ANNOTATION CONFIGURATION USING XML

ANNOTATION CONFIG :

* In this module we will walk through the configuration of Spring using XML and Annotations.
* We are going to take the sample application that we developed in the previous module and wire up that application to using annotations configured through XML.
* It was the second method available in Spring.

COPY DEMO :

* At the end of the last module , we had two projects that we completed.
* spring\_sample and spring\_sample\_xml
* In the last module we had done autowiring was up so that they could be dynamically discovered through our XML classes.
* We are not going to create beans inside our applicationContext.xml , rather we would go on to use annotations.
* Copy the spring\_sample.
* Paste
* Give name : spring\_sample\_xml\_anno
* Open the pom.xml
* Change the artifactId to spring\_sample\_xml\_anno
* Run and see whether it works as it is.

APPLICATION CONTEXT & DEMO:

* We need to place our Annotation configuration scanner somewhere in our code therefore we place it in the applicationContext.xml
* We have an Annotation scanner and it goes and looks for specific items.
* At the top of our applicationContext.xml we have our namespaces , beans defined that helps us build our application up.
* Let us add the applicationContext.xml to our new project before we start adding anymore annotations or scanners.
* RC on SRC/MAIN/RESOURCES directory.
* New -> Spring Bean Configuration File.
* File Name : applicationContext.xml -> Finish
* It will define our namespace and schemaLocation for us , its our starting point or bootstrap point for our application.

COMPONENT SCANNER AND DEMO :

* Check the folder for screenshot of Component Scanner XML Declaration within the applicationContext.xml file.
* We need to add a couple of things before initializing our Component Scanner   
  First : Context namespace  
  Second : Schema Defining context and XSD Notation
* Add both to the top of the document.
* We need to tell our application next that it is configured to using annotations.
* Where to start scanning for those annoataions. (The specific package com.pluralsight)
* Open applicationContext.xml
* Go to the namespaces tab and tell it to use “context” namespace.
* Select the spring-context.xsd from the right tab.
* I want it to be the non-version specific one. If we install 4.4 its not going to be limited to 4.3.
* Switch back to the source tab and see that its already added.
* Add   
  <context:annotation-config></context:annotation-config>  
  <context:component-scan base-package=”com.pluralsight”/>
* Our application is now ready for scanning annotations.

STEREOTYPE ANNOTATIONS & DEMO :

* There are three main annotations that are available for us in core Spring that enable us to find components or beans inside our application.
* They typically refer to these as stereotype annotations.
* The main three are @Component , @Service , @Repository. – org.springframework.stereotype
* Semantically they are the same , they don’t add any feature or functionality (@Service,@Repository they both extend @Component).
* @Component is used for regular components or beans basically any POJO.
* @Service does not stand for web service , its actually the tier where our business logic is contained. We don’t want them to be within the controller and mark them with @Service.
* @Repository is our data access layer or the DAO layer , our interaction with our database , SpringJDBCTemplate or JPA , it gets labelled with @Repository.
* Open up HibernateCustomerRepositoryImpl.java
* Above the Class - @Repository(“customerRepository”) -> Save.
* Open up CustomerServiceImpl.java
* Above the Class - @Service(“customerService”) -> Save.
* In the CustomerServiceImpl.java , we have a CustomerRepository interface that refers to the HibernateCustomerRepositoryImpl.
* Let us discuss autowiring now.

AUTOWIRED & DEMO :

* Many people think that using Autowiring is much more better and simpler.
* Autowiring a method is hidden because its tied to where we are placing the Annotation at. “Tied to a location.”
* We can autowire at three places :   
  Member Variables  
  Constructor  
  Setter Injection.
* Look in the folder for screenshot of each one of the above methods.
* @Autowired  
  private CustomerRepository customerRepository
* Using the component scanner , Spring will look for any code marked with the stereotyped annotations and the autowired annotation and wire up your application appropriately.
* Lets get on with removing the hardcoded reference for HibernateCustomerRepositoryImpl.
* Remove the reference.

MEMBER INJECTION DEMO :

* Add @Autowired at the top of the declaration of the CustomerRepository reference declaration.
* Go to Application.java
* Copy the Application.java code that we wrote in spring\_sample\_xml project and paste it within this Application.java.
* RC -> Run As -> Java Application.
* Now our beans are autowired specifically the CustomerRepository and everything was injected and its loaded using the applicationContext.xml

SETTER INJECTION DEMO :

* Autowiring at the setter level is also very straightforward , and in some ways closer to in a way where we would develop our application without Spring.
* We generate the setter and place the annotation at the top of the setter.
* The spring component scanner will look for any code that is wired with the stereotype annotation and the autowired annotation and wire up our application.
* Open CustomerServiceImpl.java
* Get rid of the autowired at the member level.
* RC on customerRepository -> Source -> Generate getters and setters.
* Select only the setters.
* Place @Autowired on top of the setter.
* Add a Syso method as the first line to just see if we are calling this method to do setter injection.
* Go to Application.java and run it.
* We get the Syso line within the setter and then returns the name as Bryan.
* We can wire up our application doing test driven development or test first development , we would call the setters.
* We can have the member level wiring.

CONSTRUCTOR LEVEL INJECTION :

* Autowiring at the constructor level is quite similar to the setter approach.
* We create a constructor of the particular type.
* We apply @Autowired to the constructor on top of it.
* We have to be careful if we shift to member or setter injection because we would have deleted our default constructor.
* Open up CustomerServiceImpl.java
* Comment out the setter autowired annotation line.
* Create a constructor with CustomerRepository reference as a argument to the constructor.
* Annotate the constructor with @Autowired at the top of the constructor.  
    
  @Autowired  
  public CustomerServiceImpl(CustomerRepository customerRepository)  
  {  
  System.out.println(“Using constructor injection”);

this.customerRepository=customerRepository;  
}

* Application.java and run it as Java Application.
* Notice that there are no beans that are wired up within our applicationContext.xml
* All we needed to do is the Annotation Config and the Component Scanner from context namespace.
* We can use setters and constructors in our application and keep out that configuration in a different place and so that this whole project can also be run without Spring.
* It’s a Hybrid approach.
* We did member level , constructor level and setter level.
* Look at the code , we don’t have any customerRepository implementations within our CustomerServiceImpl class and this shows that we have loose coupling and we are doing everything through our interfaces.

JSR 330 :

* What is the JSR-330 specification?
* Java started to see the benefits of dependency injection , it decided to import into its code some very lightweight dependency injection components.
* It is not as featurish as what Spring provides.
* It’s a method of dependency injection specification for Java.
* It has simple annotation.
* Why use Spring? It provides a lot more than the basic dependency injection and it’s a lot more feature rich libraries and makes life easier , coordinating different libraries to work together.

SUMMARY :

* In this module we still needed to have an applicationContext.xml
* Bootstrap our component scanner and look at all our package structure.
* Basic stereotype annotation and how its used within each tier.
* Autowiring methods.