

IBM WAS Liberty 서버 기반
전자정부 프레임워크(Spring)
개발 가이드 02

이정운과장 (juwlee@kr.ibm.com)
IBM Korea, WebSphere Technical Sales

목 차

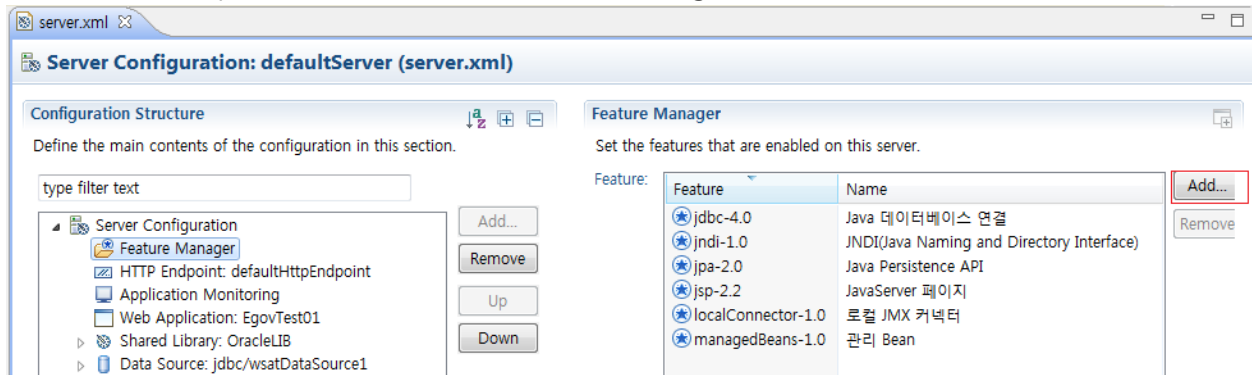
1	JAX-RS 개발	3
1.1	IBM WAS Liberty 서버의 기능 추가.....	3
1.2	JAX-RS 샘플 프로젝트 설정	4
1.3	JAX-RS 샘플 애플리케이션 개발	8
2	JAX-RS 테스트	9
2.1	JAX-RS 샘플 애플리케이션 테스트.....	9
3	JAX-WS 개발	12
3.1	IBM WAS Liberty 서버의 기능 추가.....	12
3.2	JAX-WS 샘플 애플리케이션 개발(Web Service Provider)	13
3.3	JAX-WS 샘플 애플리케이션 개발(Web Service Client).....	20
4	JAX-WS 테스트	24
4.1	JAX-WS 기반 Web Service Provider 테스트	24
4.2	JAX-WS 기반 Web Service Client 테스트.....	26

1 JAX-RS 개발

1.1 IBM WAS Liberty 서버의 기능 추가

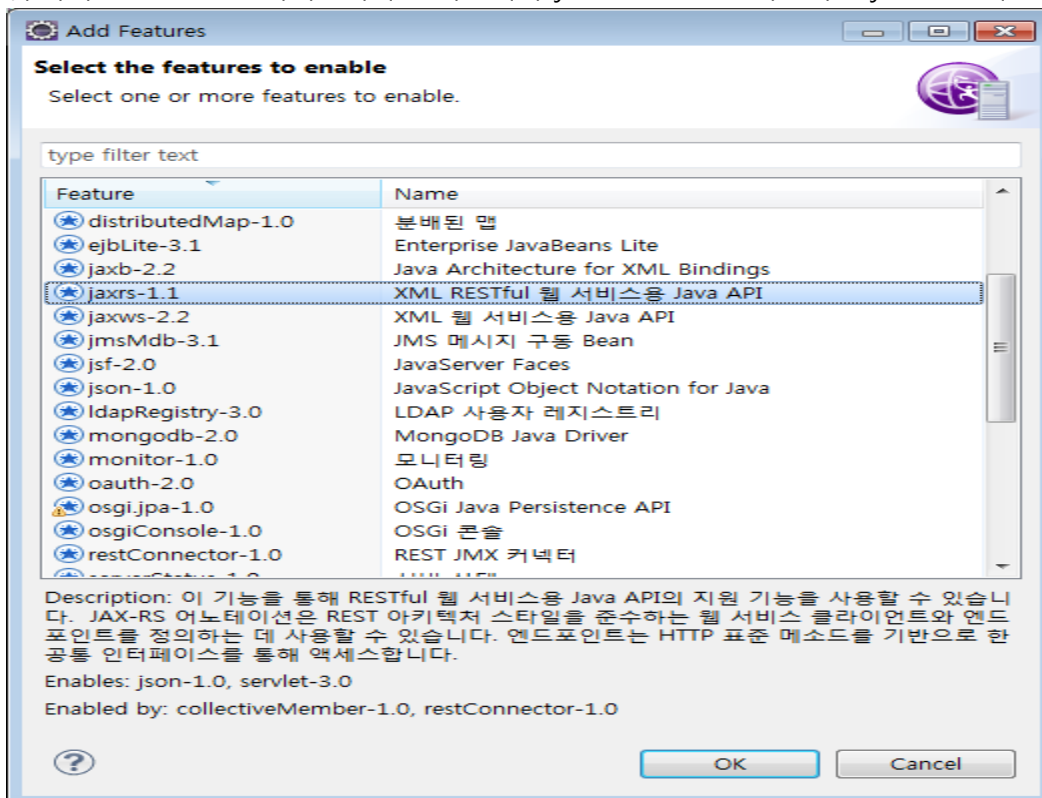
JAX-RS 스펙을 준수한 애플리케이션을 개발하기 위해서는 IBM WAS Liberty 서버의 server.xml 파일에 JAX-RS 기능추가를 해야 합니다.

server.xml 을 eclipse 를 통해서 오픈한 후 Feature Manger 에서 Add 버튼을 클릭합니다.



추가 가능한 기능 리스트에서 jaxrs-1.1 을 선택합니다.

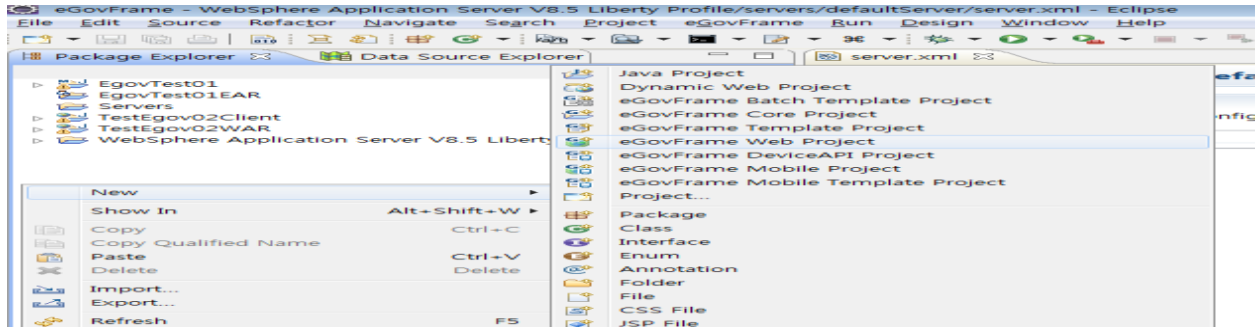
(추가적으로 JSON4J 라이브러리를 사용해서 json 을 컨트롤 하실려면 json-1.1 기능도 추가합니다.)



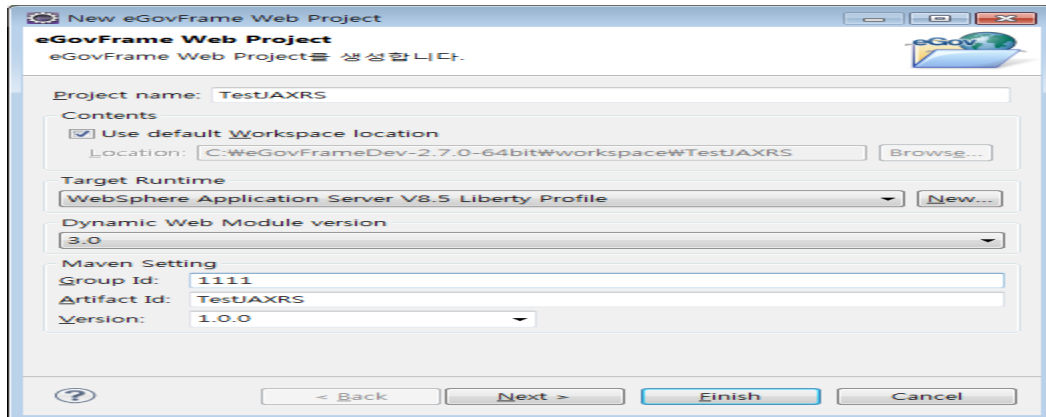
1.2 JAX-RS 샘플 프로젝트 설정

IBM WAS Liberty 서버 기반의 JAX-RS 샘플 프로젝트를 설정해 봅니다.

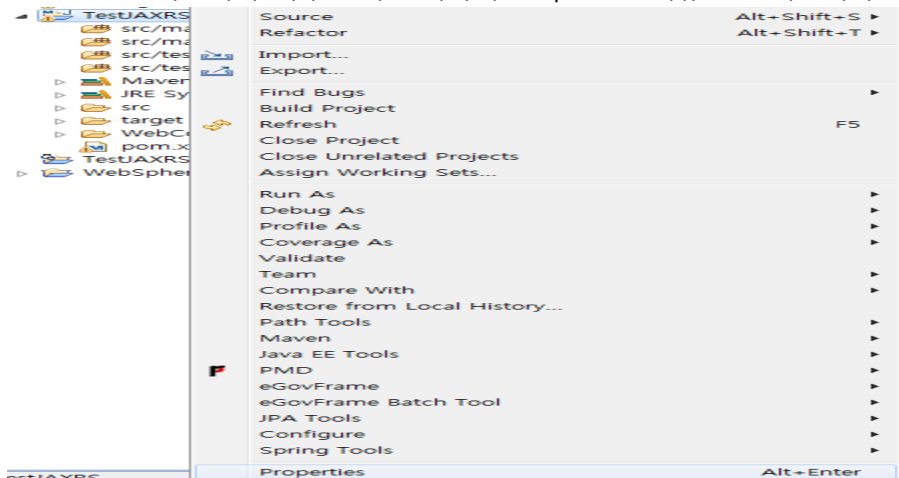
Eclipse 에서 마우스 우 클릭하여 New > eGovFrame Web Project 를 선택합니다.



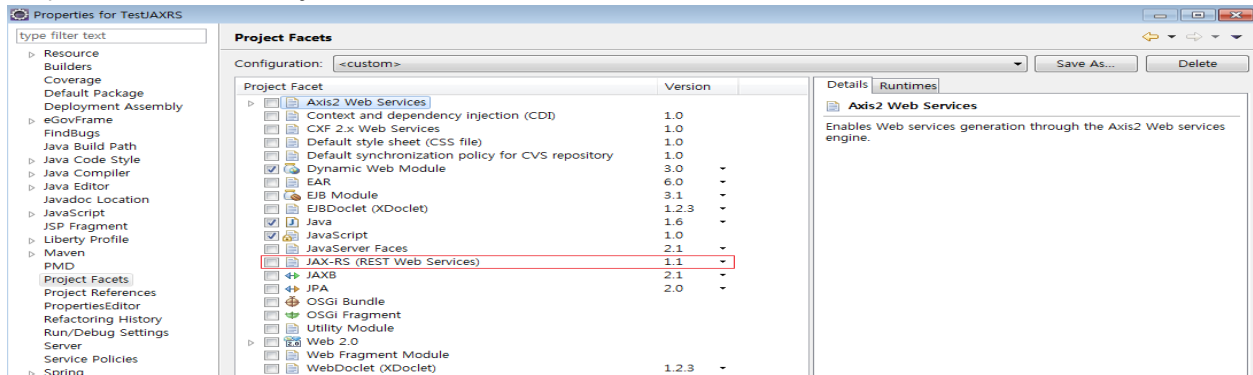
샘플로 사용하기 위해 이름을 지정하고 하단의 캡처 화면과 동일하게 설정을 입력하고 Finish 버튼을 클릭하여 프로젝트를 생성합니다.



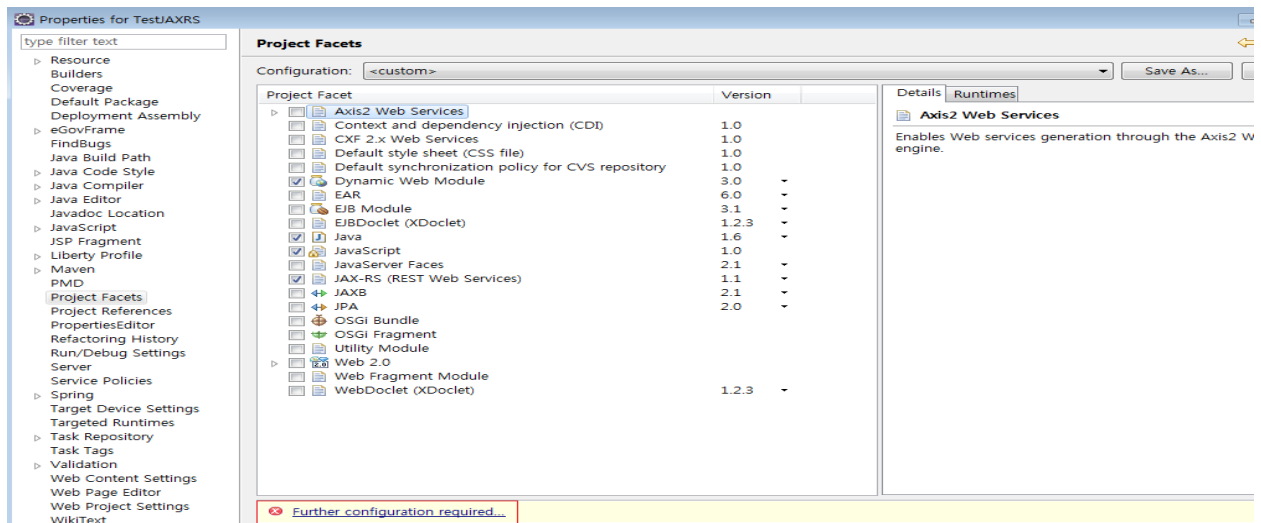
생성된 프로젝트에서 마우스 우클릭하여 Properties 메뉴를 선택합니다.



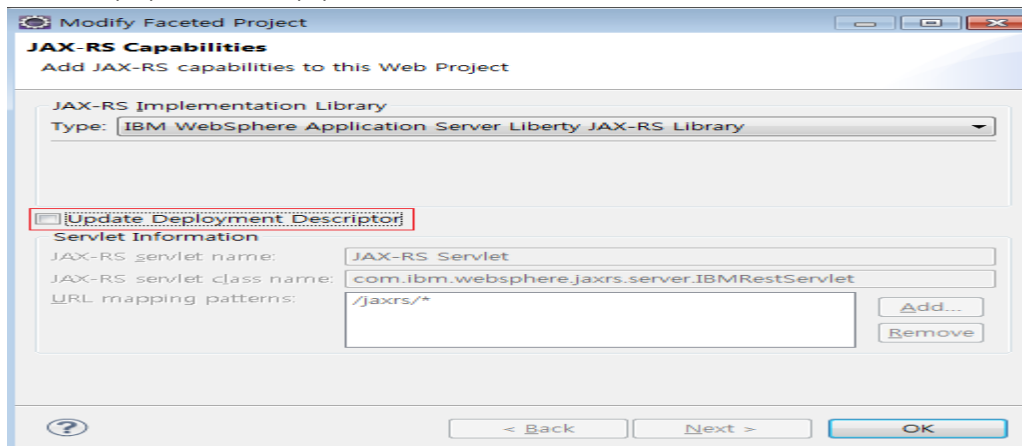
Properties 메뉴에서 Project Facets 를 선택한 후 JAX-RS 를 체크 합니다.



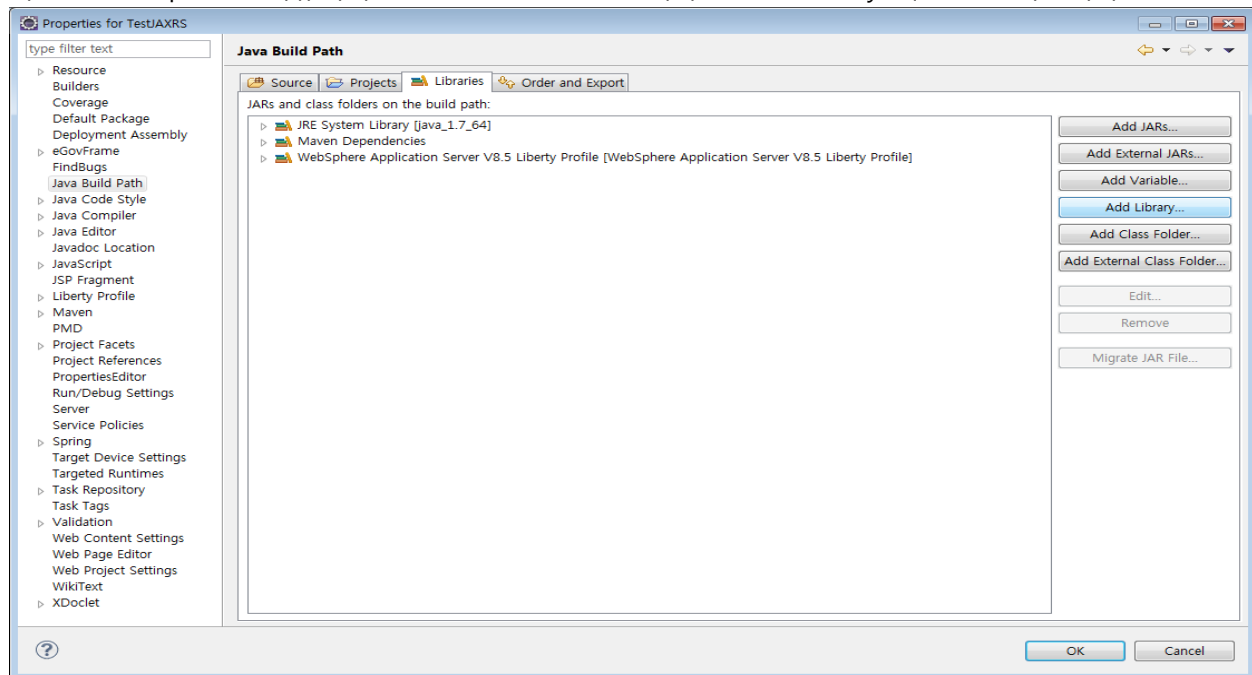
JAX-RS 를 체크하면 하단과 같이 Further configuration required 표시가 나타나는데 해당 표시를 클릭합니다.



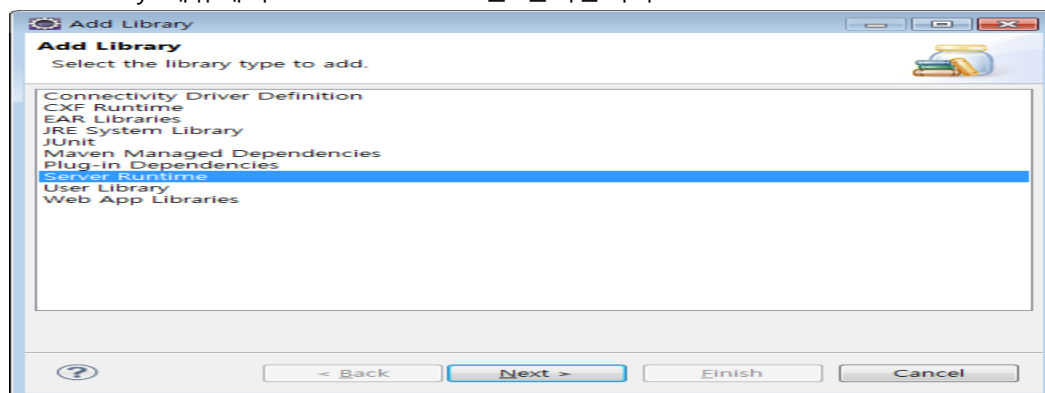
JAX-RS Capabilities 화면이 나오면 Update Deployment Descriptor 를 선택해제하고 OK 를 클릭하여 JAX-RS 추가를 완료합니다.



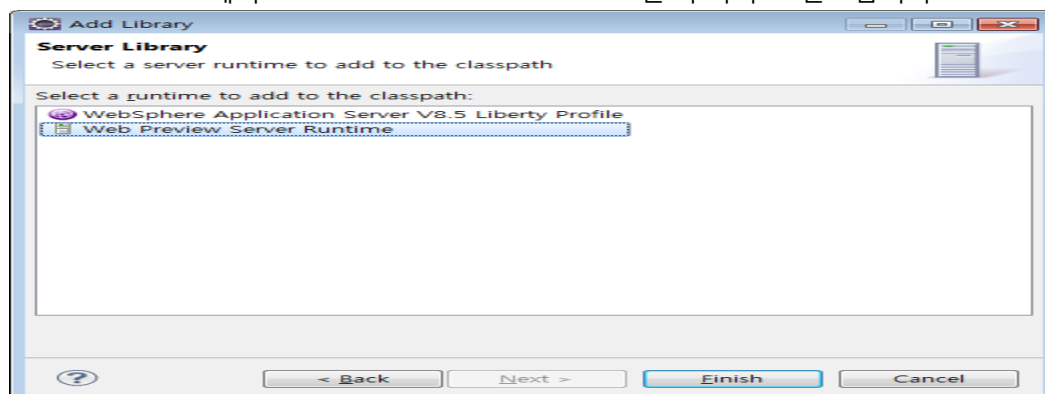
다음으로 Properties 메뉴에서 Java Build Path 를 선택하고 Add Library 버튼을 클릭합니다.



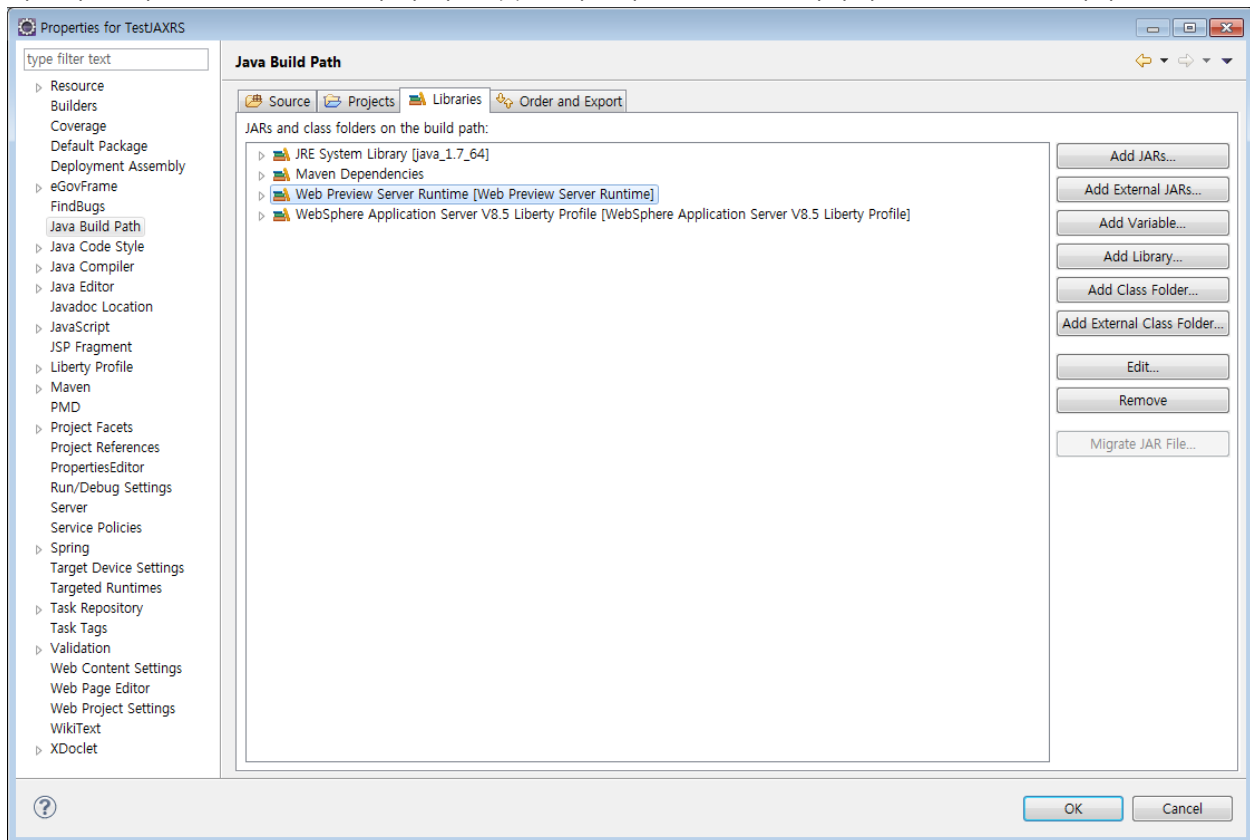
Add Library 메뉴에서 Server Runtime 을 선택합니다.



Server Runtime 에서 Web Preview Server Runtime 을 추가하고 완료합니다.



다음과 같이 Java Build Path 가 추가된 것을 확인하고 OK 를 클릭하여 설정을 완료합니다.



1.3 JAX-RS 샘플 애플리케이션 개발

IBM WAS Liberty 서버 기반의 JAX-RS 샘플 애플리케이션을 개발해 봅니다.

상단에서 만든 샘플 프로젝트에서 package 를 지정하고 하단과 같이 JAX-RS 표준에 따라서 샘플 애플리케이션을 작성해 봅니다.

```
DemoApplication.java ✕
1 package com.ibm.juwlee.jaxrs;
2
3+ import java.util.HashSet;
8
9 @ApplicationPath("/rest")
10 public class DemoApplication extends Application {
11     @Override
12     public Set<Class<?>> getClasses() {
13         Set<Class<?>> classes = new HashSet<Class<?>>();
14         classes.add(HelloResource.class);
15         return classes;
16     }
17 }
18
```

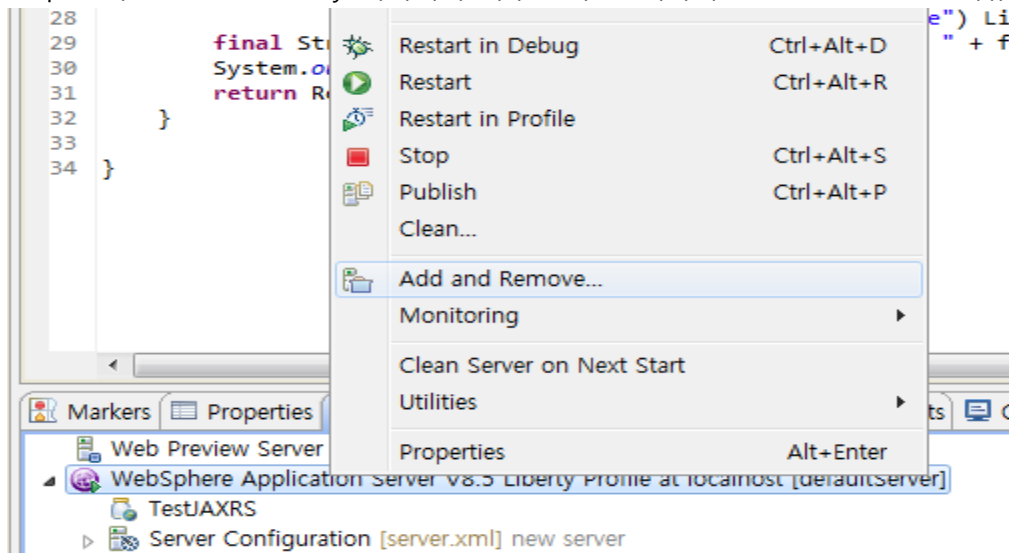
```
HelloResource.java ✕
1 package com.ibm.juwlee.jaxrs;
2
3+ import javax.ws.rs.GET;
10
11 @Path("/hello")
12 public class HelloResource {
13     @GET
14     public String helloGet() {
15         return "[GET] Hello from JAX-RS on WebSphere Liberty Profile";
16     }
17
18     @GET
19     @Path("/json")
20     @Produces(value="application/json")
21     public JSONObject getList2() {
22         JSONObject listJSONObject = new JSONObject();
23         listJSONObject.put("AAAA", "Hello Earth!");
24         listJSONObject.put("BBBB", "Hello Mars!");
25         return listJSONObject;
26     }
27
28     @POST
29     public Response helloPost() {
30         return Response.ok("[POST] Hello from JAX-RS on WebSphere Liberty Profile")
31             .header("X-Resource-Architect", "Jacek Laskowski")
32             .build();
33     }
34 }
```


2 JAX-RS 테스트

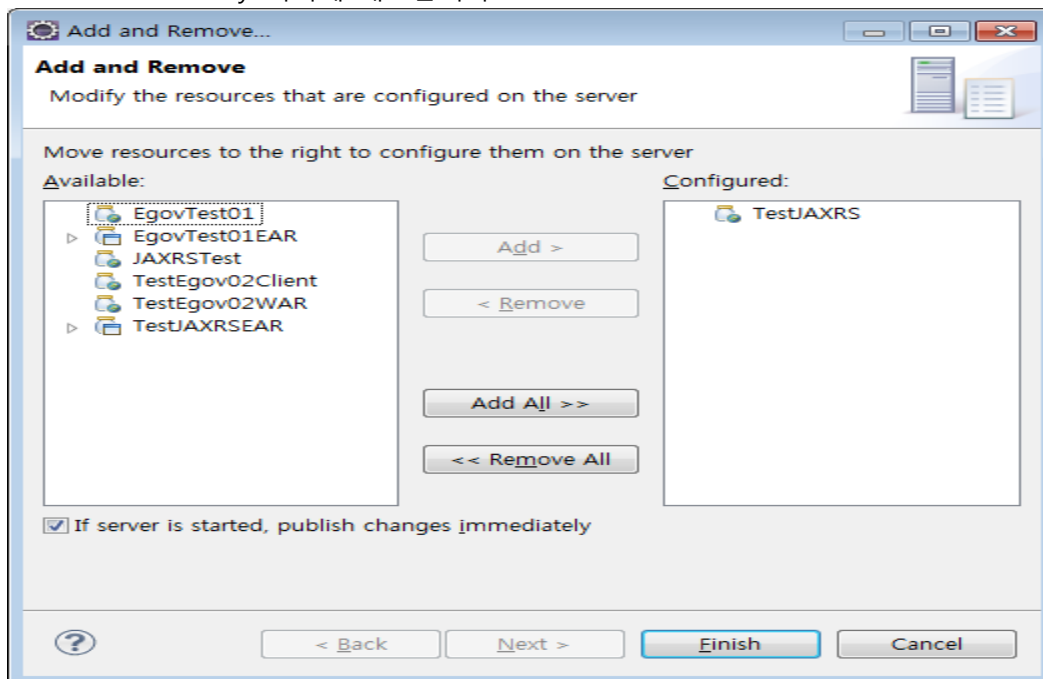
2.1 JAX-RS 샘플 애플리케이션 테스트

지금까지 작성한 JAX-RS 샘플 애플리케이션을 테스트 해 봅니다.

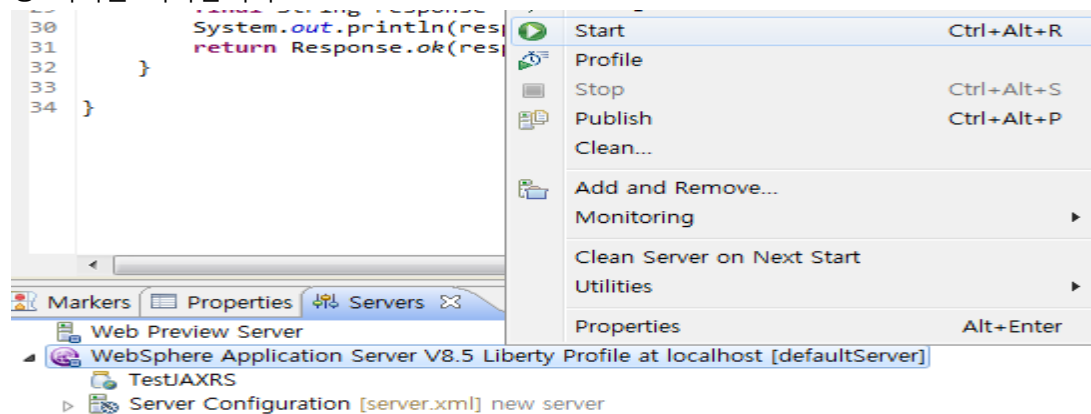
Eclipse 의 IBM WAS Liberty 서버에서 마우스 우 클릭하여 Add and Remove 메뉴를 선택합니다.



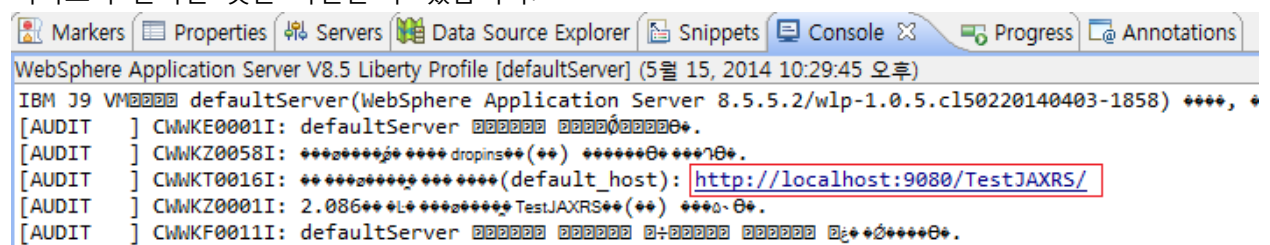
Add and Remove 메뉴가 나오면 지금 생성한 JAX-RS 샘플 프로젝트를 선택하여 Add 버튼을 클릭하고 IBM WAS Liberty 서버에 배포합니다.



IBM WAS Liberty 서버에 애플리케이션 배포가 완료되었으면 마우스 우 클릭하고 Start 를 클릭하여해당 서버를 시작합니다.

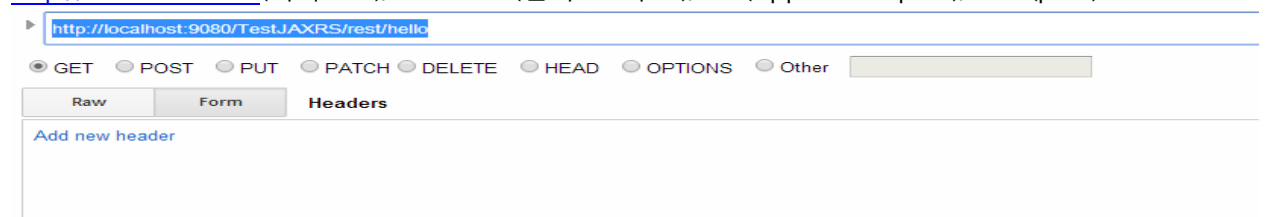


서비스가 정상적으로 구동되면 하단과 같이 console 창에서 샘플 프로젝트 이름을 컨텍스트 루트로 서비스가 올라온 것을 확인할 수 있습니다.



그럼 이제 Chrome 과 같이 rest 호출이 가능한 client 를 사용해서 하단과 같이 요청을 보내면 성공적인 데이터가 반환되는 것을 확인할 수 있습니다.

[http://localhost:9080\(서버포트\)/TestJAXRS\(컨텍스트루트\)/rest\(Application path\)/hello\(path\)](http://localhost:9080(서버포트)/TestJAXRS(컨텍스트루트)/rest(Application path)/hello(path))




<http://localhost:9080>(서버포트)/TestJAXRS(컨텍스트루트)/rest(Application path)/hello(path)/jython(path)


▶




☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

Raw Form Headers

[Add new header](#)

Status **200 OK**  Loading time: 26 ms

Request headers
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/34.0.1847.137 Safari/537.36
Content-Type: text/plain; charset=utf-8 
Accept: */*
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US, en; q=0.8

Response headers
X-Powered-By: Servlet/3.0
Content-Type: application/json 
Content-Length: 44 
Date: Fri, 16 May 2014 07:58:29 GMT 

Raw JSON Response

[Copy to clipboard](#) [Save as file](#)

```
{
  AAAA: "Hello Earth!"
  BBBB: "Hello Mars!"
}
```

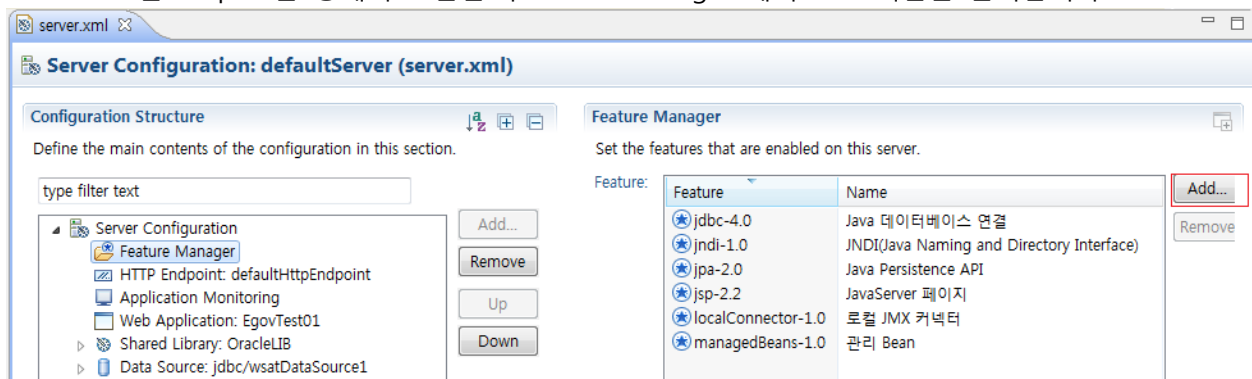
3 JAX-WS 개발

3.1 IBM WAS Liberty 서버의 기능 추가

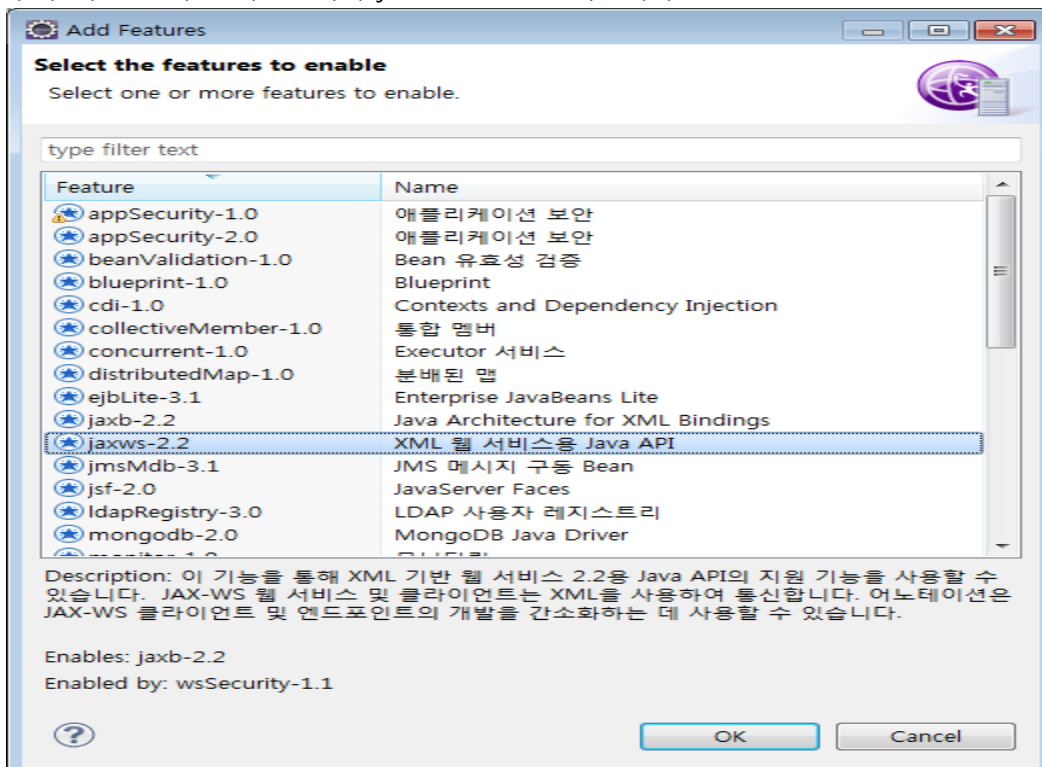
JAX-WS 스펙을 준수한 Web Service 애플리케이션을 개발하기 위해서는 IBM WAS Liberty 서버의 server.xml 파일에 JAX-WS 기능 추가를 해야 합니다.

(참고로 본 가이드는 샘플이며 JAX-WS 스펙을 준수하기 때문에 JAX-WS 표준에 정의된 모든 annotation 을 사용 가능합니다.)

server.xml 을 eclipse 를 통해서 오픈한 후 Feature Manger 에서 Add 버튼을 클릭합니다.



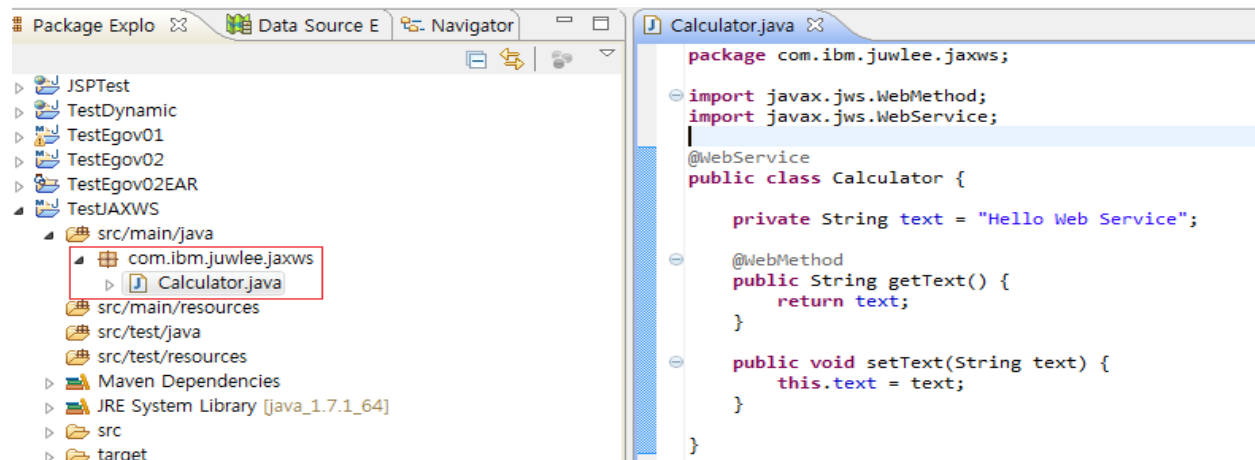
추가 가능한 기능 리스트에서 jaxws-2.2 를 선택합니다.



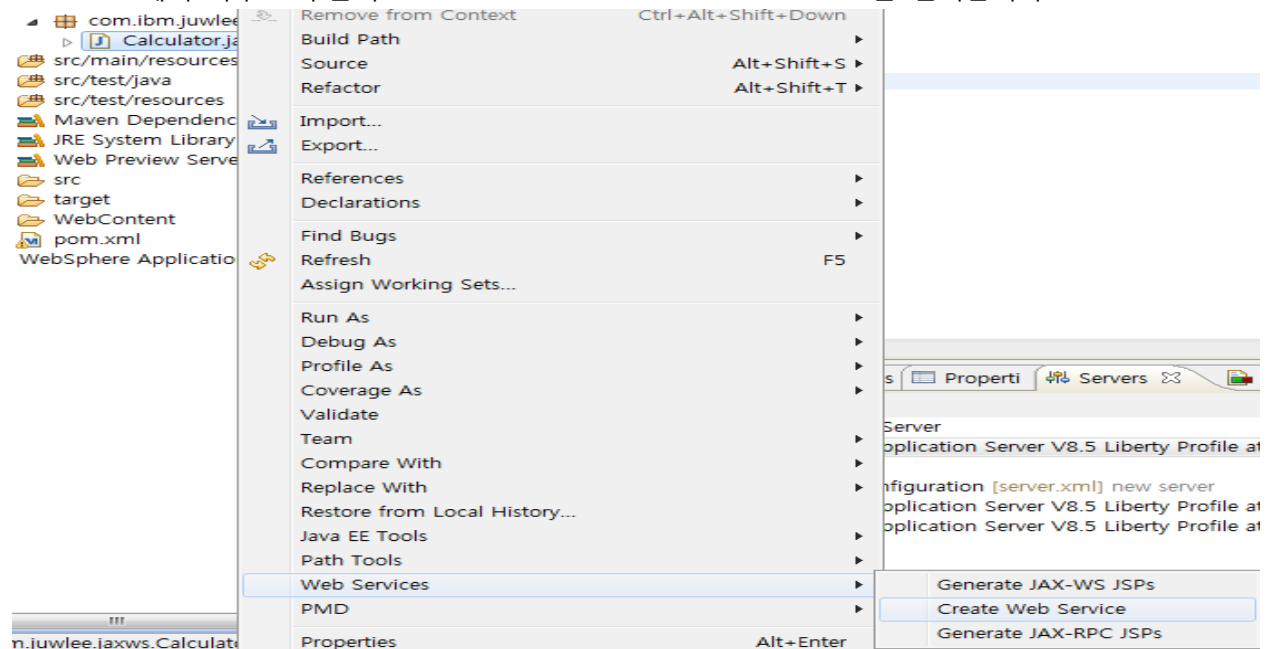
3.2 JAX-WS 샘플 애플리케이션 개발(Web Service Provider)

IBM WAS Liberty 서버 기반의 JAX-WS Web Service 샘플 애플리케이션을 개발해 봅니다. JAX-WS 를 개발할 경우에는 WSDL 을 먼저 만들고 그것을 이용해서 Web Service 를 만드는 방법이 있으며 Java class 를 먼저 만들고 그것을 이용해서 Web Service 를 만드는 방법이 있습니다. 여기서는 Java class 를 먼저 만들고 해당 class 를 이용해서 JAX-WS Web Service 를 만들어 보도록 하겠습니다.

편의성을 위해서 별도로 프로젝트를 만들지 않고 JAX-RS 샘플을 개발했던 프로젝트에서 package 와 class 만 하단과 같이 추가합니다.

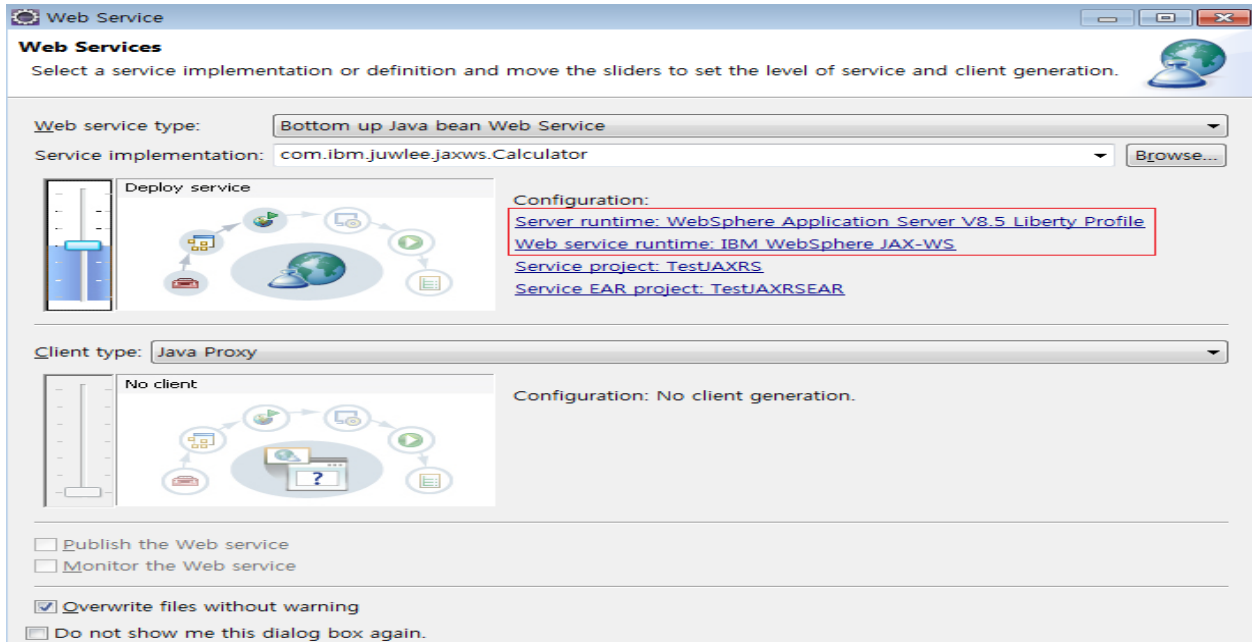


Java class 에서 마우스 우클릭 > Web Services > Create Web Service 를 선택합니다.

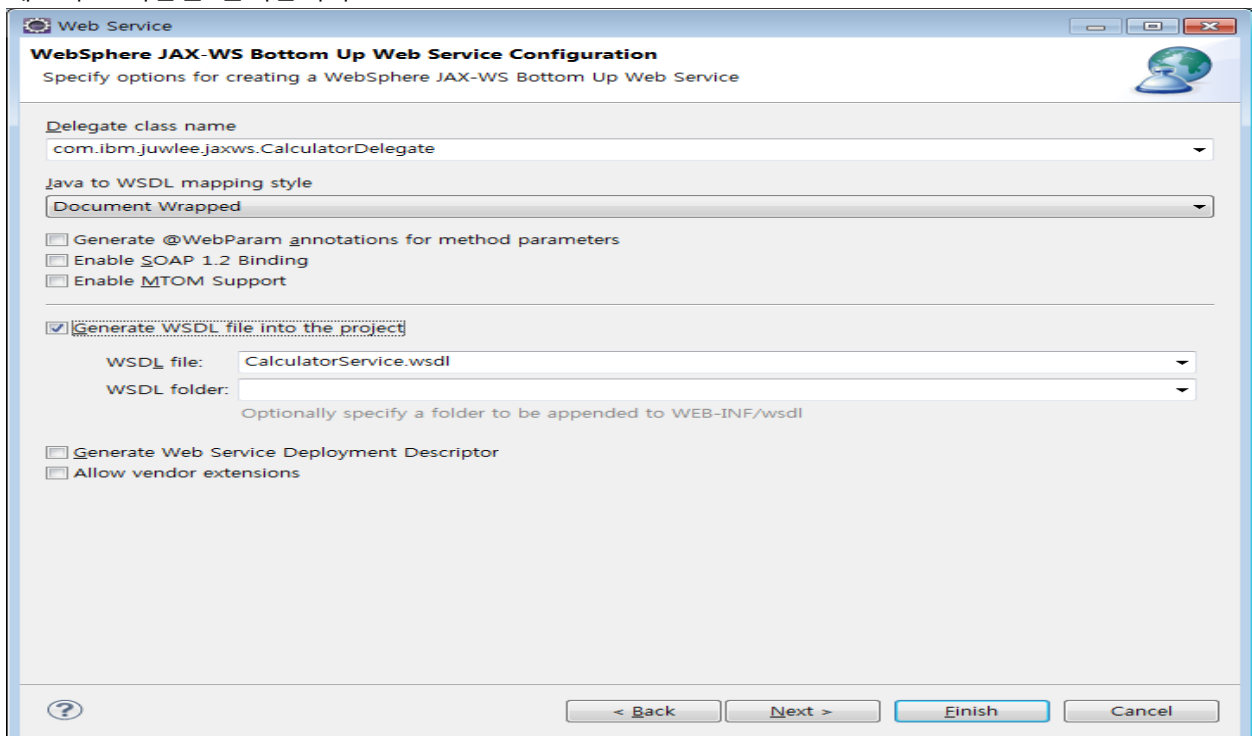


Web Service 마법사가 나오면 생성 레벨을 deploy 까지만 맞추고 Server runtime 과 Web Service runtime 설정이 제대로 하단과 같이 되어 있는지 확인합니다.

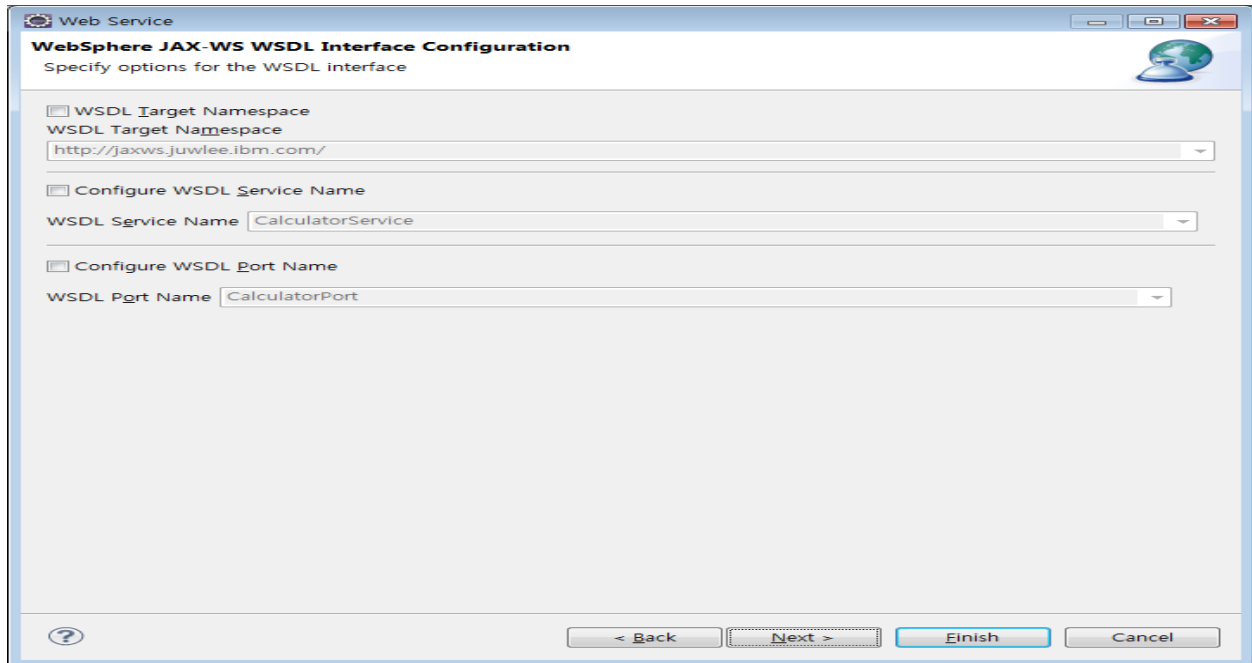
(참고로 IBM WAS Liberty 는 Web Service runtime 을 위하여 Apache CXF 구현체를 사용합니다.)



샘플 예제이기 때문에 Delegate class name 을 기본으로 두고 Generate WSDL file into the project 만 체크하고 다음을 클릭합니다.



그 외의 정보들도 기본 값으로 두고 다음을 클릭하면 Web Service 생성을 시작합니다.

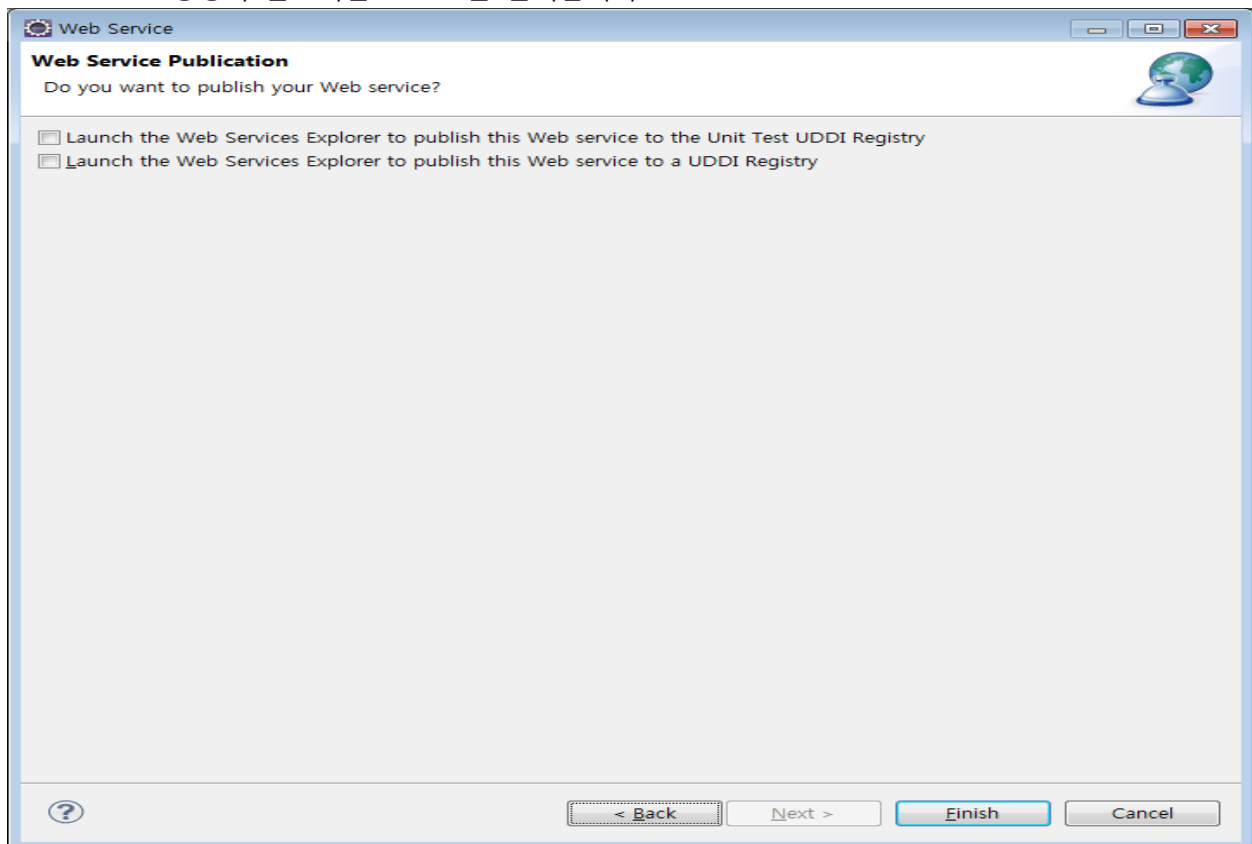


The image shows a dialog box titled "Web Service" with a sub-header "WebSphere JAX-WS WSDL Interface Configuration". Below the sub-header is the text "Specify options for the WSDL interface". The dialog contains three sections, each with a checkbox and a text field:

- ☐ WSDL Target Namespace
WSDL Target Namespace:
- ☐ Configure WSDL Service Name
WSDL Service Name:
- ☐ Configure WSDL Port Name
WSDL Port Name:

At the bottom of the dialog, there is a question mark icon on the left and four buttons: "< Back", "Next >", "Finish", and "Cancel".

Web Service 생성이 완료되면 Finish 를 클릭합니다.

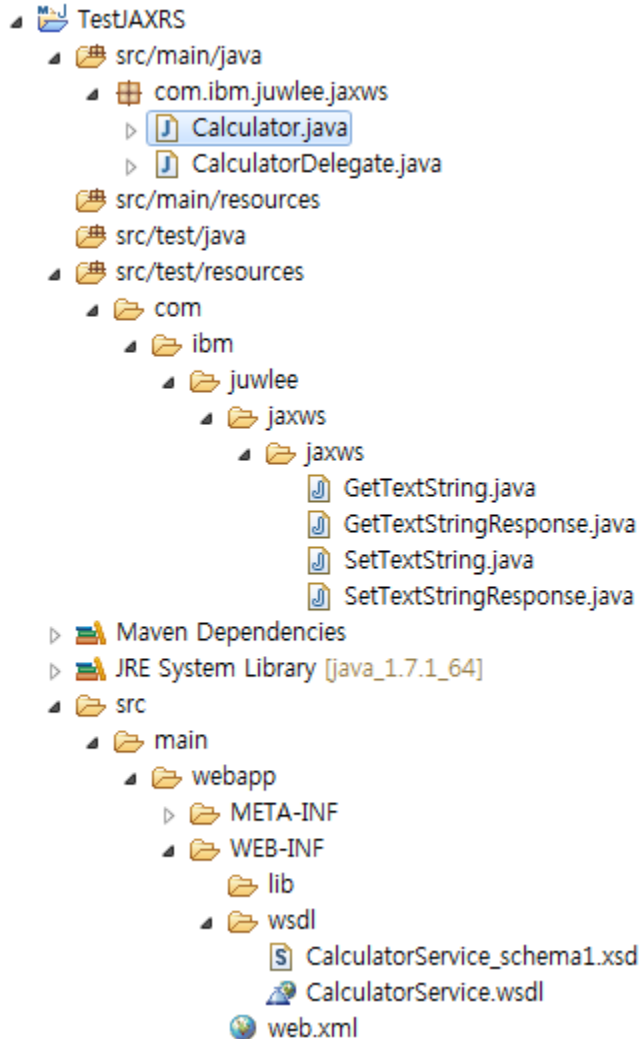


The image shows a dialog box titled "Web Service" with a sub-header "Web Service Publication". Below the sub-header is the text "Do you want to publish your Web service?". The dialog contains two checkboxes:

- ☐ Launch the Web Services Explorer to publish this Web service to the Unit Test UDDI Registry
- ☐ Launch the Web Services Explorer to publish this Web service to a UDDI Registry

At the bottom of the dialog, there is a question mark icon on the left and four buttons: "< Back", "Next >", "Finish", and "Cancel".

Java class 를 이용해서 Web Service 생성이 정상적으로 완료되었으면 하단과 같이 Delegate class 가 추가된 것과 wsdl 파일이 생성된 것을 확인할 수 있습니다.

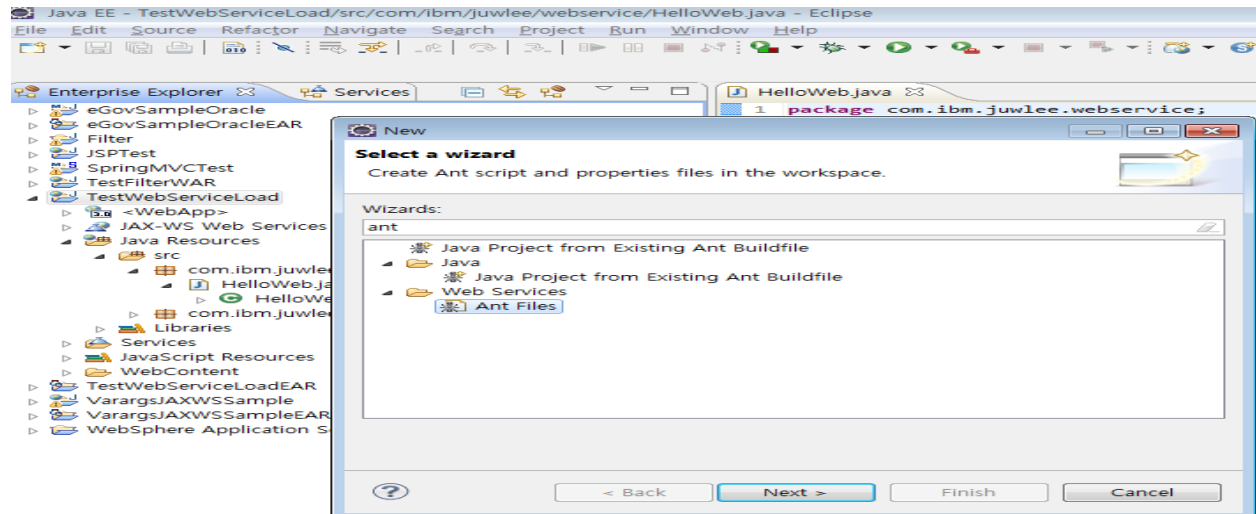


참고 #1) Delegate class 의 경우에는 Java class 와 같은 패키지 안에 생성되지만 Web Service 를 위한 JAXB 구현 class 들은 패키지의 맨 마지막 순서에 있는 folder 패키지 안에 생성하는 delegate class 패키지 이름에 jaxws 를 추가하여 자동 생성됩니다. (예: /src/test/resources/)

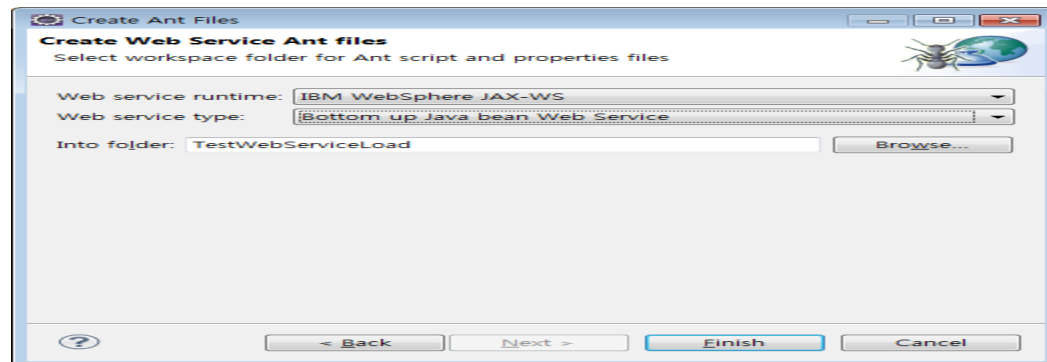
그러므로 properties > Java Build Path > Order and Export 메뉴를 통해서 위치를 지정할 수 있으며 어쨌든 eGov 설정에 따라 해당 폴더도 결국 build 시에 WEB-INF/class 로 class 파일이 생성되므로 그대로 두어도 됩니다. 아니면, 단순히 프로젝트 마다의 규칙에 따라 특정 위치의 패키지로 그대로 옮겨도 됩니다.

참고 #2) Webservice Wizard 대신에 wsgen 명령어와 ant 를 이용한 Delegate class 와 WSDL 생성 방안

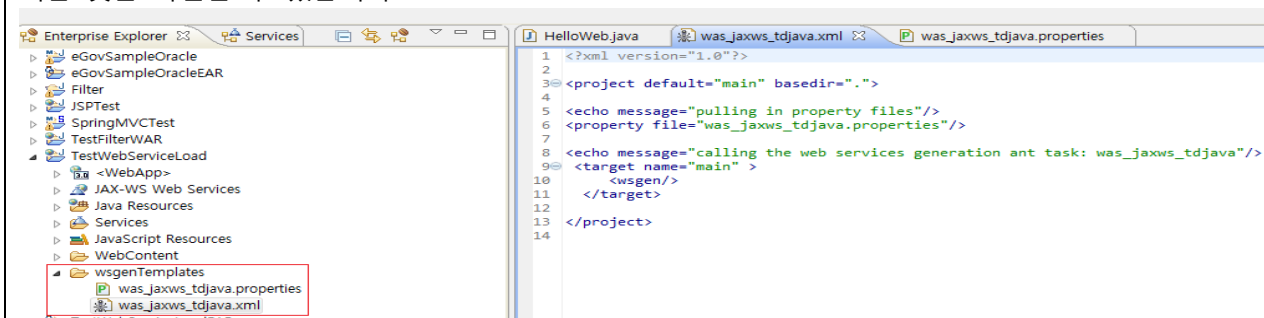
만들고자 하는 프로젝트에서 마우스 우 클릭후에 New 를 선택하고 Ant Files 를 선택합니다.



Create Ant Files 메뉴가 나오면 IBM WebSphere JAX-WS 와 Bottom up java bean Web service 를 선택한 후 Finish 로 Ant file 생성을 완료합니다.



Ant file 생성을 완료하면 하단과 같이 실행 파일과 프로퍼티 파일로 분리된 2 개의 Ant file 이 생성 되는 것을 확인할 수 있습니다.



프로퍼티 파일에서 Web Service 로 만들 Java 파일이 full path 를 수정하고 ServiceProjectName 과 ServiceEarProjectName 을 입력합니다. 추가로 Service.ServerId 를 com.ibm.ws.st.server.v85.was 로 변경하고 WSDL 을 생성하기 위해서 GenerateWSDL 옵션을 true 로 변경합니다.

(필요하다면 다른 옵션들도 상황에 맞게 수정합니다. 하단의 링크에 옵션 설명 참고..)

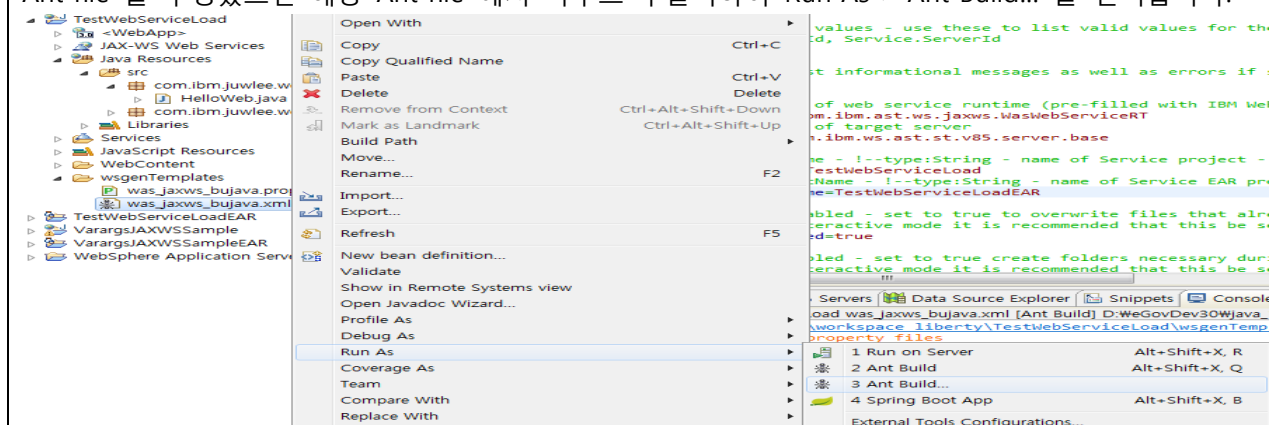
<http://pic.dhe.ibm.com/infocenter/rsawshlp/v7r5m0/index.jsp?topic=%2Fcom.ibm.webservice.jaxws.creation.doc%2Ftopics%2Ftimpantjaxws.html>

```

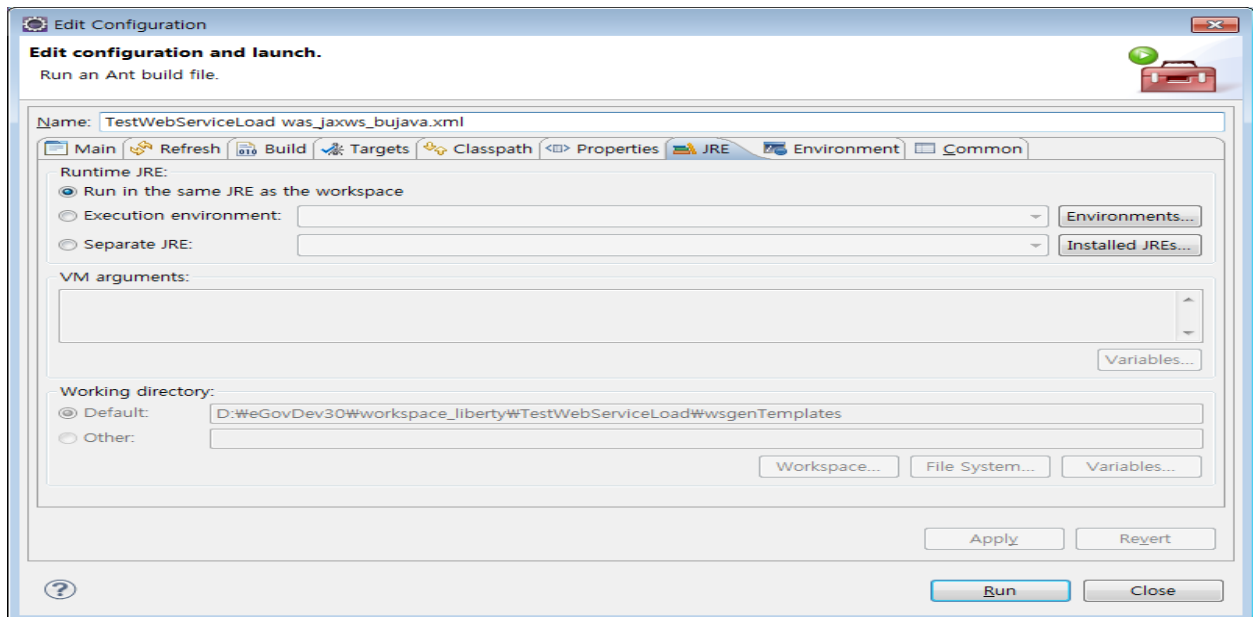
10 #####
11 !-- REQUIRED. Differentiates from "client" scenario.
12 ScenarioType=service
13
14 !-- REQUIRED. Workspace relative URI to the input Java
15 !-- InitialSelection=/dynamicWebProjectName/src/packageName/MyBean.java
16 InitialSelection=/TestWebServiceLoad/src/com/ibm/juwlee/webservice/HelloWeb.java
17
18 !--Utility property values - use these to list valid values for the following pro
19 !-- Service.RuntimeId, Service.ServerId
20 ListRuntimes=true
21 ListServers=true
22 !-- Verbose will list informational messages as well as errors if set to true
23 Verbose=true
24
25 !--type:String - ID of web service runtime (pre-filled with IBM WebSphere JAX-RPC
26 Service.RuntimeId=com.ibm.ast.ws.jaxws.WasWebServiceRT
27 !--type:String - ID of target server
28 Service.ServerId=com.ibm.ws.st.server.v85.was
29
30 !--ServiceProjectName - !--type:String - name of Service project - this property
31 ServiceProjectName=TestWebServiceLoad
32 !--ServiceEarProjectName - !--type:String - name of Service EAR project - this pr
33 ServiceEarProjectName=TestWebServiceLoadEAR
34
35
36
37
38
39
40 !-- note for non-interactive mode it is recommended that this be set to true
41 CreateFoldersEnabled=true
42
43 !--CheckoutFilesEnabled - set to true to check out files with no warning to the user type:boolean
44 !-- note for non-interactive mode it is recommended that this be set to true
45 CheckoutFilesEnabled=true
46
47 !--GenerateWSDL - set to true to generate a wsdl file into the project, the default is false
48 GenerateWSDL=true
49
50 !--WSDLFile !--type:String - the name of the WSDL file that will be generated.
51 !-- Please note that this field will only take effect if GenerateWSDL=true

```

Ant file 을 수정했으면 해당 Ant file 에서 마우스 우클릭하여 Run As > Ant Build... 을 선택합니다.



JRE 탭에서 Run in the same JRE as the workspace 를 선택하고 Run 을 수행하면 Ant 스크립트가 정상적으로 수행됩니다.



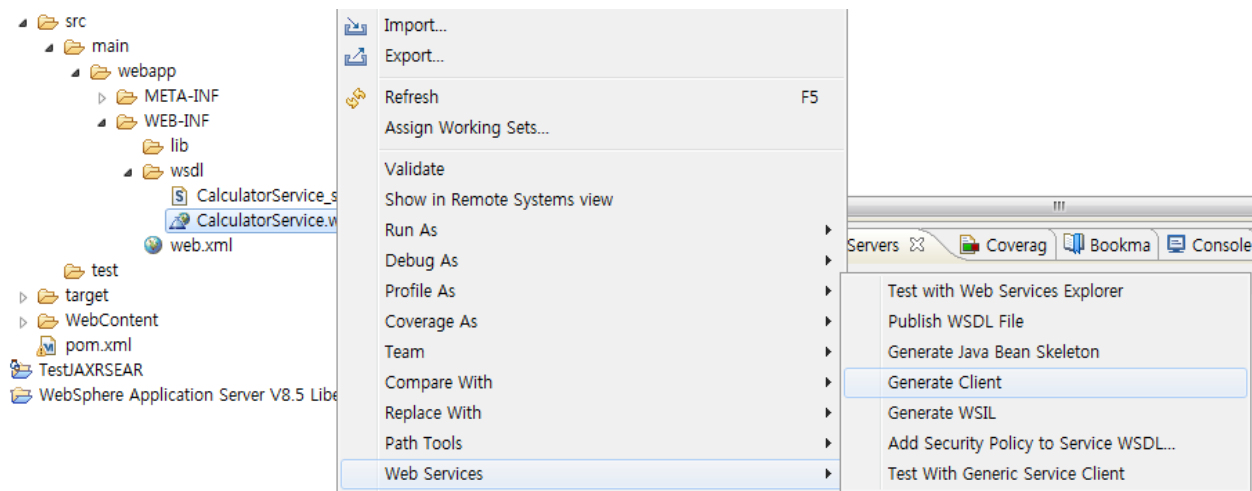
정상적으로 완료되면 Build Successful 메시지와 class, wsdl 파일을 하단과 같이 확인 가능합니다.



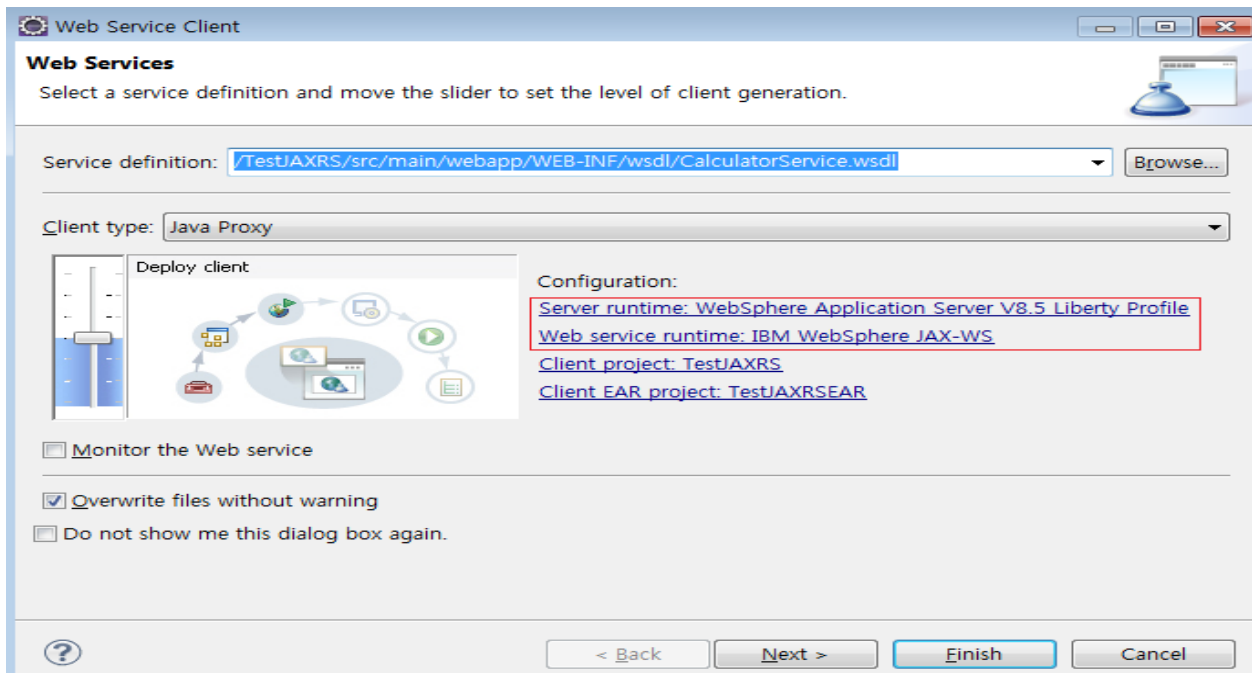
3.3 JAX-WS 샘플 애플리케이션 개발(Web Service Client)

이전 챕터에서 JAX-WS 기반으로 Web Service Provider 샘플 애플리케이션을 만들었으니 이제는 해당 서비스를 호출하기 위한 Web Service Client 를 만들어보도록 하겠습니다.

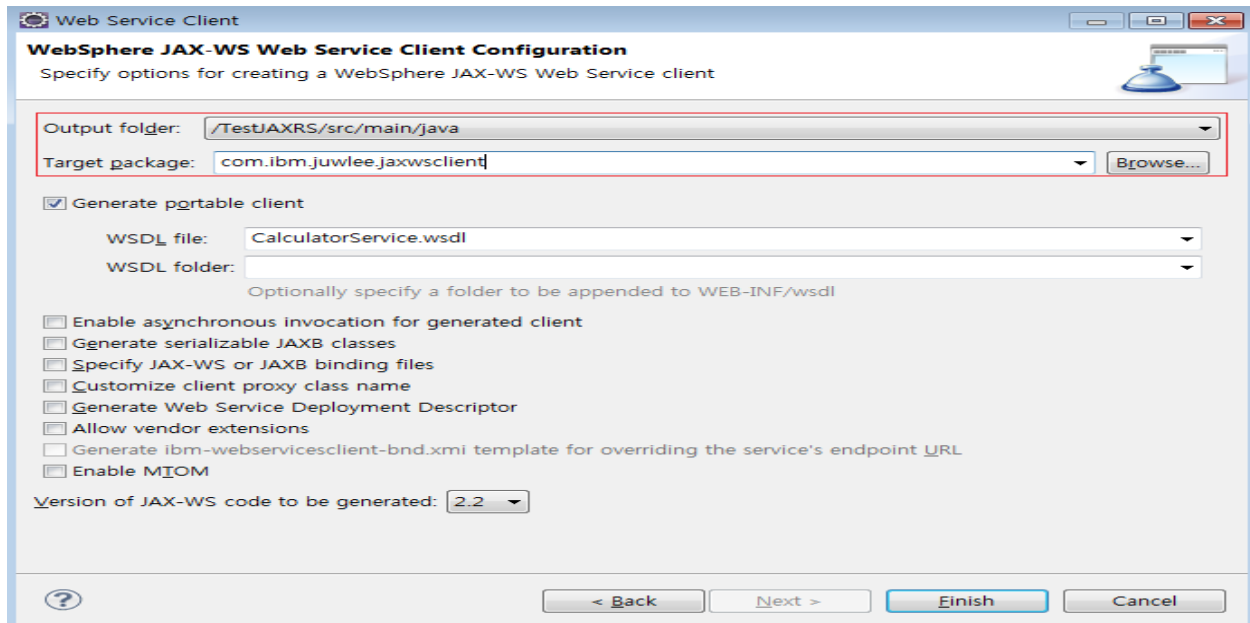
JAX-WS 기반 Web Service Client 를 생성하기 위해서 이전에 만들어진 서비스 정의 파일인 wsdl 파일에서 마우스 우클릭 > Web Services > Generate Client 를 선택합니다.



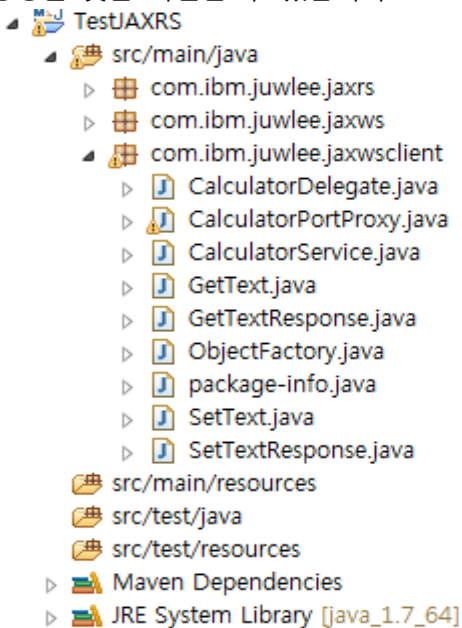
Web Service 마법사가 나오면 client 레벨을 deploy 까지만 맞추고 나머지 설정을 확인한 후 이상이 없으면 Next 를 클릭합니다.



다음으로 Output folder 의 위치를 조정하고 Target package 명을 새롭게 하나 생성합니다. (반드시 이렇게 해야하는 것은 아니고 Web Service Provider 랑 Client 를 같은 프로젝트에 위치시켰기 때문에 구분을 위해서 분리한 것입니다.) 설정에 문제가 없으면 Finish 버튼을 클릭하여 Web Service Client 를 생성합니다.



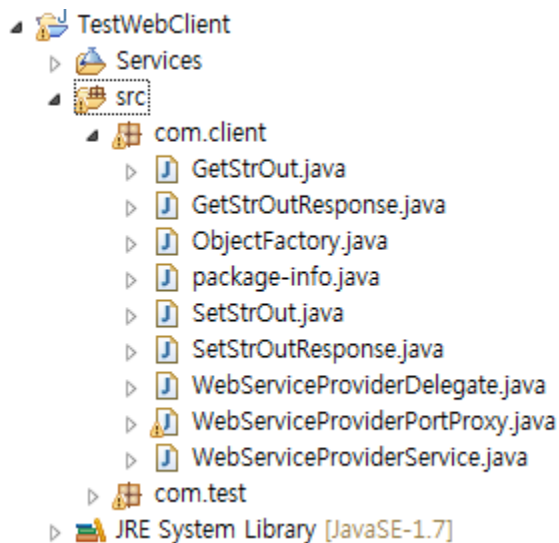
Web Service Client 생성이 문제없이 완료되면 하단과 같이 Java proxy 형태로 Web Service client 가 생성된 것을 확인할 수 있습니다.



3.4 JAX-WS 샘플 애플리케이션 개발2 (Web Service Client)

이전 챕터에서는 wsdl 을 이용해서 바로 Web Service Client 를 만들었습니다. 이번에는 그와 비슷한 형태로 아예 java client 에서 호출 가능한 형태로 개발을 해보도록 하겠습니다.

Java 프로젝트를 하나 생성한 후에 wsimport 명령을 사용해도 되고 좀 더 단순하게 작업하시려면 이전에 wsdl 을 이용해서 생성한 class 들을 eclipse 에서 copy 해오면 됩니다.



이렇게 하시면 Web Service 를 호출하기 위한 client 준비가 완료되었습니다.

Web Service client 개발 준비가 완료되면 하단과 같이 Service class 와 Delegate class 를 생성하여 필요한 method 를 call 하면 됩니다. Java client 형태로 개발되었기 때문에 바로 실행 가능합니다.

```
public void call() {  
  
    URL url = null;  
    try {  
        url = new URL("http://localhost:9080/WebServiceProviderService?wsdl");  
    } catch (MalformedURLException e) {  
        e.printStackTrace();  
    }  
  
    WebServiceProviderService wp = new WebServiceProviderService(url);  
    WebServiceProviderDelegate wd = wp.getWebServiceProviderPort();  
  
    System.out.println(wd.getStrOut());  
}
```

참고로 wsdl 로 client 를 위한 class 들을 생성하게 되면 패키지명은 변경할 수 없습니다. 클라이언트 마다 패키지명을 동적으로 변경해서 사용하고자 한다면 wsimport 를 통해서 원하시는 이름으로 클라이언트를 생성하면 됩니다.

```
wsimport -keep -s D:\eclipses\WeGovFrameDev-2.7.0-64bit_Liberty\workspace\TestWebClient\src -p com.client -d D:\eclipses\WeGovFrameDev-2.7.0-64bit_Liberty\workspace\TestWebClient\src http://localhost:9080/WebServiceProviderService?wsdl)
```

또는, Ant 를 수행하기 위한 build.xml 을 하단과 같이 만들고 ant 를 수행해도 됩니다.

(만약, Ant 수행후에 결과가 바로 반영되지 않는다면 해당 프로젝트에서 마우스 우클릭하여 refresh 버튼을 누르면 변경 결과가 바로 반영됩니다.)

```
<project default="wsimport">
  <target name="wsimport">
    <exec executable="${java.home}\..\bin\wsimport">

      <arg line="-keep -s .\src -p com.ibm.test -d .\src
http://localhost:9080/WebServiceProviderService?wsdl"/>

    </exec>

  </target>
</project>
```

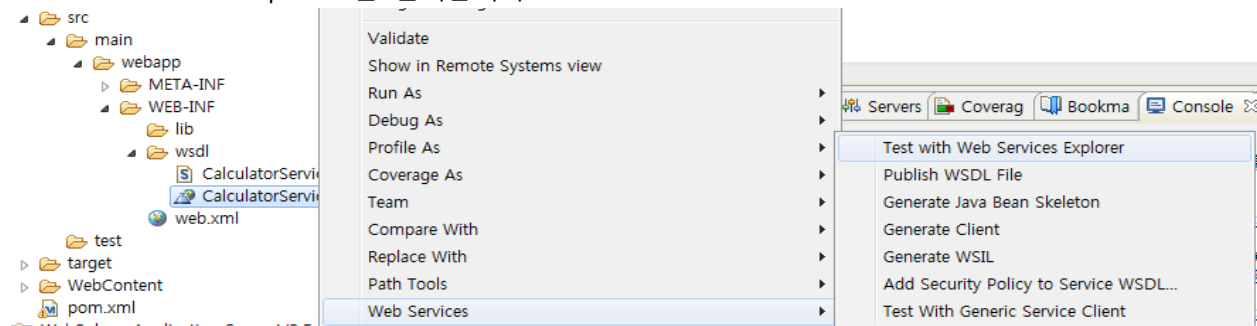
4 JAX-WS 테스트

4.1 JAX-WS 기반 Web Service Provider 테스트

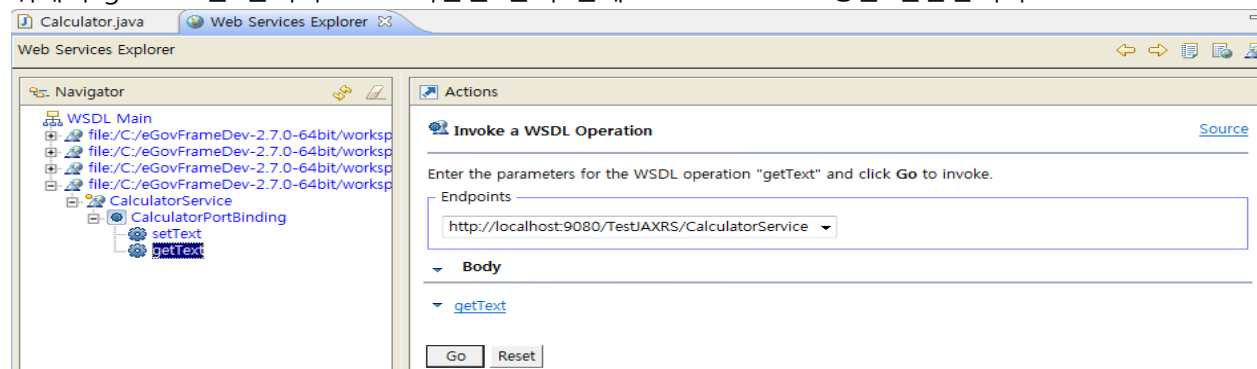
JAX-WS 표준의 Web Service Provider 를 테스트 하기 위해서는 해당 샘플 프로젝트가 IBM WAS Liberty 서버에 배포되어야 하고 구동되어야 합니다.

(여기서는 별도로 다루지 않고 JAX-RS 부분을 참고하세요)

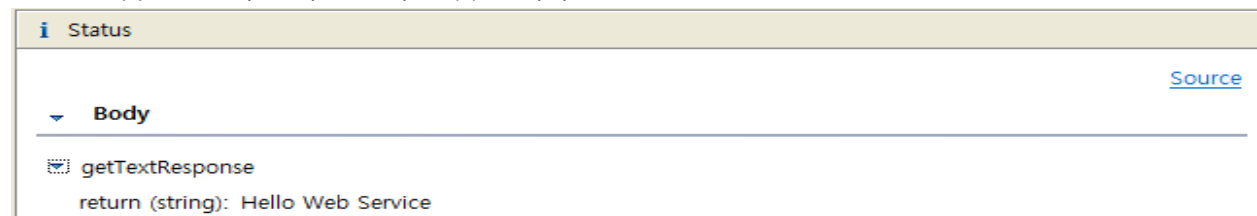
서버가 구동되었으면 이전에 생성했던 wsdl 파일을 찾은 후 마우스 우클릭 > Web Services > Test with Web Services Explorer 를 선택합니다.



그러면, 테스트를 위한 Web Services Explorer 가 실행되고 Web Service Provider 의 동작을 확인하기 위해서 getText 를 클릭하고 Go 버튼을 눌러 실제 Web Service 요청을 전달합니다.



하단과 같이 정상적으로 String 값을 return 하면 정상적으로 JAX-WS 기반으로 Web Service Provider 를 생성 및 WAS 배포/테스트 하신 것 입니다.



참고로 서비스로 등록된 wsdl 파일을 웹 브라우저를 통해서 확인하길 원한다면 하단과 같이 호출하여 wsdl 파일을 확인할 수 있습니다.

[http://localhost:9080/TestJAXRS\(컨텍스트루트\)/CalculatorService\(등록된 웹서비스명\)?wsdl](http://localhost:9080/TestJAXRS(컨텍스트루트)/CalculatorService(등록된 웹서비스명)?wsdl)



4.2 JAX-WS 기반 Web Service Client 테스트

이전 챕터에서는 JAX-WS 기반 Web Service 를 테스트 하기 위해서 별도의 도구를 이용했다면 이미 Web Service client 도 만들어두었기 때문에 이번에는 간단하게 Web Service Client 를 이용해서 Web Service 를 호출 테스트를 해보도록 하겠습니다.

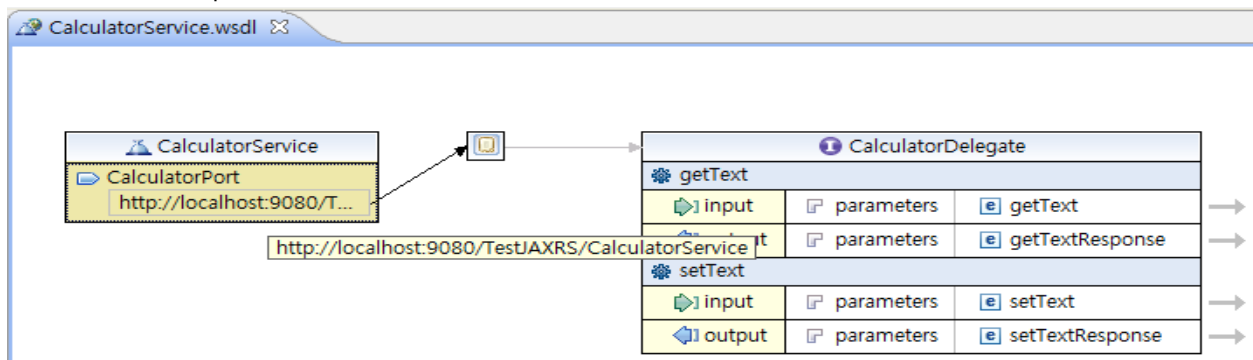
테스트를 하는 방법은 아주 간단하며 하단과 같이 Servlet class 를 하나 만들어서 Web Service client 를 통해서 Web Service 를 호출하도록 합니다.

```
protected void doWork(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("<html><body>");
    out.println("Web Service call <br>");

    CalculatorPortProxy proxy = new CalculatorPortProxy();
    //proxy._getDescriptor().setEndpoint("http://localhost:9080/TestJAXRS/CalculatorService");

    out.println("out : " + proxy.getText());
    out.println("</body></html>");
}
```

여기서 중요한 부분 중의 하나는 Web Service 의 위치를 지정하는 setEndpoint() 메소드이며 <http://Web> Service provider 가 위치한 서버주소:WAS서비스포트/컨텍스트루트/Web Service 서비스명으로 되어있습니다. Eclipse 에서 wsdl 파일을 열어보면 하단과 같이 확인할 수 있습니다. 서버의 변경이 없었으면 web service client 로 그대로 호출되지만 만약 Web Service Provider 를 다른 서버에 배포하거나 wsdl 로 client 작성 후 해당 부분에 대한 다른 변경 작업을 했다면 setEndpoint() 메소드를 이용해서 endpoint 를 변경해야 합니다.



이후 해당 servlet 을 웹 브라우저를 이용해서 호출하면 하단과 같이 결과를 바로 확인할 수 있습니다.

