ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στα Δίκτυα Υπηρεσιών

Assisting Lecture 8 - Bottom up SOAP Web Services

Μύρων Παπαδάκης
Τμήμα Επιστήμης Υπολογιστών

CS-592: Introduction Service Networks 2015

Assisting Lecture: SOAP Web Services and Bottom-up Examples

Myron Papadakis (myrpap@gmail.com)

Outline

- Web Services Components
- SOAP
- WSDL
- WSDL Invocation Tools
- Netbeans IDE: JAX-WS Web Services (Bottom-up)

Web services (1/3)

- **Web Service (in short) is a resource identified by URI and returns XML.**
  - Software program that uses XML to exchange information with other software via common Internet protocols.
  - It has an interface described in a machine-processable format (specifically WSDL).
  - Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.
- ***[W3C Definition] "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."***

Web services (2/3)

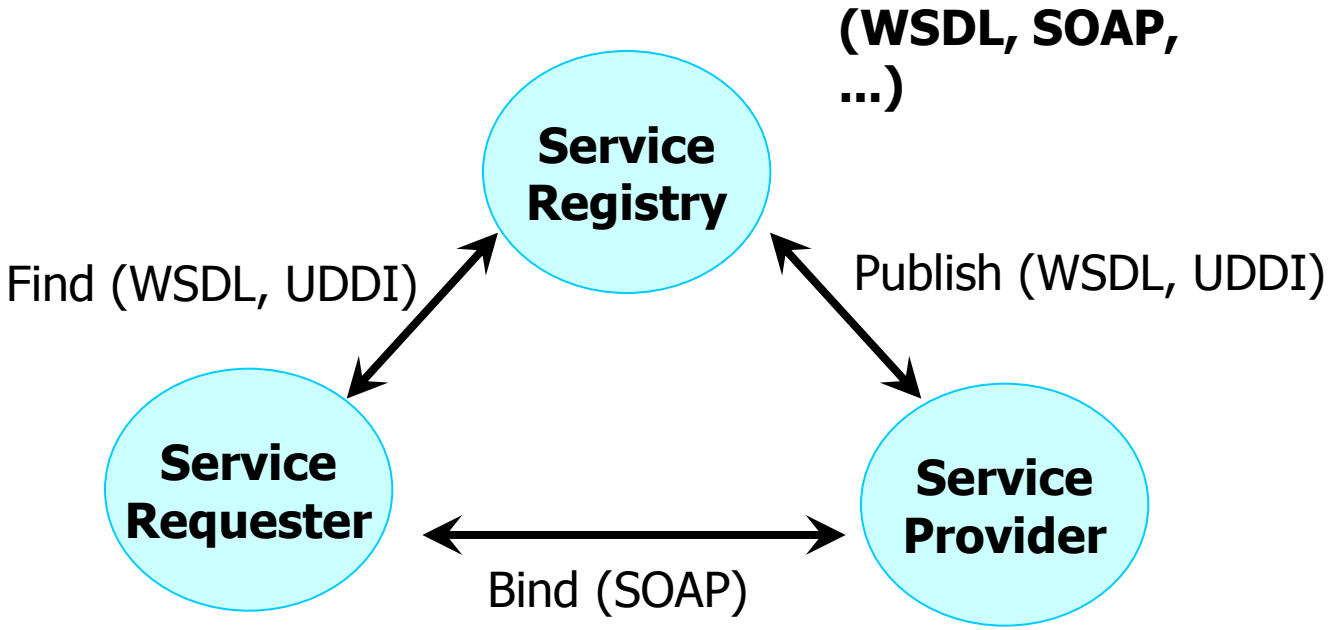
- Unlike components, web services do not have to be downloaded and deployed so as to be used by clients.

- **Web services basic technologies:**
  - SOAP (Simple Object Access Protocol)
  - WSDL (Web Services Description Language)
  - UDDI (Universal Description, Discovery and Integration)

Web services (3/3)

- **What is SOAP?**
  - SOAP is an XML-based protocol to let applications exchange information (send and receive messages) over HTTP.
    - Why HTTP?
      - Supported by every web browser and server
      - Effective technology for transferring text, graphics and other information
  - A SOAP message is a kind of XML document
  - SOAP has its own schema, namespaces and processing rules
- **What is WSDL** (Web Services Description Language) **?**
  - WSDL is an *XML-based language for locating and describing Web services*.
  - It is actually document written in XML, which describes a Web service.
  - It specifies the location of the service and the operations (or methods) the service exposes.
- **What is UDDI?**
  - UDDI (Universal Description, Discovery and Integration) is a directory (of web service interfaces described by WSDL) where companies can register and search for Web services

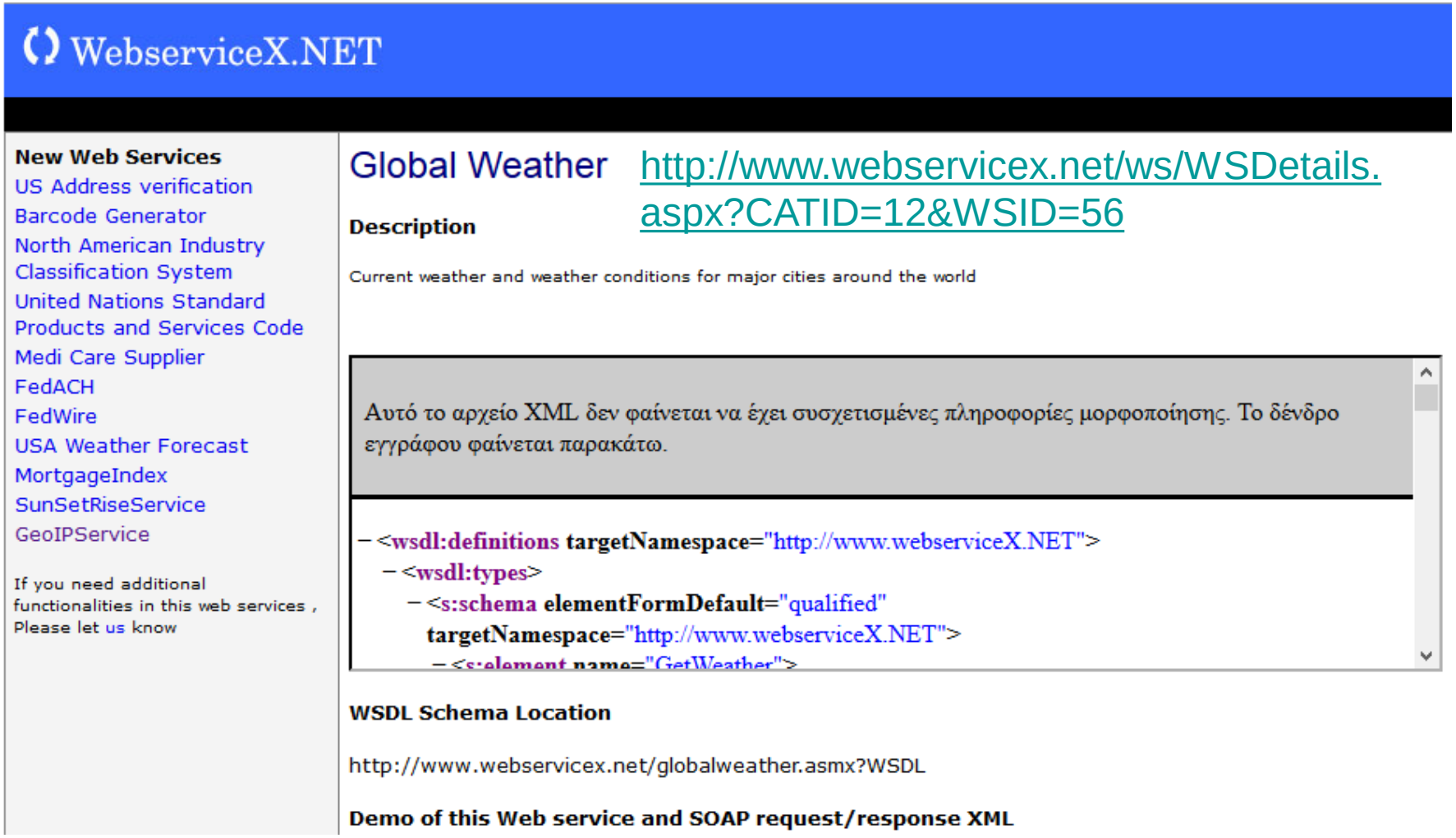
Web Services Model



Classic student's mood when talking about WS

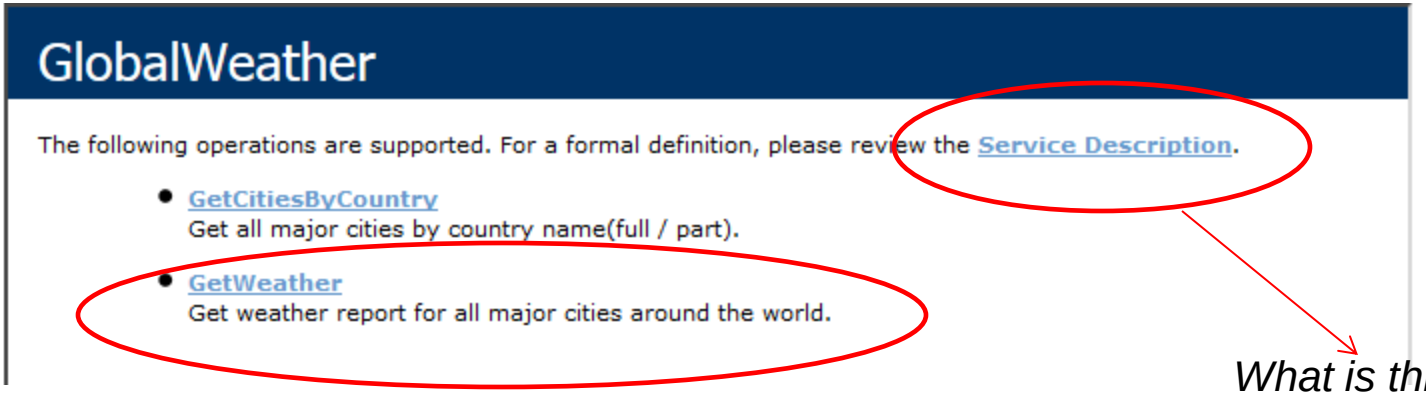
Not this time a classic theoretical WS lecture (hopefully)

Weather in Heraklion Today?



Weather in Heraklion (Greece) Today?

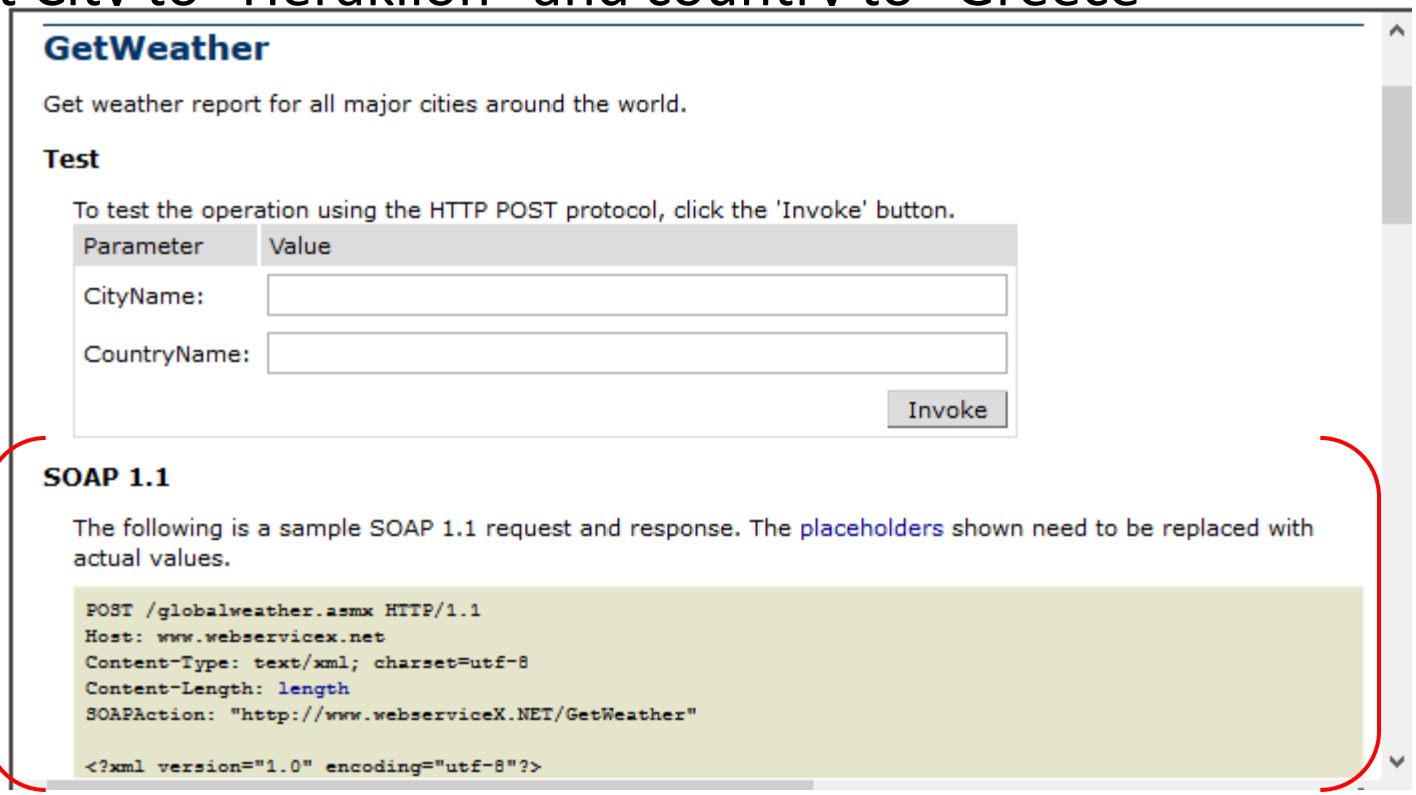
- http://www.webservicex.net/ws/WSDetails.aspx?CATID=12&WSID=56



What is this? (More on this later)…

Weather in Heraklion Today?

- Scroll down a bit
- Set City to “Heraklion” and country to “Greece”



→ *To be explained later…*

Weather in Heraklion Today?

- Invoke the Web Service
- A new tab in your browser will open and you will get the result (it is xml…, but this is not always the case)

```
− <string>
    <?xml version="1.0" encoding="utf-16"?> <CurrentWeather> <Location>Heraklion
    Airport , Greece (LGIR) 35-20N 025-11E 39M</Location> <Time>Feb 15, 2015 - 02:20
    PM EST / 2015.02.15 1920 UTC</Time> <Wind> from the SW (220 degrees) at 6 MPH
    (5 KT):0</Wind> <Visibility> greater than 7 mile(s):0</Visibility> <SkyConditions>
    mostly cloudy</SkyConditions> <Temperature> 51 F (11 C)</Temperature> <DewPoint>
    41 F (5 C)</DewPoint> <RelativeHumidity> 66%</RelativeHumidity> <Pressure> 30.06
    in. Hg (1018 hPa)</Pressure> <Status>Success</Status> </CurrentWeather>
  </string>
```

Using http (GET)

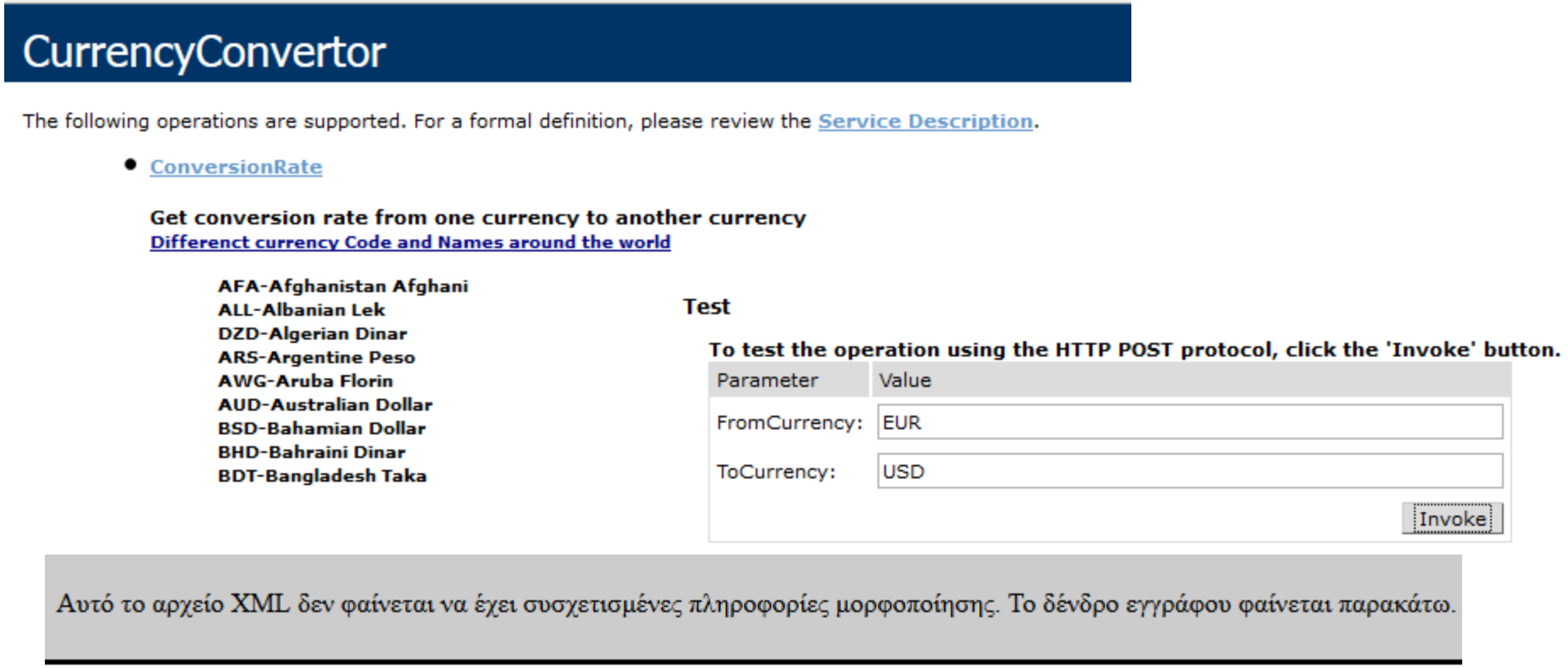
- http://www.webservicex.net/globalweather.asmx/GetWeather?CityName=Heraklion&CountryName=Greece

---

<string>
<?xml version="1.0" encoding="utf-16"?> <CurrentWeather> <Location>Heraklion Airport , Greece (LGIR) 35-20N 025-11E 39M</Location> <Time>Feb 15, 2015 - 02:20 PM EST / 2015.02.15 1920 UTC</Time> <Wind> from the SW (220 degrees) at 6 MPH (5 KT):0</Wind> <Visibility> greater than 7 mile(s):0</Visibility> <SkyConditions> mostly cloudy</SkyConditions> <Temperature> 51 F (11 C)</Temperature> <DewPoint> 41 F (5 C)</DewPoint> <RelativeHumidity> 66%</RelativeHumidity> <Pressure> 30.06 in. Hg (1018 hPa)</Pressure> <Status>Success</Status> </CurrentWeather>
</string>

Web Service Example: Temperature Conversion

- http://www.webservicex.net/currencyconvertor.asmx



Web Service Example: Temperature Conversion

- Try for example this one:
  - http://www.w3schools.com/webservices/tempconvert.asmx?op=CelsiusToFahrenheit

- There are a lot of available web services online…
  - However, a lot of old WS are now down…
  - http://www.webservicex.net/ws/default.aspx is usually up-to-date

- So far we have just used a browser for sending a request and getting a response from a Web Service
  - **We really do not know the structure of the information/messages that is being exchanged, the technologies used,etc,etc**…

Outline

- Web Services Components
- SOAP
- WSDL
- WSDL Invocation Tools
- Netbeans IDE: JAX-WS Web Services

SOAP

- Simple, lightweight **protocol** for exchanging XML messages over the Web using HTTP, SMTP, etc. (in a decentralized, distributed environment).
  - **The most common way to exchange SOAP messages is via HTTP**

- It is a way for a program running in one kind of operating system (such as Windows 2000) to communicate with a program in the same or another kind of an operating system (such as Linux) by using the HTTP and XML as the mechanisms for information exchange.

- SOAP has a clear purpose: exchanging data over networks.

- Specifically, it concerns itself with **encapsulating and encoding XML data and defining the rules for transmitting and receiving that data**.

- In short: **HTTP + XML = SOAP** (SOAP is based on existing technology)

SOAP

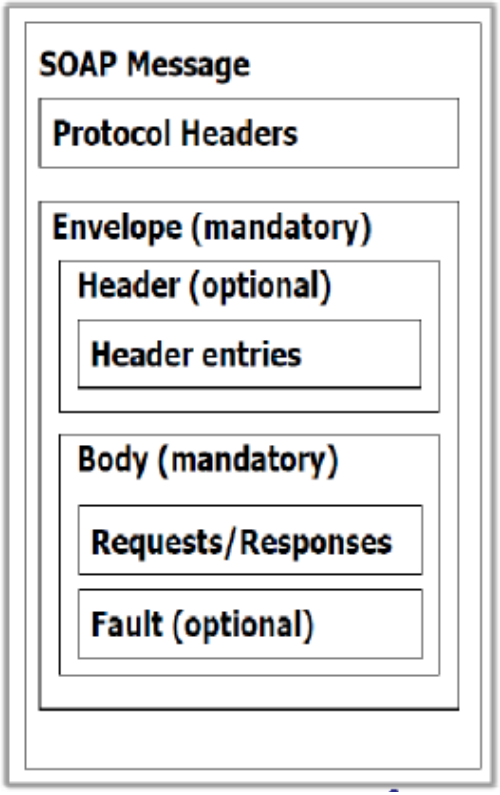
- SOAP is just another XML markup language accompanied by rules that dictate its use.

- SOAP is not tied to any particular transport protocol (though HTTP is popular).

- **Platform independent**: not tied to any particular operating system or programming language:
  - clients and servers in these dialogues can be running on any platform and written in any language as long as they can formulate and understand SOAP messages.

SOAP Messages

- Messages are structured within an envelope where the application encloses the data to be sent
- Analogous to an envelope used in traditional postal service (contains XML data instead of a paper envelope…)
- SOAP Elements
  - Envelope: top XML element representing the message
  - Header: (optional), contains application-specific information (like authentication, routing instructions, etc) about the SOAP message
  - Body: (mandatory): contains the actual SOAP message call and response information
    - Fault (optional): used to indicate error messages
- A SOAP conversation includes:
  - SOAP Request: Specify method name, method parameters
  - SOAP Response: Specify return values or error conditions



Skeleton Soap Message

- The **namespace** defines the Envelope as a SOAP Envelope.
- SOAP messages are written entirely in XML

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

SOAP HTTP Binding

- A SOAP method is an HTTP request/response that complies with the SOAP encoding rules.
- **HTTP + XML = SOAP**
- A SOAP request could be an HTTP POST or an HTTP GET request.
- The HTTP POST request specifies at least two HTTP headers: Content-Type and Content-Length

HTTP Post

- If a user needs to
  - submit large amount of data to a Web server, or
  - the user doesn't like the user data to appear as a query string on the URL, or
  - the user needs to download some dynamic information from a Web server

then HTTP POST should be used to access the Web server.

SOAP Example

- A GetStockPrice request is sent to a server

- The request has a StockName parameter, and a Price parameter that will be returned in the response

Soap request

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

Soap response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Wait

- Did you notice any SOAP messages when we invoked the Web Service?
- We just filled in the values (Heraklion, Greece) and invoked the WS through a browser
  - we are clients (WS consumers), no need to know everything...
- We can actually verify it using either a WSDL invocation tool, or a hand-written client (on next slides)

Outline

- Web Services Components
- SOAP
- WSDL
- WSDL Invocation Tools
- Netbeans IDE: JAX-WS Web Services

WSDL

- WSDL is an XML-based (interface description) language used to describe Web services by specifying:
  - the **location of the service**,
  - the **operations** exposed and
  - the **message data types** (functionality of a WS).
- A WSDL service description (also referred to as a WSDL file)
  - indicates how potential clients are intended to interact with the described service.
  - provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns.
  - It thus serves a roughly similar purpose as a method signature in a programming language.
- **A complete WSDL definition contains all of the information necessary to invoke a Web service**
- A service producer implements a service according to the WSDL definition.
- A service consumer implements a client according to the WSDL definition.
- WSDL is often used in combination with SOAP and an XML Schema to provide Web services over the Internet.

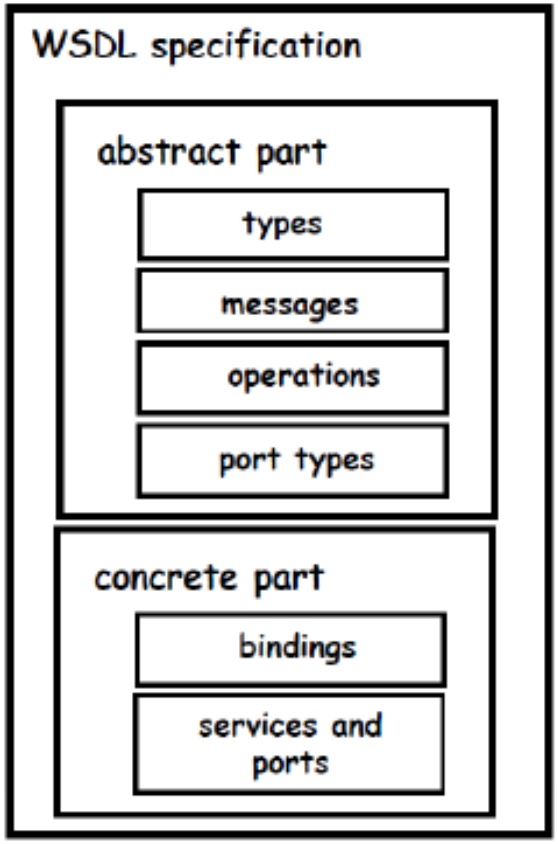
Remember the Weather Example?

WSDL Part

http://www.webservicex.net/globalweather.asmx?WSDL

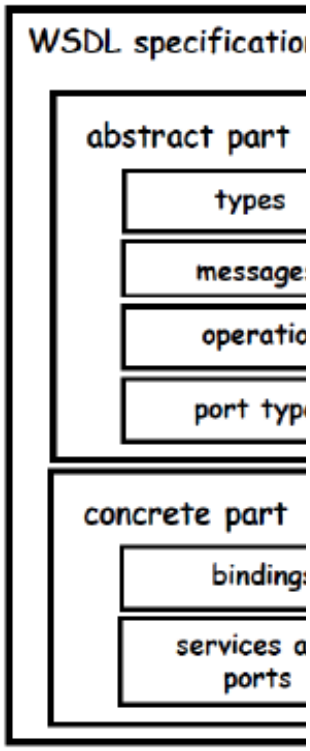
```
- <wsdl:definitions targetNamespace="http://www.webserviceX.NET">
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET">
      - <s:element name="GetWeather">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="CityName" type="s:string"/>
              <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="GetWeatherResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="GetWeatherResult" type="s:string"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="GetCitiesByCountry">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="CountryName" type="s:string"/>
```

WSDL Essentials



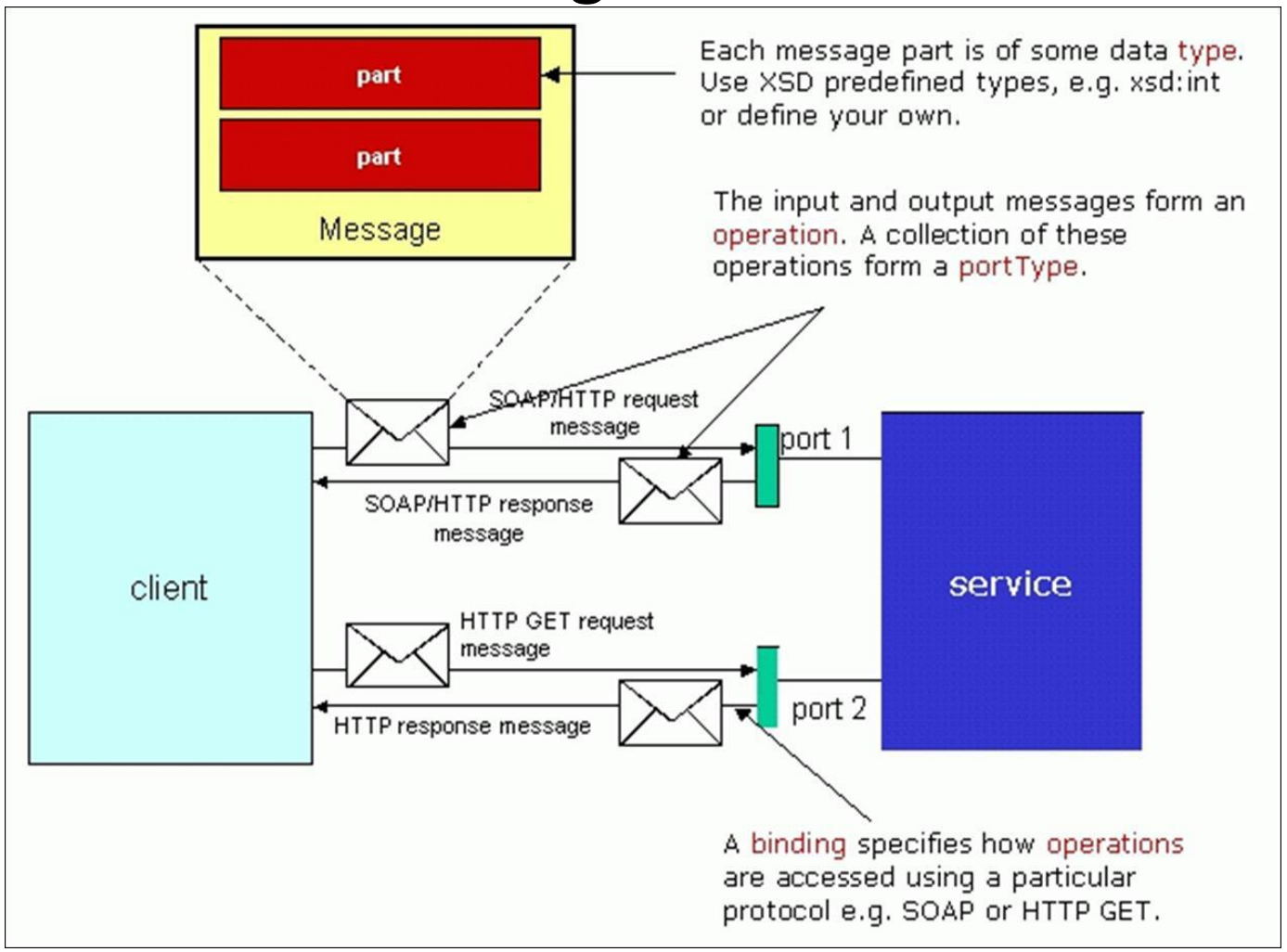
- **(XML) Data Elements for V1.1**
- **<wsdl:definitions>**: root element, declaration of namespaces, targetNamespace, name of the WS (Description in v1.2)
- **<wsdl:types>**: datatypes used by the Web service (XML Schema)
- **<wsdl:messages>:** messages used by the Web service, data eleements of an operation. Each message can consist of one or more parts (parts compared to the parameters of a function call). Parts that have can be simple types (primitives) or more complicated types (defined in a schema)
- **<wsdl:portType>**: operations performed by the Web service (interface in v1.2), messages involved. Like a function library (class, module in typical programming languages)
- **<wsdl:binding>**: how the operations are invoked, communication protocols used by the web service, protocol details, message format
- **<wsdl:service>**: address for invoking the service

WSDL Essentials



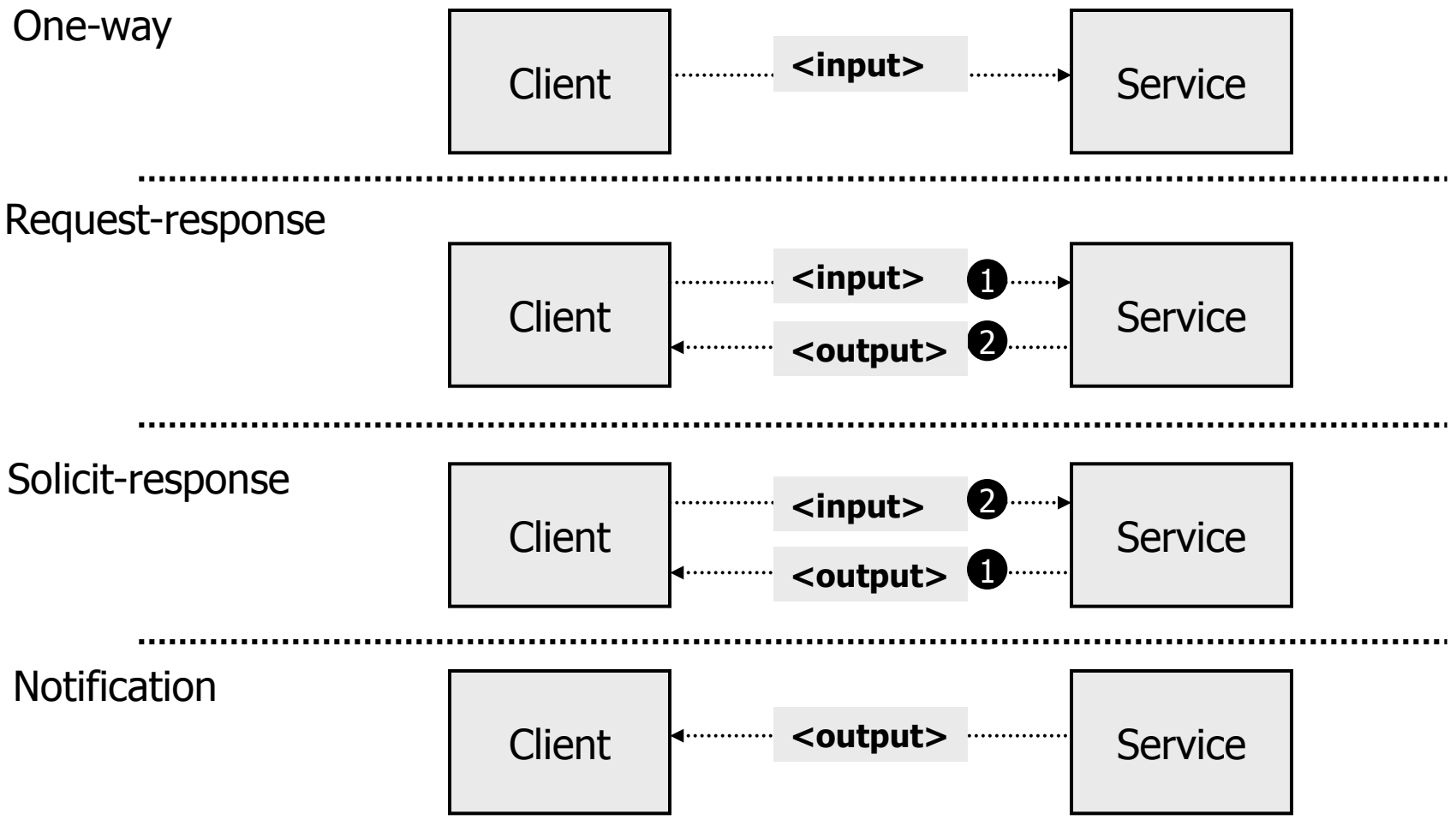

```
− <wsdl:definitions targetNamespace="http://www.webserviceX.NET">
  − <wsdl:types>
    + <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET"></s:schema>
    </wsdl:types>
  + <wsdl:message name="GetWeatherSoapIn"></wsdl:message>
  + <wsdl:message name="GetWeatherSoapOut"></wsdl:message>
  + <wsdl:message name="GetCitiesByCountrySoapIn"></wsdl:message>
  + <wsdl:message name="GetCitiesByCountrySoapOut"></wsdl:message>
  + <wsdl:message name="GetWeatherHttpGetIn"></wsdl:message>
  + <wsdl:message name="GetWeatherHttpGetOut"></wsdl:message>
  + <wsdl:message name="GetCitiesByCountryHttpGetIn"></wsdl:message>
  + <wsdl:message name="GetCitiesByCountryHttpGetOut"></wsdl:message>
  + <wsdl:message name="GetWeatherHttpPostIn"></wsdl:message>
  + <wsdl:message name="GetWeatherHttpPostOut"></wsdl:message>
  + <wsdl:message name="GetCitiesByCountryHttpPostIn"></wsdl:message>
  + <wsdl:message name="GetCitiesByCountryHttpPostOut"></wsdl:message>
  + <wsdl:portType name="GlobalWeatherSoap"></wsdl:portType>
  + <wsdl:portType name="GlobalWeatherHttpGet"></wsdl:portType>
  + <wsdl:portType name="GlobalWeatherHttpPost"></wsdl:portType>
  + <wsdl:binding name="GlobalWeatherSoap" type="tns:GlobalWeatherSoap"></wsdl:binding>
  + <wsdl:binding name="GlobalWeatherSoap12" type="tns:GlobalWeatherSoap"></wsdl:binding>
  + <wsdl:binding name="GlobalWeatherHttpGet" type="tns:GlobalWeatherHttpGet"></wsdl:binding>
  + <wsdl:binding name="GlobalWeatherHttpPost" type="tns:GlobalWeatherHttpPost"></wsdl:binding>
  + <wsdl:service name="GlobalWeather"></wsdl:service>
  </wsdl:definitions>
```

Working of WSDL



Source: http://acet.rdg.ac.uk/~mab/Education/Undergraduate/CS2K7/Lectures/

WSDL PortType



WSDL Port Types Examples

One-Way (1)

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

Request-Response Operation (2)

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

- **Each message can consist of one or more parts**.
  - The parts can be compared to the parameters of a function call
- The portType "glossaryTerms“ (1) defines a **one-way** operation called "setTerm".
- The portType "glossaryTerms" (2) defines a **request-response** operation called "getTerm".
- The "getTerm" operation requires an input message called "getTermRequest" with a parameter called "term", and will return an output message called "getTermResponse" with a parameter called "value".
- The input and output describe the structure of the messages

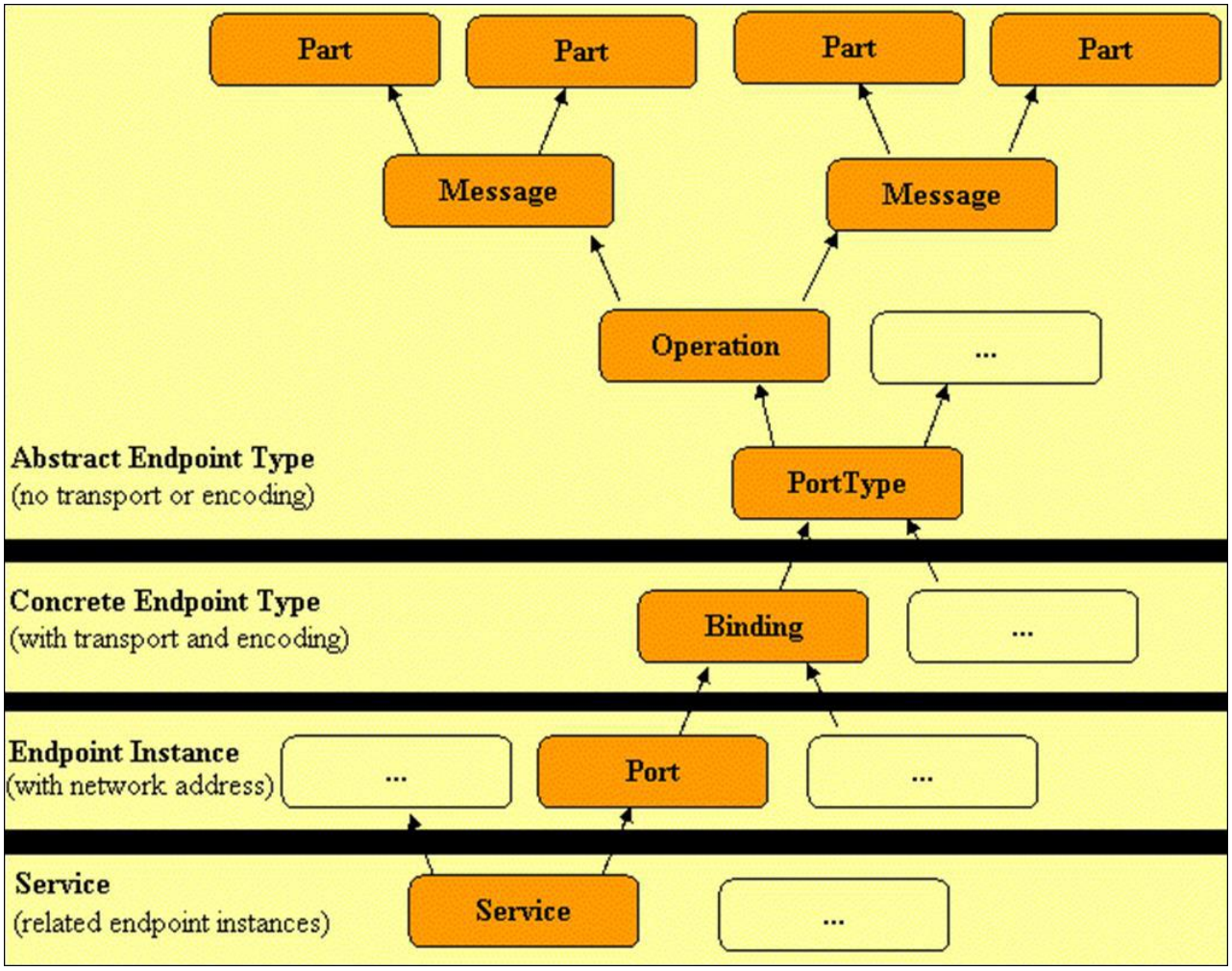
WSDL Binding

- WSDL Binding: How operations are accessed using a particular protocol e.g. HTTP, which portType it is describing

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>

<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

Transport attribute, indicates that SOAP messages should be send over HTTP

WSDL Diagram



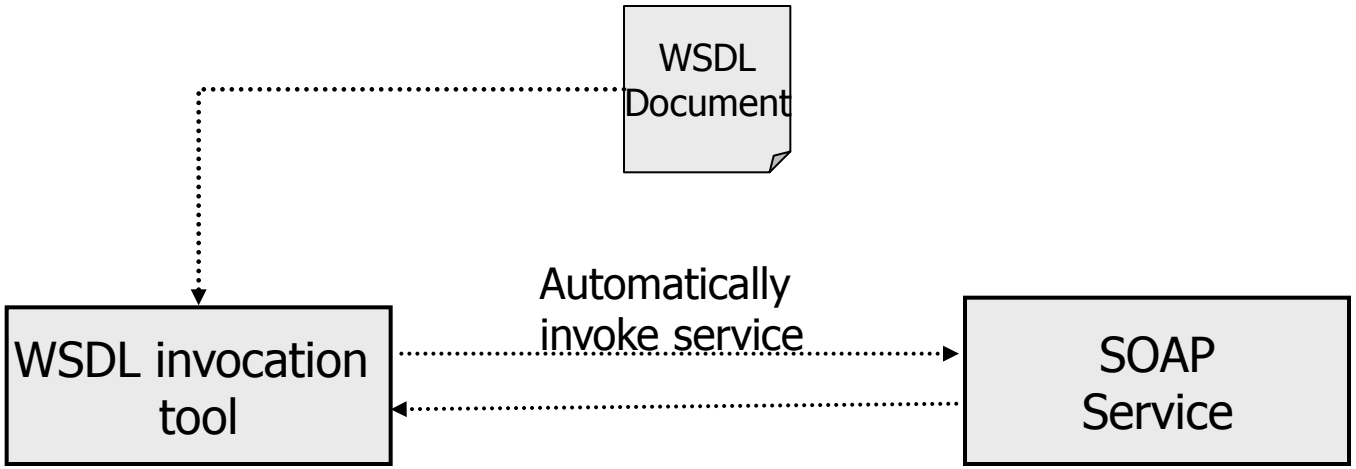
Source: http://msdn.microsoft.com/

Outline

- Web Services Components
- SOAP
- WSDL
- WSDL Invocation Tools
- Netbeans IDE: JAX-WS Web Services

WSDL Invocation Tools (1/2)

- Given a WSDL file one could manually create a SOAP client to invoke the service.
- A better alternative is to *automatically* invoke the service via a **WSDL invocation tool** (WSIF, SOAP::Lite, SOAPUI)
- SOAP UI comes with extensive support for testing WSDL / SOAP based services and it is easy to use.

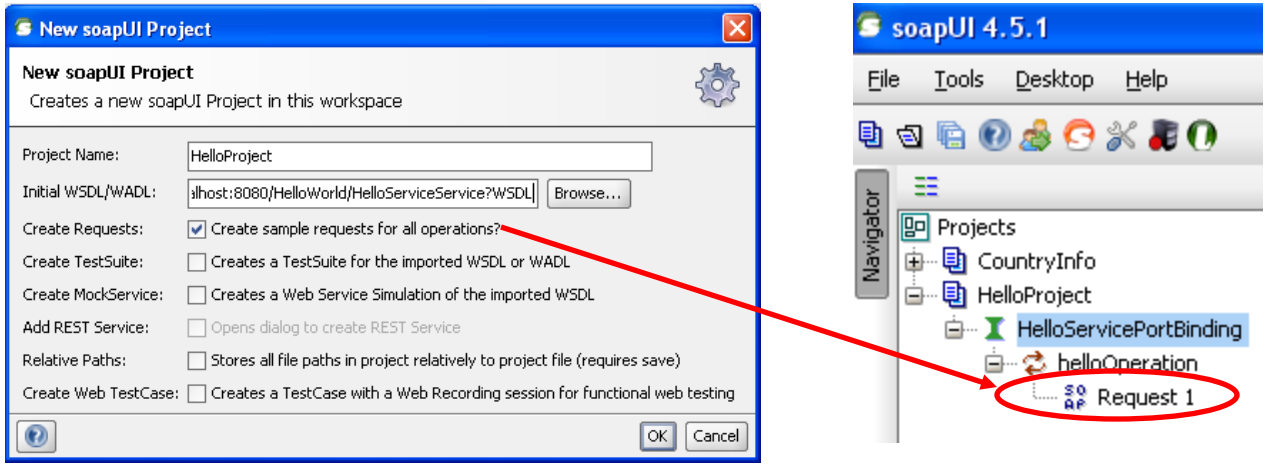


WSDL Invocation Tools (2/2)

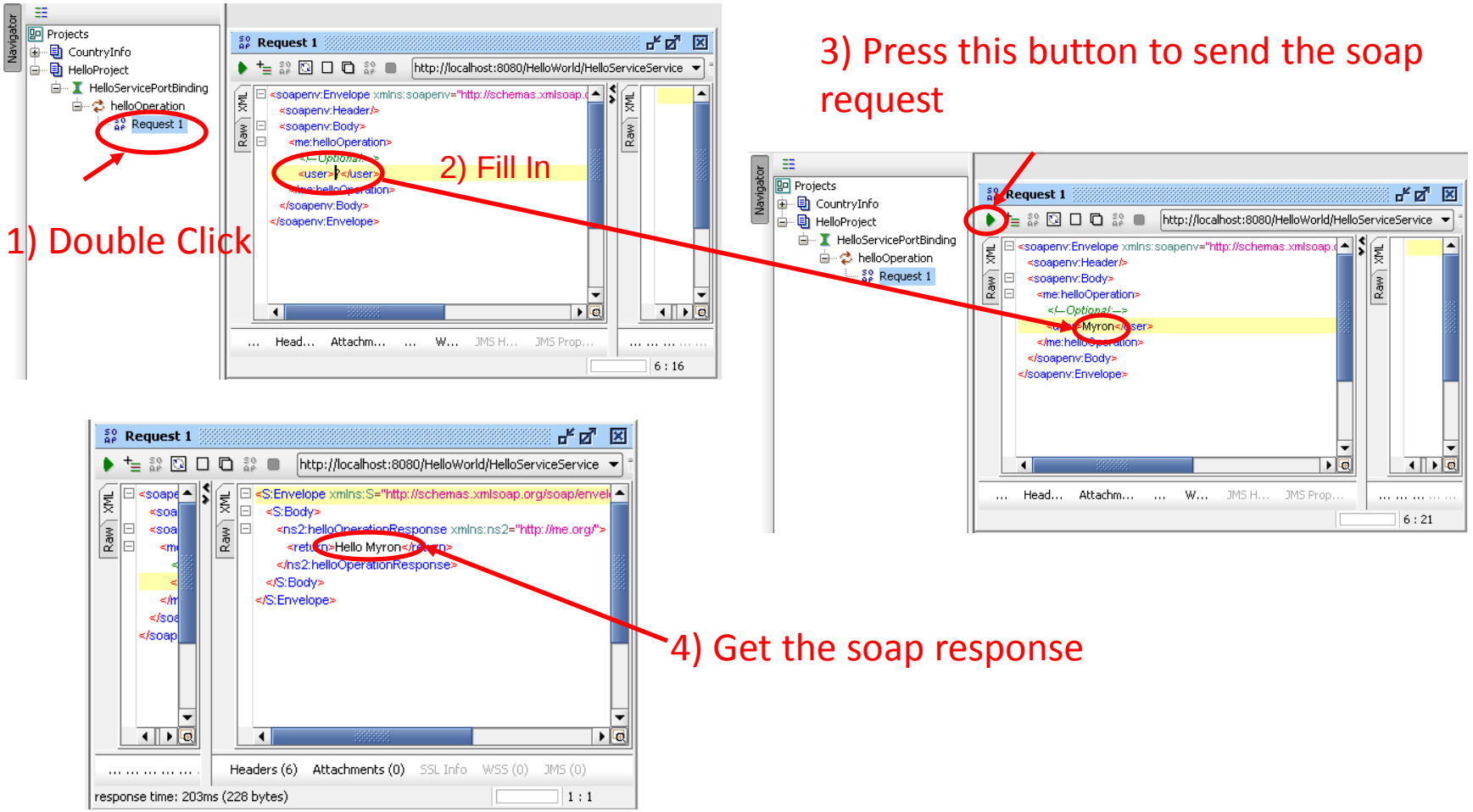
- Information flow when we use an invocation tool
  1. Enter URL for the WSDL file
  2. Get the overview of the web service
     - Description of the web service
     - Public operations
  3. Use of service
- Download and install SOAP UI
  - http://sourceforge.net/projects/soapui/files/

SOAPUI Example 1 (1/2)

- Assume a web service “HelloService” that has a single operation which takes as input a single string from a user (e.g “Myron”) and returns a greeting message to the user (e.g “Hello Myron”)
- Assume that the WSDL of this web service is located in the following url: http://localhost:8080/HelloWorld/HelloServiceService?WSDL
- Invoke this web service using the SOAPUI tool
  - Create a new project (*File > New soapUI Project),* then set the project’s name and the url of the WSDL and press ok

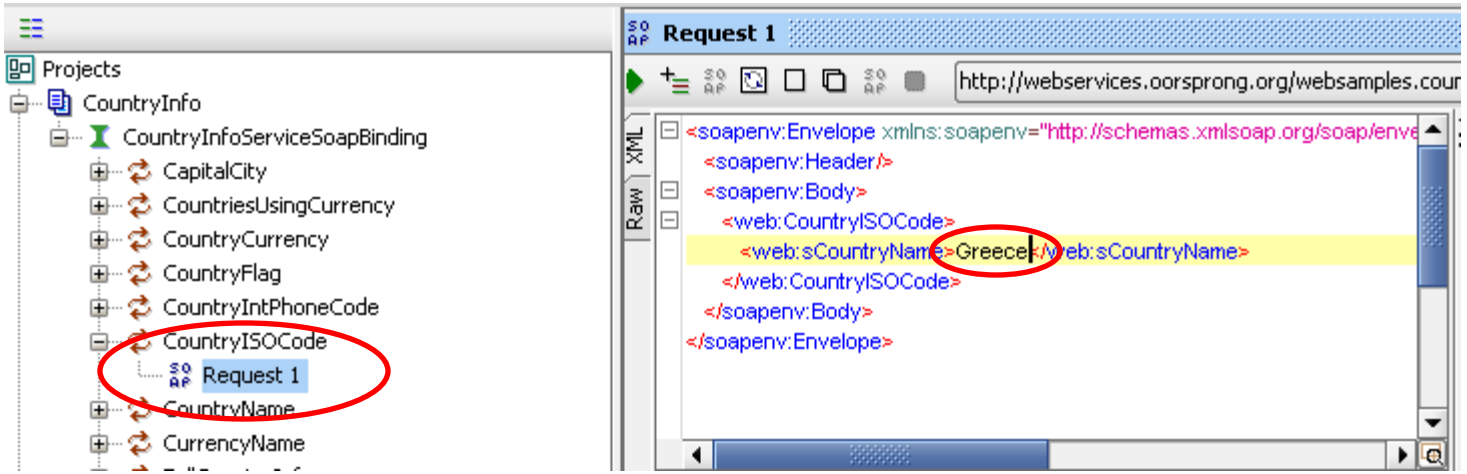


SOAPUI Example 1 (2/2)

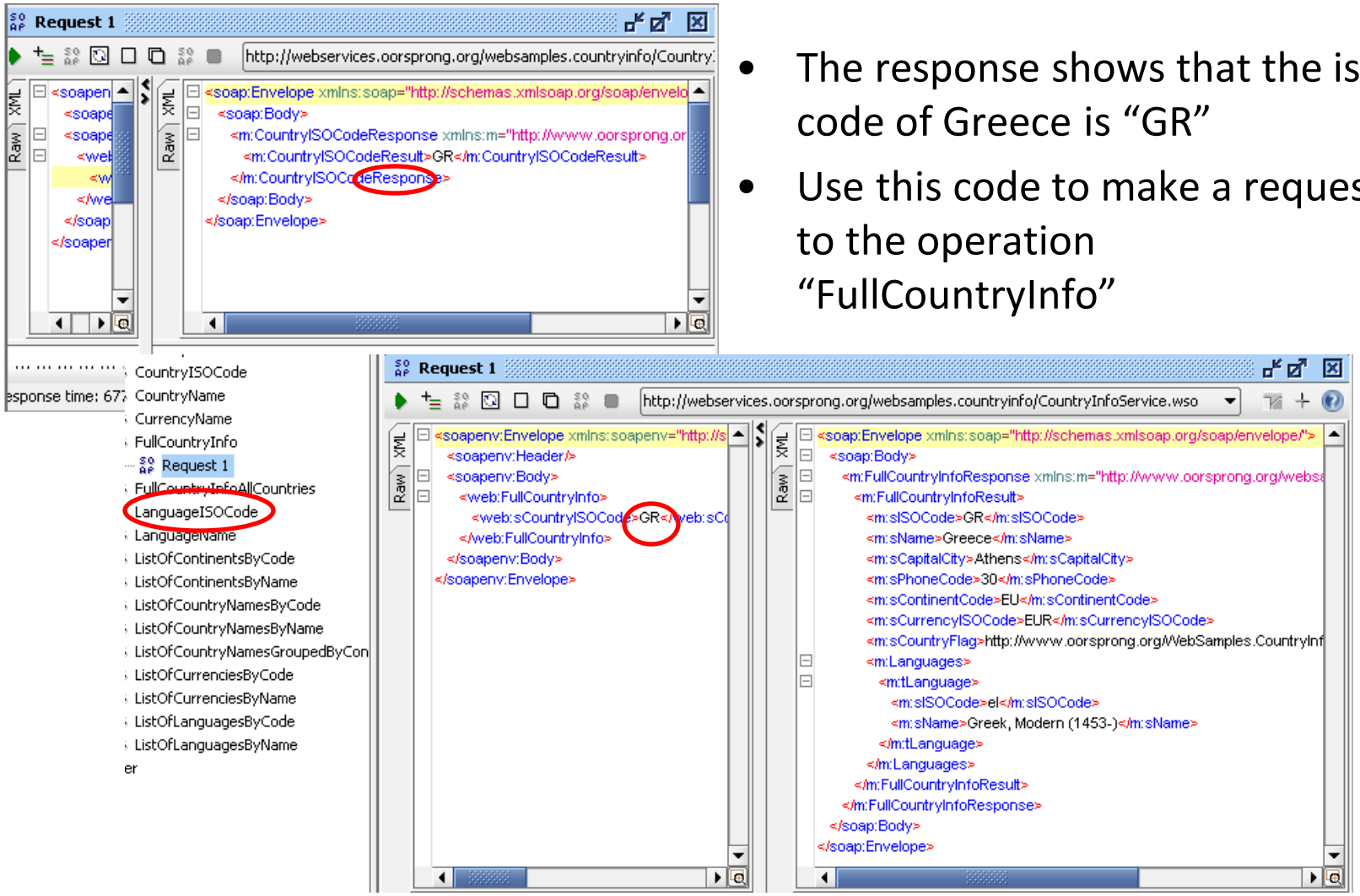


SOAPUI Example 2 (1/2)

- Create a new project “CountryInfo” & set the WSDL to the following url:
  - http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL
  - As you will see, this WS provides several operations for retrieving information about a particular country
  - Firstly find the iso code of “Greece” by sending a request (soap message)



SOAPUI Example 2 (2/2)



- The response shows that the iso code of Greece is “GR”
- Use this code to make a request to the operation “FullCountryInfo”

World Cup 2014 Example

- Do you remember at all the world cup football 2014?
- Who was the top scorer?

http://www.bbc.com/sport/football/world-cup/2014/top-scorers

Top Scorers

James Rodríguez

Colombia

67 Minutes per goal

400 Minutes played

6 Goals scored

2 Assists

Shots on target 89% 8 Total 9

Thomas Müller

Germany

136 Minutes per goal

682 Minutes played

5 Goals scored

3 Assists

Shots on target 57% 8 Total 14

Neymar

Brazil

114 Minutes per goal

457 Minutes played

4 Goals scored

1 Assists

Shots on target 64% 9 Total 14

http://www.bbc.com/sport/football/world-cup/2014/top-scorers

- I would like to access a WS (a .wsdl file) to get some info..

http://footballpool.dataaccess.eu/data/info.wso?wsdl

Top Scorers

James Rodríguez

Colombia

67 Minutes per goal

400 Minutes played

6 Goals scored

2 Assists

Shots on target 89% 8 — Total 9

Thomas Müller

Germany

136 Minutes per goal

682 Minutes played

5 Goals scored

3 Assists

Shots on target 57% 8 — Total 14

Neymar

Brazil

114 Minutes per goal

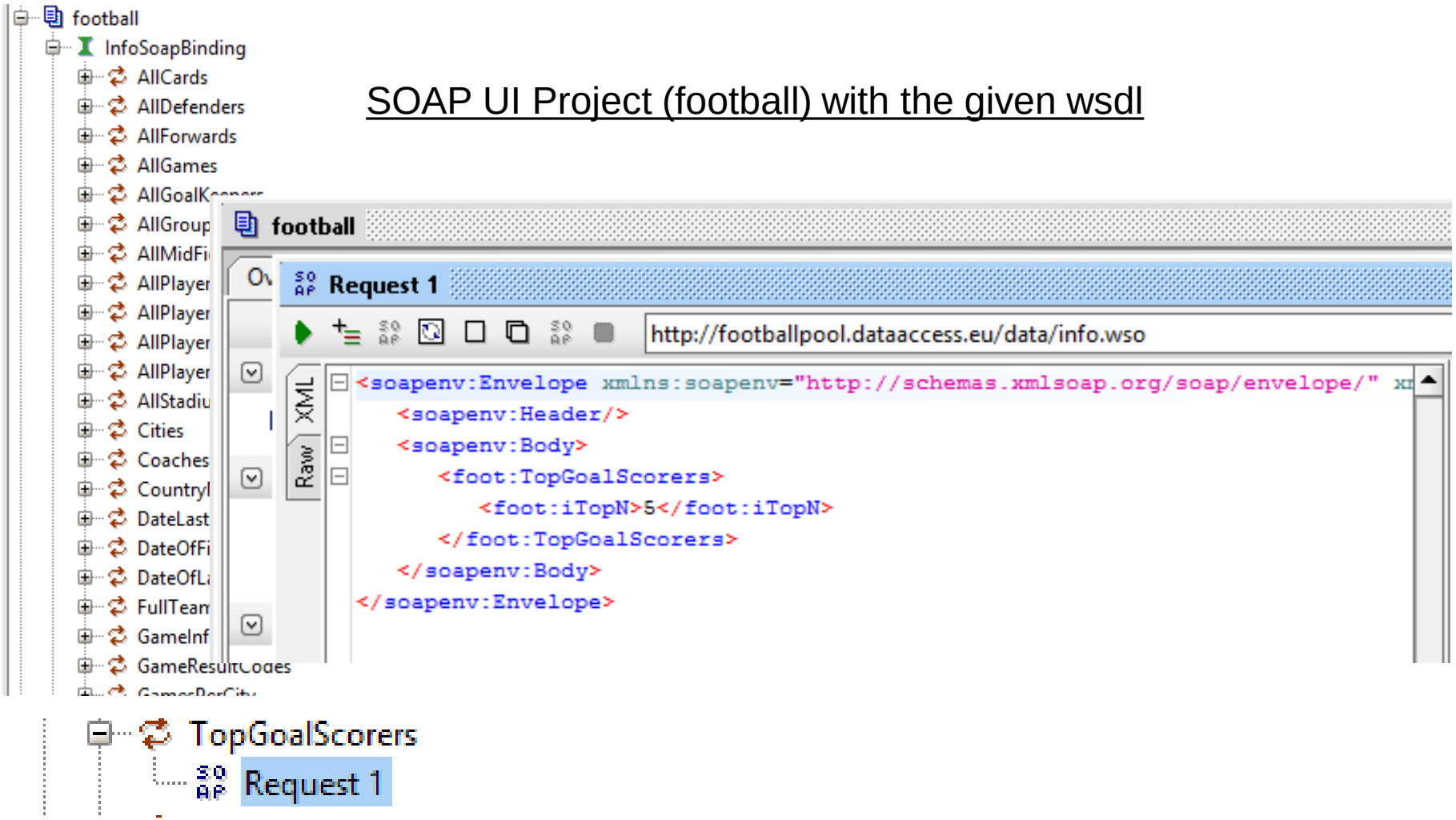
457 Minutes played

4 Goals scored

1 Assists

Shots on target 64% 9 — Total 14

World Cup 2014 Example - Request



SOAP UI Project (football) with the given wsdl

World Cup 2014 Example - Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <m:TopGoalScorersResponse xmlns:m="http://footballpool.dataaccess.eu">
         <m:TopGoalScorersResult>
            <m:tTopGoalScorer>
               <m:sName>James Rodríguez</m:sName>
               <m:iGoals>6</m:iGoals>
               <m:sCountry>Y</m:sCountry>
               <m:sFlag>http://footballpool.dataaccess.eu/images/flags/co.gif</m:sFlag>
               <m:sFlagLarge>http://footballpool.dataaccess.eu/images/flags/co.png</m:sFlagLarge>
            </m:tTopGoalScorer>
            <m:tTopGoalScorer>
               <m:sName>Thomas Mueller</m:sName>
               <m:iGoals>5</m:iGoals>
               <m:sCountry>Y</m:sCountry>
               <m:sFlag>http://footballpool.dataaccess.eu/images/flags/de.gif</m:sFlag>
               <m:sFlagLarge>http://footballpool.dataaccess.eu/images/flags/de.png</m:sFlagLarge>
            </m:tTopGoalScorer>
            <m:tTopGoalScorer>
               <m:sName>Messi</m:sName>
               <m:iGoals>4</m:iGoals>
               <m:sCountry>Y</m:sCountry>
               <m:sFlag>http://footballpool.dataaccess.eu/images/flags/ar.gif</m:sFlag>
```

Outline

- Web Services Components
- SOAP
- WSDL
- WSDL Invocation Tools
- Netbeans IDE: JAX-WS Web Services

Creating (SOAP) Web Services

- Web services can be created using two methods:
  - **top-down approach**: first you design the implementation of the Web service by creating a WSDL file, then use the Web services wizard to create the Web service and skeleton Java classes to which you can add the required code
  - **bottom-up approach**:first you create a Java bean or EJB bean and then use the Web services wizard to create the WSDL file and Web service
- Although bottom-up Web service development may be faster and easier, especially if you are new to Web services, **the top-down approach is the recommended way of creating a Web service.**

Netbeans IDE: Developing a simple web service

- Download and install Netbeans IDE
  - http://netbeans.org/downloads/6.5/index.html

- We discuss a scenario where we develop the Web Service and not access an existing one

- Briefly, the steps involved in creating a WS follow:
  - Create the Web Service
  - Design the Web Service
  - Deploy and Test the Web Service

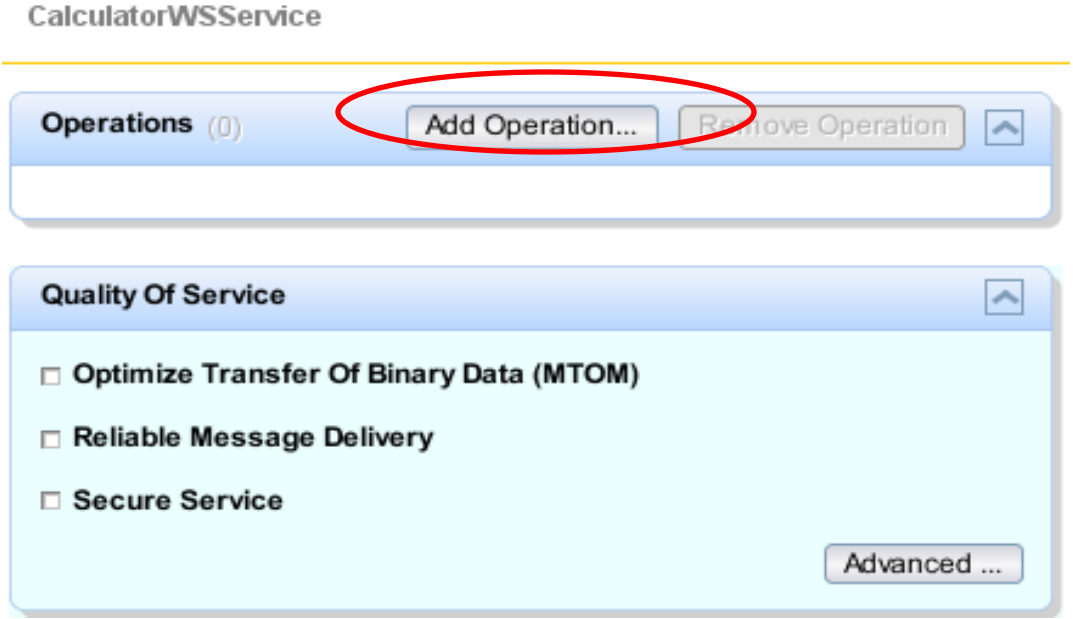
Netbeans: Creating a Simple Web Service (Simple Calculator)

- Create a new Java Web Application by following the next steps

- File > New Project> Java Web > Web Application
  - *Name: SimpleCalculator*
  - *Server: GlassFish v2*
  - *Java EE Version: Java EE 5*
  - *Context Path:/SimpleCalculator*
  - Do not include any additional frameworks

Adding an Operation to the Web Service

- Create a package in the src folder
- **Right click, New > Web Services > Web Service**
- Add 2 simple operations to this WS
  - An **Add** operation: adds 2 integers, and a
  - A **Multiply** operation: multiplies 2 integers
- You can either add these operations manually (writing java code) or by using the design view of the java file or by right-clicking the Web Service in the project and selecting the add operation



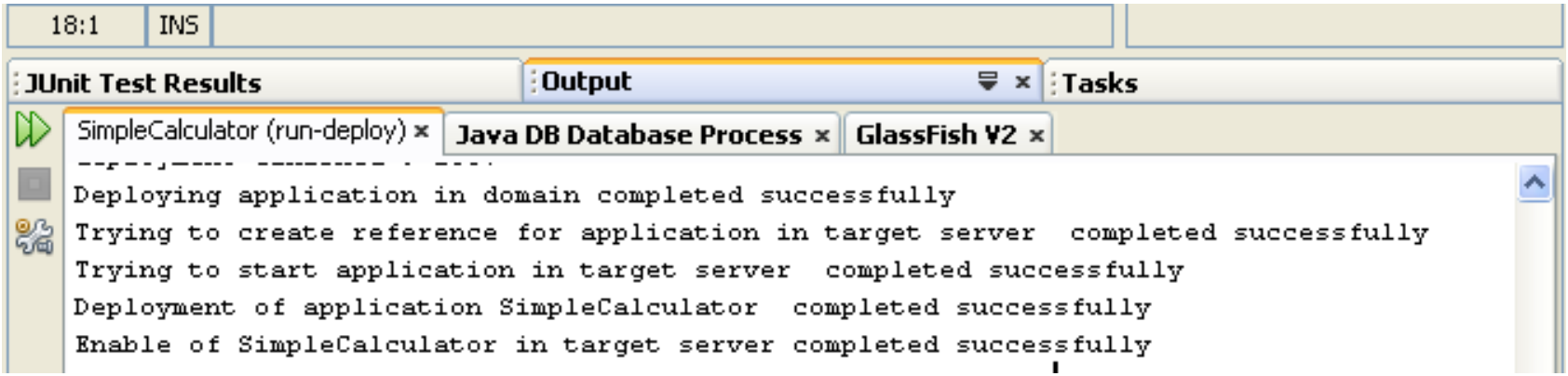
Adding an Operation to the Web Service

- Edit the source code to implement the logic of your operations (actually in this example just code the “return” statements
- Our Web Service is now complete
- We have to deploy it and afterwards to test it

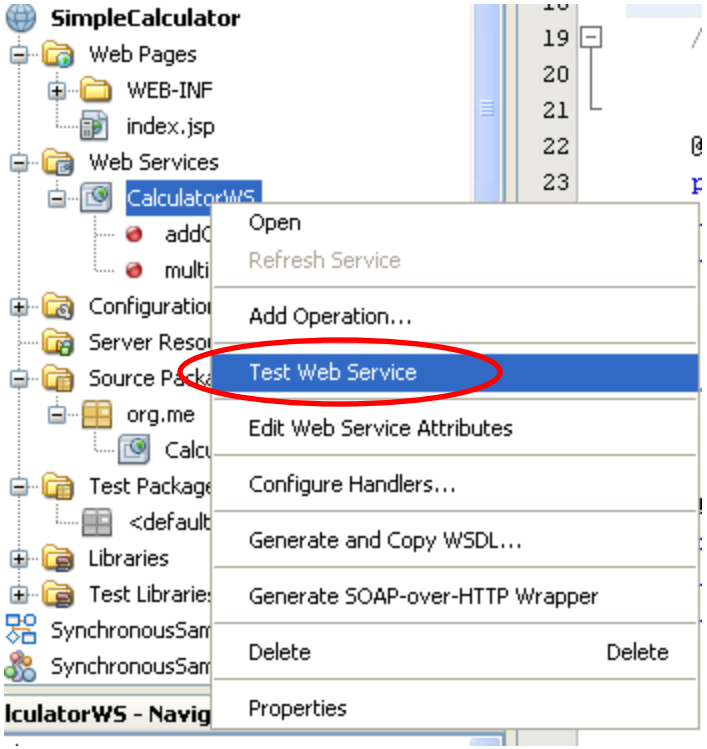
Deploying the Web Service

- Right-click the project SimpleCalculator and choose "**Deploy**" (it will be deployed to the GlassFish v2 Server)

- Look at the output console to check whether your project has been deployed successfully or not



Testing the Web Service



- The IDE opens the tester page in your browser http://localhost:8080/SimpleCalculator/CalculatorWSService?Tester

- You can see the WSDL File of your Web Service by selecting the link

Viewing the generated WSDL

```
- <!--
    Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version
  -->
- <!--
    Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version
  -->
- <definitions targetNamespace="http://me.org/" name="CalculatorWSService">
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://me.org/" schemaLocation="http://localhost:8080
        /SimpleCalculator/CalculatorWSService?xsd=1"/>
      </xsd:schema>
    </types>
  - <message name="addOperation">
      <part name="parameters" element="tns:addOperation"/>
    </message>
  - <message name="addOperationResponse">
      <part name="parameters" element="tns:addOperationResponse"/>
    </message>
  - <message name="multiplyOperation">
      <part name="parameters" element="tns:multiplyOperation"/>
    </message>
  - <message name="multiplyOperationResponse">
      <part name="parameters" element="tns:multiplyOperationResponse"/>
    </message>
  - <portType name="CalculatorWS">
    - <operation name="addOperation">
        <input message="tns:addOperation"/>
        <output message="tns:addOperationResponse"/>
      </operation>
    - <operation name="multiplyOperation">
        <input message="tns:multiplyOperation"/>
        <output message="tns:multiplyOperationResponse"/>
      </operation>
    </portType>
  - <binding name="CalculatorWSPortBinding" type="tns:CalculatorWS">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="addOperation">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
      - <output>
          <soap:body use="literal"/>
        </output>
      </operation>
    - <operation name="multiplyOperation">
        <soap:operation soapAction=""/>
      - <input>
          <soap:body use="literal"/>
        </input>
      - <output>
          <soap:body use="literal"/>
        </output>
      </operation>
    </binding>
  - <service name="CalculatorWSService">
    - <port name="CalculatorWSPort" binding="tns:CalculatorWSPortBinding">
        <soap:address location="http://localhost:8080/SimpleCalculator
        /CalculatorWSService"/>
      </port>
    </service>
</definitions>
```

Calculator Web Service

http://localhost:8080/SimpleCalculator/CalculatorWebServiceService

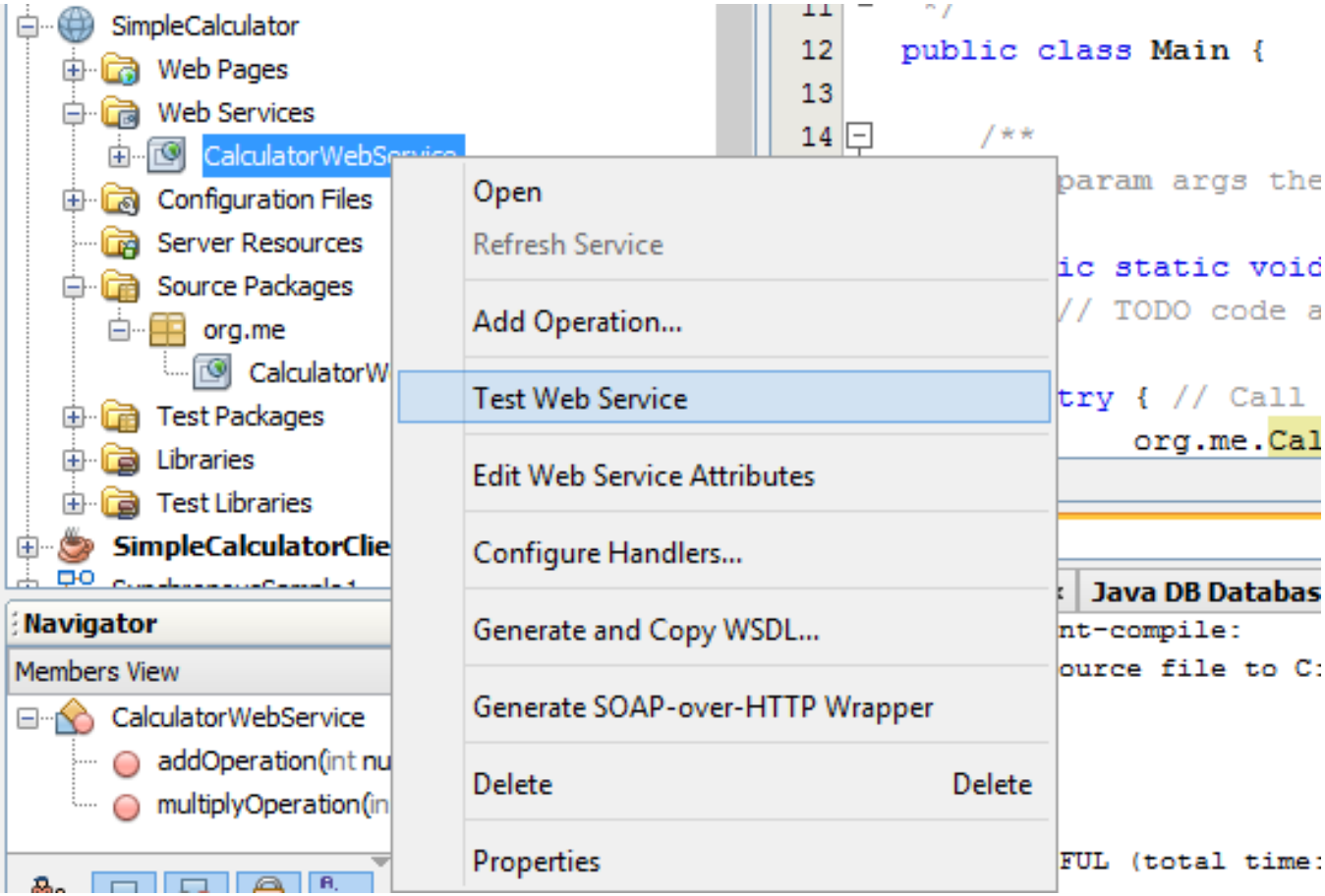
Web Services

Endpoint	Information
Service Name: {http://me.org/}CalculatorWebServiceService Port Name: {http://me.org/}CalculatorWebServicePort	Address: /CalculatorWebServiceService WSDL: /CalculatorWebServiceService?wsdl Implementation class: org.me.CalculatorWebService

http://localhost:8080/SimpleCalculator/CalculatorWebServiceService?xsd=1

```
<xs:schema version="1.0" targetNamespace="http://me.org/">
  <xs:element name="addOperation" type="tns:addOperation"/>
  <xs:element name="addOperationResponse" type="tns:addOperationResponse"/>
  <xs:element name="multiplyOperation" type="tns:multiplyOperation"/>
  <xs:element name="multiplyOperationResponse" type="tns:multiplyOperationRespon
- <xs:complexType name="multiplyOperation">
  - <xs:sequence>
      <xs:element name="number1" type="xs:int"/>
      <xs:element name="number2" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
- <xs:complexType name="multiplyOperationResponse">
  - <xs:sequence>
      <xs:element name="return" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
- <xs:complexType name="addOperation">
  - <xs:sequence>
      <xs:element name="number1" type="xs:int"/>
      <xs:element name="number2" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
- <xs:complexType name="addOperationResponse">
  - <xs:sequence>
      <xs:element name="return" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Testing the Web Service



Testing the Web Service

http://localhost:8080/SimpleCalculator/CalculatorWebServiceService?Tester

Testing the Web Service > Add Operation

- After each method is invoked you can view the SOAP Request and the SOAP Response of the addOperation and the multiplyOperation respectively

addOperation Method invocation

Method parameter(s)

Type	Value
int	23
int	28

Method returned

int : "**51**"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:addOperation xmlns:ns2="http://me.org/">
            <number1>23</number1>
            <number2>28</number2>
        </ns2:addOperation>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:addOperationResponse xmlns:ns2="http://me.org/">
            <return>51</return>
        </ns2:addOperationResponse>
    </S:Body>
</S:Envelope>
```

Testing the Web Service > Multiply Operation

multiplyOperation Method invocation

Method parameter(s)

Type	Value
int	23
int	28

Method returned

int : "**644**"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:multiplyOperation xmlns:ns2="http://me.org/">
            <number1>23</number1>
            <number2>28</number2>
        </ns2:multiplyOperation>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:multiplyOperationResponse xmlns:ns2="http://me.org/">
            <return>644</return>
        </ns2:multiplyOperationResponse>
    </S:Body>
</S:Envelope>
```

Developing JAX-WS Web Service Clients

- Now we will create manually a client to consume to make use of the operations of the Web Service

- The client can be ….
  - A java SE application
  - A servlet
  - A jsp page

- In this presentation we will create a java SE application client

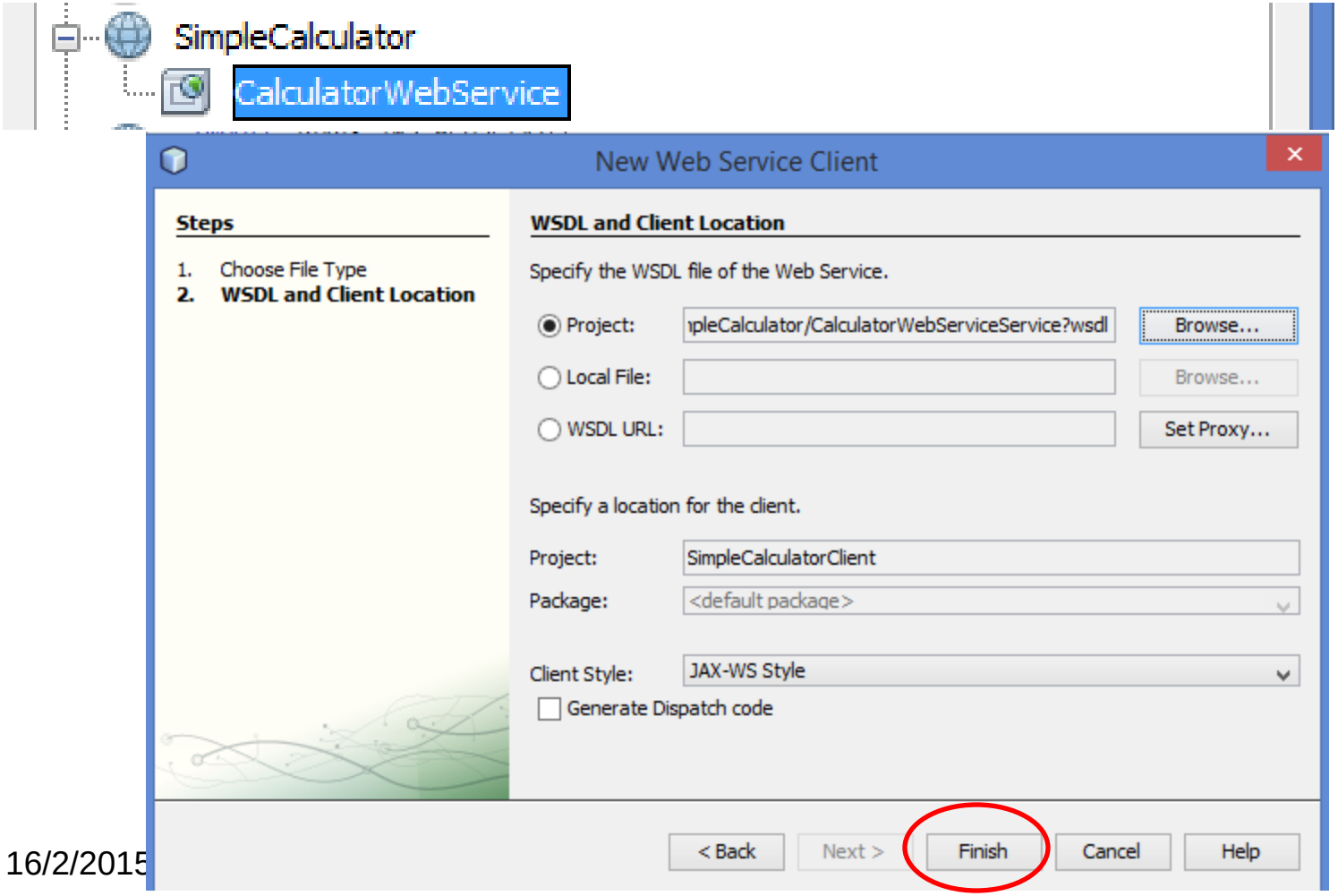
Developing JAX-WS Web Service Clients

1. Choose *File > New Project*
   - Select Java Application from the Java category.
   - Name the project *SimpleCalculatorClient*.
   - Leave Create Main Class selected and accept all other default settings. Click Finish.

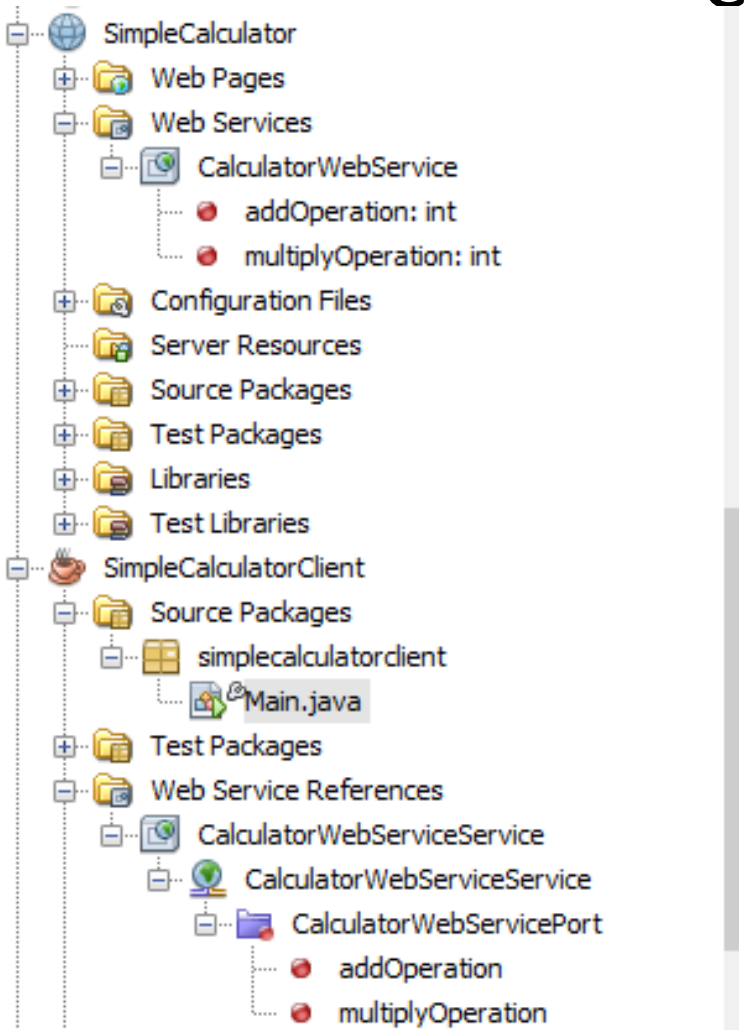
2. Right-click the *SimpleCalculatorClient* node
   - Choose New > Web Service Client.
   - The New Web Service Client wizard opens.

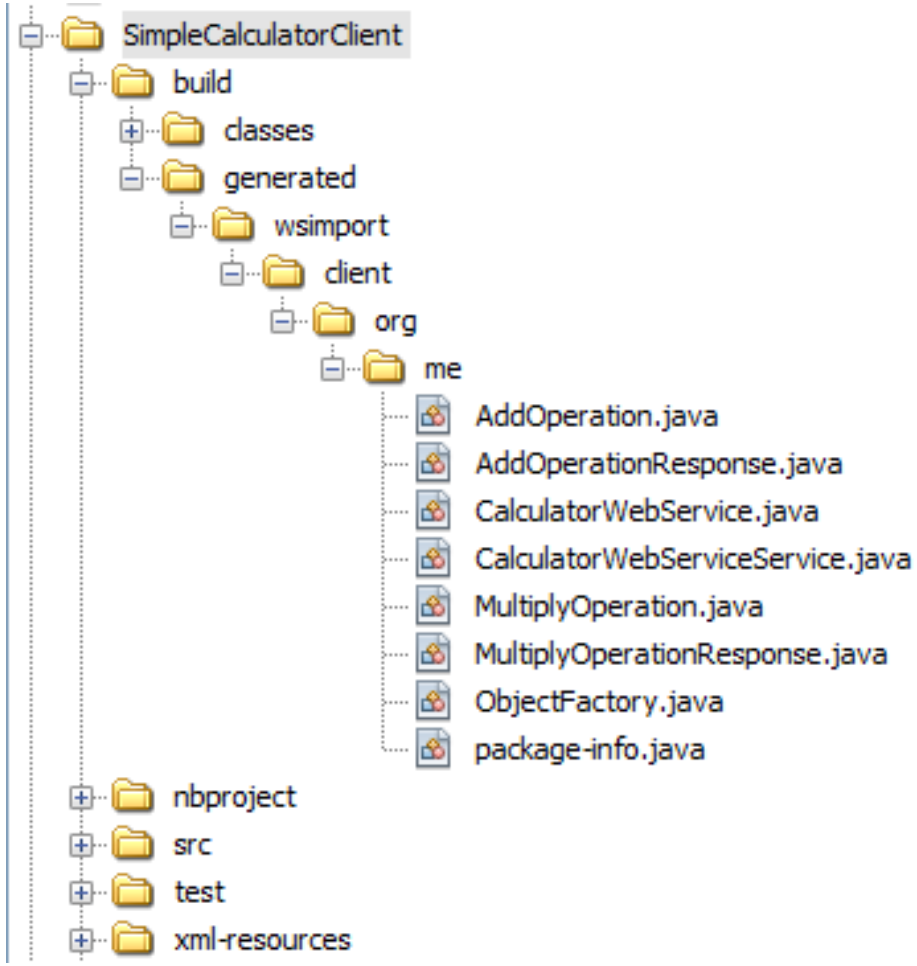
3. Select Project as the WSDL source. Click Browse. Browse to the CalculatorWS web service in the SimpleCalculator project. When you have selected the web service, click OK.

Developing JAX-WS Web Service Clients

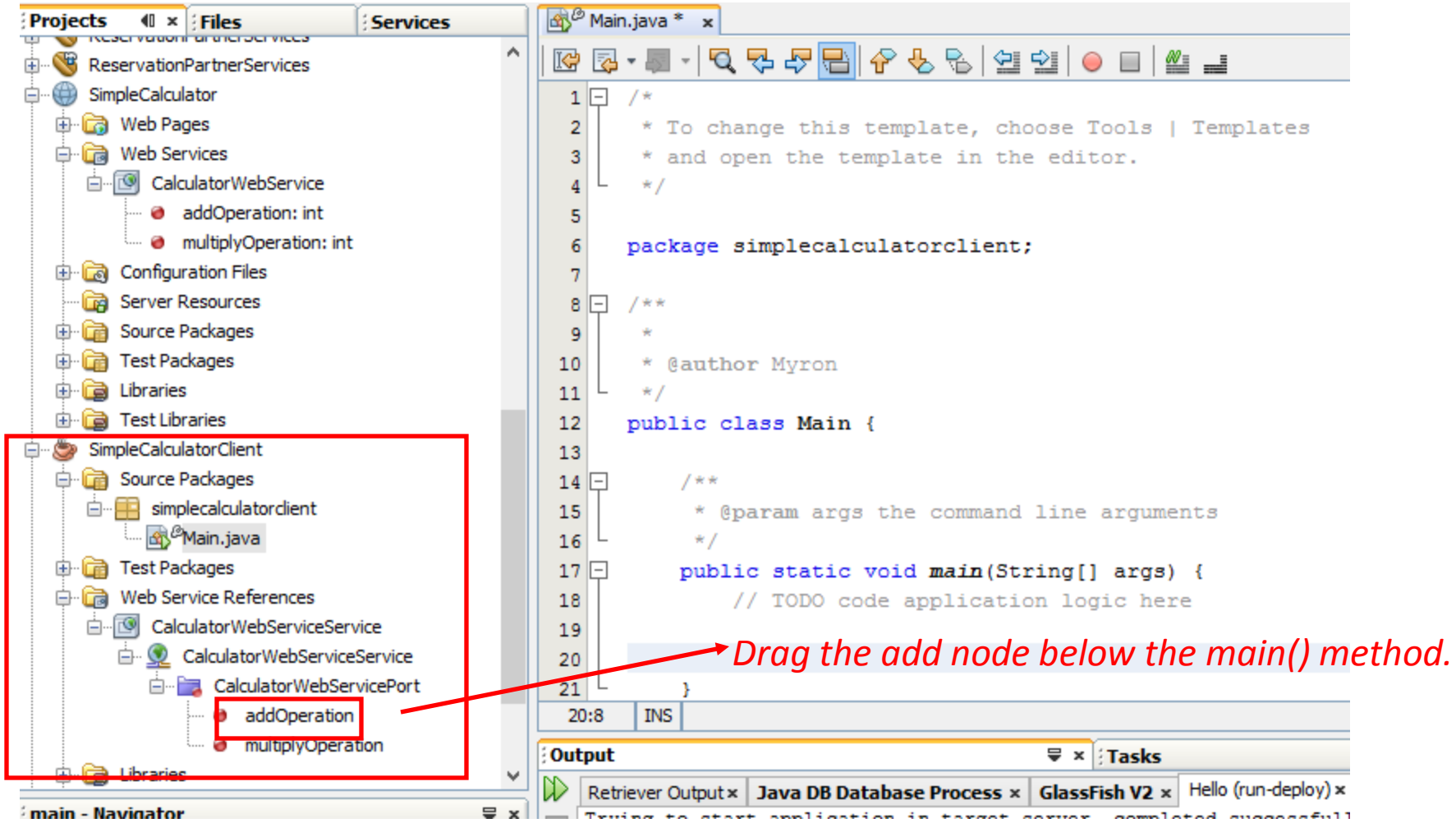


Developing JAX-WS Web Service Clients





Developing JAX-WS Web Service Clients



Developing JAX-WS Web Service Clients

Main.java *

```
11    */
12   public class Main {
13
14       /**
15        * @param args the command line arguments
16        */
17       public static void main(String[] args) {
18           // TODO code application logic here
19
20           try { // Call Web Service Operation
21               org.me.CalculatorWebServiceService service = new org.me.CalculatorWebServiceService();
22               org.me.CalculatorWebService port = service.getCalculatorWebServicePort();
23               // TODO initialize WS operation arguments here
24               int number1 = 0;
25               int number2 = 0;
26               // TODO process result here
27               int result = port.addOperation(number1, number2);
28               System.out.print                                    result);
29           } catch (Exception ex) {
30               // TODO handle custom exceptions here
31           }
```

Variable ex is not used

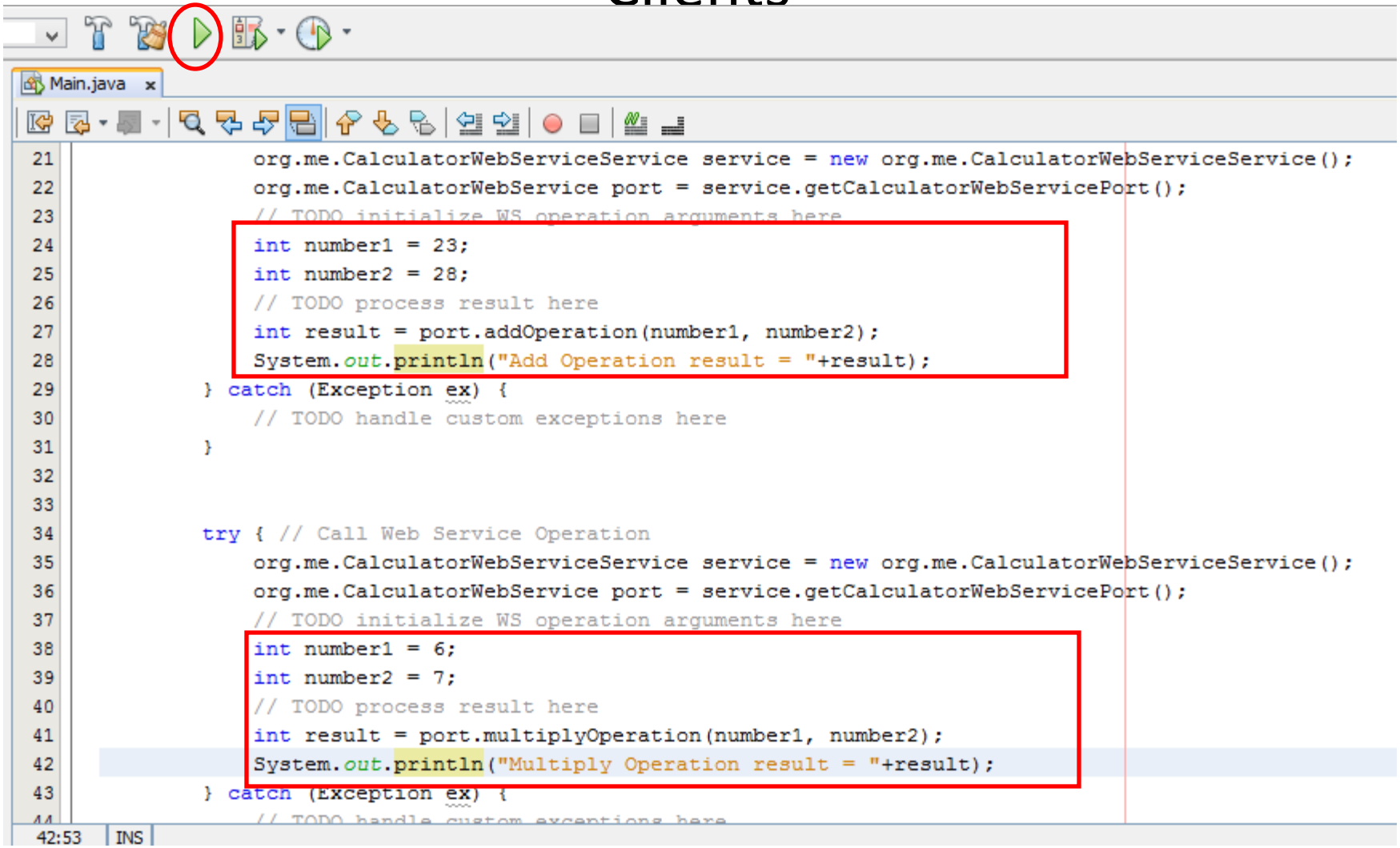
After dragging and dropping

Developing JAX-WS Web Service Clients

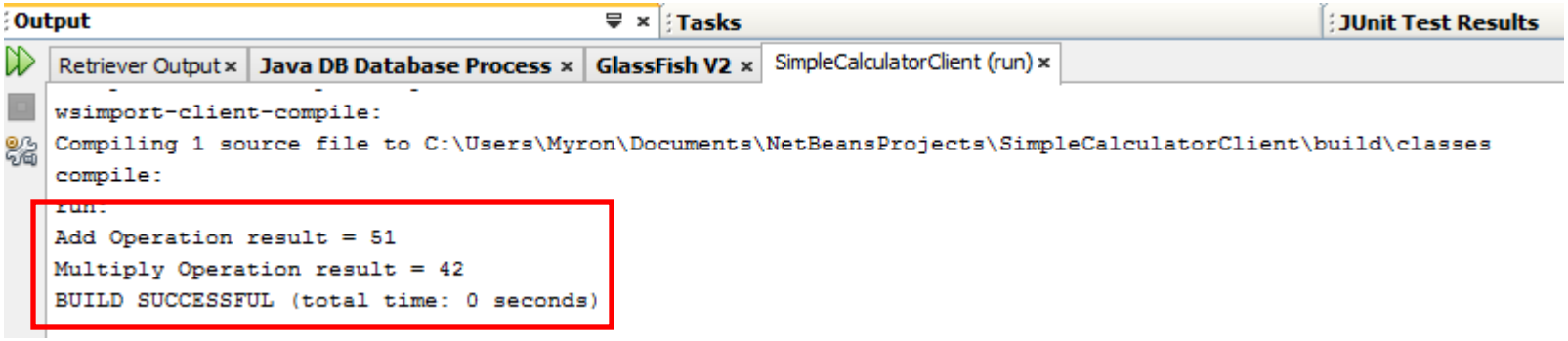
Dragging similarly the multiply operation

```
31          }
32
33
34          try { // Call Web Service Operation
35              org.me.CalculatorWebServiceService service = new org.me.CalculatorWebServiceService();
36              org.me.CalculatorWebService port = service.getCalculatorWebServicePort();
37              // TODO initialize WS operation arguments here
38              int number1 = 0;
39              int number2 = 0;
40              // TODO process result here
41              int result = port.multiplyOperation(number1, number2);
42              System.out.println("Result = "+result);
43          } catch (Exception ex) {
44              // TODO handle custom exceptions here
45          }
```

Developing JAX-WS Web Service Clients



Developing JAX-WS Web Service Clients



Example: Developing JAX-WS Web Service Clients for an External Web Service with Netbeans

- Next we will create a web service for an external Web Service

- Create a Web Service Client Project in Netbeans for an external Web Service

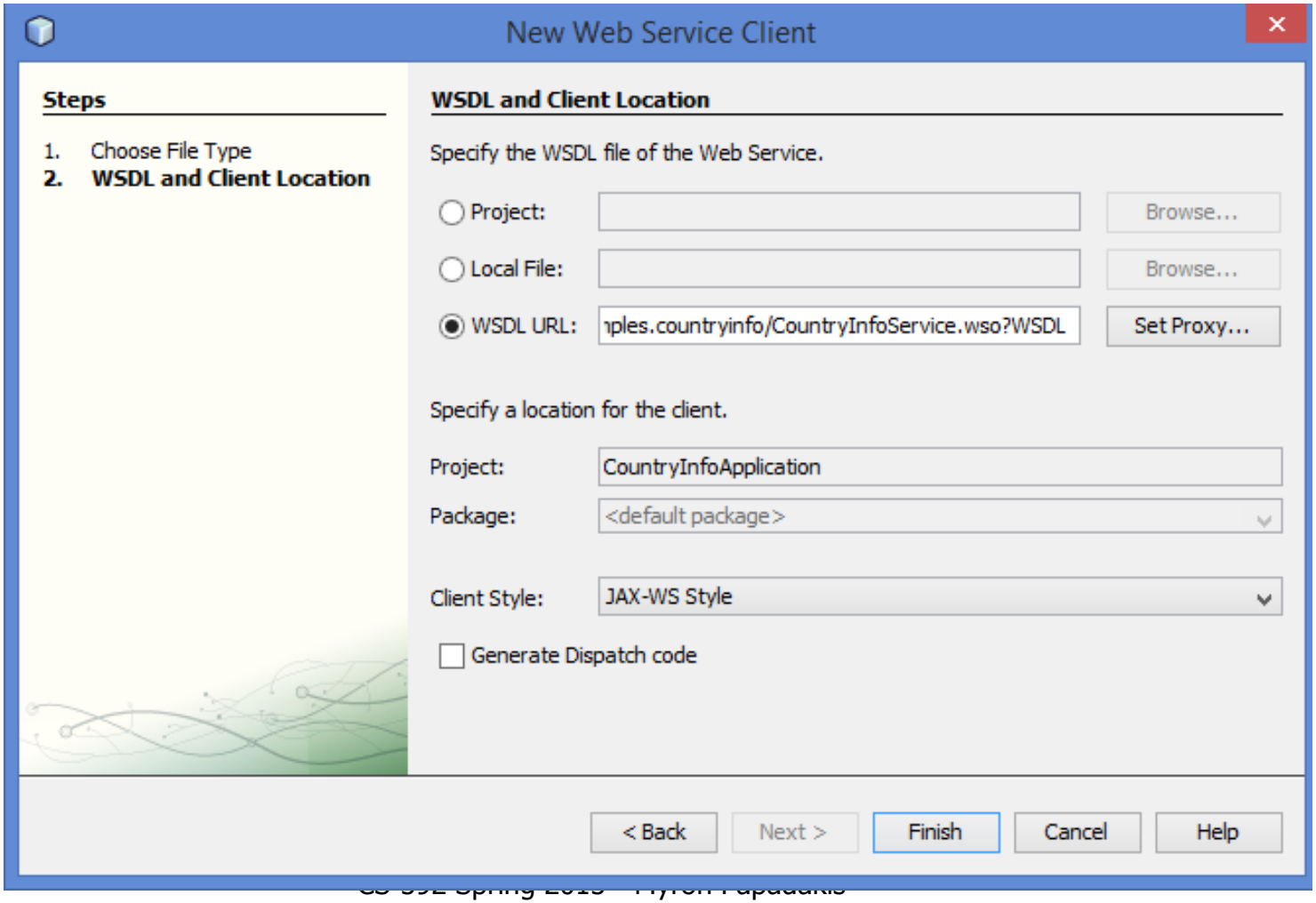
- Suppose for example that we want to create a client in Netbeans that consumes the WS provided by
  - http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL

- In the previous slides we tested this particular WS using soapUI

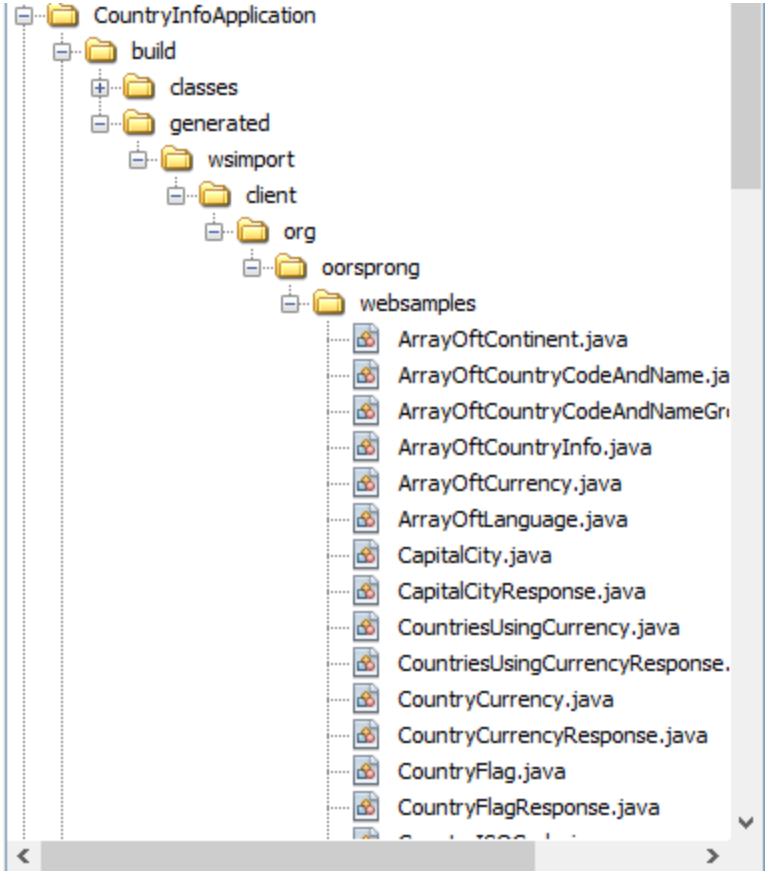
Example: Developing JAX-WS Web Service Clients for an External Web Service with Netbeans

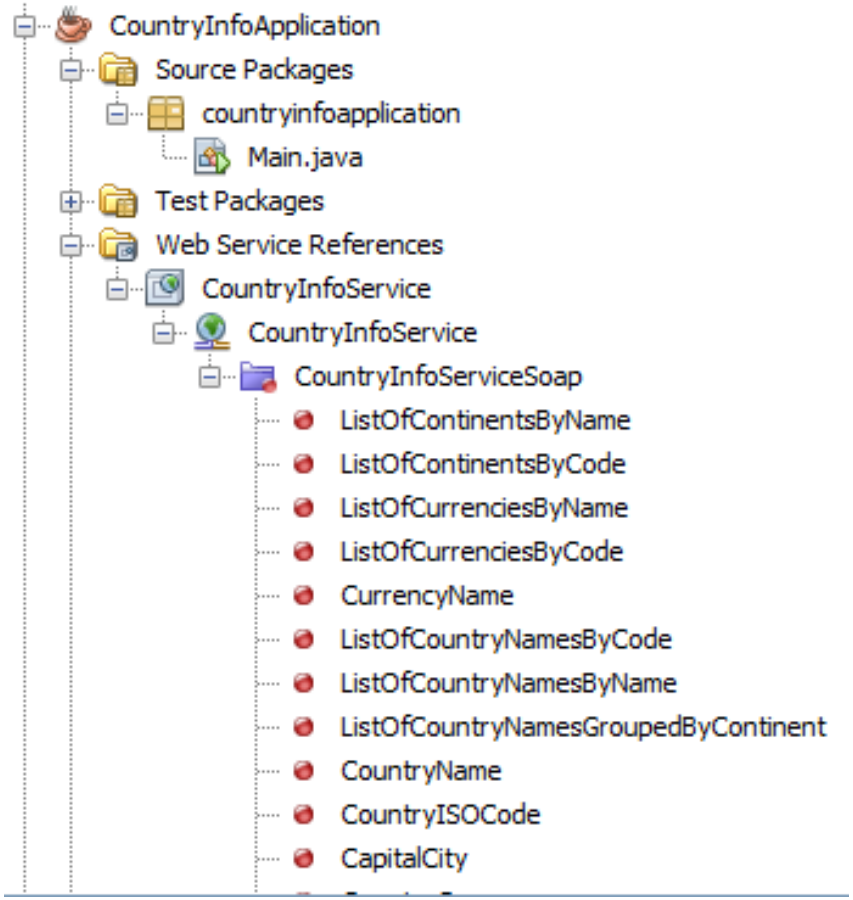
- Create a new Java Application Project “CountryInfoApplication”
- Right click the project node and create a Web Service Client
- Set the WSDL URL to http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?WSDL
- Select Finish

Example: Developing JAX-WS Web Service Clients for an External Web Service with Netbeans

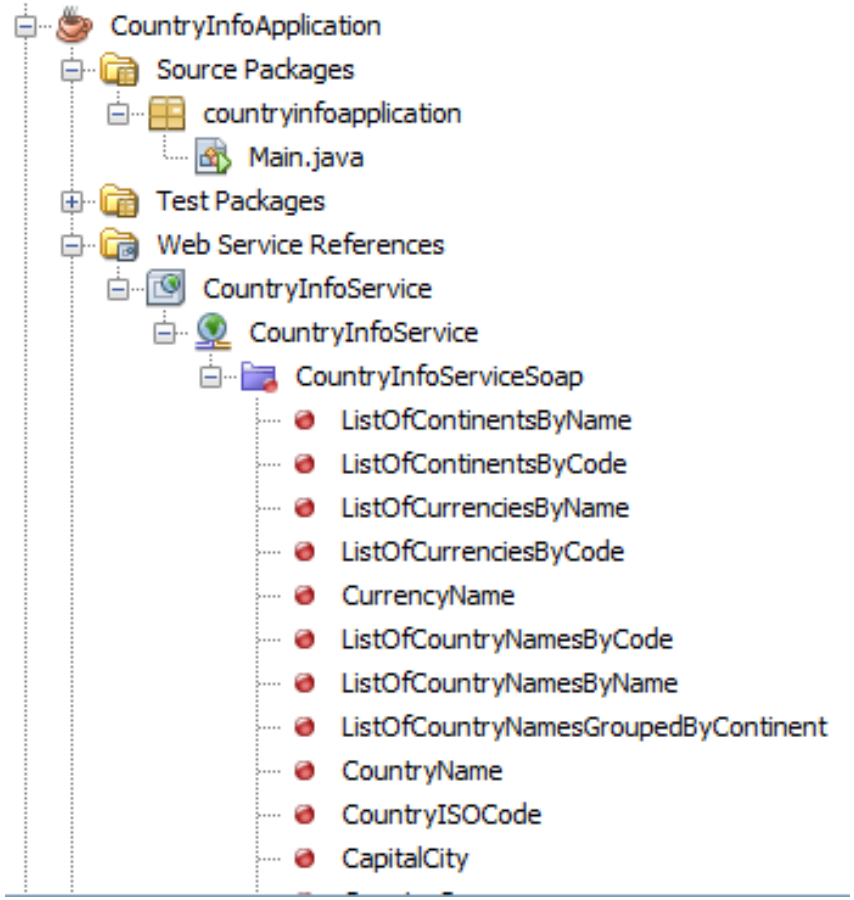


Example: Developing JAX-WS Web Service Clients for an External Web Service with Netbeans





Example: Developing JAX-WS Web Service Clients for an External Web Service with Netbeans



- As demonstrated in the previous slides we can drag any of the available operation to Main.java in order to invoke it
- Drag
  - CountryISOCode
  - FullCountryInfo
- Test Country: Greece

Example: Developing JAX-WS Web Service Clients for an External Web Service with Netbeans

Main.java

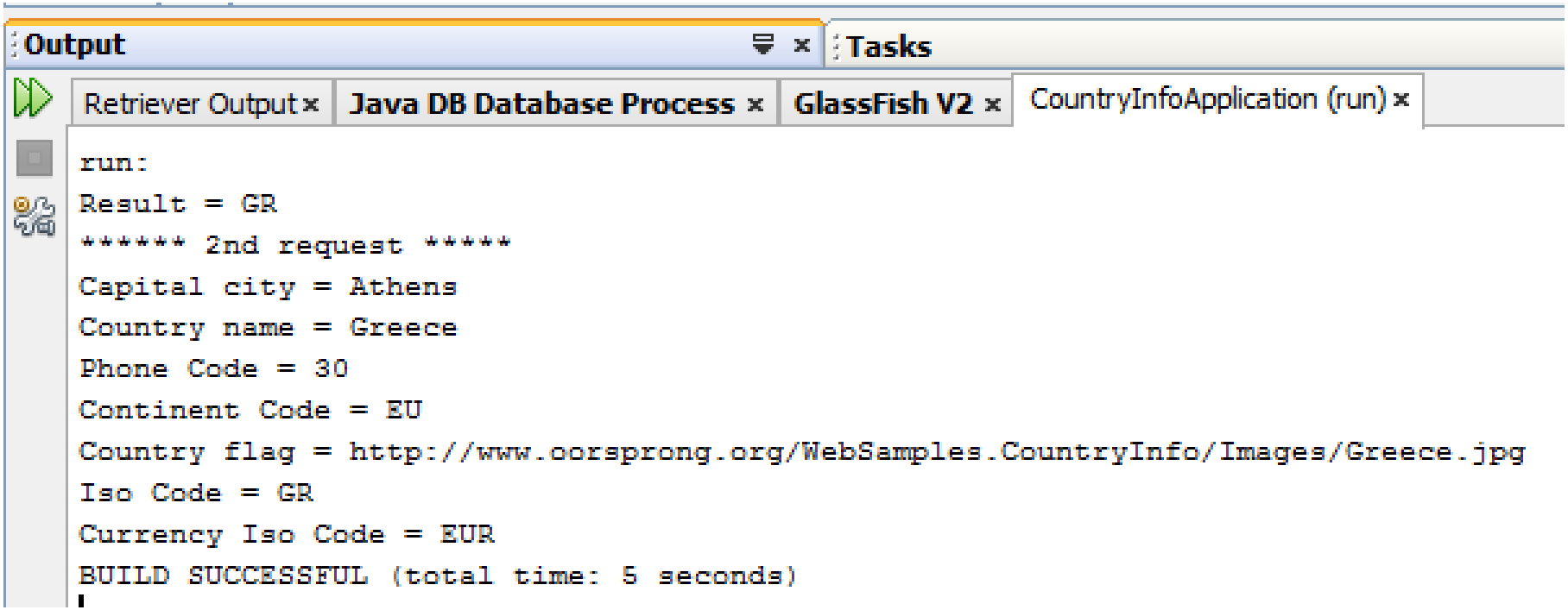
```
16          */
17      public static void main(String[] args) {
18          // TODO code application logic here
19          try { // Call Web Service Operation
20              org.oorsprong.websamples.CountryInfoService service = new org.oorsprong.websamples.CountryInfoSe
21              org.oorsprong.websamples.CountryInfoServiceSoapType port = service.getCountryInfoServiceSoap();
22              // TODO initialize WS operation arguments here
23              java.lang.String sCountryName = "Greece";
24              // TODO process result here
25              java.lang.String result = port.countryISOCode(sCountryName);
26              System.out.println("Result = "+result);
27          } catch (Exception ex) {
28              // TODO handle custom exceptions here
29          }
30
31          try { // Call Web Service Operation
32              org.oorsprong.websamples.CountryInfoService service = new org.oorsprong.websamples.CountryInfoSe
33              org.oorsprong.websamples.CountryInfoServiceSoapType port = service.getCountryInfoServiceSoap();
34              // TODO initialize WS operation arguments here
35              java.lang.String sCountryISOCode = "GR";
36              // TODO process result here
37              org.oorsprong.websamples.TCountryInfo result = port.fullCountryInfo(sCountryISOCode);
38
```

Example: Developing JAX-WS Web Service Clients for an External Web Service with Netbeans

```
try { // Call Web Service Operation
    org.oorsprong.websamples.CountryInfoService service = new org.oorsprong.websamples.CountryInfoSe
    org.oorsprong.websamples.CountryInfoServiceSoapType port = service.getCountryInfoServiceSoap();
    // TODO initialize WS operation arguments here
    java.lang.String sCountryISOCode = "GR";
    // TODO process result here
    org.oorsprong.websamples.TCountryInfo result = port.fullCountryInfo(sCountryISOCode);


    System.out.println("****** 2nd request *****");
    System.out.println("Capital city = "+result.getSCapitalCity());
    System.out.println("Country name = "+result.getSName());
    System.out.println("Phone Code = "+result.getSPhoneCode());
    System.out.println("Continent Code = "+result.getSContinentCode());
    System.out.println("Country flag = "+result.getSCountryFlag());
    System.out.println("Iso Code = "+result.getSISOCode());
    System.out.println("Currency Iso Code = "+result.getSCurrencyISOCode());
```

Example: Developing JAX-WS Web Service Clients for an External Web Service with Netbeans

- Running the client



JAXB *(Just some notes….)*

JAXB Introduction

- NetBeans 6.5 provides support for web services using JAX-WS and JAXB.
- JAX-WS delegates the mapping of the Java language data types to JAXB API.
  - JAXB stands for Java Architecture for XML Binding.
  - JAXB converts XML schemas to Java Content trees and vice versa.
  - Java Architecture for XML Binding (JAXB) is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of Web service

JAXB Introduction

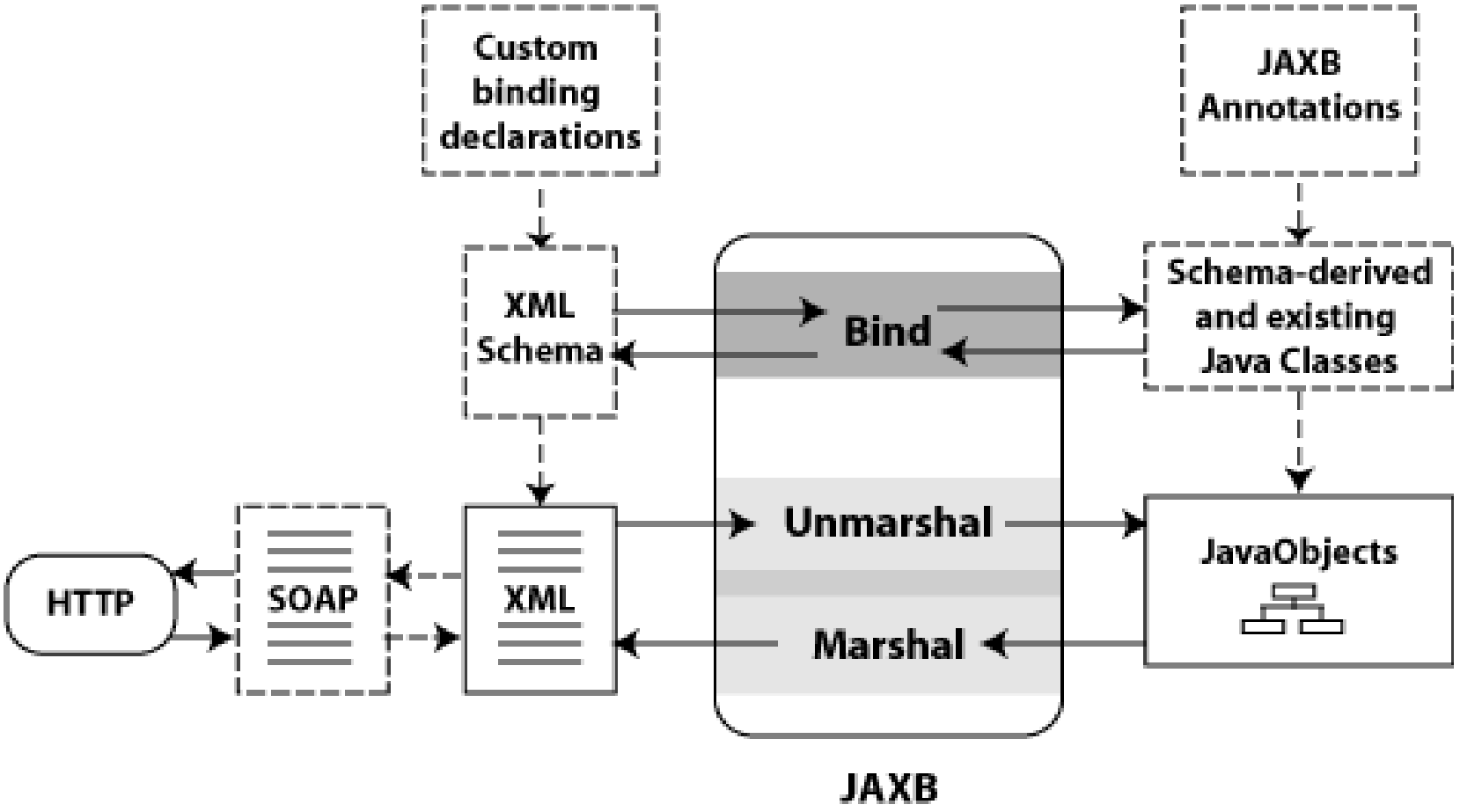
- Basically JAXB is the translator of XML schemas (and data types) to Java and vice versa.
  - Since the WSDL describes the XML schema there is a need to translate that schema to Java.
  - It is the job of JAXB to do just that

- JAXB uses annotations to indicate the central elements.

Overview of Data Binding Using JAXB

- Web Service applications need a way to access data that are in XML format directly from the Java application.
- Specifically, the XML content needs to be converted to a format that is readable by the Java application.
- **Data binding describes the conversion of data between its XML and Java representations.**
- **JAX-WS uses Java Architecture for XML Binding (JAXB) to manage all of the data binding tasks**.
  - Specifically, JAXB binds Java method signatures and WSDL messages and operations and allows you to customize the mapping while automatically handling the runtime conversion.

Data Binding with JAXB



JAXB in Bottom-up (and Top-down)

- **Start from Java**: Using this programming model, you create the Java classes.
  - At run-time, JAXB ***marshals*** the Java objects to generate the XML content which is then packaged in a SOAP message and sent as a Web Service request or response.

- **Start from WSDL**: Using this programming model, the XML Schemas exist and JAXB *unmarshals* the XML document to generate the Java objects.

JAXB Architecture

XML Schema Type	Java Data Type
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd.long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float

JAXB vs. DOM and SAX

- JAXB is not part of the standard Java distribution, but is available from Sun as part of the Java Web Services Developer Pack (Java WSDP)

- JAXB is a higher level construct than DOM or SAX
  - DOM represents XML documents as generic trees
  - SAX represents XML documents as generic event streams
  - JAXB represents XML documents as Java classes with properties that are specific to the particular XML document
    - E.g. book.xml becomes Book.java with getTitle, setTitle, etc.

- JAXB thus requires almost no knowledge of XML to be able to programmatically process XML documents!

Τέλος Ενότητας

Χρηματοδότηση

•Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

•Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

•Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημείωμα αδειοδότησης

- Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».

[1] http://creativecommons.org/licenses/by-nc-nd/4.0/

•Ως **Μη Εμπορική** ορίζεται η χρήση:

–που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο

–που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο

–που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

•Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

.

Σημείωμα Αναφοράς