PROPERTIES

PROPERTIES INTRODUCTION :

* In this module , we will discuss how to import properties files using the Spring Framework.
* Properties files are a great way to abstract out values that can change with each environment.
* Passwords , URLs , Connection statements
* Using properties files we will be able to keep the above code separated from our business logic.
* If we have to change to another environment , the hardcoded code in our application needs to be changed every single time.

XML CONFIG & DEMO :

* The XML configuration for loading a properties file into our applicaton is quite simple.
* XML config is simpler than Java config.
* We need to declare a property placeholder in our XML  
  <context:property-placeholder location = “app.properties”/>
* Behind the XML snippet , it creates a property placeholder configure is loaded from the location attribute.
* To utilize the properties we use the   
  value = “${dbUsername}”
* To inject that value into our bean.   
  <bean name=”customerRepository” class=”com.pluralsight.repository.HibernateCustomerRepositoryImpl”>  
  <property name=”dbUsername” value=”${dbUsername}”/>  
  </bean>
* The above statement mentions that we are going to call the property name called “dbUsername” from our app.properties file that we place under SRC/MAIN/RESOURCES and the value that we are going to inject into our customerRepository bean is defined inside the value attribute called as ${dbUsername}.
* Open up sample\_spring\_xml
* RC on SRC/MAIN/RESOURCES
* New -> File -> app.properties -> Finish
* Add the following lines :   
  dbUsername = mysqlusername
* Open up applicationContext.xml
* Go to the namespaces tab and choose context namespace.
* Choose the namespace without the version number.
* Type the following  
  <context : property-placeholder location=”app.properties”/>
* Go within the customerRepository bean.
* Add the following  
  <property name=”dbUsername” value=”${dbUsername}”/>
* Open up HibernateCustomerRepositoryImpl.java
* Add a field.   
  private String dbUsername;
* Source generate getters and setters -> select only setter.
* Go inside the findAll() method of return type list  
  Add a Syso line within it ,   
  System.out.println(dbUsername);
* Go to Application.java
* Run As – Java Application
* We can see that it has brought that value from app.properties for dbUsername injection.

XML CONFIG INJECTION DEMO :

* Instead of injecting that value with hardwired XML we can now use annotations.
* Get rid of the setter for dbUsername.
* Go to the field   
  @Value(“${dbUsername}”)  
  private String dbUsername;
* We have got rid of the setter and used the @Value annotation.
* We have to change two things within our applicationContext.xml
* Get rid of the <property> tag.
* Since we are now using Annotations , we have to tell Spring that there are annotations to look out for.
* Therefore we add the following  
  <context: annotation-config/>
* Go and run the Application.java
* We can see the property now injected.

JAVA CONFIG & DEMO:

* The java configuration for loading a property file in your application , is very similar to the XML configuration.
* To load a properties file , we need to declare an annotation @PropertySource(“app.properties”)
* This tells the container to look for app.properties file.
* We then need to make the properties available to our container , we need to create a bean that returns a PropertySourcesPlaceholderConfigurer();  
  @Bean public static PropertySourcesPlaceholderConfigurer getPropertySourcesPlaceholderConfigurer() {  
  return new PropertySourcesPlaceholderConfigurer();  
  }
* This is the equivalent for what we were doing with XML and <context:property-placeholder location = “app.properties”/>
* It creates an instance of a Bean to store those values in.
* When this is done , we use that @Value annotation.
* Open up spring\_sample\_java
* RC on SRC/MAIN/RESOURCES directory and Say New-> File
* FileName : app.properties
* Add the following  
  dbUsername=mysqldbuser
* Open up the AppConfig.java
* Add @PropertySource(“app.properties”) on top of the classname.  
  @Bean public static PropertySourcesPlaceholderConfigurer getPropertySourcesPlaceholderConfigurer() {  
  return new PropertySourcesPlaceholderConfigurer();  
  }
* What this class does , is take all those properties that we loaded in from the PropertySource  
  location and puts them in this class and makes them available in the context for our application to use. This is what is behind the scenes of the XML snippet.
* Open up HibernateCustomerRepositoryImpl.java
* Add a new field  
  private String dbUsername;  
  Add a @Value(“${dbUsername}”) or use a setter.
* Add a System.out.println(dbUsername) into the findAll() method of return Type List.
* Go open up Application.java
* Run As -> Java Application.
* In the console , we can see the value being injected.

SUMMARY :

* In this short module , we discussed import and use properties inside of Spring.
* Key way of configuring Spring instead of having hardwired code.
* XML and Java configuration also used Annotations like @Value @PropertySource and in XML we used the <context:property-placeholder location=””/>