1. **What is the dependency injection?**

It’s a a technique of creating software in which objects does not need to create their dependencies on itself, instead it will be provided by external object.

So we don’t have to change the code every time we change the dependency.

Types of dependency injection:

1. Constructor injection.
2. Setter injection.
3. Interface injection.
4. **What is a pattern? and what is anti-pattern?**

Pattern is a way to solve the software problem and to write code that can be reuse , and it’s the best practice that developer can use.

Examples:

Builder.

Factory method.

Template method.

Strategy method.

Anti-pattern is the opposite of a pattern .

1. **What is application-context?**

It’s a central part of spring it hold beans definitions.

It do the following :

1. Init beans.
2. Assemblies beans.
3. Manage bean lifecycle.
4. It’s a bean factory.
5. It’s a resource loader.
6. Has the ability to push events to registered event listener.
7. **What is container? and what is the life cycle of container?**

It’s a code technique that allow us to focus on creating business aspect while technical aspects like manage (http,database,soap…etc) will be handled by spring containers.

Container lifecycle:

1. Application started.
2. Spring container is created.
3. Container reads configurations.

AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(ApplicationConfiguration.class)

1. Beans definitions are created from configs.
2. BeanFactoryPostProcessors are processing bean definitions.

public class CustomBeanFactoryPostProcessor implements BeanFactoryPostProcessor {  
 @Override  
 public void postProcessBeanFactory(ConfigurableListableBeanFactory configurableListableBeanFactory) throws BeansException {  
 System.*out*.println(getClass().getSimpleName() + "::postProcessBeanFactory Listing Beans Start");  
 Arrays.*stream*(configurableListableBeanFactory.getBeanDefinitionNames()).forEach(System.*out*::println);  
 System.*out*.println(getClass().getSimpleName() + "::postProcessBeanFactory Listing Beans End\n");  
 }  
}

1. instances of spring beans are created.
2. Spring beans are configured and assembled-resolve property and inject dependencies.
3. BeanPostProcessors are called.
4. Application runs.
5. Application shutdown.
6. Spring context closed.
7. Destruction callback are invoked.
8. **How are we going to create a new instance of an ApplicationContext?**
9. Non-web applications:

* AnnotationConfigApplicationContext (from annotation config)
* ClassPathXmlApplicationContext (from xml file config)
* FileSystemXmlApplicationContext (from a xml file).

1. Web Applications (wildfly,jboss,glassfish)

* Web.xml ContextLoadedListener and DispatcherServlet.

ContextLoaderListener will load the beans from xml bean file provided.

The initialization is done thro the web.xml

* XmlWebApplicationContext.

The bean initialization done thro WebApplicationInitializer class.

* AnnotationConfigWebApplicationContext.

The bean initialization done thro AnnotationConfigWebApplicationContext and here we need to pass Config class annotated with @ComponantScan it will scan all the beans in the same package.

1. Spring Boot:

* Spring Boot ConsoleApplication – CommandLinerRunner.
* SpringBootWebApplication – Embaded Tomcat.

Refer to module01-q6

@Bean can be defined without a problem in @ComponentScan class because of Configuration Class Candidate Indicators. Any class annotated with:

- @Component , @ComponentScan ,@Import , @ImportResource () referring to module01-q5

1. **Can u describe the lifecycle of Spring bean in application context?**

Context is created.

* Bean definitions are created on spring bean configuration.
* beanFactoryPostProcessors are invoked.

Bean is created:

* instance of bean is created.
* Properties and dependencies are set.
* BeanPostProcessor::beforeInit called.
* @PostConstruct method called
* InitilizationBean:afterProperiesSet method get called.
* @Bean(initMethod) method called.
* BeanPostProcessor:aferInint called.

Bean ready to user.

Bean destroyed.

* @preDestroy called.
* DisposalBean::destroy method called.
* @Bean Destory method called.

Module01-q07 important to read and execute.

1. **How are you going to create an ApplicationContext in an integration test?**

Make sure to add spring-test dependency .

Add spring runner to test @RunWith(SpringRunner.class)

Add context configuration to the test @ContextConfiguration (<your configuration class>)

1. **Whats the preferred way to close an application context? Does spring boot do this for you?**

* Non-web applications:

Register a shutdown hook by calling ConfigurableApplicationContext::registerShutdownHook.

Or call ConfigurableApplicationContext::close()

AnnotationConfigApplicationContext will auto close context.

* Web application( glassfish,jboss..) and spring boot:

Application Context will be auto closed and shutdown will be auto registered.

Refer to module01-q09

1. **Can u describe dependency injection in spring using configuration?**

When using Dependency Injection using Java Configuration you need to explicitly define all your **beans** in **configuration** and you need to use @Autowire on @Bean method level to inject dependencies.

Text

Description automatically generated

So in this example we inject 3 beans and springbean1 has 2 dependencies (springbean2,springbean3) that’s why we add @Autowired.

Here in the above example springBean1,2,3 classes is just normal classes without any annotations like (component,…etc).

1. **Can u describe dependency injection in spring using annotation (@component,@Autowired)?**

Text

Description automatically generated

Injection can be:

1. Field injection.
2. Constructor injection.
3. Setter injection.
4. **Can u describe more component scanning ?**

Graphical user interface, text

Description automatically generated

1. **What is the stereostypes?**

Graphical user interface, text, application

Description automatically generated

1. **What is meta controller ?**

Graphical user interface, text, application

Description automatically generated

1. **What is scope types in spring?**

Table

Description automatically generated

Text

Description automatically generated

Everytime we ask for springBean2 it will create a new instance,

1. **What is BeanFactoryPostProcessor?**

Its an interface allows u to modify bean metadata and its defined always as a static so it will be initialized fist before any bean.

1. **What is BeanPostProcessor?**

It’s a class allows you to modify bean object and it has two method:

1. Before init

In between before init and after init we have:

@PostConstructor will get called.

@AfterProperiesSet will get called.

@Bean(initMethod) will get called.

1. After init.

It should be static also so it will be initialized first.

1. **What is destroy method in bean and how is it declared?**
2. @PreDestory method.
3. @destory method.
4. @Bean(destoryMethod)

Text

Description automatically generated

Text

Description automatically generated

1. **Are beans lazily or eagerly initiated ?**

Graphical user interface, text, application, email

Description automatically generated

1. **What is a property source?**

Graphical user interface, text, application

Description automatically generated

Note: that jvm,system env,jndi,servelet params is already read for you.

Note: @PropertySource is defined with ComponantScan or Configuration annotations.

Text

Description automatically generated

Text

Description automatically generated

1. **How to enable JSR-250 annotation like @PostConstruct and @PreDestory ?**

When we use AnnotationConfigApplicationContext JSP-250 will be already supported.

1. **What is the behaviour of @Autowired?**

It can be used on top of constructor , field and set methods.

Text

Description automatically generated

1. **@Autowired with field injection?**

Text

Description automatically generated

Text

Description automatically generated

Note: than RecordsHash and RecordsUtil and RecordsValidator are just interfaces without any annotations on top on them (they are not beans) that’s why we mark as optional or nullable or required=false, but for other classes like DbRecordsReader and DbrecordsBackup those classes marked with annotation @Componant that’s why we use @Autowired safely with them.

1. **@Autowired with constructor?**

Text

Description automatically generated

NOTE: if we have multiple constructor then we must use @Autowired with ONE of them only.

Text

Description automatically generated

This will give complier error as spring does not know which constructor to use to inject the beans.

Unless we remove @Autowired from the second constructor then it will works fine.

1. **@Autowired with setter method?**

Graphical user interface, text

Description automatically generated

Graphical user interface, text

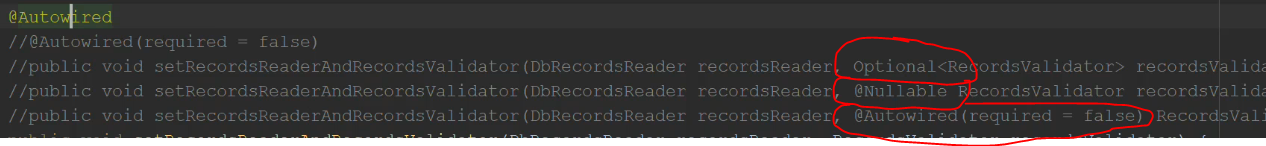
Description automatically generated

Here is this example we can see that @Autowired(required=false) means if spring failed to resolve on of the params in the header then it will **NOT** call this setter method at **ALL.**

as we cann see DBRecordsReader is annotated with @componant but RecordsValidator is not annotated with @componant and its just a normal class, that’s why it will fail to resolve these dependencies and it will not call this setter at all.

If we set @Autowired without required=false then it will thro an exception bcoz RecordsValidator is not annotated with @component and could not be resolved,

Unless we make RecordsValidator @Nullable or optional.



Final note about @Autowired,

Autowired assumed that all param in the header are resolvable (means annotated with a stereotype like @componant), in one of these params not resolved (the class not annotated) then it will thro exceptions.

We can avoid exceptions by setting @Autowired(required=false) but this will not be called if one of the param is not resolved,

Like this case:

Text

Description automatically generated

The above method will not get called bcoz RecordsValidator is not annotated with @componant,

Text

Description automatically generated

The above method will get called coz we mark RecordsValidator with @Nullable.

1. **How do you inject into a private field from unit test ?**

Text

Description automatically generated

So here we want to test this class ReportService:

Text

Description automatically generated

We can use springrunner to test:

Text

Description automatically generated

Or we could use MokitoJUnitRunner

Text

Description automatically generated

Or we use reflection with using the name, but the problem in this test if we change the field name then this test will not work.

Text

Description automatically generated

1. **What is @Qualifier and how is it works?**

Text, letter

Description automatically generated

1. Inject by bean name:

Graphical user interface, text

Description automatically generated

Here is RecordsProcessor is an interface and there are two classes implement this interface that’s why we need to use the @Qualifier, dbRecordsProcessor match the class name so we don’t have to specify the qualifier name .

Graphical user interface

Description automatically generated

1. Inject using @Qualifier:

A picture containing graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Or we could use @Qualifier on top of the class:

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

1. Using custom annotation:

Text

Description automatically generated

Text

Description automatically generated

1. **What is proxy objects? And what is the types of proxy objects?**

Text, chat or text message

Description automatically generated

Text

Description automatically generated

1. **What’s the advantages and disadvantages of using the proxy?**

Graphical user interface, text, application, email

Description automatically generated

1. **Can u define how jdk implemented?**
2. We need the invoker handler that handle the execution before and after the execution:

Text

Description automatically generated

PersonDao is the interface, and invoke code will invoke the original method.

1. Code to initialize the jdk proxy:

Text

Description automatically generated

Here PersonalDaoImp is the implementation of PersonDao interface.

1. **How to you implement cglib proxy?**
2. We need the method interceptor:

Text

Description automatically generated

1. Code to initialize cglib proxy:

Text

Description automatically generated

Note: that DepartmentDao is just normal class (not interface),

1. **What are the advantages of java config?**

Text

Description automatically generated

We cant make @Configuration class final bcoz during the runtime spring will create a cglib proxy for this class and as we know that cglib does not work with final class and final methods.

Text

Description automatically generated

Text

Description automatically generated

1. **What is bean annotation do? (important)**

Text

Description automatically generated

As we can see in the example above SpringBean1 has 2 dependencies first one springBean2 and SpringBean3 , springBean2 its resolvable bcoz we have only one bean Type SpringBean2 so it will compile without any issue, but for SpringBean3 its an **interface** and there are 3 bean in config class returning SpringBean3.

So what will spring do is first:

1. Try to match the type -> he will fail bcoz we have three beans match the same type.
2. Try to match the name -> it will work coz the param name is springBean3rd and it match the bean alias name @Bean(name=”springBean3rd”).
3. Try to match with the method name ->

Text

Description automatically generated

And the results will be:

Graphical user interface, text, application

Description automatically generated

There is also a way to solve this by using also a @Quialifier and we set the name of the bean

Text

Description automatically generated

Or we make one bean available by making the other 2 beans candidate=false, so that spring will inject 1 springbean3 only since the other 2 beans we set **autowirecandidate=false**

Text

Description automatically generated

1. **What is the default bean id?**

Graphical user interface, text, application

Description automatically generated

1. **Why you are not allowed to annotate a final class with @configration ?**

Text

Description automatically generated

Text

Description automatically generated

As we can see we have two bean the first one is using the default scope (singleton) and the second one used prototype scope, the first bean will get initialized only once, but the second bean will be initialized every time get called.

Text

Description automatically generated

And the results will be:

Text

Description automatically generated

1. **How do you configure a profiles in spring?**

Text

Description automatically generated

1. With component annotation (@componantscan will scan the annotations in classpath)

Here we have FinancialDataDao is an interface and there are two classes implemented it

Which is DatabaseStore and FileStore,

Graphical user interface, text

Description automatically generated

Then lets open DatabaseStoreFinancialDao

Text

Description automatically generated

We can see that this class annotated with @profile(database) means when applicationContext set “database” as active profile then this class will get scanned.

Text

Description automatically generated

And we see the FileStoreDataDao not called bcoz its annotated with @Profile(file)

Text

Description automatically generated

1. Using the @Configuration:

We define two config file each config annotated with the profile:

Graphical user interface, text

Description automatically generated

DatabaseApplicationConfig:

Text

Description automatically generated

FileApplicationConfig:

Graphical user interface, text, website

Description automatically generated

Then we have the main config that import these two configs:

A picture containing website

Description automatically generated

The FinancialReportService does not have any profile so it will be injected for all profiles.

1. Using bean in the configuration:

Text

Description automatically generated

1. Using custom annotation:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

These custom annotations can be used with @componant or @beans

Graphical user interface, text, website

Description automatically generated

1. **How You activated @Profile in spring?**

Text, letter

Description automatically generated

1. Programmatically:

Text

Description automatically generated

1. In junit test:

Text

Description automatically generated

1. In springboot using SpringApplicationBuilder:

Text

Description automatically generated

1. In springBoot using application.properties:



1. From vm property:

Text

Description automatically generated

1. **What are the use cases of using @Profile?**

Text

Description automatically generated

1. **Can you use @Bean together with @profile?**

Graphical user interface, text

Description automatically generated

1. **Can you use @Componant together with @Profile?**

Graphical user interface, text, application

Description automatically generated

1. **How many profiles can you have?**

Text

Description automatically generated

1. **How do you inject scalar/literal values into Spring beans?**

Text

Description automatically generated

1. Simple value:



1. spEl String, note that spel sttril start with #

Text

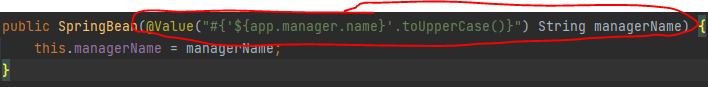
Description automatically generated

1. property reference:

A picture containing text, device, orange, meter

Description automatically generated

1. constructor method:



1. on top of method, note that all params in method param header will be injected with this value:

Graphical user interface, text

Description automatically generated

1. method param, note that it should be annotated with @Autowired as well,

Text

Description automatically generated

1. for array values we should implement a proper conversion :

Text

Description automatically generatedA picture containing text, device, meter, gauge

Description automatically generated

1. if the property not available or present then we can set default value:

Graphical user interface, text

Description automatically generated

1. **where can properties values come from?**
2. For standalone application:

Text

Description automatically generated

1. For web application:

A picture containing text, indoor, screenshot

Description automatically generated

1. For spring boot application:

A close-up of a document

Description automatically generated with low confidence

1. We can set the property file name as application-<profile name> and we can activate the profile:

In application.properties file:

A picture containing graphical user interface

Description automatically generated

1. **What are the reference of using SpEL?**

Text

Description automatically generated

Must check the code module01-q33

1. **Whats the difference between $ and # in @value?**

