BEAN SCOPES

BEAN SCOPE INTRODUCTION :

* In this module , we will discuss the various scopes that are available within the Spring framework , and how you can use that while configuring your beans.
* Bean scopes are a very important part of the bean life cycle.
* We will see how to use different scopes so that we will get different behavior from our application.
* We will be modifying our existing projects.

PATTERNS :

* Bean Scopes and Patterns are not the same thing , they usually go hand in hand.
* Spring implements a lot of patterns for us , so that we don’t do any mistakes or pitfalls while implementing those patterns on our own.
* Look at the design patterns course , behavioral , creational and structural design patterns.

SCOPES :

* There are 5 scopes that are available inside a Spring application for us to configurate bean to use  
  Valid in any configuration – Singleton – Prototype   
  Other configuration available for Bean in Web Aware projects – Request – Session and Global.

SINGLETON :

* The singleton design pattern restricts the instantiation of a class to one object.
* Singleton is the default bean scope inside of Spring.
* So , if we do not assign a scope , then it will be default set to Singleton.
* That means that there is only one instance per Spring Container , ie per context.
* Some people say that its only one instance per JVM , but we can have more than one Spring Container spun up in the JVM.

SINGLETON JAVA CONFIG & DEMO :

* We are going to use the Java Config approach of annotating our class with @Scope(“singleton”).
* In the previous version of this course , we were supposed to add a separate AOP jar for utilizing this annotation , but now since we have Maven , things are easier.
* Have spring\_sample\_java alone in the workspace and Close all other projects.
* Open up CustomerServiceImpl.java
* Add @Scope after @Service as   
  @Service(“customerService”)  
  @Scope(“singleton”) – org.springframework.context.annotation.Scope
* If we don’t add the text in there saying singleton , it would default to singleton.
* Open up Application.java
* Add  
  System.out.println(customerService) - print out the object address
* Copy the CustomerService instance and paste it again with the System.out.println() statement of the object address.
* Give another CustomerService instance called customerService2.
* RC -> Run As -> Java Application.
* We can see that we are getting the same object address each time.
* Refer the screenshot in the folder.
* Even though we have two different CustomerService instances , we get the same object address each time from container.
* Now that is Singleton , its giving the same object back.

SINGLETON JAVA CONFIG USING A CONSTANT DEMO :

* We have had people complaining that we are having raw strings within our Annotations inside our application.
* We can definitely create our own Enum , our own static constants and store that String within the @Scope.
* I would like to tell you that Spring has a public static final String called the ConfigurableBeanFactory.SCOPE\_SINGLETON that is available for us to use inside our code for that String.
* It resides in the package – org.springframework.beans.factory.config.ConfigurableBeanFacotry
* Go ahead and run the application.
* It still prints the Singleton output we got when we did @Scope(“singleton”).

SINGLETON XML CONFIG & DEMO :

* <bean name=”customerService” class=”com.pluralsight.service.CustomerService”  
  autowire=”byType” scope=”singleton”/>
* The above code is for creating a singleton scope for bean of type customerService within our XML configuration.
* We need to add the attribute scope=”singleton”.
* Unlike Java the AOP jar is not required for XML but Spring keeps it as a transitive dependency in Maven.
* Open up spring\_sample\_xml and close all other projects.
* Open up SRC/MAIN/RESOURCES and open up applicationContext.xml
* Go inside the bean called customerService , add the attribute scope=”singleton”
* That’s all I have to do to add the scope to our XML declaration.
* All beans again here by default are singleton. Our CustomerRepository that we defined above is by default singleton cause we haven’t specified any scope.
* Open up Application.java
* Add a System.out.println() for both of the CustomerService instance that we create and check whether the object addresses are the same.
* RC -> Run As -> Java Application.
* Lets now look at the Prototype scope.

PROTOTYPE

* We are going to get a different object address everytime we request for an object from the container.
* The prototype design pattern guarantees a unique instance per request , each time we request a bean from the container , you are guaranteed a unique instance.
* It is essentially the opposite of a singleton.
* Since the configuration of a prototype is identical to a singleton , lets dive into the code and see how it works.

PROTOTYPE XML DEMO :

* Have your spring\_sample\_xml project open.
* Go back to applicationContext.xml
* Go to where we defined the scope=”singleton” and change it to scope=”prototype”
* That’s all I have to do to change it from singleton to prototype.
* It guarantees a unique instance with each request now.
* Go back to Application.java
* Run -> Run As -> Java Application.
* Now when you see the console , we see two different object address created for each instance of CustomerService.

PROTOTYPE JAVA CONFIG DEMO :

* To show the prototype configuration in the Java Project , open up spring\_sample\_java
* Go to CustomerServiceImpl.java
* Go to where we defined our scope as @Scope(ConfigurableBeanFactory.SCOPE\_SINGLETON) and change it to @Scope(ConfigurableBeanFactory.SCOPE\_PROTOTYPE)
* There is that public static final String in the ConfigurableBeanFactory for us to utilize.
* Open up Application.java
* Run -> Run As -> Java Application.
* You can see on the console , that we are getting two different object addresses for the different instances that we created of the CustomerService.
* If we are not using the public static final String, we can directly specify as @Scope(“prototype”)
* Look at the power that Spring provides you with these annotations , I can go on and change it from singleton to prototype and directly change the behavior of my application.

WEB SCOPES :

* Web scopes are beyond what we are going to cover in this course.
* They are covered more in the introduction to Spring MVC course.
* We have to set up an entire web application for you to see how they interact with the object.
* Request – Which returns a bean / HTTP request which sounds a lot like the prototype but for the lifecycle of a Request.   
  Session – Which returns a single bean / HTTP Session and will live as long as the user session is alive. We can also set a timeout of 10min or 20min or however long they are on the website.  
  Global Session – Which will return a single bean / Application and will be shared across all users of that application. Once I access it , its available for the duration of my application.

SUMMARY :

* In this short module , we discussed the various scopes that are available within the Spring Container.
* We saw the configuration or Java and XML
* We saw above the web scopes.
* Singleton , Prototype , Request , Session , Global Session.