**REST API using JERSEY:**

1. Open eclipse and right click and say new 🡪 maven project and choose archetype using Rest Jersey.
2. Default format which Jersey supports is Xml. We can enable dependencies(jersey-media-moxy) in pom.xml to support json.
3. All request will go through Servlet Container in web.xml. You can check the url pattern here.
4. Annotations :
5. @Path: This specifies the path
6. @GET, @PUT, @DELETE, @POST – HTTP methods
7. @Produces, @Consumes – specifies the input/output formats
8. @XmlRootElement – To specify the rootelement in xml format
9. @PathParam – parameter for a method
10. Refer workspace C:\Users\stkra\Documents\REST\_API\_JERSEY\Jersey\_workspace
11. Execution is done through postman. Example : <http://localhost:8080/demorest/webapi/aliens/>

Get method

Workspace : C:\Users\stkra\Documents\REST\_API\_JERSEY\Jersey\_workspace

Git url: <https://github.com/RaajeeKumar/Code.git>

**REST API using Spring:**

1. Open STS, create a Boot starter project with JPA, Web, sql
2. @Path at method level in Jersey is equivalent to @RestMapping here
3. @Path at class level in Jersey is equivalent to @RestController here
4. @GET, @PUT, @DELETE, @POST…. All http methods is equivalent to @RequestMapping or @GetMapping..
5. Pojo class should have @Entity and primary key column should have @Id
6. @PathParam is equivalent to @ParamVariable. @RequestParam is used in Spring MVC. @Param is used in Spring data JPA. @WebParam is used in SOAP

**SOAP**

1. **Soap Terminologies :** 
   1. **WSDL –** Webservice Definition Language : This has the details about the webservice in xml format. What are the methods, input, output etc..
   2. **UDDI –** Universal Description Discovery and Integration – Any publisher can publish their web service and any consumer can query the service
   3. **SOAP –** Simple Object Access Protocol – Language used to encode and decode messages. This is the language which the web service can understand
   4. **SEI –** Service Endpoint Interface – Interface to the webservice. Depending upon the technology, it creates the service endpoint. For example, if its Java, it will convert the java object to the SOAP message
2. **Writing a web service client :**

Step 1: Stub Generation : stubs can be generated from the wsdl url using wsimport tool(This comes as part of Java SE version. )This step is to basically create SEI which is part of stub.

This is the command. Java & class files will be generated in the respective path(C:\Users\stkra\Documents\SOAP\SEI)

C:\Users\stkra\Documents\SOAP\SEI>wsimport -keep -s src <http://wsgeoip.lavasoft.com/ipservice.asmx?WSDL>

Step 2: copy the created stub to the project with the same package structure. Use the service class name to get the port name class. You can call methods on port name class methods

Project Name : C:\Users\stkra\Documents\SOAP\SOAP\_Workspace\_new1\IPLocationFinder

Execution : Run this as a Java application

1. **Writing a web service set up:**

Step 1: Eclipse set up & Glass fish server set up :

Download the glass fish app server and configure it in eclipse. You have to install glassfish tools through eclipse market place, then you have to refer to the glass fish installation path(download “Java Platform, Enterprise Edition 8 SDK” from Oracle site) while creating your new server.

Create a dynamic web project and create a sample jsp page. Then create a simple java class with a method. We are going to expose this method as webservice by using @WebService annotation. Now when we build and deploy it to the glassfish server, you can see the webservice endpoint url, tester url from the admin console.

1. **Two ways of implementing web services:**

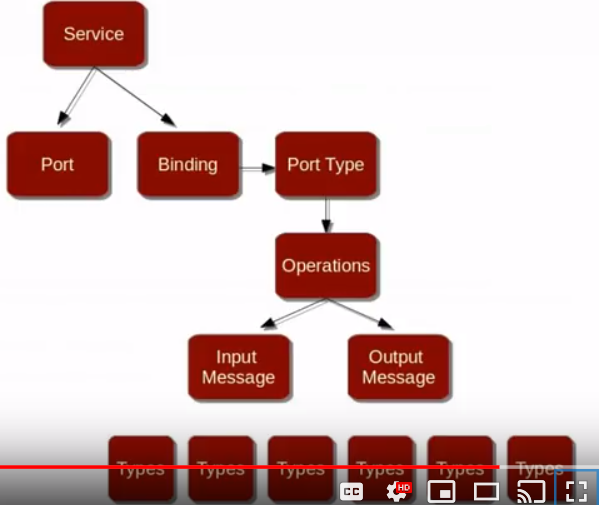
Service First/Code First : we first write the implementation/code first and annotate with @WebService annotation which will generate the wsdl for us

Contract First : we first write the wsdl first and write the implementation from the wsdl

1. **Understanding wsdl:**

How does Glass fish generates the wsdl for us?

As soon as, it sees that a java class has a @WebService annotation, it creates a service, port and binding(ways to access the webservice). Binding creates a port type with the set of operations(methods) with input and output. Input and referred to messages here. For each operation, you have 2 message tags, one for input and other for output



Default names can be changed with the following annotations,

@WebService(name="TestMart",portName="TestMartPort",serviceName="TestMartService",targetNamespace="http://testmart.com")

@WebMethod(action="fetch\_categories",operationName="fetchCategories",exclude=”true”)

@SOAPBinding(style=Style.***RPC***)

@WebResult(partName="lookupOutput")

**public** List<String> getProductList(@WebParam(partName="lookupInput") String category)

Design should be in such a way, you should have a webservice Interface and an implementation class. Interface class is called SEI. All annotations will go to the interface. Only annotation that is needed in implementation class is

@WebService(endpointInterface="org.raajee.ProductCatalogInterface",portName="TestMartPort",serviceName="TestMartService")

1. **JAXB:**

Java Architecture for XML Binding. Converting java object to xml and vice versa using JAX B annotation. For JAX B to do the conversion, a no arg constructor is required in the class

* To declare a root element, you have to provide the following in the webservice Interface

@WebResult(name="Product")

+

@XmlRootElement(name = “Product”)

* To specify the order,

@XmlType(propOrder={"price","sku","name"})

* To change the name of the element in resultant xml,

@XmlElement(name="ProductName")

1. **Handling Faults:**

You have create an exception class and throw that custom exception from the service class. In wsdl, exception class is referred as fault and it will be present within operations tag along with input and output messages, as the throws exception is available as part of method signature.

Unfortunately, you cant check the fault in glassfish tester url. But there are tools which helps to test the fault exception

1. **Tools for checking soap webservice:**

You have a eclipse plugin for soap.

Soap UI

You also have webservice Explorer as part of eclipse to view the wsdl and import it to the workbench

1. **How does all SOAP annotations work.?**

JAX WS is an interface which will inspect the code + annotation and see if all the standards are followed. Metro is the JAX WS implementation which is bundled as part of glass fish and JDK. That’s why our simple JAVA application(EndpointMainTestApplication) is able to test the wsdl.

Workspace : C:\Users\stkra\Documents\SOAP\SOAP\_Workspace\_new1

Actual Code is in Git Location. So be cautious. Don’t delete any folder.

Dont delete the git repository, soap\_jaxws\_code\_impl, soap\_jaxws\_code\_implementation

Don’t delete the workspace, C:\Users\stkra\Documents\SOAP\SOAP\_Workspace\_new1

C:\Users\stkra\Documents\SOAP\SOAP\_Workspace\_new

Don’t delete the folders, C:\Users\stkra\git\soap\_jaxws\_code\_impl,

C:\Users\stkra\git\soap\_jaxws\_code\_implementation