

JAVA EE COURSE

HELLO WORLD WITH JAVA WEB SERVICES JAX-WS



By the expert: Eng. Ubaldo Acosta



JAVA EE COURSE

www.globalmentoring.com.mx

EXERCISE OBJECTIVE

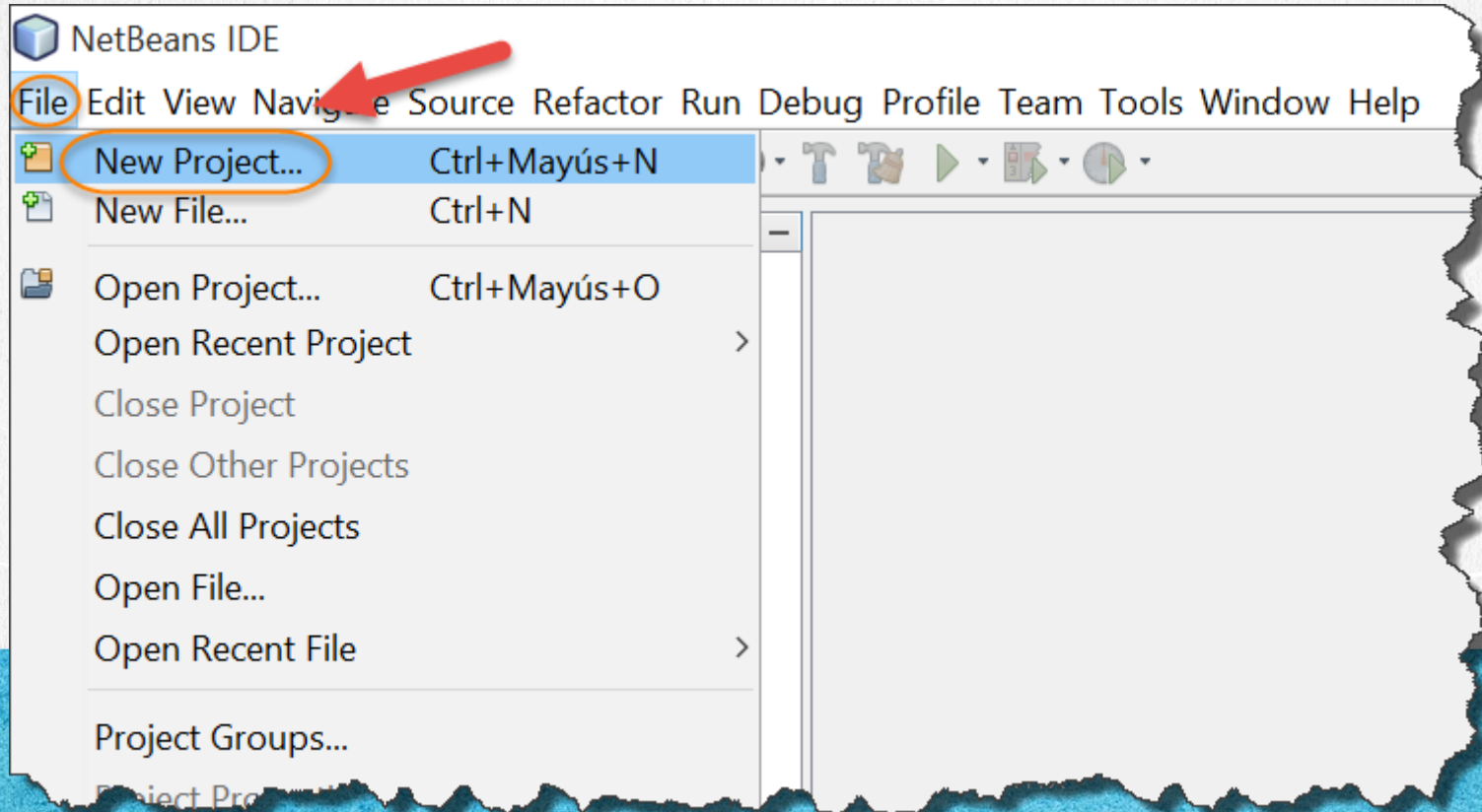
The objective of the exercise is to create our web service with JAX-WS and EJBs of the Stateless Session Bean type. The result is shown below:

The screenshot displays the GlassFish Server Open Source Edition web console. The top navigation bar includes 'Home', 'About...', and 'Help' buttons. Below the navigation bar, the user information is shown: 'User: admin | Domain: domain1 | Server: localhost'. The main title is 'GlassFish™ Server Open Source Edition'. On the left, a 'Tree' view shows the server structure, with 'Applications' selected. The right pane, titled 'Web Service Endpoint Information', provides details for a web service endpoint. A 'Back' button is located in the top right of this pane.

Web Service Endpoint Information	
View details about a web service endpoint.	
Application Name:	add-ws
Tester:	/ServiceAddWSImplService/ServiceAddWSImpl?Tester
WSDL:	/ServiceAddWSImplService/ServiceAddWSImpl?wsdl
Endpoint Name:	ServiceAddWSImpl
Service Name:	ServiceAddWSImplService
Port Name:	ServiceAddWSImplPort
Deployment Type:	109
Implementation Type:	EJB
Implementation Class Name:	beans.ServiceAddWSImpl
Endpoint Address URI:	/ServiceAddWSImplService/ServiceAddWSImpl
Namespace:	http://beans/

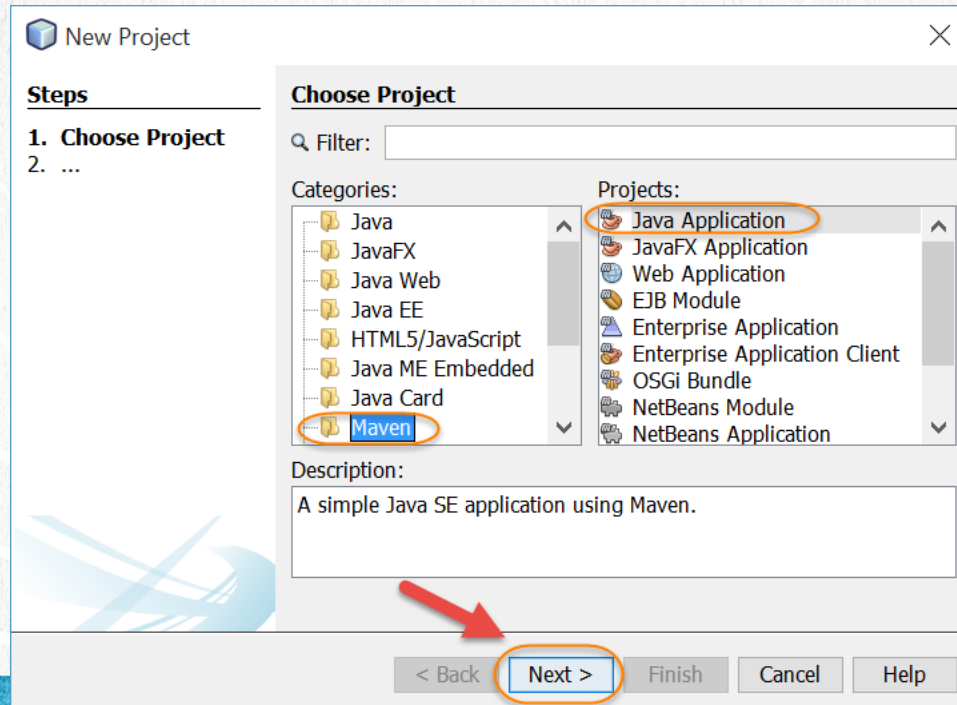
1. CREATE A NEW PROJECT

We create the add-ws project:



1. CREATE A NEW PROJECT

We create the add-ws project as a maven project:



JAVA EE COURSE

www.globalmentoring.com.mx

1. CREATE A NEW PROJECT

We create the add-ws project as a maven project:

New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: sumaws

Project Location: C:\Cursos\JavaEE\Leccion11 Browse...

Project Folder: C:\Cursos\JavaEE\Leccion11\sumaws

Artifact Id: sumaws

Group Id: mx.com.gm

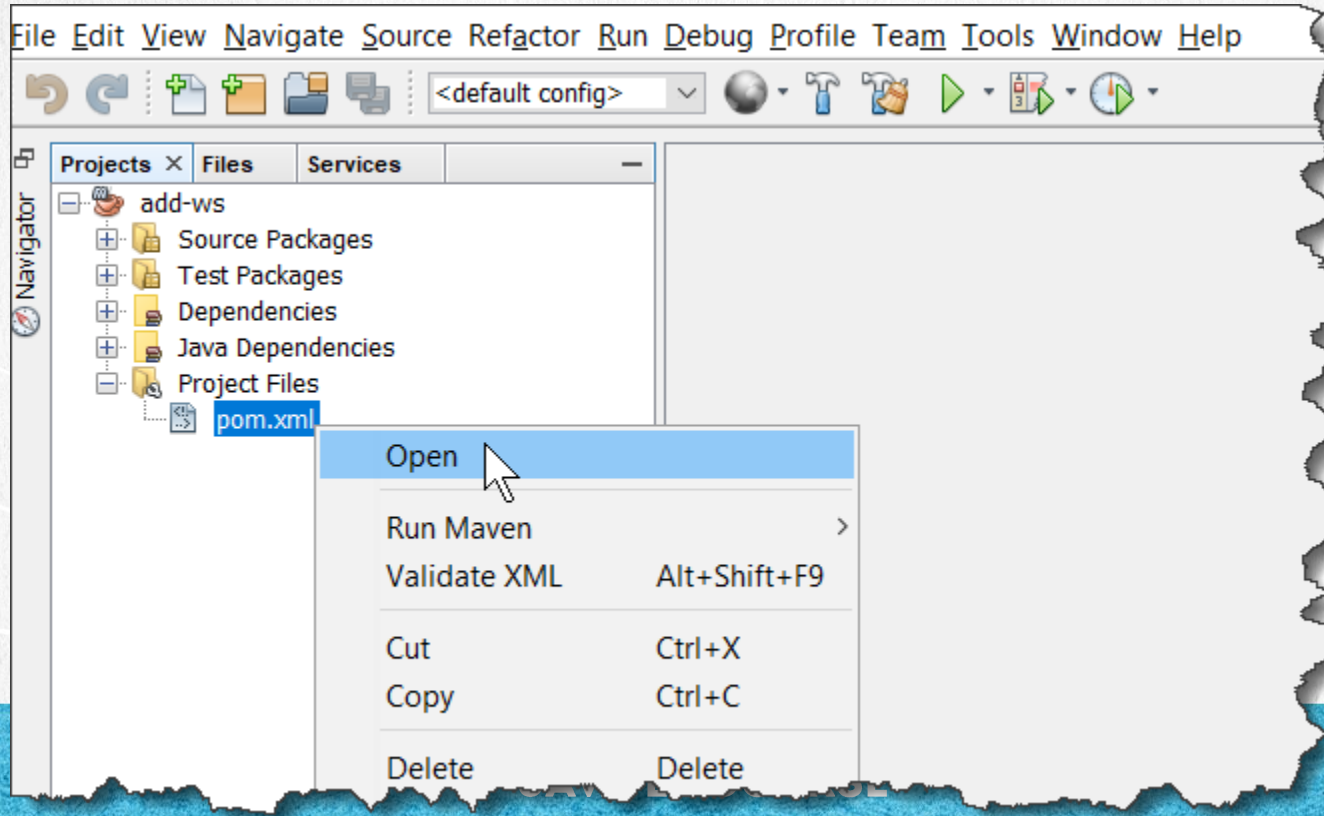
Version: 1.0

Package: (Optional)

< Back Next > **Finish** Cancel Help

2. MODIFY THE POM.XML

Modify the pom.xml:



2. MODIFY THE FILE

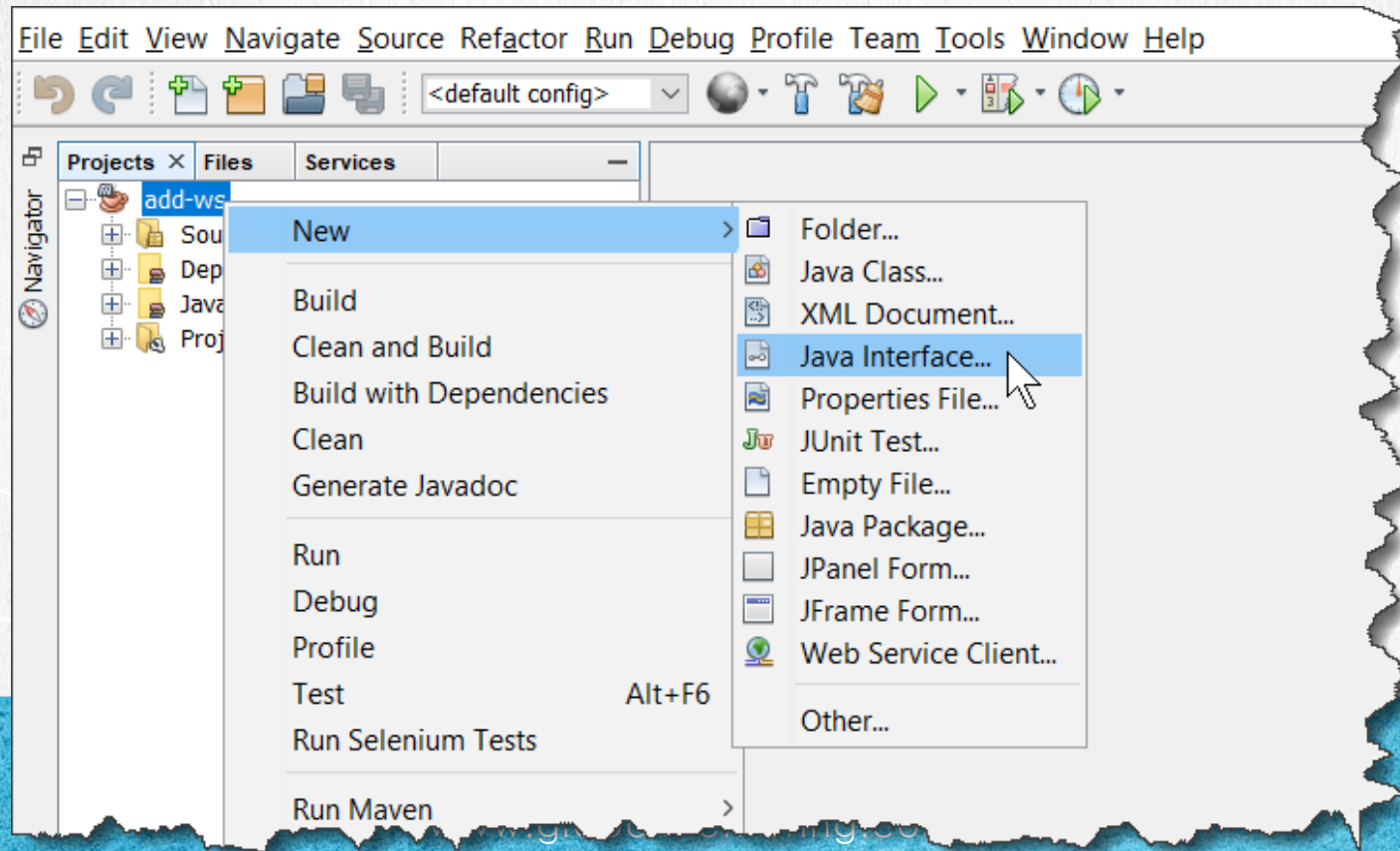
pom.xml:

Click to download

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>beans</groupId>
  <artifactId>add-ws</artifactId>
  <version>1</version>
  <packaging>jar</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>8.0</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
</project>
```

3. CREATE A JAVA INTERFACE

We create the ServiceAddWS.java interface:



3. CREATE A JAVA INTERFACE

We create the ServiceAddWS.java interface:

New Java Interface

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: ServiceAddWS

Project: add-ws

Location: Source Packages

Package: beans

Created File: C:\Courses\JavaEE\Lesson06\add-ws\src\main\java\beans\ServiceAddWS.java

< Back Next > **Finish** Cancel Help

4. MODIFY THE CODE

[ServiceAddWS.java:](#)

[Click to download](#)

```
package beans;

import javax.ws.WebMethod;
import javax.ws.WebService;

@WebService
public interface ServiceAddWS {

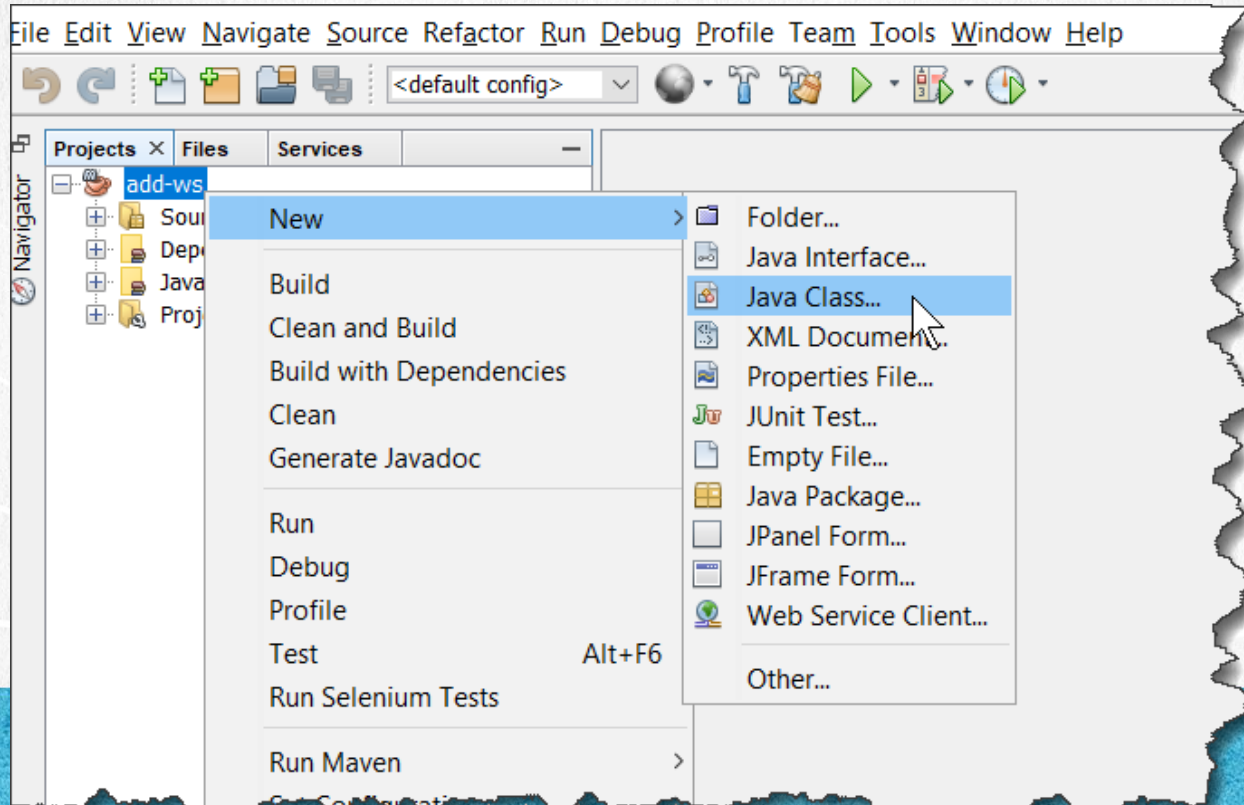
    @WebMethod
    public int add(int a, int b);
}
```

JAVA EE COURSE

www.globalmentoring.com.mx

5. CREATE A JAVA CLASS

We create the class ServiceAddWSImpl.java:



5. CREATE A JAVA CLASS

We create the class ServiceAddWSImpl.java:

New Java Class

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: ServiceAddWSImpl

Project: add-ws

Location: Source Packages

Package: beans

Created File: C:\Courses\JavaEE\Lesson06\add-ws\src\main\java\beans\ServiceAddWSImpl.java

< Back Next > **Finish** Cancel Help

6. MODIFY THE FILE

ServiceAddWSImpl.java:

[Click to download](#)

```
package beans;

import javax.ejb.Stateless;
import javax.jws.WebService;

@Stateless
@WebService(endpointInterface = "beans.ServiceAddWS")
public class ServiceAddWSImpl implements ServiceAddWS {

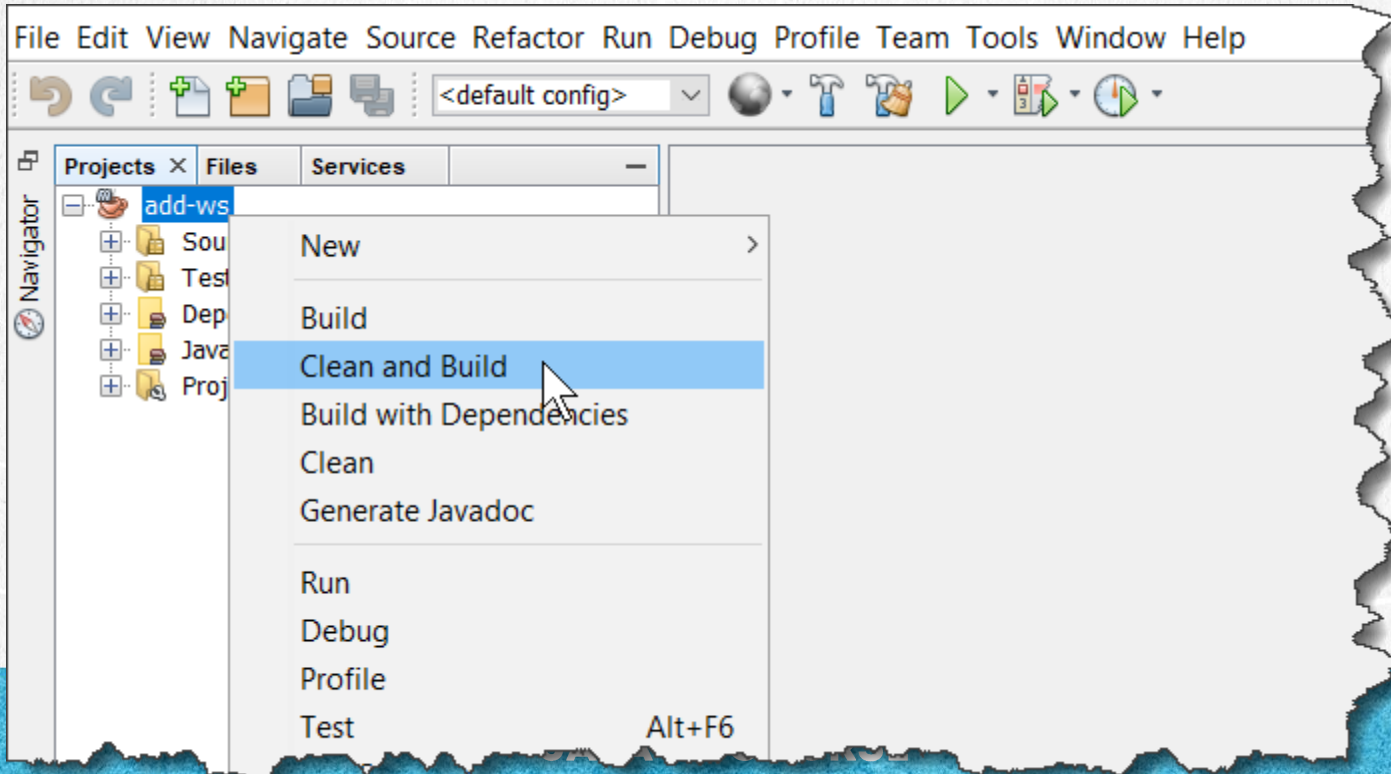
    @Override
    public int add(int a, int b) {
        return a + b;
    }
}
```

JAVA EE COURSE

www.globalmentoring.com.mx

7. WE CREATE THE .JAR FILE

We do clean & build to create the .jar file:



7. CREATE THE .JAR FILE

We do clean & build to create the .jar file:



```
Output - Build (add-ws) X
-----
T E S T S
-----

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

--- maven-jar-plugin:2.3.2:jar (default-jar) @ add-ws ---
Building jar: C:\Courses\JavaEE\Lesson06\add-ws\target\add-ws-1.jar

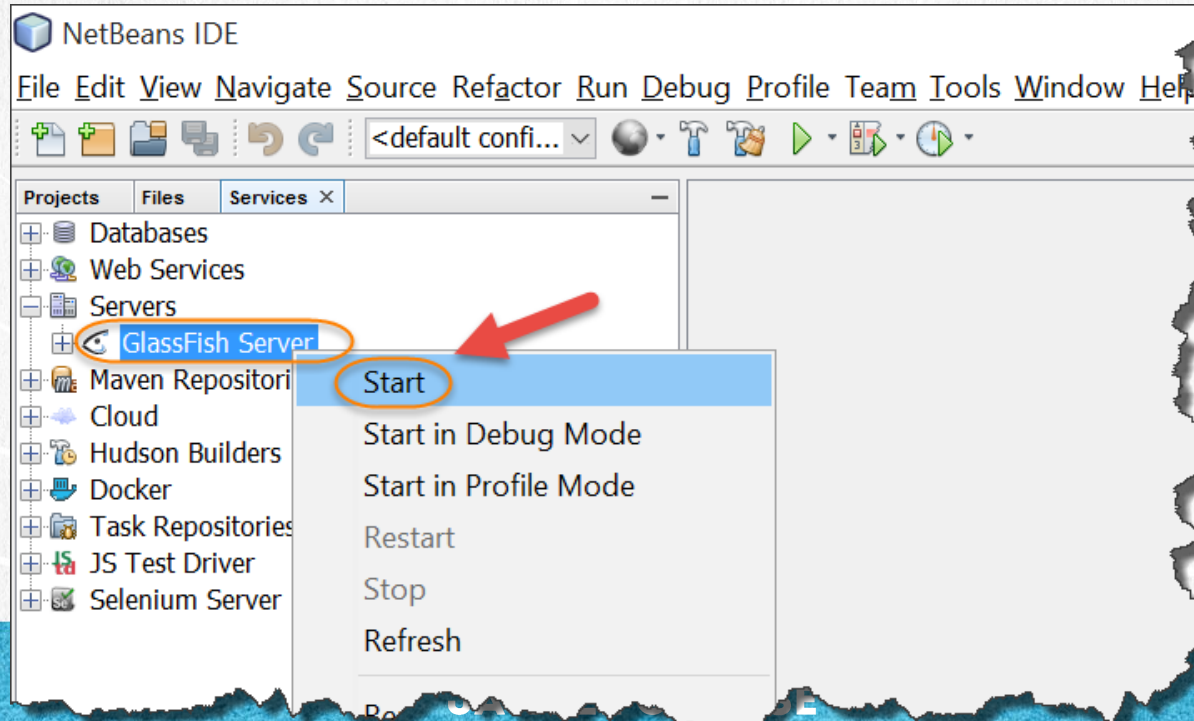
--- maven-install-plugin:2.3.1:install (default-install) @ add-ws ---
Installing C:\Courses\JavaEE\Lesson06\add-ws\target\add-ws-1.jar to C:\Users\user\.m2\repository\beans\add-ws\1\add-ws-1.jar
Installing C:\Courses\JavaEE\Lesson06\add-ws\pom.xml to C:\Users\user\.m2\repository\beans\add-ws\1\add-ws-1.pom

BUILD SUCCESS
-----

Total time: 4.645s
Finished at: Sat Oct 20 20:24:33
Final Memory: 15M/489M
-----
```

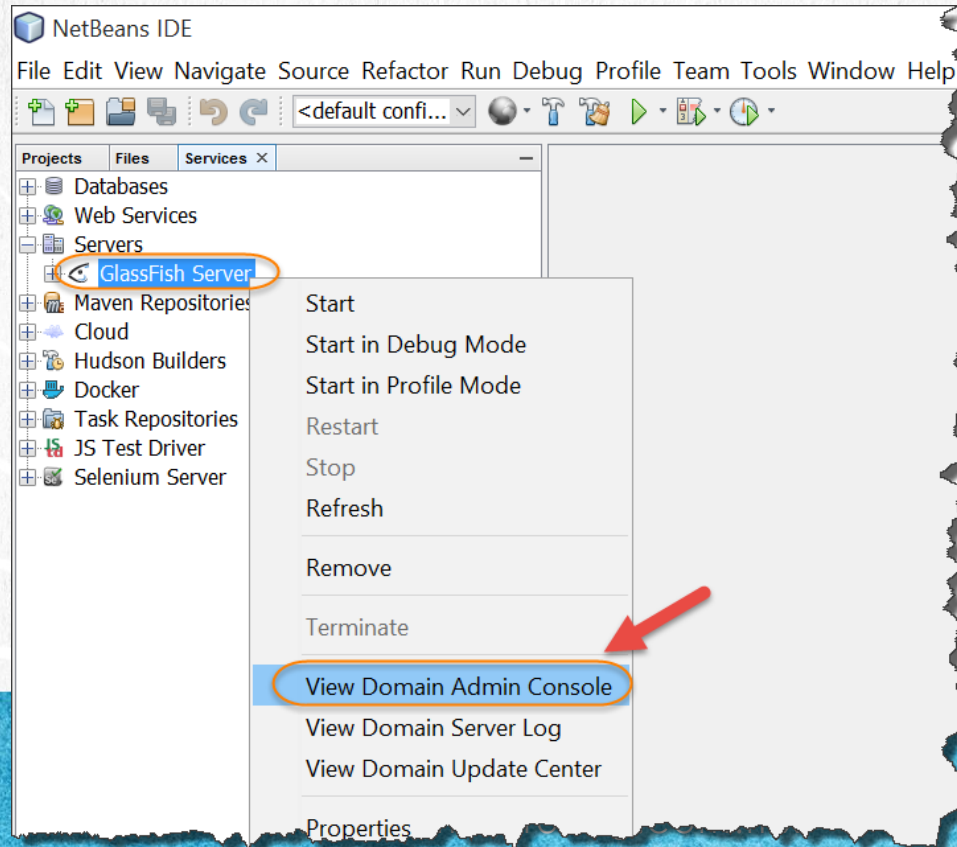
8. DEPLOY THE .JAR FILE

We deploy the .jar file generated in Glassfish. We start up the Glassfish server in case it is turned off:



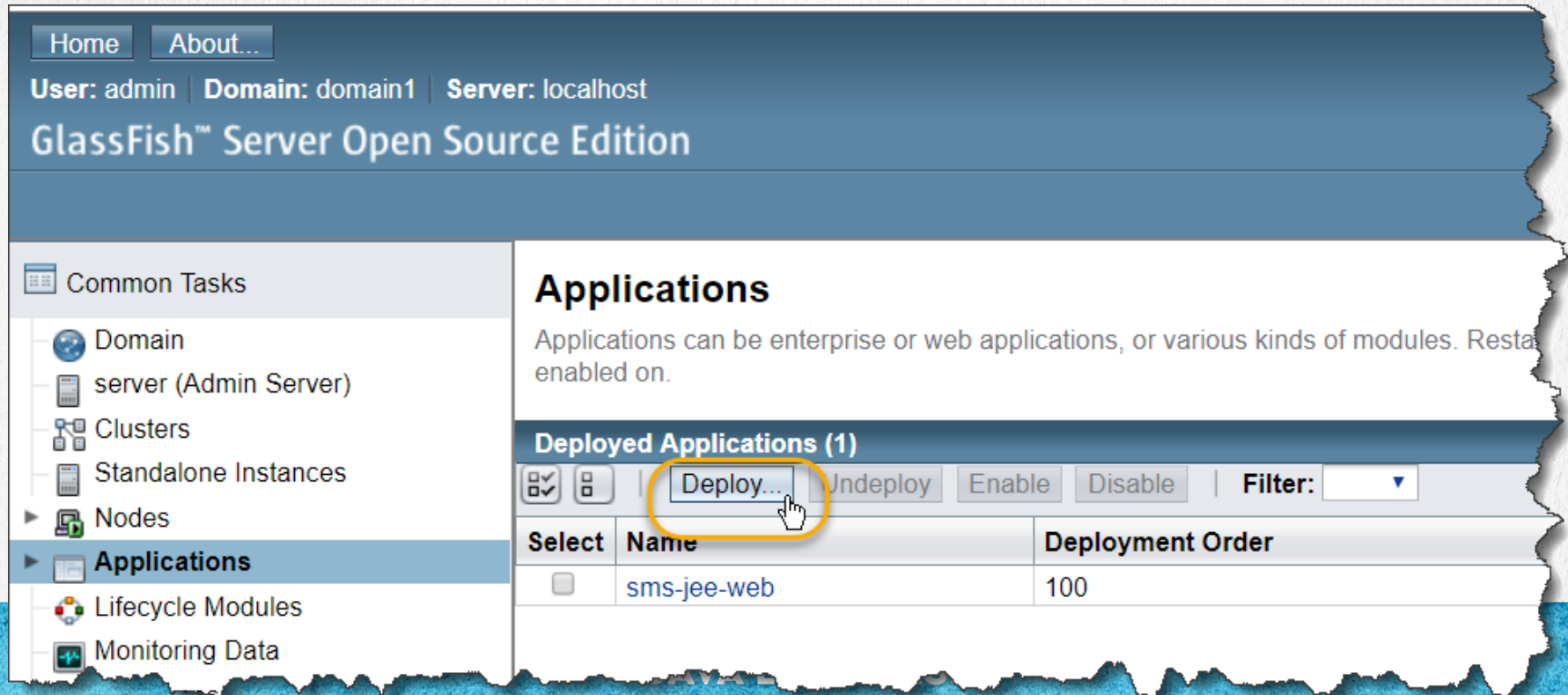
8. DEPLOY THE .JAR FILE

We enter the Glassfish console:



8. DEPLOY THE .JAR FILE

We deploy the .jar file generated in Glassfish.

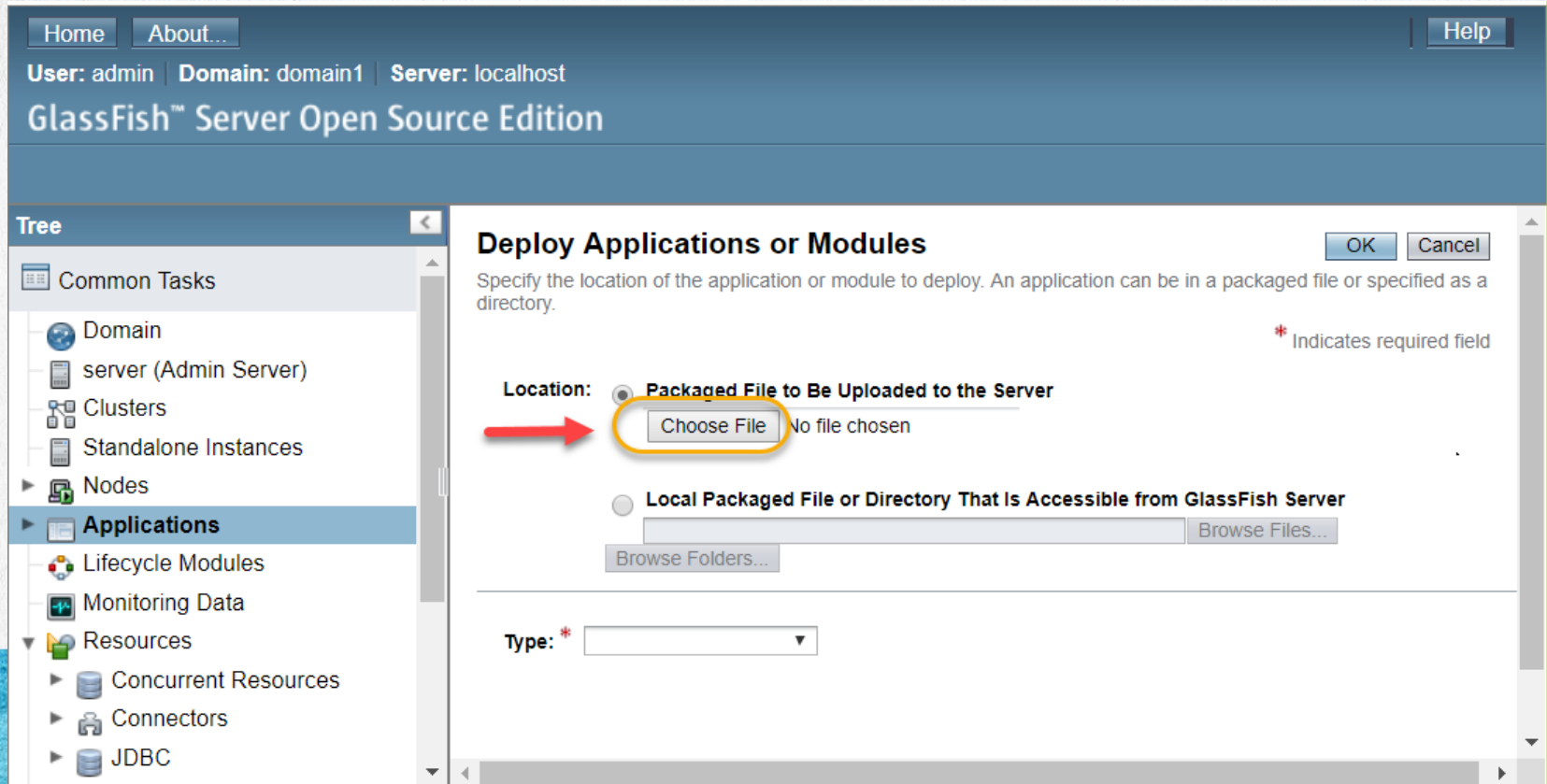


The screenshot shows the GlassFish Server Open Source Edition web console. The top navigation bar includes 'Home' and 'About...' buttons. Below this, the user information is displayed: 'User: admin | Domain: domain1 | Server: localhost'. The main title is 'GlassFish™ Server Open Source Edition'. On the left, a sidebar lists 'Common Tasks' with a tree view containing 'Domain', 'server (Admin Server)', 'Clusters', 'Standalone Instances', 'Nodes', 'Applications' (selected), 'Lifecycle Modules', and 'Monitoring Data'. The main content area is titled 'Applications' and contains the text: 'Applications can be enterprise or web applications, or various kinds of modules. Restart enabled on.' Below this, a section titled 'Deployed Applications (1)' features a toolbar with buttons for 'Deploy...', 'Undeploy', 'Enable', and 'Disable', along with a 'Filter:' dropdown. The 'Deploy...' button is highlighted with a yellow circle and a mouse cursor. Below the toolbar is a table with the following data:

Select	Name	Deployment Order
<input type="checkbox"/>	sms-jee-web	100

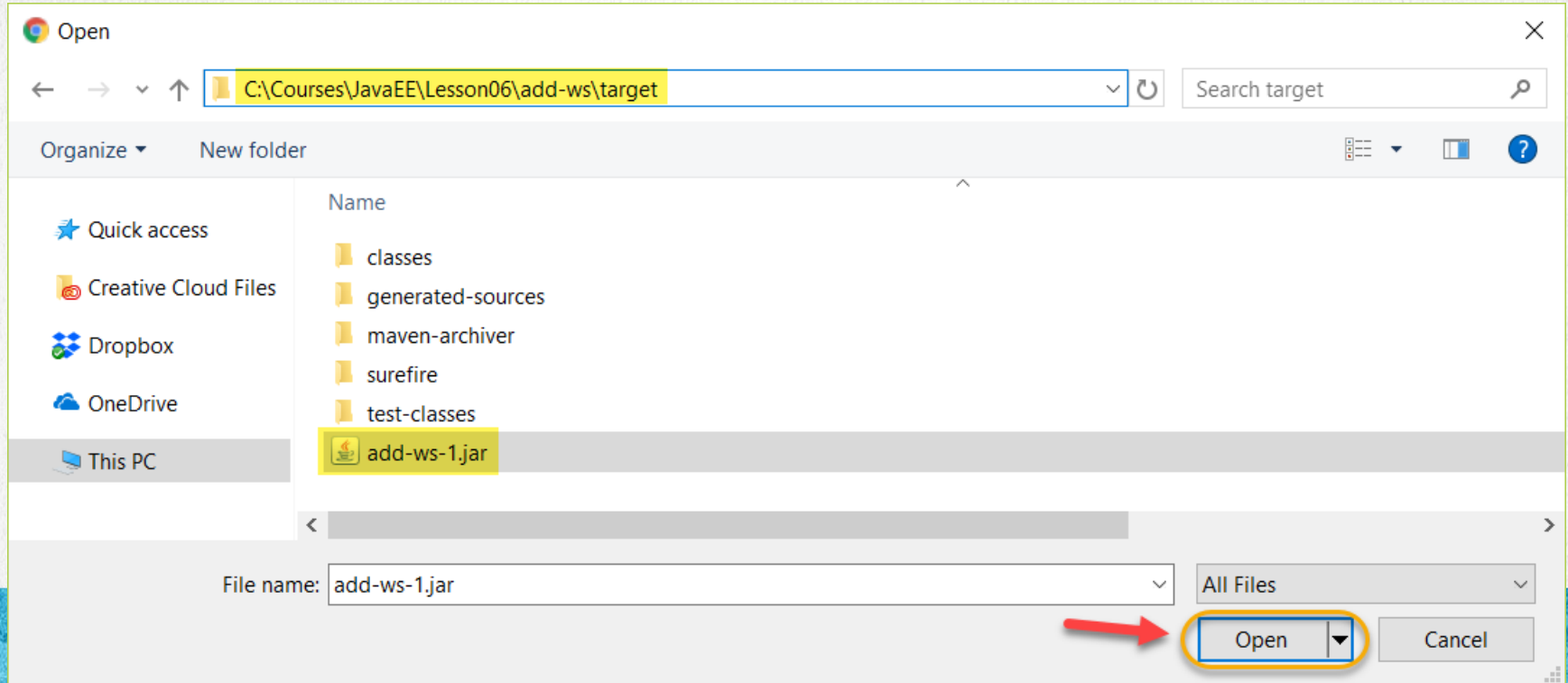
8. DEPLOY THE .JAR FILE

We deploy the .jar file generated in Glassfish.



8. DEPLOY THE .JAR FILE

We deploy the .jar file generated in Glassfish.



8. DEPLOY THE .JAR FILE

We rename the file as we observed (add-ws), and click on ok:

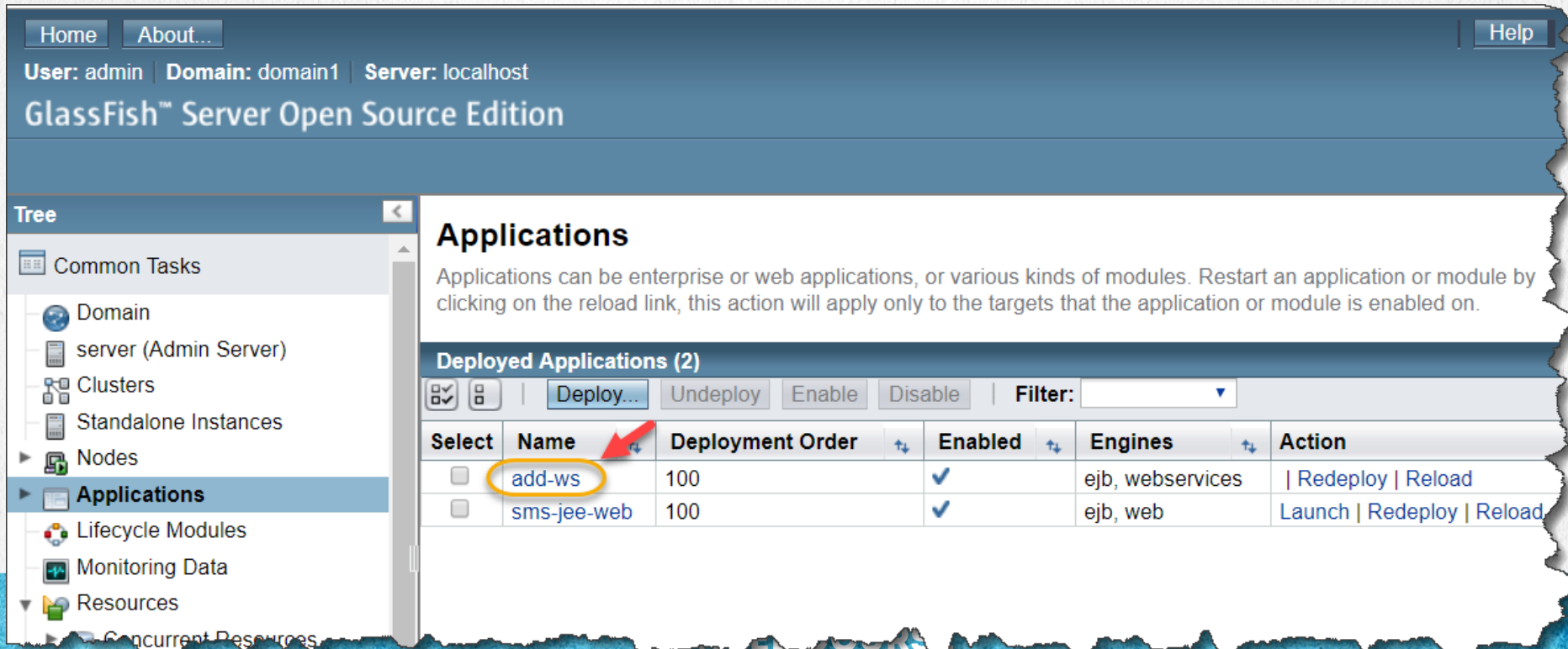
The screenshot shows the GlassFish Server Open Source Edition deployment wizard. The left sidebar contains a tree view with the following items: Common Tasks, Domain, server (Admin Server), Clusters, Standalone Instances, Nodes, Applications (selected), Lifecycle Modules, Monitoring Data, and Resources. The main panel is titled 'Deploy Applications or Modules' and contains the following fields:

- Location:** ☒ **Packaged File to Be Uploaded to the Server**
Choose File: add-ws-1.jar
- ☐ **Local Packaged File or Directory That Is Accessible from GlassFish Server**
Browse Files... Browse Folders...
- Type:** * EJB Jar
- Application Name:** * add-ws
- Status:** ☒ **Enabled**
Allows users to access the application.
- Implicit CDI:** ☒ **Enabled**
Implicit discovery of CDI beans

The 'OK' button is highlighted with a red arrow and a yellow circle. A red asterisk indicates required fields.

9. VERIFY THE CLIENT WS AND THE WSDL

We verify the WS client and the generated WSDL:



Home About... Help

User: admin | Domain: domain1 | Server: localhost

GlassFish™ Server Open Source Edition

Applications

Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking on the reload link, this action will apply only to the targets that the application or module is enabled on.

Deployed Applications (2)

Select	Name	Deployment Order	Enabled	Engines	Action
<input type="checkbox"/>	add-ws	100	✓	ejb, webservices	Redeploy Reload
<input type="checkbox"/>	sms-jee-web	100	✓	ejb, web	Launch Redeploy Reload

9. VERIFY THE CLIENT WS AND THE WSDL

We verify the WS client and the generated WSDL:

The screenshot shows the GlassFish Server Open Source Edition administration console. The top navigation bar includes 'Home', 'About...', and 'Help' buttons. Below the navigation bar, the user information is displayed: 'User: admin | Domain: domain1 | Server: localhost'. The main title is 'GlassFish™ Server Open Source Edition'.

The left sidebar contains a 'Tree' view with the following structure:

- Common Tasks
- Domain
 - server (Admin Server)
 - Clusters
 - Standalone Instances
 - Nodes
 - Applications**
 - Lifecycle Modules
 - Monitoring Data
 - Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - JNDI
 - JavaMail Sessions
 - Resource Adapter Configs
 - Configurations
 - default-config

The main content area is titled 'Modify an existing application or module.' and contains the following configuration fields:

- Name:** add-ws
- Status:** ☒ **Enabled**
- Implicit CDI:** ☒ **Enabled**
Implicit discovery of CDI beans
- Location:** \${com.sun.aas.instanceRootURI}/applications/add-ws/
- Deployment Order:** 100
A number that determines the loading order of the application at server startup. Lower numbers are loaded first. The default is 100.
- Libraries:**
- Description:**

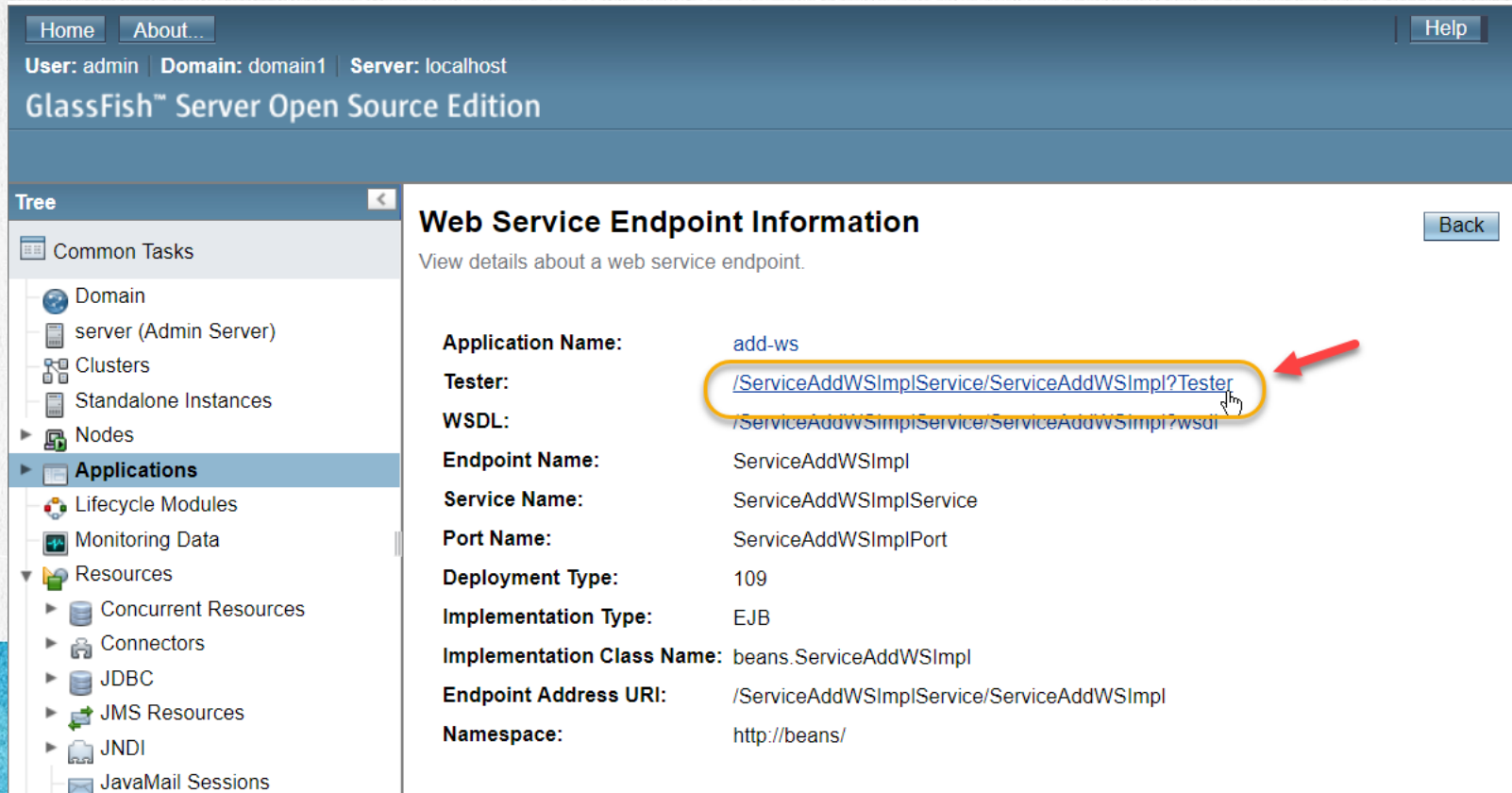
Below the configuration fields is a table titled 'Modules and Components (2)':

Module Name	Engines	Component Name	Type	Action
add-ws	[ejb, webservices, weld]	-----	-----	View Endpoint
add-ws		ServiceAddWSImpl	StatelessSessionBean	

A red arrow points from the 'View Endpoint' link in the table to the 'Type' column header.

9. VERIFY THE CLIENT WS AND THE WSDL

We verify the WS client and the generated WSDL:



Home About... Help

User: admin | Domain: domain1 | Server: localhost

GlassFish™ Server Open Source Edition

Tree

- Common Tasks
- Domain
 - server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
- Applications**
 - Lifecycle Modules
 - Monitoring Data
 - Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - JNDI
 - JavaMail Sessions

Web Service Endpoint Information

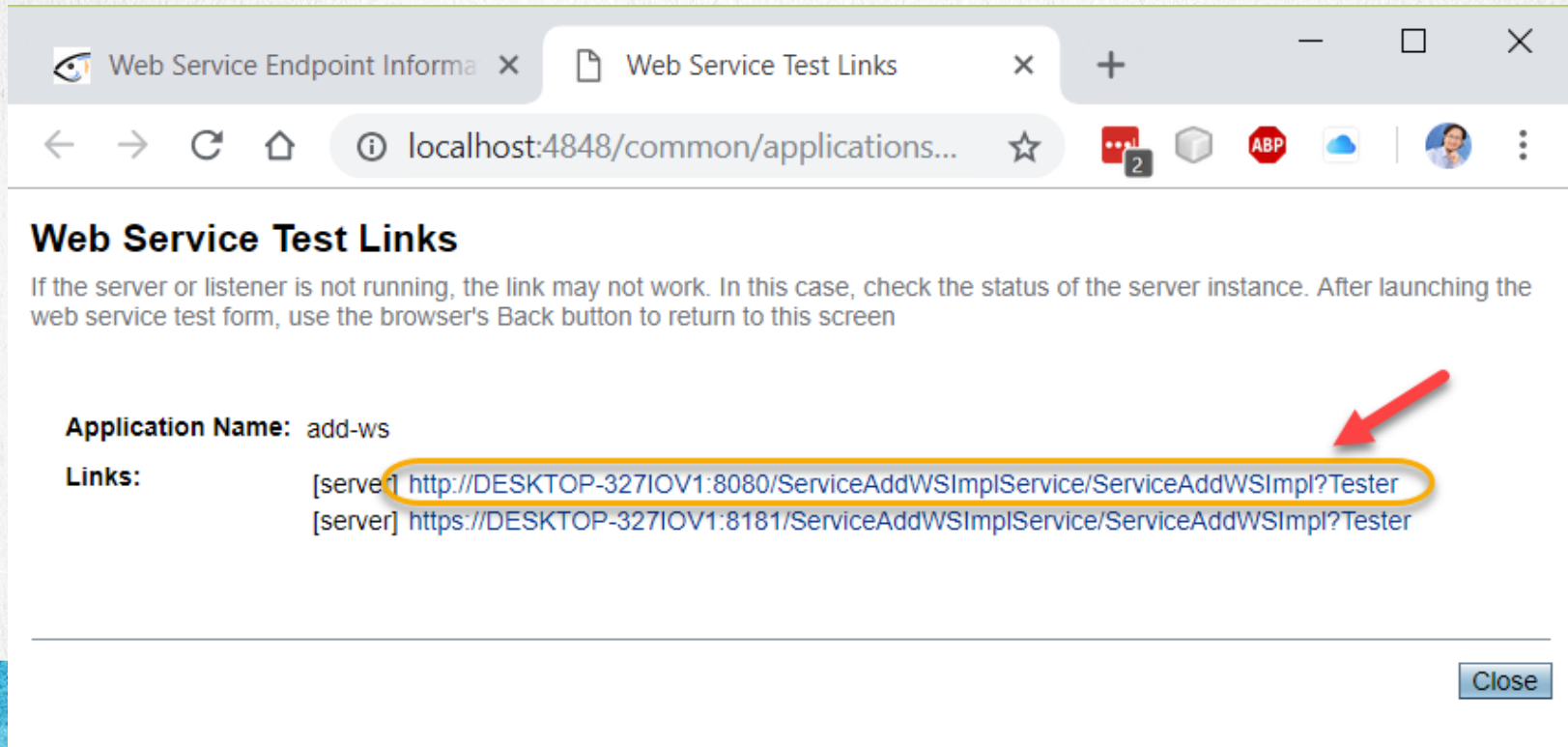
View details about a web service endpoint.

Back

Application Name:	add-ws
Tester:	/ServiceAddWSImplService/ServiceAddWSImpl?Tester
WSDL:	/ServiceAddWSImplService/ServiceAddWSImpl?wsdl
Endpoint Name:	ServiceAddWSImpl
Service Name:	ServiceAddWSImplService
Port Name:	ServiceAddWSImplPort
Deployment Type:	109
Implementation Type:	EJB
Implementation Class Name:	beans.ServiceAddWSImpl
Endpoint Address URI:	/ServiceAddWSImplService/ServiceAddWSImpl
Namespace:	http://beans/

9. VERIFY THE CLIENT WS AND THE WSDL

We verify the WS client:



Web Service Test Links

If the server or listener is not running, the link may not work. In this case, check the status of the server instance. After launching the web service test form, use the browser's Back button to return to this screen

Application Name: add-ws

Links:

- [server1] <http://DESKTOP-327IOV1:8080/ServiceAddWSImplService/ServiceAddWSImpl?Tester>
- [server] <https://DESKTOP-327IOV1:8181/ServiceAddWSImplService/ServiceAddWSImpl?Tester>

Close

9. VERIFY THE CLIENT WS AND THE WSDL

We verify the WS client:

Web Service Endpoint Informa x ServiceAddWSImplService Wel x

← → ↻ 🏠 ⓘ Not secure | desktop-327iov1:8080/... ☆

ServiceAddWSImplService Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract int beans.ServiceAddWS.add(int,int)

add (1, 2)

9. VERIFY THE CLIENT WS AND THE WSDL

We verify the WS client:

add Method invocation

Method parameter(s)

Type	Value
int	1
int	2

Method returned

int : "3"

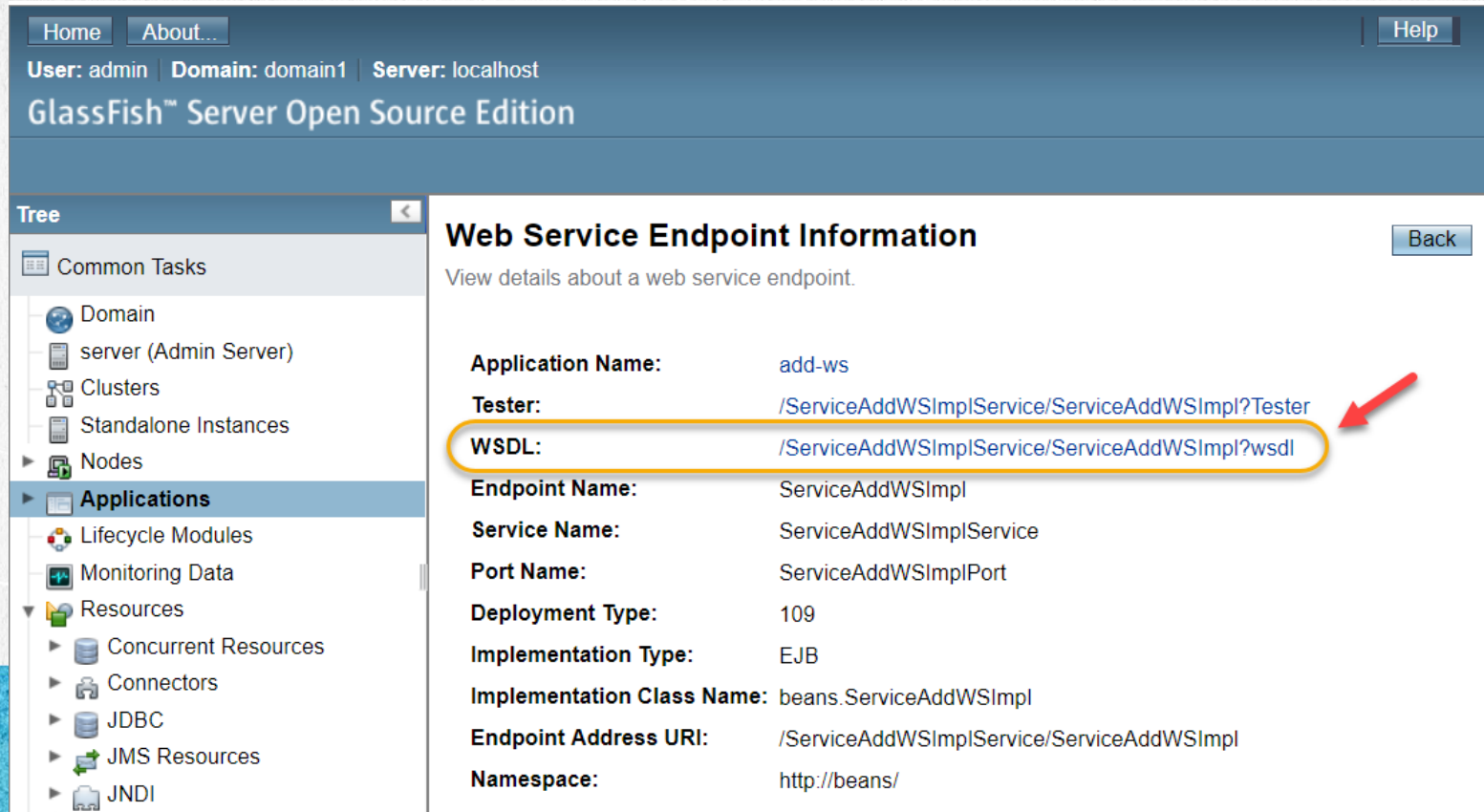
SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body xmlns:ns2="http://beans/">
    <ns2:add>
      <arg0>1</arg0>
      <arg1>2</arg1>
    </ns2:add>
  </S:Body>
</S:Envelope>
```

SOAP Response

9. VERIFY THE CLIENT WS AND THE WSDL

We verify the WSDL file:



The screenshot displays the GlassFish Server Open Source Edition interface. The top navigation bar includes links for 'Home', 'About...', and 'Help'. Below this, the user information is shown: 'User: admin | Domain: domain1 | Server: localhost'. The main title is 'GlassFish™ Server Open Source Edition'.

The left sidebar contains a 'Tree' view with the following structure:

- Common Tasks
- Domain
 - server (Admin Server)
- Clusters
- Standalone Instances
- Nodes
- Applications**
 - Lifecycle Modules
 - Monitoring Data
 - Resources
 - Concurrent Resources
 - Connectors
 - JDBC
 - JMS Resources
 - JNDI

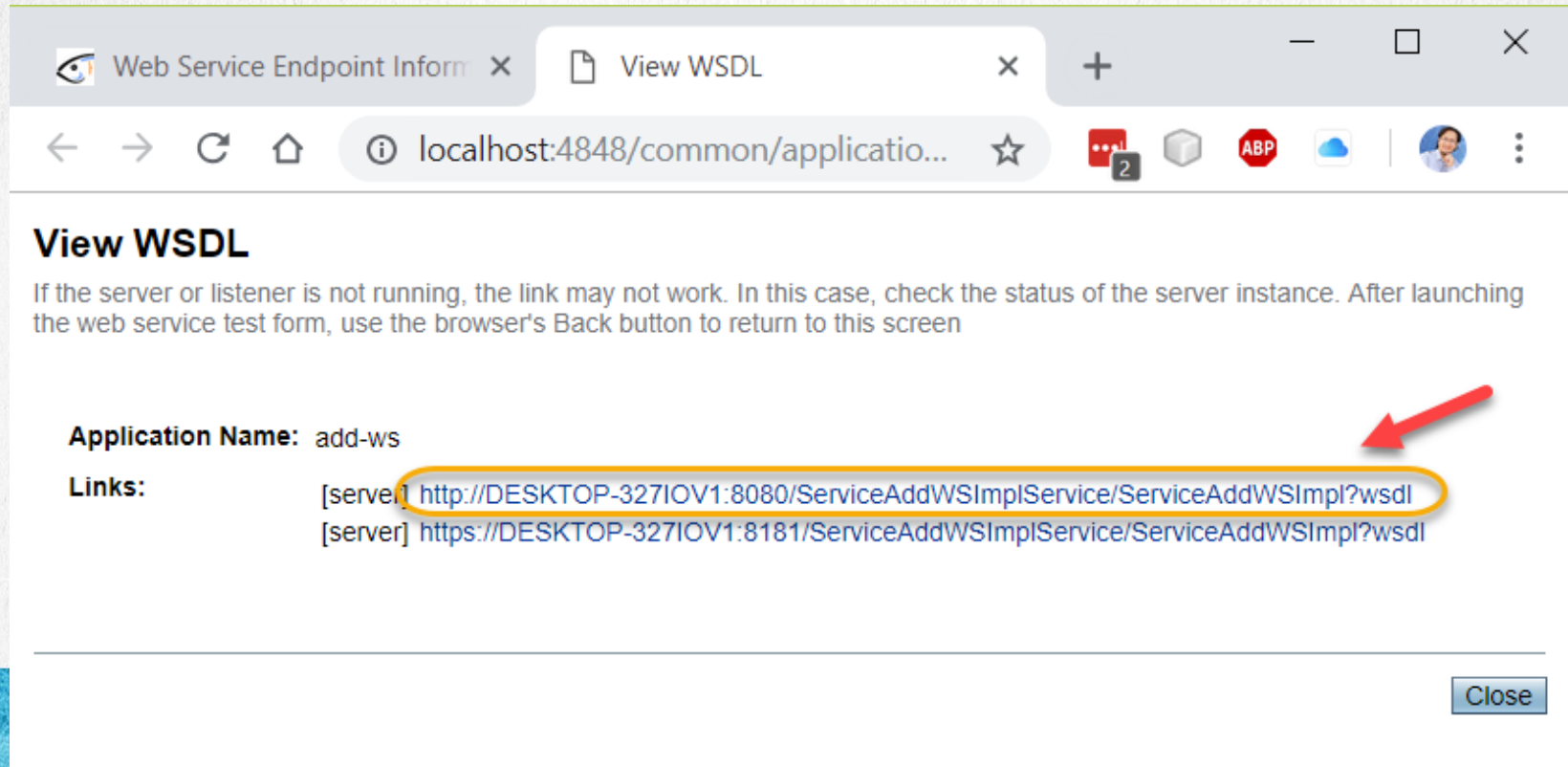
The main content area is titled 'Web Service Endpoint Information' and includes a 'Back' button. Below the title, it says 'View details about a web service endpoint.' The details are as follows:

Application Name:	add-ws
Tester:	/ServiceAddWSImplService/ServiceAddWSImpl?Tester
WSDL:	/ServiceAddWSImplService/ServiceAddWSImpl?wsdl
Endpoint Name:	ServiceAddWSImpl
Service Name:	ServiceAddWSImplService
Port Name:	ServiceAddWSImplPort
Deployment Type:	109
Implementation Type:	EJB
Implementation Class Name:	beans.ServiceAddWSImpl
Endpoint Address URI:	/ServiceAddWSImplService/ServiceAddWSImpl
Namespace:	http://beans/

A red arrow points to the WSDL URL, which is highlighted with a yellow circle.

9. VERIFY THE CLIENT WS AND THE WSDL

We verify the WSDL file:



View WSDL

If the server or listener is not running, the link may not work. In this case, check the status of the server instance. After launching the web service test form, use the browser's Back button to return to this screen

Application Name: add-ws

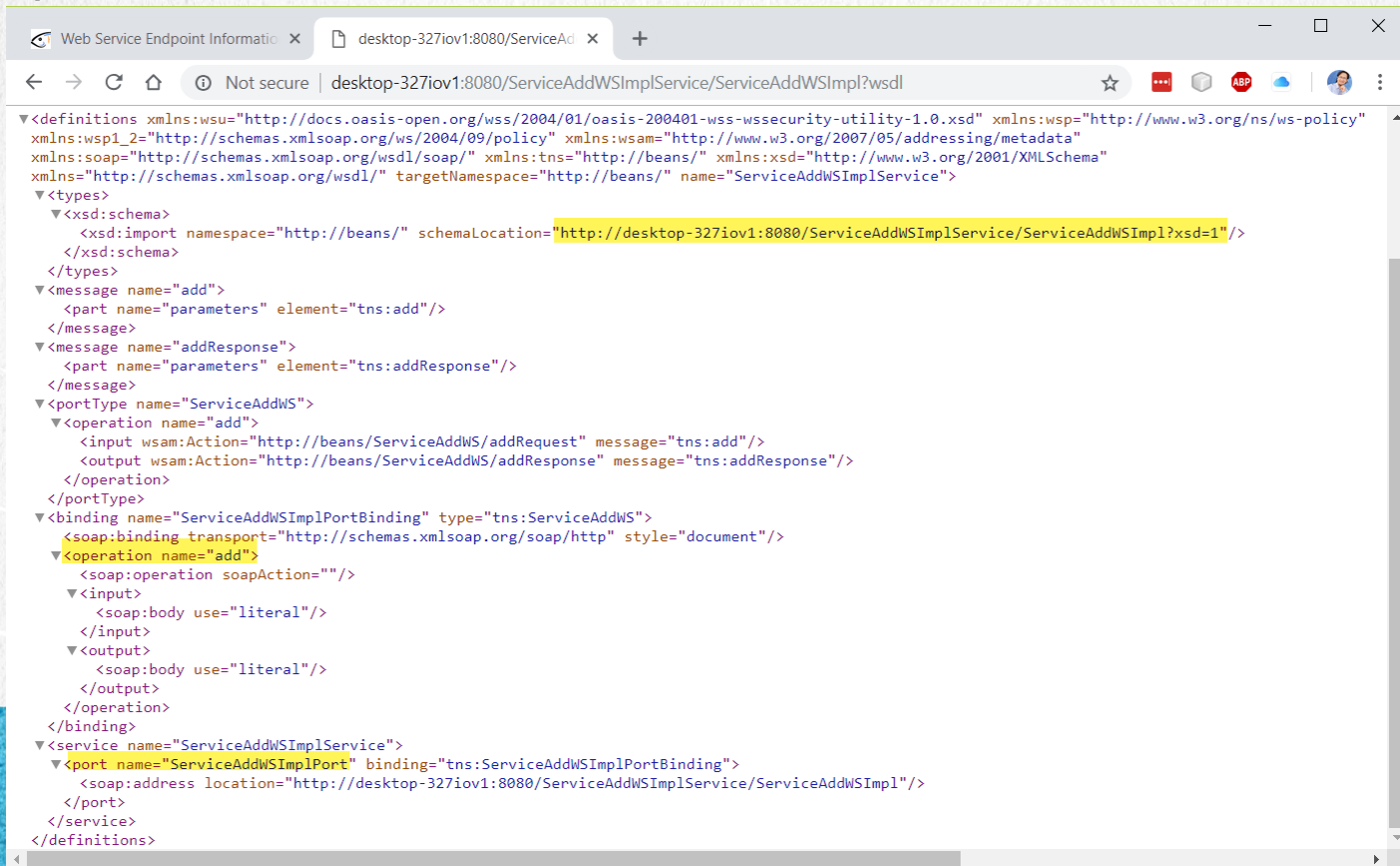
Links:

- [server] <http://DESKTOP-327IOV1:8080/ServiceAddWSImplService/ServiceAddWSImpl?wsdl>
- [server] <https://DESKTOP-327IOV1:8181/ServiceAddWSImplService/ServiceAddWSImpl?wsdl>

Close

9. VERIFY THE CLIENT WS AND THE WSDL

We verify the WSDL file:



The screenshot shows a web browser window with the address bar displaying the URL: `desktop-327iov1:8080/ServiceAddWSImplService/ServiceAddWSImpl?wsdl`. The browser's address bar also shows "Not secure" and a "Web Service Endpoint Information" tab. The main content area displays the WSDL XML document, which is partially expanded to show the `<types>` section. The XML content is as follows:

```
<?xml version='1.0' encoding='utf-8'>
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://beans/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://beans/" name="ServiceAddWSImplService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://beans/" schemaLocation="http://desktop-327iov1:8080/ServiceAddWSImplService/ServiceAddWSImpl?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="add">
    <part name="parameters" element="tns:add"/>
  </message>
  <message name="addResponse">
    <part name="parameters" element="tns:addResponse"/>
  </message>
  <portType name="ServiceAddWS">
    <operation name="add">
      <input wsam:Action="http://beans/ServiceAddWS/addRequest" message="tns:add"/>
      <output wsam:Action="http://beans/ServiceAddWS/addResponse" message="tns:addResponse"/>
    </operation>
  </portType>
  <binding name="ServiceAddWSImplPortBinding" type="tns:ServiceAddWS">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="add">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="ServiceAddWSImplService">
    <port name="ServiceAddWSImplPort" binding="tns:ServiceAddWSImplPortBinding">
      <soap:address location="http://desktop-327iov1:8080/ServiceAddWSImplService/ServiceAddWSImpl"/>
    </port>
  </service>
</definitions>
```

10. CHECK THE XSD FILE

We verify the xsd file:

<http://localhost:8080/ServiceAddWSImplService/ServiceAddWSImpl?xsd=1>



The screenshot shows a web browser window with the address bar displaying the URL `desktop-327iov1:8080/ServiceAddWSImplService/ServiceAddWSImpl?xsd=1`. The browser's developer tools are open, showing the XML content of the XSD file. The XML is a schema definition for a web service, including a complex type for the 'add' operation and a response type 'addResponse'.

```
<?xml version='1.0' encoding='UTF-8'>
<xs:schema xmlns:tns="http://beans/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" version="1.0" targetNamespace="http://beans/">
  <xs:element name="add" type="tns:add"/>
  <xs:element name="addResponse" type="tns:addResponse"/>
  <xs:complexType name="add">
    <xs:sequence>
      <xs:element name="arg0" type="xsd:int"/>
      <xs:element name="arg1" type="xsd:int"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="addResponse">
    <xs:sequence>
      <xs:element name="return" type="xsd:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

EXERCISE CONCLUSION

With this exercise created a Web Service called add, which receives two integer arguments, and returns as a result the sum of those arguments.

This is a very simple example of a Stateless EJB that exposes one of its methods as a Web service, but it is the same process that we will perform to create more complex Web Services from any method of an EJB that we need to expose.

In the following exercise we will see how to create a client to consume the web service already published and exposed on the Glassfish server.

ONLINE COURSE

JAVA EE

JAKARTA EE

By: Eng. Ubaldo Acosta



JAVA EE COURSE
www.globalmentoring.com.mx