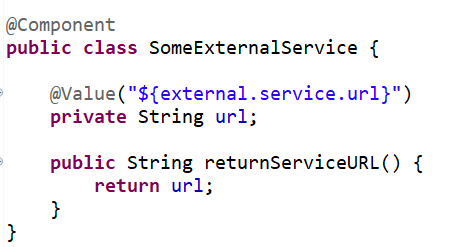
* 1. Mixing Xml Context with Component scan for Bean definition (in place of @ComponentScan in xml)
     1. Add <context:component-scan> tag in xml file as below



* + 1. It will scan for bean in base package and load them.



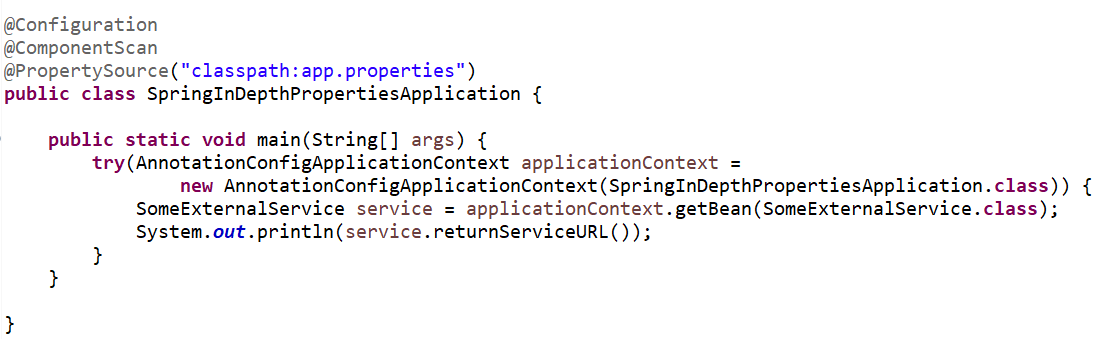
* 1. IOC Container vs Application Context vs Bean Factory
     1. IOC Container
        1. It will manage beans and auto wire dependency.
        2. Its not an actual implementation.
     2. Application Context
        1. Its an implementation of IOC Container
        2. Spring recommends using Application Context instead of Bean Factory
        3. Its an extension of Bean Factory
        4. With bean factory, its also provide other features like
           1. Spring AOP,
           2. I18n capability,
           3. WebApplicationContext for web application etc.
     3. Bean Factory
        1. Its an implementation of IOC Container
  2. Component Annotations
     1. @Component – generic component
     2. @Repository – encapsulating storage, retrieval, and search behavior typically from a relational database
     3. @Service – Business Service Facade
     4. @Controller – Controller in MVC pattern
  3. Read values from external properties file
     1. Use @Value annotation to map with properties value



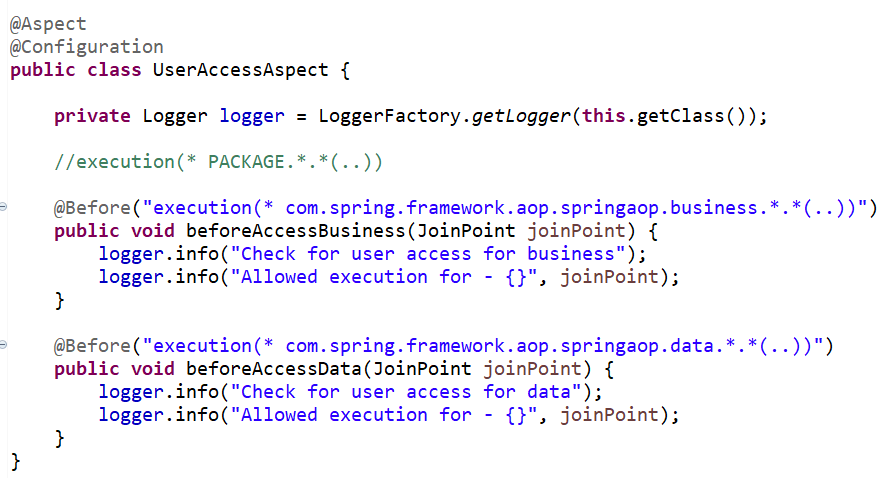
* + 1. Create app.properties file in src/main/resources and set property



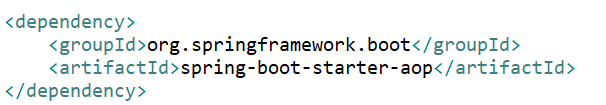
* + 1. Use @PropertySource annotation to use properties file



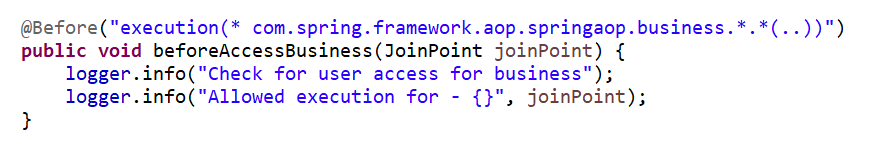
1. Spring AOP
   1. Introduction
      1. AOP is the best approach for implementing cross-cutting concerns like logging, security etc.
      2. Point Cut – The expression which defines what kind of method It would want to intercept is called a point cut.
      3. Advice – The logic which execute when a method intercepted is called advice
      4. Aspect – An aspect is a combination of your joint point + your advice. What kind of methods to intercept + what to do that your aspect.
      5. Joint Point – the intercepted method is called joint point.
      6. Proceeding Joint Point- used in case in @Around aspect. It intercept the method and again using proceed() allow the execution.
      7. Weaving & Weaver - Spring AOP framework is ensuring that this is getting executed at the right moment. This process is called Weaving. And the framework which does this is called the weaver.



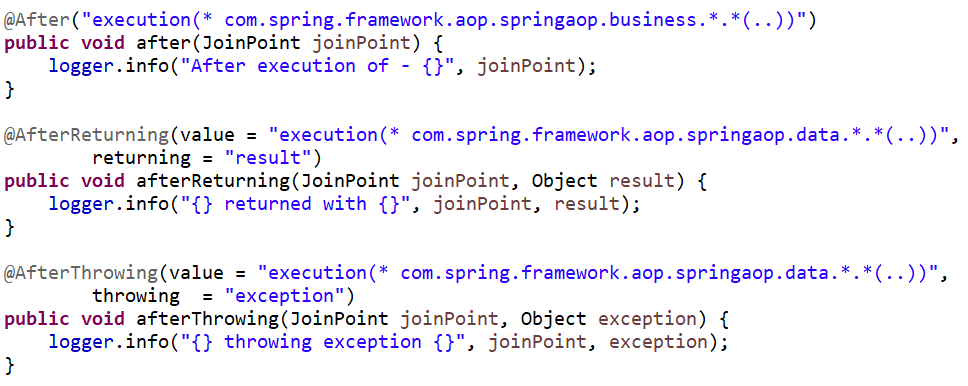
* 1. Create Spring Project and add below dependency



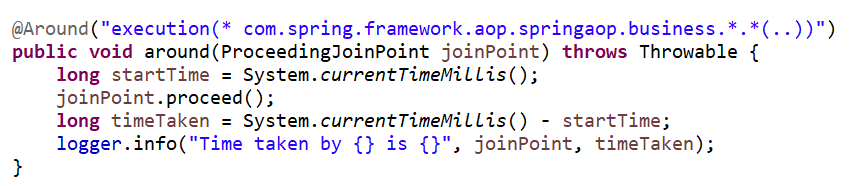
* 1. @Before advice
     1. It is used to execute some logic before intercepting the method.
     2. It can used to perform security check for method level.



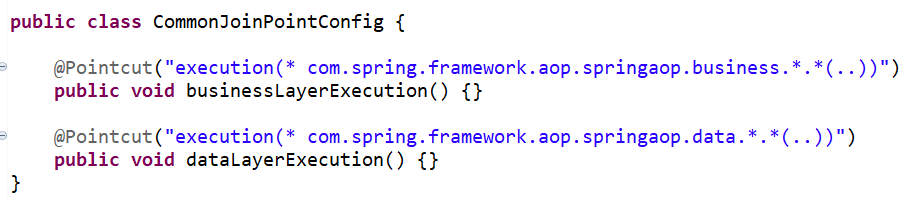
* 1. @After advice
     1. It is used to execute some logic after method execution.
     2. @After – it is called after executing the method either return or throw exception
     3. @AfterReturing – it is called after returning the value
     4. @AfterThrowing – it is called after throwing the exception



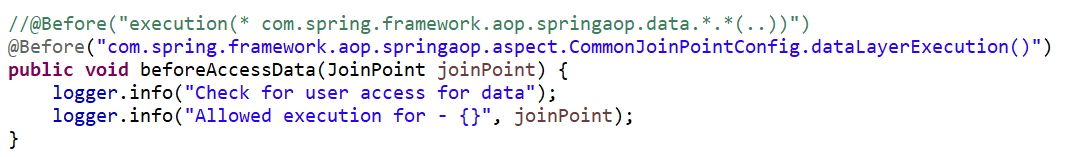
* 1. @Around Advice
     1. It can be used to implement performance tracing
     2. Uses ProceedingJointPoint instead of JointPoint to proceed execution.
     3. Run both times before executing method and after execution completion.



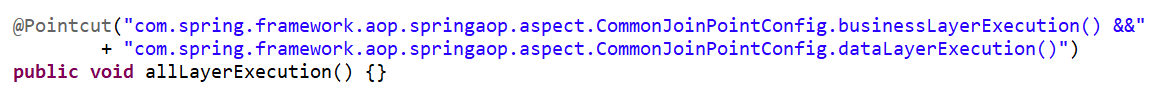
* 1. Use common point cut configuration - @Pointcut
     1. Use @Pointcut annotation to define point cut



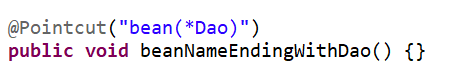
* + 1. Use qualified name of function instead of execution expression



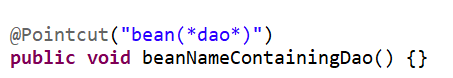
* + 1. Other pointcuts
       1. Combining two point cuts using &&



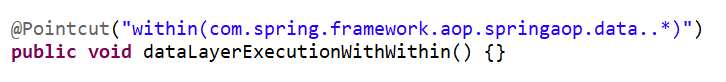
* + - 1. Check for bean ending with “Dao”



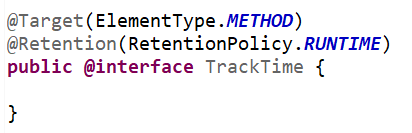
* + - 1. Check for bean containing “dao”



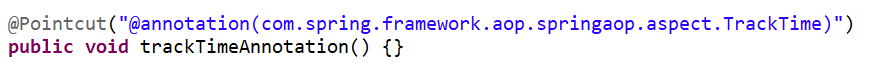
* + - 1. Within the specific package



* 1. Creating Custom Annotation and an Aspect for Tracking Time
     1. Create TrackTime annotation and add @Target for which level it will use and @Retention for executing during run time



* + 1. Create point cut for this annotation



* + 1. Use this point for advice

