

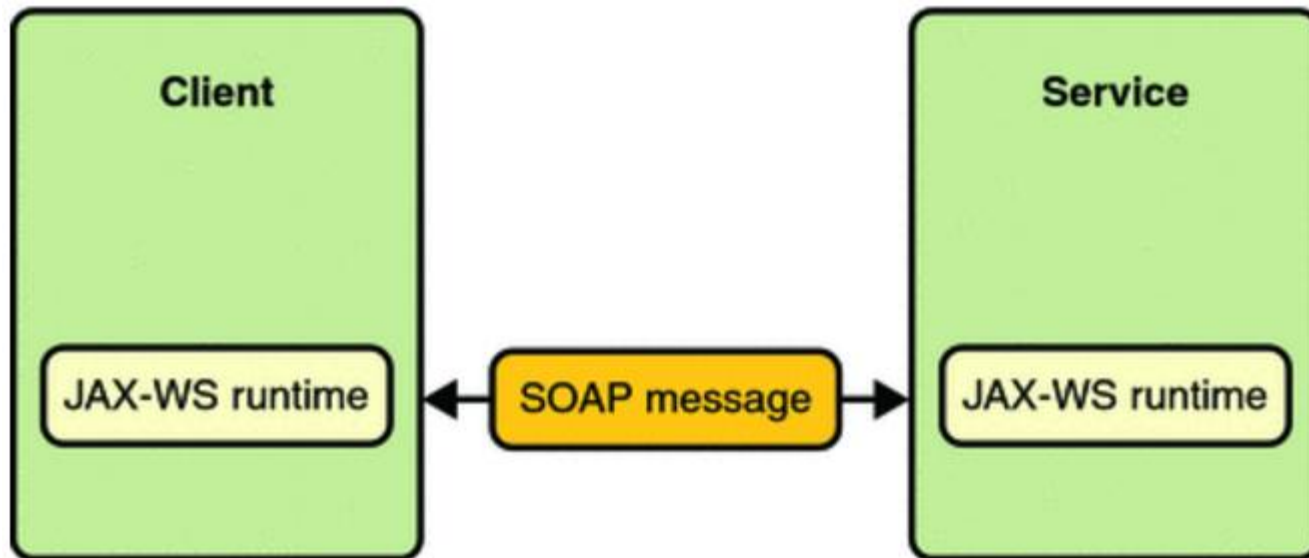
Creating a WS using Java

د. عبدالناصر ضياف

The Start

- ▶ Short introduction to NetBeans IDE
- ▶ Basic requirements for developing a simple WS
 - NetBeans IDE (Java EE download bundle)
 - Java Development Kit (JDK7 or JDK8)
 - Java EE-compliant web or application server (GlassFish Server)
 - The Java API for XML Web Service (JAX-WS): It is a java programming language API for creating SOAP web services.
 - It defines a standard Java- to-WSDL mapping which determines how WSDL operations are bound to Java methods when a SOAP message invokes a WSDL operation.

Communication between a JAX-WS Web Service and a Client



Web Service Parties

- ▶ The Web Service

- from the provider (exports a WSDL document)

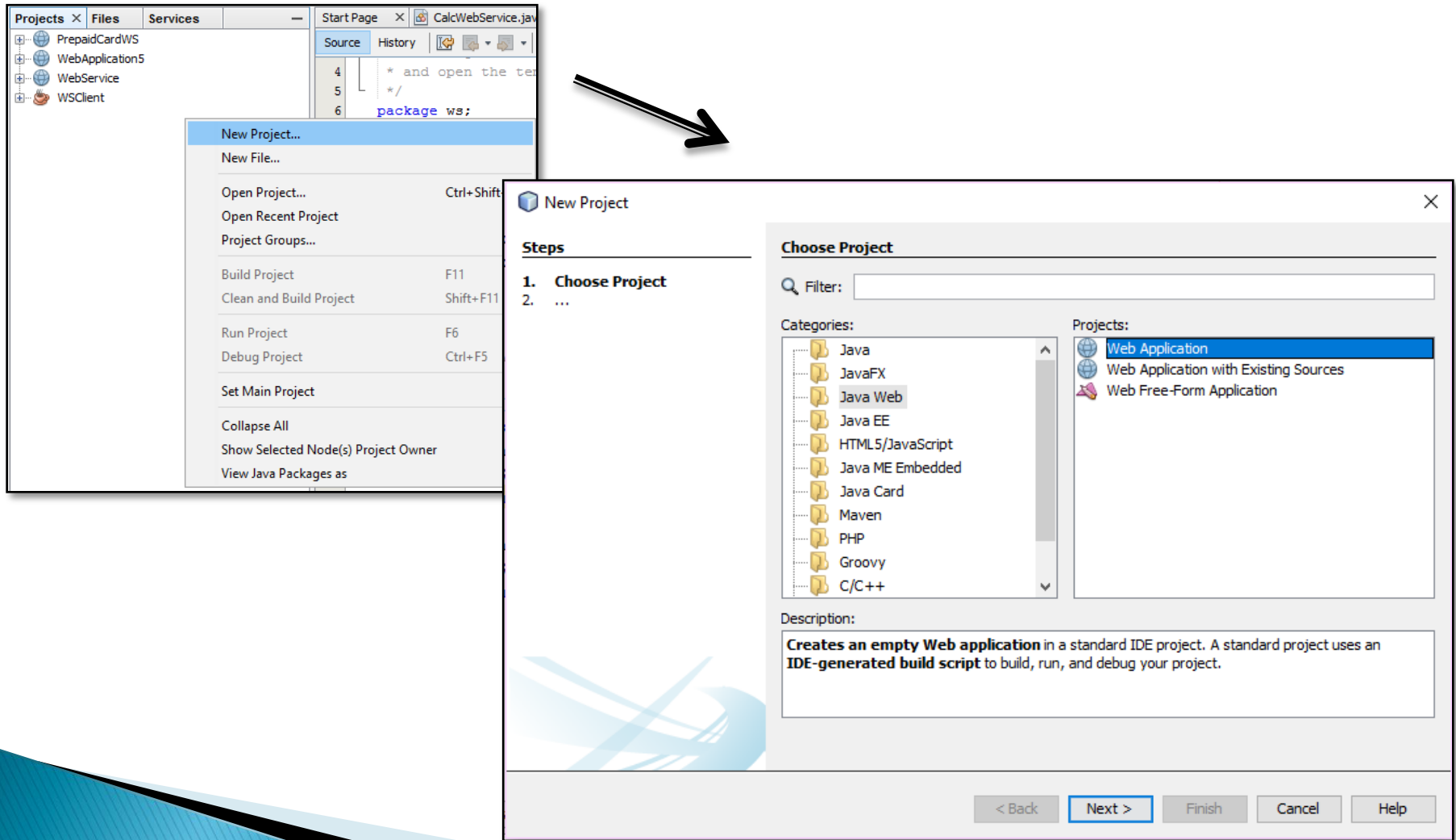
- ▶ The Client Application

- from the consumer (imports the WSDL document)

Creating A Simple WS

1. Create a new “Java Web Application” Project
2. Give a name and specify a location to the project
3. Create a simple JAVA class and some public methods
4. Convert the class to a WS by adding @WebService annotation
 - *Public methods become WS operations*
5. Build and deploy the project
6. Test the WS and view the generated WSDL document via GlassFish Admin Console.

1. Create a new Java Web Application project



The image shows a screenshot of an IDE interface. On the left, the 'Projects' pane lists several projects: PrepaidCardWS, WebApplication5, WebService, and WSCient. A context menu is open over the 'WebApplication5' project, with 'New Project...' selected. An arrow points from this menu to the 'New Project' dialog box on the right.

The 'New Project' dialog box has a 'Steps' section with the following items:

1. Choose Project
2. ...

The 'Choose Project' section includes a 'Filter' input field, a 'Categories' list, and a 'Projects' list. The 'Categories' list is expanded to 'Java Web', and 'Web Application' is selected in the 'Projects' list. The 'Description' field contains the following text:

Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

At the bottom of the dialog box, there are buttons for '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

2. Give a name and specify a location to the project

New Web Application

Steps

1. Choose Project
- 2. Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name:

Project Location:

Project Folder:

Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

< Back **Next >** Finish Cancel

New Web Application

Steps

1. Choose Project
2. Name and Location
- 3. Server and Settings**
4. Frameworks

Server and Settings

Add to Enterprise Application:

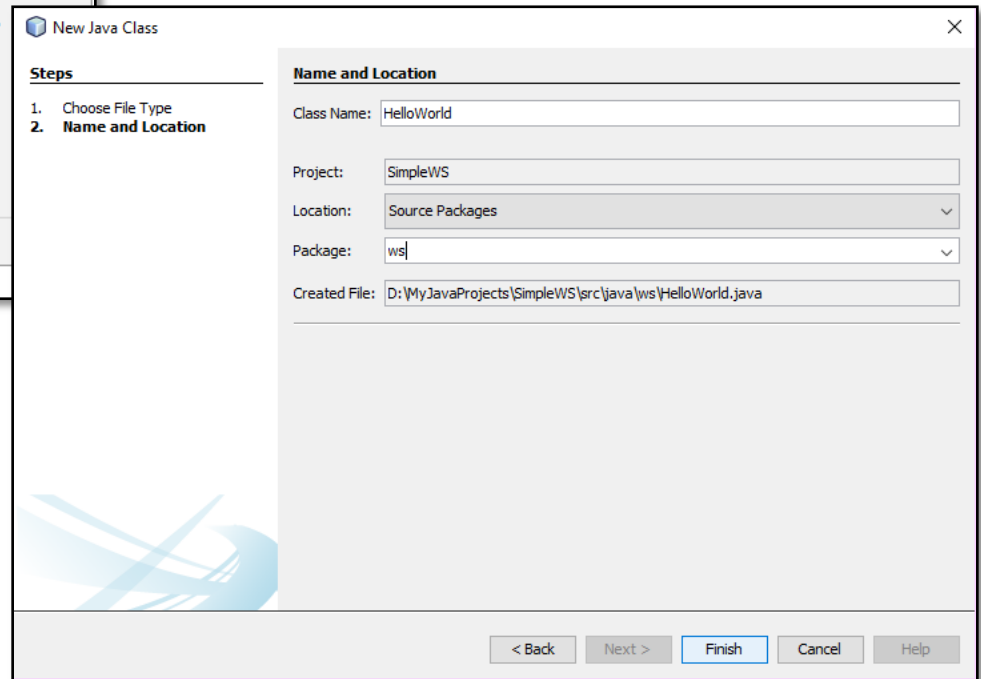
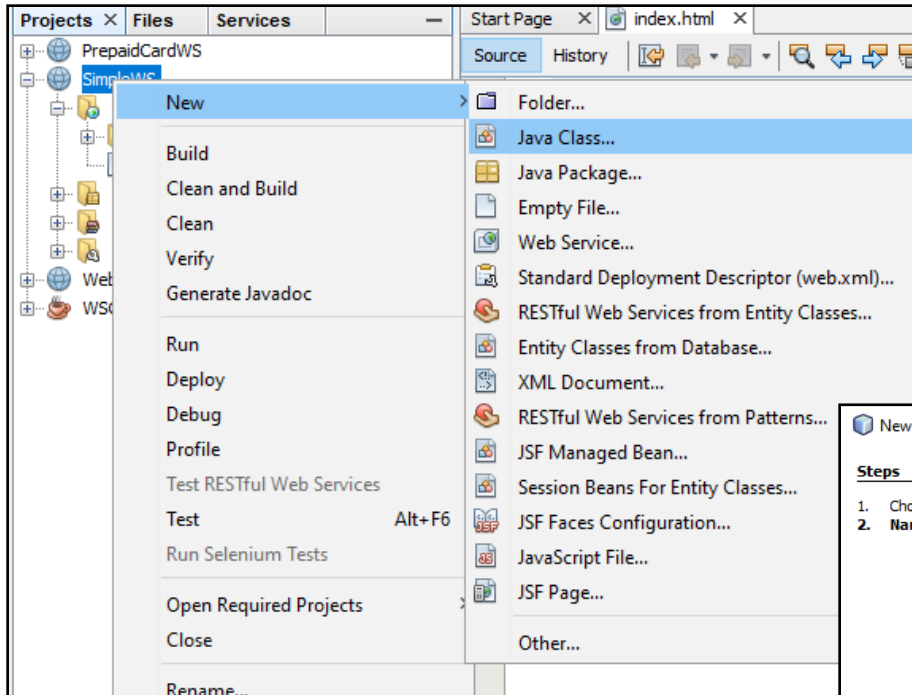
Server:

Java EE Version:

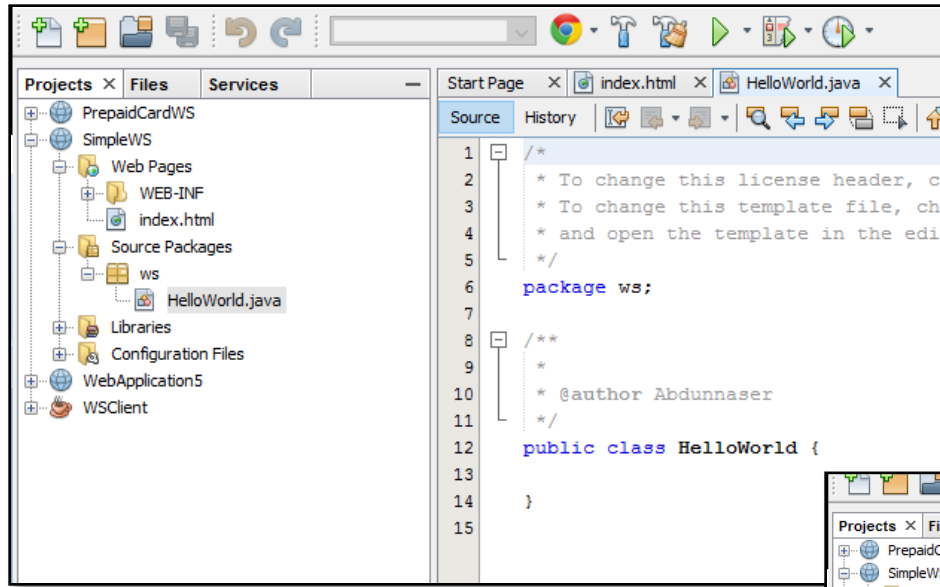
Context Path:

< Back Next > **Finish** Cancel Help

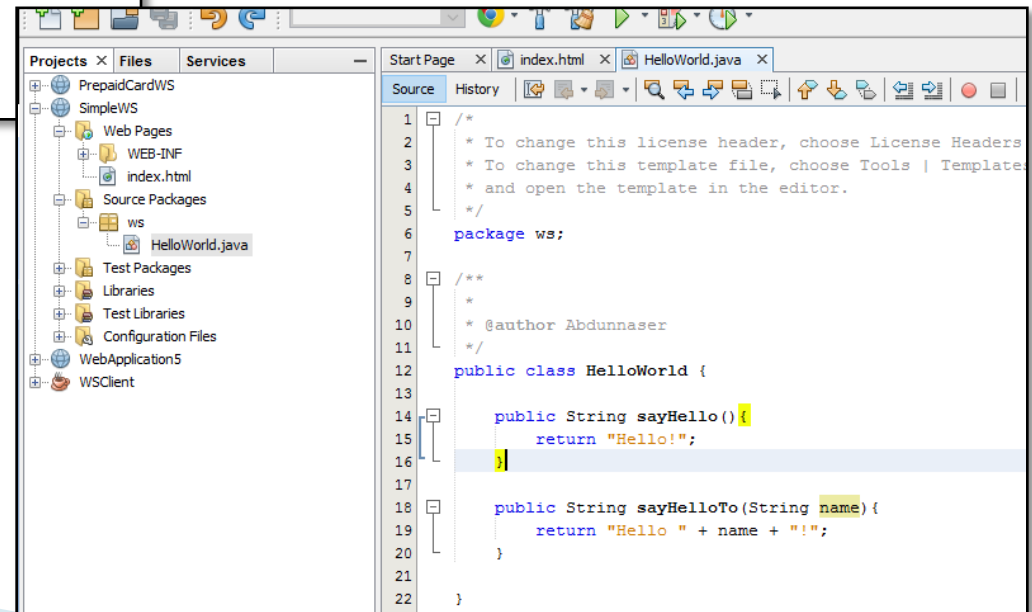
3. Create a JAVA class and some methods



3. Create a JAVA class and some methods (cont.)

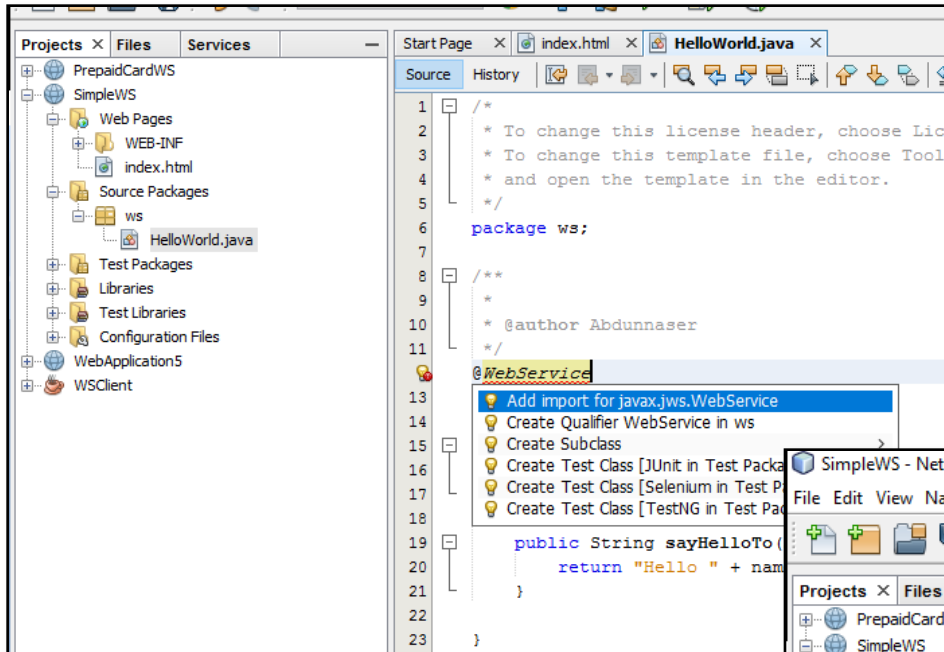


```
1  /*
2  * To change this license header, choose License Headers
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package ws;
7
8  /**
9  *
10 * @author Abdunnaser
11 */
12 public class HelloWorld {
13
14 }
15
```



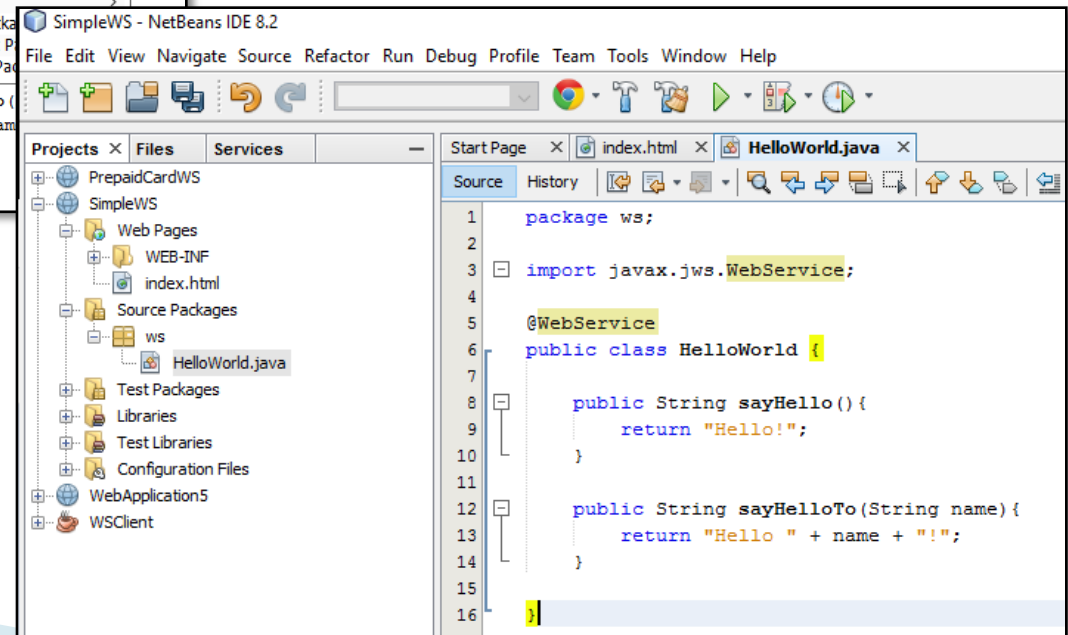
```
1  /*
2  * To change this license header, choose License Headers
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package ws;
7
8  /**
9  *
10 * @author Abdunnaser
11 */
12 public class HelloWorld {
13
14     public String sayHello() {
15         return "Hello!";
16     }
17
18     public String sayHelloTo(String name) {
19         return "Hello " + name + "!";
20     }
21
22 }
```

4. Convert the class to a WS by adding @WebService annotation



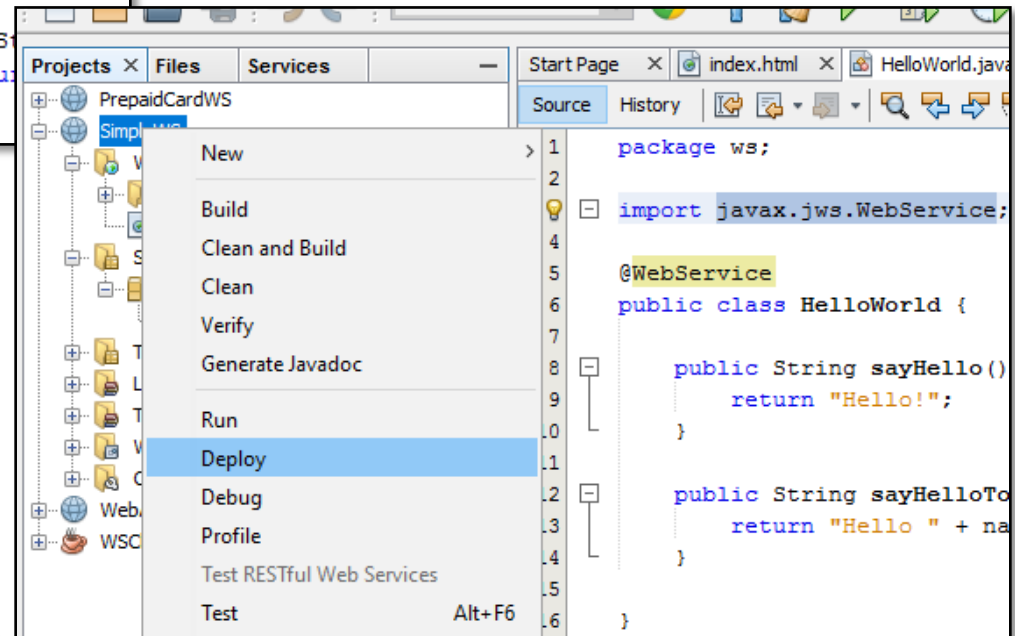
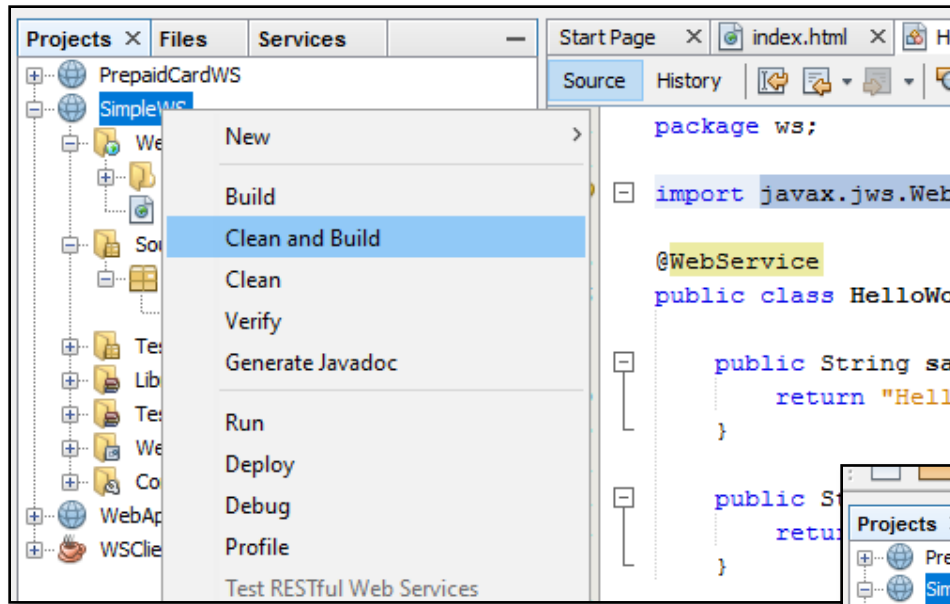
```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates | the JSP template under
4  * src/main/webapp/WEB-INF and open the template in the editor.
5  */
6  package ws;
7
8  /**
9   *
10  * @author Abdunnaser
11  */
12  @WebService
13
14
15
16
17
18
19  public String sayHelloTo(
20      return "Hello " + name
21  }
22
23  }
```

Do not forget to import
"javax.jws.WebService"



```
1  package ws;
2
3  import javax.jws.WebService;
4
5  @WebService
6  public class HelloWorld {
7
8      public String sayHello() {
9          return "Hello!";
10     }
11
12     public String sayHelloTo(String name) {
13         return "Hello " + name + "!";
14     }
15
16 }
```

5. Build and deploy the project



6. Test the WS and view the generated WSDL document via GlassFish Console

The image shows a workflow for testing a web service and viewing its WSDL document. On the left, an IDE window displays a Java source file named `HelloWorld.java` with the following code:

```
1 package ws;  
2  
import javax.ws.rs.WebService;  
  
@WebService  
public class HelloWorld {  
  
    public String sayHello()  
        return "Hello!";  
}  
  
public String sayHello()  
    return "Hello ";
```

A black arrow points from the IDE to the GlassFish Console on the right. The console is titled "GlassFish Console - Common Tasks" and shows the following structure:

- 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
 - Resource Adapter Configs
 - Configurations
 - default-config
 - server-config
 - Update Tool

The main content area of the console displays various management tasks:

- GlassFish News**: GlassFish News
- Deployment**: List Deployed Applications, Deploy an Application
- Administration**: Change Administrator Password, List Password Aliases
- Monitoring**: Monitoring Data
- Documentation**: Open Source Edition Documentation Set, Quick Start Guide, Administration Guide, Application Development Guide, Application Deployment Guide
- Update Center**: Installed Components, Available Updates, Available Add-Ons
- Resources**: Create New JDBC Resource, Create New JDBC Connection Pool

6. Test WS & view WSDL via GlassFish Console

Console (cont.)

GlassFish Console - Common Tasks

GlassFish News

GlassFish News

Deployment

List Deployed Applications

Deploy an Application

Administration

Change Administrator Password

List Password Aliases

Monitoring

Monitoring Data

1. Collapse “Application”
2. Click on your project name “SimpleWS”
3. Scroll down until you see the “Modules and Components” table
4. You’ll see your WS name “HelloWorld”

server

Associates an Internet domain name with a physical server.

Context Root: /SimpleWS
Path relative to server's base URL.

Implicit CDI Enabled
Implicit discovery of CDI beans

Location: file:/D:/MyJavaProjects/SimpleWS/build/web/

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:

Modules and Components (4)

Module Name	Engines	Component Name	Type	Action
SimpleWS	[web, webservices]	-----	-----	Launch
SimpleWS		default	Servlet	
SimpleWS		jsp	Servlet	
SimpleWS		HelloWorld	Servlet	View Endpoint

6. Test WS & view WSDL via GlassFish Console

Console (cont.)

Context Root: /SimpleWS
Implicit CDI: Enabled
Location: file:/D:/MyJavaProjects/SimpleWS/build/web/
Deployment Order: 100

Module Name	Engines	Component Name	Type	Action
SimpleWS	[web, webservices]	-----	-----	Launch
SimpleWS		default	Servlet	
SimpleWS		jsp	Servlet	
SimpleWS		HelloWorld	Servlet	View Endpoint

1. Click on "View Endpoint"
2. Test WS operations by clicking on: </SimpleWS/HelloWorldService?Tester>



User: admin Role: domain1 Server: localhost
GlassFish™ Server Open Source Edition
Total # of available updates : 45

Tree: Domain > server (Admin Server) > Clusters > Standalone Instances > Nodes > Applications > SimpleWS

Web Service Endpoint Information

View details about a web service endpoint.

Application Name: SimpleWS
Tester: /SimpleWS/HelloWorldService?Tester
WSDL: /SimpleWS/HelloWorldService?wsdl
Endpoint Name: HelloWorld
Service Name: HelloWorldService
Port Name: HelloWorldPort
Deployment Type: 109
Implementation Type: SERVLET
Implementation Class Name: ws.HelloWorld
Endpoint Address URI: /SimpleWS/HelloWorldService
Namespace: http://ws/

6. Test WS & view WSDL via GlassFish Console

Console (cont.)

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 tester screen

Application Name: SimpleWS

Links:

- [server] <http://Nasser-Acer-LT:8080/SimpleWS/HelloWorldService?Tester>
- [server] <https://Nasser-Acer-LT:8181/SimpleWS/HelloWorldService?Tester>

You can see and test your WS operation "sayHello" and "sayHelloTo"

HelloWorldService 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 java.lang.String ws.HelloWorld.sayHello()
sayHello ()

public abstract java.lang.String ws.HelloWorld.sayHelloTo(java.lang.String)
sayHelloTo ()

6. Test WS & view WSDL via GlassFish Console

Console (cont.)

Web Service Endpoint In x Method invocation trace x

nasser-acer-It:8080/SimpleWS/HelloWorldService?Tester

sayHello Method invocation

Method parameter(s)

Type	Value
------	-------

Method returned

java.lang.String : "Hello!"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Header/><S:Body><ns2:sayHello xmlns:ns2="http://ws/"/></S:Body></S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Header/>
```

A red arrow points to the returned value "Hello!".

After clicking on "sayHello" and "sayHelloTo", respectively.

nasser-acer-It:8080/SimpleWS/HelloWorldService?Tester

sayHelloTo Method invocation

Method parameter(s)

Type	Value
java.lang.String	Nasser

Method returned

java.lang.String : "Hello Nasser!"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Header/><S:Body><ns2:sayHelloTo xmlns:ns2="http://ws/"><arg0>Nasser</arg0></ns2:sayHelloTo></S:Body></S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Header/>
```

A red arrow points to the parameter value "Nasser".

A red arrow points to the returned value "Hello Nasser!".

6. Test WS & view WSDL via GlassFish Console (cont.)

Web Service Endpoint In... x HelloWorldService Web S... x

← → ↻ 🏠 ⓘ nasser-acer-lt:8080/SimpleWS/HelloWorldService?Tester

HelloWorldService Web Service Tester

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

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

Methods :

public abstract java.lang.String ws.HelloWorld.sayHello()
 ()

public abstract java.lang.String ws.HelloWorld.sayHelloTo(java.lang.String)

You can view the generated WSDL document by clicking on “WSDL File”

Web Service Endpoint In... x nasser-acer-lt:8080/Simp... x

← → ↻ 🏠 ⓘ nasser-acer-lt:8080/SimpleWS/HelloWorldService?WSDL

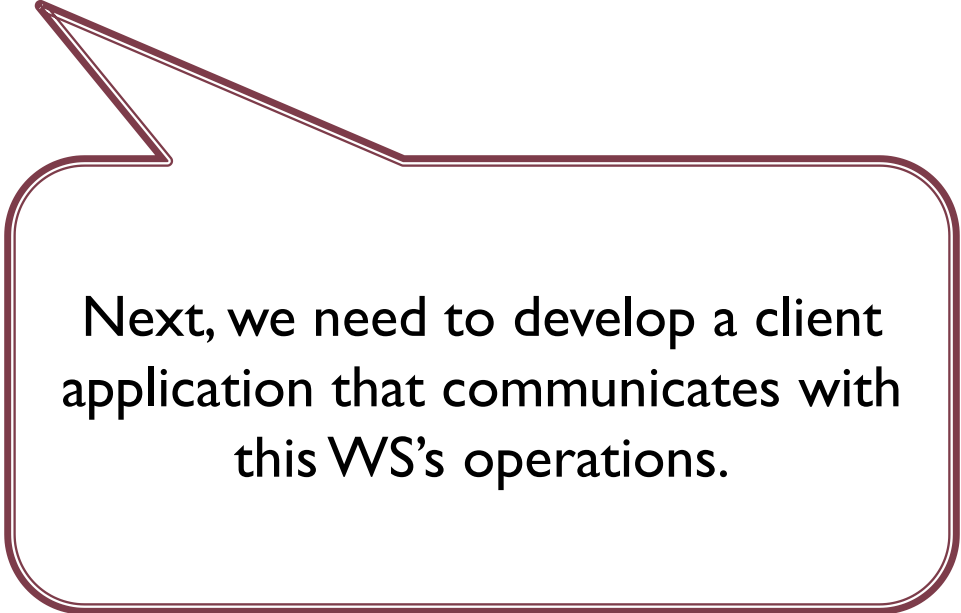
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!--
  Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.1-b419 (branches/2.3.1.x-7937; 2014-08-04T08:11:03+0000) J
-->
<!--
  Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.1-b419 (branches/2.3.1.x-7937; 2014-08-04T08:11:03+0000) J
-->
<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://ws/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://ws/"
  name="HelloWorldService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://ws/" schemaLocation="http://nasser-acer-lt:8080/SimpleWS/HelloWorldService?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="sayHello">
    <part name="parameters" element="tns:sayHello"/>
  </message>
  <message name="sayHelloResponse">
    <part name="parameters" element="tns:sayHelloResponse"/>
  </message>
  <message name="sayHelloTo">
    <part name="parameters" element="tns:sayHelloTo"/>
  </message>
  <message name="sayHelloToResponse">
    <part name="parameters" element="tns:sayHelloToResponse"/>
  </message>
  <portType name="HelloWorld">
    <operation name="sayHello">
      <input wsam:Action="http://ws/HelloWorld/sayHelloRequest" message="tns:sayHello"/>
      <output wsam:Action="http://ws/HelloWorld/sayHelloResponse" message="tns:sayHelloResponse"/>
    </operation>
    <operation name="sayHelloTo">
      <input wsam:Action="http://ws/HelloWorld/sayHelloToRequest" message="tns:sayHelloTo"/>
      <output wsam:Action="http://ws/HelloWorld/sayHelloToResponse" message="tns:sayHelloToResponse"/>
    </operation>
  </portType>
</definitions>
```

Other things, covered

- ▶ `@WebService(serviceName=“...”)`
- ▶ `@WebMethod`
- ▶ `@WebMethod(operationName=“...”)`

The WS is done and ready



Next, we need to develop a client application that communicates with this WS's operations.