ARCHITECTURE AND PROJECT SETUP

SAMPLE APP INTRO :

* Let us talk about the architecture of Spring and the application we will be building in this course.
* We will look at the basic project setup and a historical approach to what Spring solves.

ARCHITECTURE :

* We will take a minute to discuss why Spring was built the way it was.
* Spring was developed to make existing tasks easier.
* Before Spring came we used some design patterns that came from the J2EE Blueprint that helped us establish cleaner code and reusable processes.
* The J2EE blueprint made code much cleaner but made is brittle and untestable.
* If I ever had to say , recompile code because we are moving to a different environment or change things like URL or Connection because of your environment , then you will realize these are things we are trying to fix.
* The above problem is generally referred to as Write Once Read Anywhere (WORA).
* You may actually want the implementation not hardcoded inside your application.

PREREQUISITES :

* Java 8
* Maven
* Spring STS/Eclipse
* Tomcat

SAMPLE APP SETUP

* Open Spring STS -> Choose workspace.
* RC Package Explorer -> New -> Other -> Maven -> Maven Project.
* Click the checkbox to create a simple project.
* Group ID : com.pluralsight
* Artifact ID : spring\_sample
* Finish
* We have a particular folder structure that is followed by Maven.   
  SRC/MAIN -> JAVA & RESOURCES  
  SRC/TEST -> JAVA & RESOURCES  
  POM.XML

SAMPLE APP POM.XML DEMO :

* The first thing that we are going to do with our POM.xml is fix the compiler version that our IDE is using.
* If you look at the left to the Project Manager -> We will see that our compiler is currently set to J2SE 1.5
* Add a   
  <build>  
  <plugins><plugin>  
  <groupId>org.apache.maven.plugins</groupId>  
  <artifactId>maven-compiler-plugin</artifactId>  
  <version>3.2</version>  
  <configuration>  
  <source>1.8</source>  
  <target>1.8</target>  
  </configuration>  
  </plugin></plugins>  
  </build>
* Once this is done , save the POM.xml file and usually what STS or Eclipse does is update the project for you once you save it.
* But still when you look at the Project Manager it is still stuck at J2SE 1.5
* RC the project -> Maven -> Update Project -> OK.

SAMPLE APP ADD MODEL DEMO :

* Lets now add a class file to our application.
* RC on SRC/MAIN/JAVA -> New -> Class.
* Package Name : com.pluralsight.model
* Class Name : Customer
* Finish
* Add these two within the Customer.java  
  private String firstName;  
  private String lastName;  
  RC -> Source -> Generate Getters and Setters  
  RC -> Source -> Sort Members  
  RC -> Source -> Format
* We are currently still using the default no-arg constructor.
* We are not going to do any constructor injection and this is what Spring does early on  
  public Customer() {}

SAMPLE APP ADD REPOSITORY DEMO :

* The repository tier is a little bit more complex than our model tier but not that much.
* RC on SRC/MAIN/JAVA -> New -> Class.
* Package Name : com.pluralsight.repository
* Class Name : HibernateCustomerRepositoryImpl
* Use Ctrl+Shift+O to import.
* Within the HibernateCustormerRepositoryImpl implements CustomerRepository -> class add the following :

public List<Customer> findAll(){  
List<Customer> customers=new ArrayList<>();  
Customer customer = new Customer();  
customer.setFirstName(“Bryan”);  
customer.setLastName(“Hansen”);  
customers.add(customer);  
return customers;  
}

* Therefore we had a Class called HibernateCustomerRepositoryImpl where we added a method that returned a List<Customer> where we specified a List<Customer> as a new ArrayList to store all the values given by the Customer Object and we added it to the list and returned the list.
* If we have to use this in Spring , then we need to code the above into a Interface.
* RC the com.pluralsight.repository package -> New -> Interface and code that hardcoded.
* The other thing we could do is :   
  RC the HibernateCustomerRepositoryImpl -> Refactor -> Extract Interface.
* Interface Name : CustomerRepository and click the methods that needed to be added to the Interface.
* Click Ok.  
  It went on to create a CustomerRepository Interface right beside the HibernateCustomerRepositoryImpl.
* The code within the Interface looks like :  
  package com.pluralsight.repository  
  import java.util.\*  
  public interface CustomerRepository {  
  List<Customer> findAll();  
  }

SAMPLE APP ADD SERVICE DEMO :

* The next tier is our CustomerService tier where our business logic would be created and stored.
* Always the business logic resides in the service tier. For a web service the interface may be in the WEB directory but its implementation and business logic should be in the service tier.
* RC on SRC/MAIN/JAVA -> New -> Class
* Package Name : com.pluralsight.service
* Class Name : CustomerServiceImpl
* Within the CustomerServiceImpl implements CustomerService -> Add the following code :  
    
  private CustomerRepository customerRepository = new HibernateCustomerRepositoryImpl();  
  public List<Customer> findAll() {   
  return customerRepository.findAll();  
  }
* Notice here that we have a hardcoded reference to our CustomerRepository.
* This is where Spring starts to step in so that we don’t have these pieces hardcoded into our application.
* Just like our repository tier we are going to extract an interface.
* RC on CustomerServiceImpl -> Refactor -> Extract Interface
* Interface Name : CustomerService
* Select the methods you need to include into the Interface and click OK
* The interface looks like :   
    
  package com.pluralsight.service  
  import java.util.\*;  
  public interface CustomerService {  
  List<Customer> findAll();  
  }

SAMPLE APP ADD APPLICATION DEMO :

* To create our application RC on SRC/MAIN/JAVA -> New -> Class
* We are going to use the default package on this.
* Class Name : Application and click the checkbox for MAIN() method.
* The code is displayed like :  
    
  public class Application   
  {  
  public static void main (String[] args)  
  {  
  CustomerService service=new CustomerServiceImpl();  
  System.out.println(service.findAll().get(0).getFirstName()); // Knowing that we have only 1 record and since its returning a List<Customer> we use the .get() method to get the value stored.  
  }   
  }
* Save the above and Click on Run As and say Java Application
* We get “Bryan” as output on the console.
* We have all our pieces wired together , we have our application , service and repository tier.
* Now we will look at all our injection methods so that we don’t have to hardcode at some places.

CONFIGURATION :

* Spring is all about removing configuration code from your application.
* Why is configuration code such a bad thing in your application?
* It make things brittle , which means its hard to move to a different location or environment.
* If you had to recompile your code because your were moving to a different server environment , We don’t do unit test because its hard to test our code base.
* We have really complex code.
* Its not the complexity of the code that makes it hard to test , it’s the way the code is written.
* It’s the configuration of the code that’s hard.
* The code that doesn’t have to flow with the normal flow of business logic , if suppose your code interacts with the database or JNDI.

PAIN POINTS THROUGH OUR APPLICATION :

* To go through an illustrate the pain points in our application :
* We want to remove this “new HibernateCustomerRepositoryImpl” , The CustomerService doesn’t need to know that it is exactly calling the HibernateCustomerRepositoryImpl because we have already extracted it to an interface in the previous section through which it can interact.
* Here we have used a design pattern called as Factory to abstract that code.
* In our HibernateCustomerRepositoryImpl , we have some hardcoded data in there like setFirstName() , setLastName() and such. We should try to abstract that code so that there is no hardcoding scenario anywhere.

SPRING DOWNLOAD :

* Spring quit offering a direct download for the compiled jars.
* Nothing is simpler than clicking a link on the project page.
* We are going to use Maven. Spring wants you to download the jars using Maven.
* A project of some complexity should use Maven or Gradle to manage its dependencies.
* The Maven Repo has the Source , JavaDocs and Binaries all available to download and integrate to IDE.
* Its mainly because of the transitive dependencies that are required to run the project.
* Spring Boot could be used to run our application , but it acts as a black box and people generally don’t understand whats running in the background.
* We will set up a Spring Boot application at the last and compare it with our application.

SPRING DOWNLOAD MAVEN DEMO :

* Open chrome.
* Navigate to Spring.io
* Click on Projects tab.
* Click on the Spring Framework.
* Scroll down the page where we can grab the dependency for the Spring Framework for Maven and Gradle.
* Check at the version which is 4.3.2
* The good thing about Spring is that it releases current updates all the time
* We currently don’t have a dependency section in our application , come over to your application and double click on POM.xml
* Paste the following below <version> tag :

<dependencies>  
<dependency>  
<groupId>org.springframework</groupId>  
<artifactId>spring-context</artifactId>  
<version>4.3.2</version>  
</dependency>  
</dependencies>

* Save the above code and click on Ctrl+Shift+F for formatting.
* When you save the code , you may notice somethings happening in your application , some files would get downloaded.
* On the left in the Project Manager , you would see that there is a new Maven Dependencies section that is added.
* It has downloaded the following Jar’s  
  Spring-Context  
  Spring-AOP  
  Spring-Beans  
  Spring-Core  
  Commons-logging  
  Spring-Expression
* That’s all it took to add the Spring Dependencies into our application.

SUMMARY :

* In this module , we covered a lot of things which set us up for the upcoming modules.
* WHY factor of Spring.
* The Prerequisites that we had for our application.
* We built a Demo App and looked at the pain points and WHY Spring.
* Downloaded Spring and incorporated that inside our application , how quickly it was done using Maven.
* We will next see about the XML configuration. Please don’t skip this.