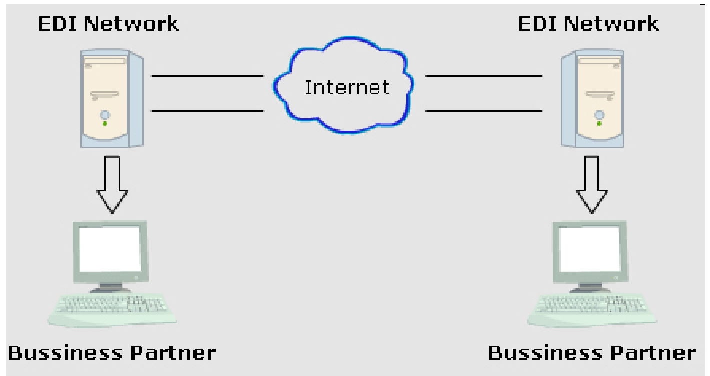
DEVELOPING WEB SERVICES WITH JAVA

SIMPLE OBJECT ACCESS PROTOCOL – SOAP

CONTENTS

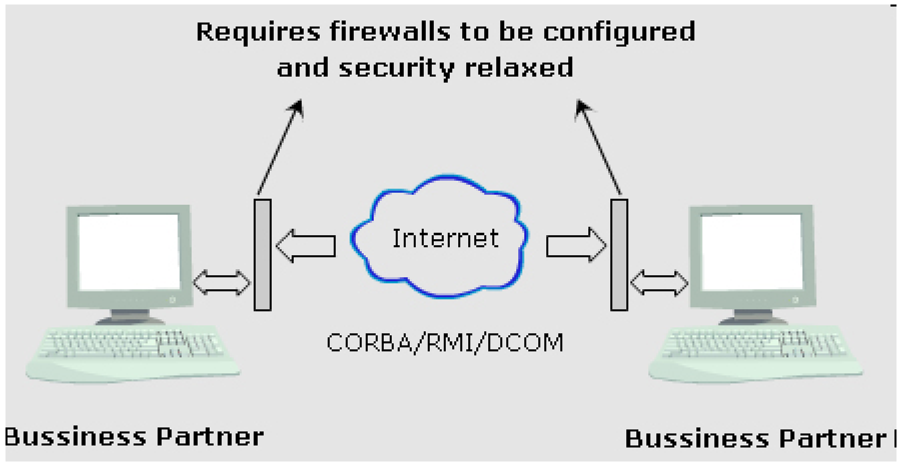
INFORMATION EXCHANGE APPROACHES

- Electronic Data Interchange (EDI)
  - Is a technique used by business partners to exchange business documents that included purchase orders, invoices, shipping notification, financial payments, and so on.
  - To send an EDI document
    - Install translation software on your system that is used to convert business documents into X12 format.
    - Next, you set up a private wide area network to send and receive the documents.
    - The same process is repeated at the receiver’s end.
    - Drawbacks:
      - Cost involved in setting up private wide area networks was too high.
      - Bus partners had to buy proprietary software for transmission of messages from their system to private network.
      - Each business partner had to buy propriety software to translate business document to X12 format.



INFORMATION EXCHANGE APPROACHES (cont)

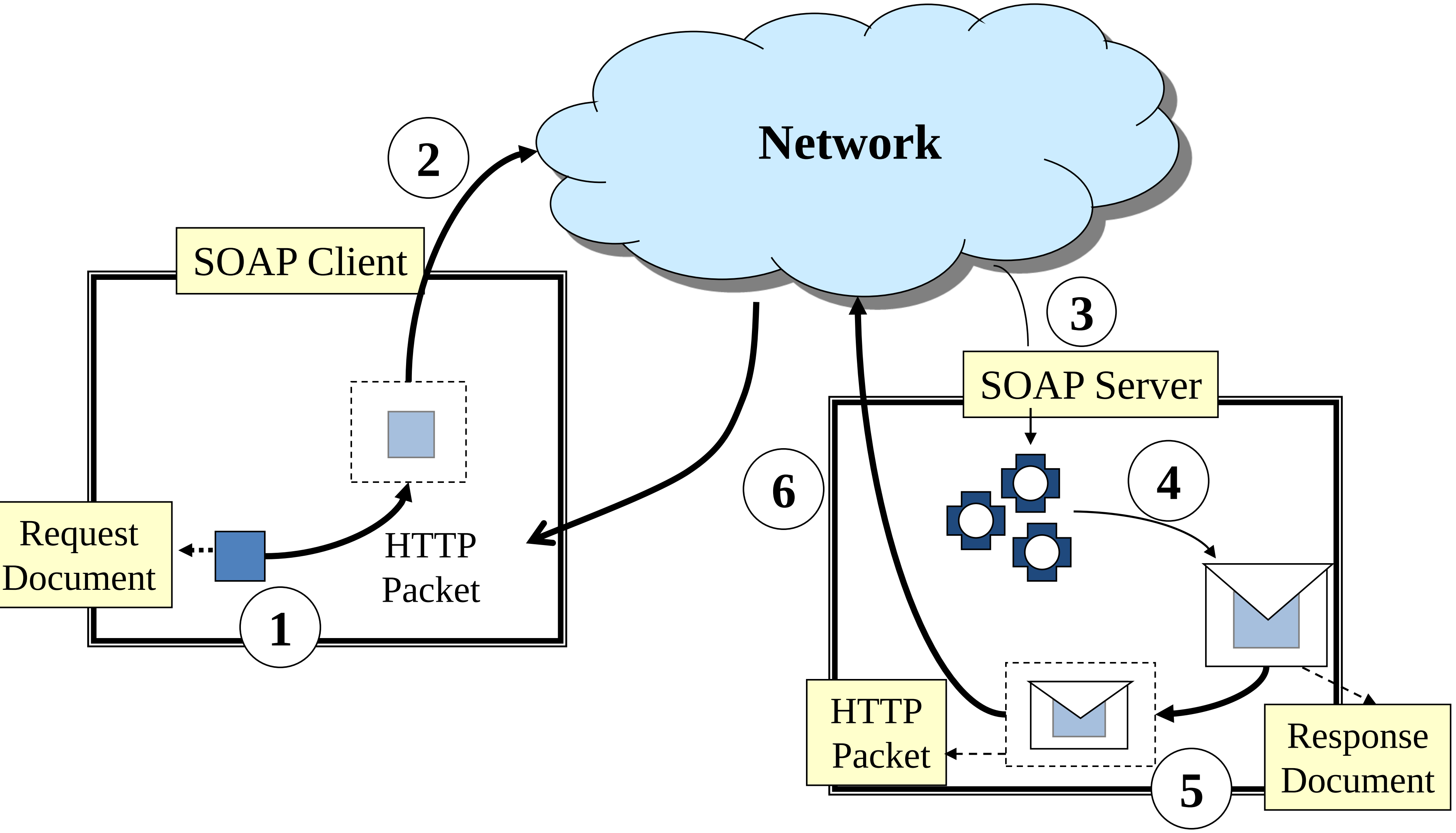
- Remote Procedure Call (RPC)
  - Involved invoking remote methods that allowed information exchange in the form of parameters and returned values.
  - Approach standardized the communicate protocol and eliminated the need of private networks.
  - Drawbacks:
    - Several vendors came up with RPC-based technologies, such as CORBA, RMI, and DCOM.
    - Communication between distributed systems required relaxation of security features.



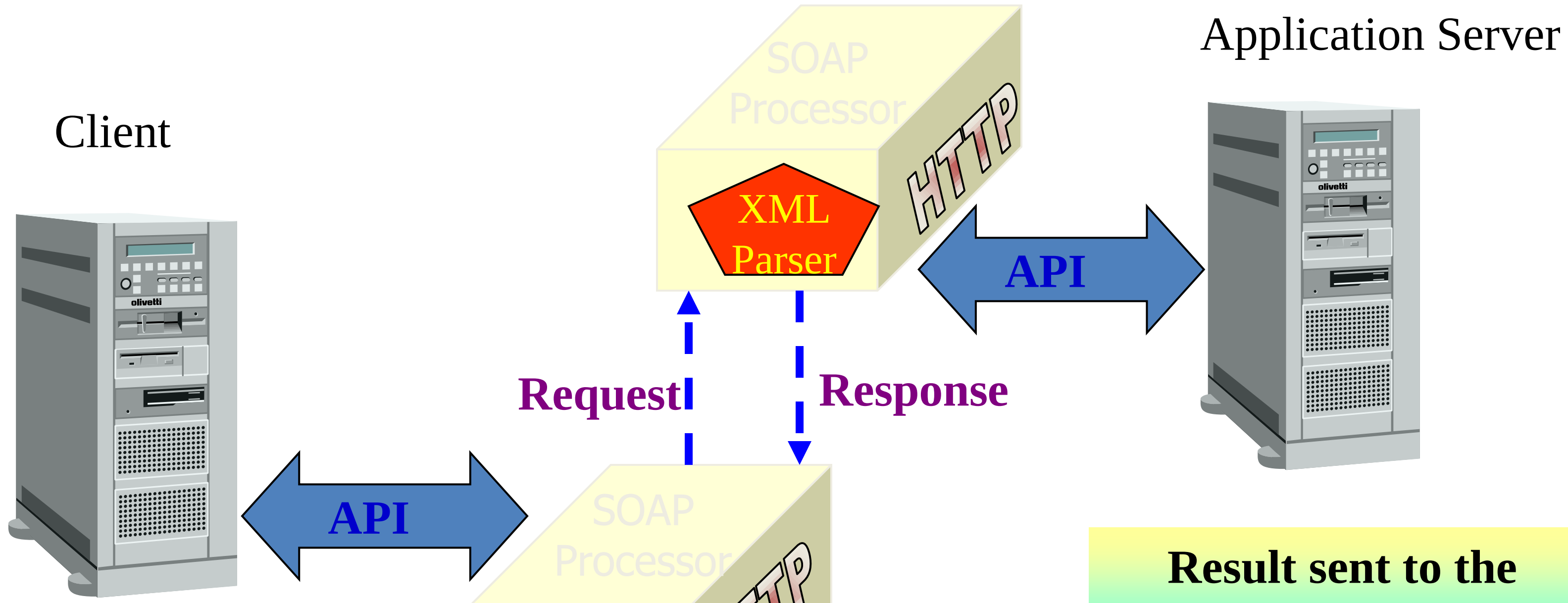
SOAP

- Provides interoperability between applications by using XML & HTTP.
- Uses HTTP protocol to transfer messages between applications. HTTP is recognized by all browsers, proxies, firewalls and servers. Furthermore, it uses port 80 to transmit data.
- Advantages:
  - **Vendor Neutral-SOAP** is defined by W3C.
  - **Transport Protocol Independent-SOAP messages** can be transmitted over HTTP, Simple transfer Protocol (SMTP), File Transfer Protocol (FTP) and Post Office Protocol (POP). However, Web Service Interoperability (WS-I) organization allows the use of only HTTP.
  - **Platform Independent-XML** is platform independent.
  - **Language Independent-SOAP messages** are plain XML documents. Several languages, such as Java, Perl, C++ provide support to create XML documents.
  - I**nteroperability-SOAP** uses XML and HTTP. This enables distributed applications to communicate without any hassles.
  - **Simple-SOAP** is simple to use in that it does not require any change in network infrastructure or security configurations.

SOAP (con’t)


Network
2
SOAP Client
3
SOAP Server
6
4
Request
Document
HTTP
Packet
1
HTTP
Packet
5
Response
Document

SOAP (con't)

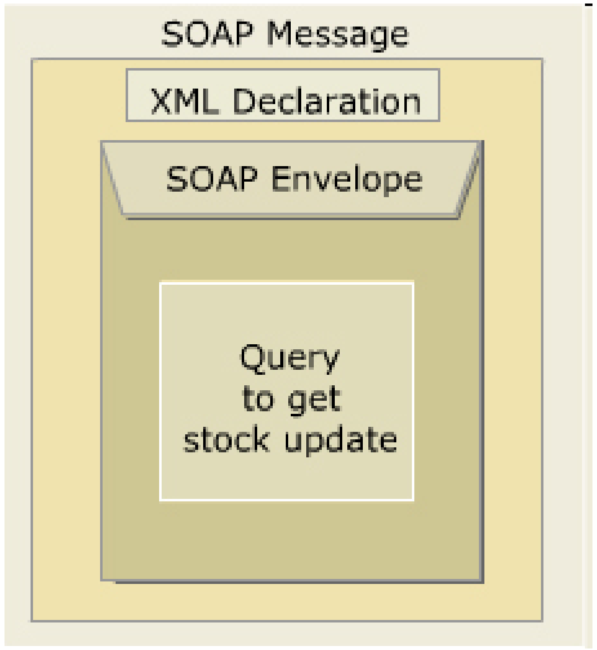


Client command creates action on the application server

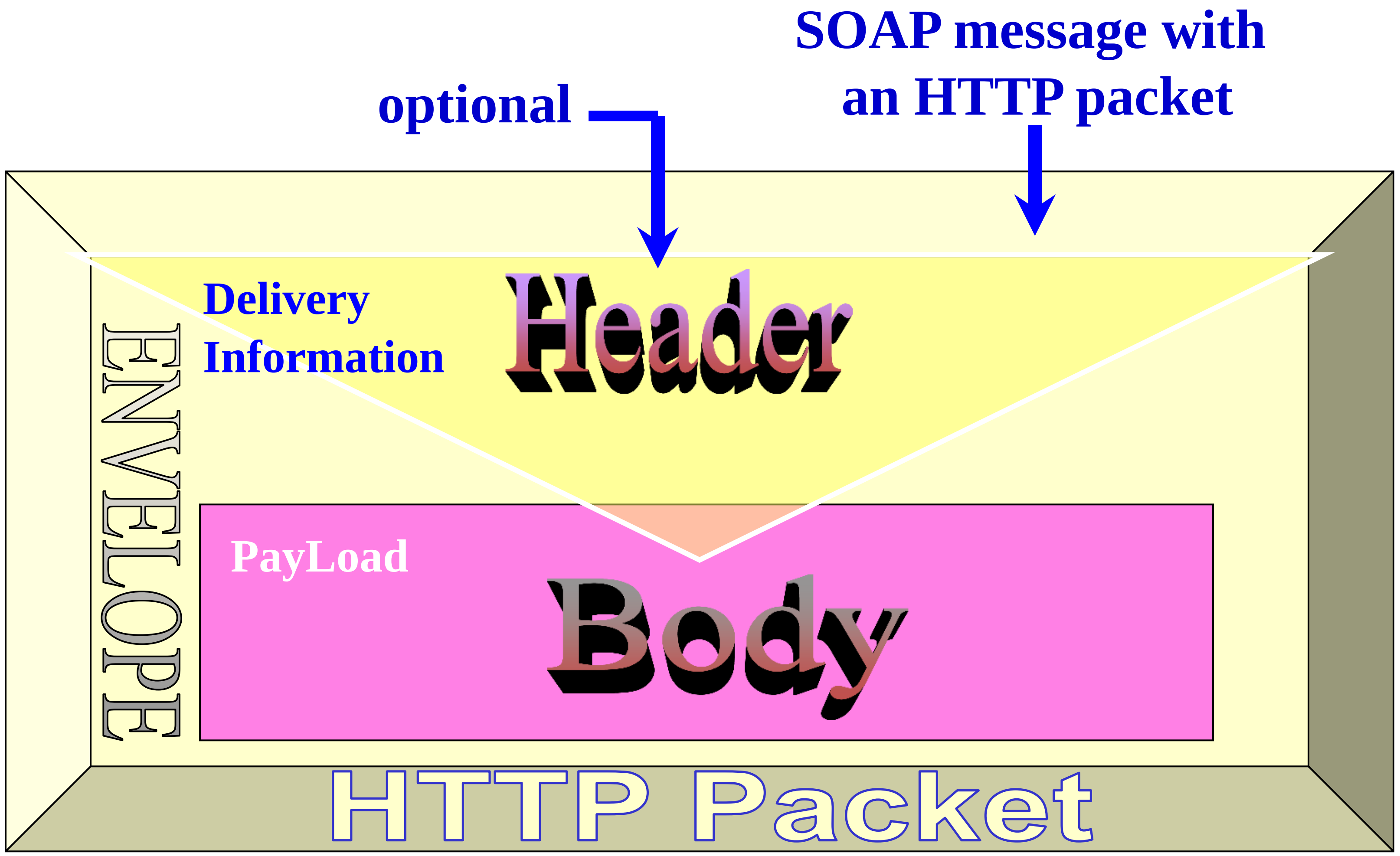
Result sent to the client that had called the Web Service

SOAP MESSAGES

- Is an XML document.
- Is used to exchange data between applications.
- Can contains the requesting service, querying of searching, and so on.
- Contains two parts:
  - ***XML declaration***:
    - Is a beginning of SOAP message.
    - States the version of XML and encoding used.
  - ***Envelope***:
    - Comprises ***Header*** and ***Body*** where the data is enclosed.
    - Is analogous to a paper envelope.
    - Forms a message from one application to another application (from a sender to receiver).
    - Acts as a container for the message.
    - The ***Envelope*** element:
      - Acts as a root element of the message.
      - Acts a processing node to the receiving application.



ARCHITECTURE of SOAP MESSAGES



SOAP MESSAGES (cont)

- Envelope:
  - SOAP Namespace:
    - SOAP uses the namespace ***http://schemas.xmlsoap.org/ soap/envelope/*** to qualify these elements and attributes in a SOAP message.
    - The Envelope element is qualified by ***http://schemas.xmlsoap.org/ soap/envelope/*** namespace. The prefix of this namespace is ***SOAP-ENV*** or ***soapenv***.
    - The namespace ***http://schemas.xmlsoap.org/ soap/envelope/*** also indicates that the message adheres to SOAP 1.1. Therefore, if the receiving application finds a different namespace, it can flag and error. This ensures that both the sender and the receiver adhere to same version of SOAP.
    - The second namespace is used to qualify the data elements of the SOAP message with other prefix.
    - Ex:

```
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:ns1="http://ws.sample/">
```

SOAP HEADER

- SOAP Header:
  - Are used to extend SOAP messages to include additional information and functionality to process a message.
  - Is information included in the header part of the SOAP message such as identity of the person requesting the service, account information, and so on.
  - Includes in headers are digital signatures for password-protected service and information, such as authentication, authorization, transaction management, and routing path.
  - Is the child element of SOAP message’s Envelope element.
  - Is an optional element.
  - If present it should be the immediate child of Envelope element. The immediate child elements of the Header element are referred as header entries.
  - Each header entry and its child elements must be qualified with a namespace.
  - The header entries can have attributes:
    - ***actor*** is used to specify the URI or role of such intermediaries and the ultimate destination of a message.
    - ***mustUnderstand***
      - Is used to indicate to the recipient whether is mandatory for it process the header entry.
      - Allows two values 0 (equivalent to omitting this attribute) and 1 (recipient should process the header entry).

EXAMPLE

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

  <SOAP-ENV:Header
    xlmns:au="http://www.flamingo.com/books/
     authentication">

    <!--Header entry -->
  <au:Requestor mustUnderstand="1">
      <name>John Smith</name>
  </au:Requestor>
```

```
<SOAP-ENV:Header>
  <mid:message-id
   soap:actor="http://www.flamingo.com/logger" >
    11546544ea:b134534:f3sdas5342:4354
  </mid:message-id>
  <m:monitored-by
    SOAP-ENV:actor=
     "http://schemas.xmlsoap.org/soap/actor/next">
    <node>
      <time>1078753670000</time>
      <identity>austria</identity>
    </node>
  </m:monitored-by>
 </SOAP-ENV:Header>
```

BODY ELEMENT

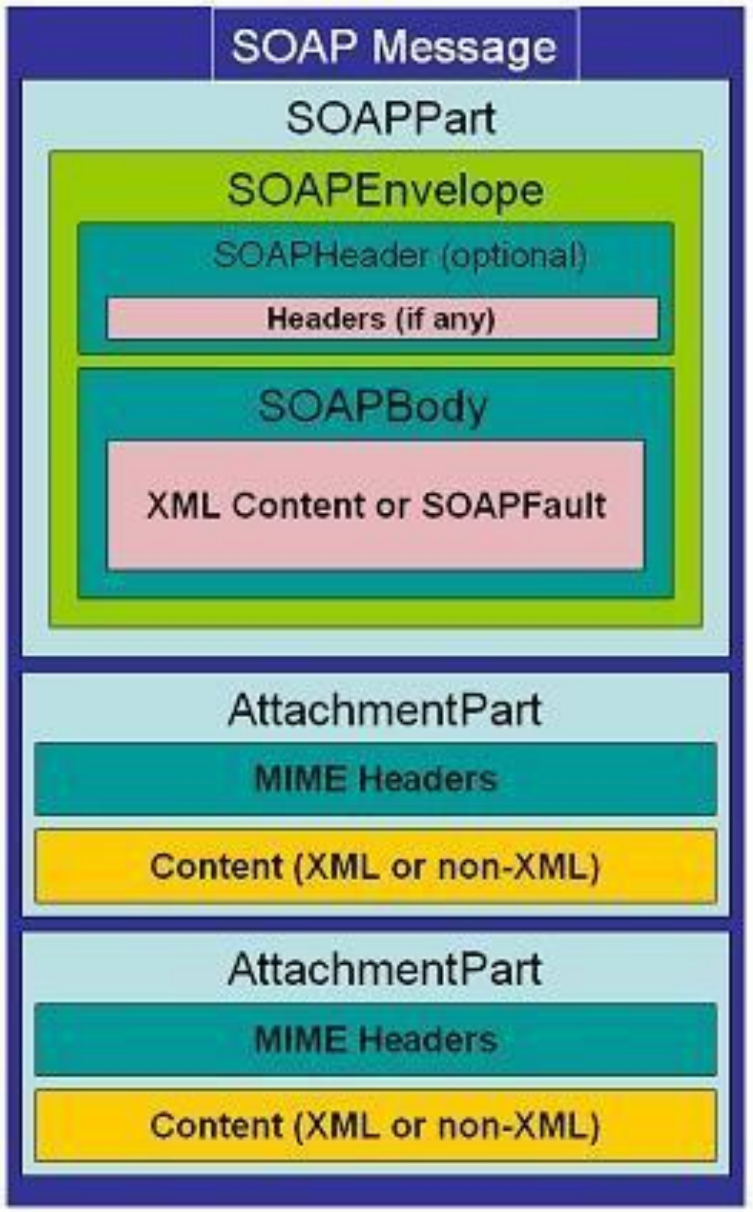
- Contains application-specific data to be exchanged between applications as parameters to a method call.
- Is the mandatory element of Envelope element.
- The immediate child elements of Body element must be namespace-qualified.
- If the Header element is not present, then Body element should be the immediate child of Envelope element. However, in the presence of Header element, the Body element should immediately follow the Header element.

```
<soapenv:Body>
    <ns1:add>
        <num1>3</num1>
        <num2>5</num2>
    </ns1:add>
</soapenv:Body>
```

SOAP WITH ATTACHMENT

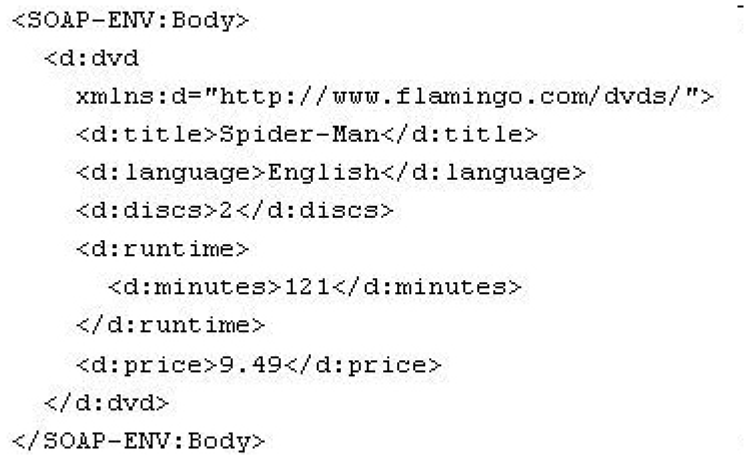
- Need of attachment:
  - SOAP does not allow binary data such as images in the message.
  - Messages that require binary data are converted to a Multipurpose Internet Mail Extensions (MIME) message format and then sent. A MIME message can contain multiple parts and supports binary data as well.
- MIME Message Structure:
  - Was designed to enable the inclusion of attachments in emails.
  - Each MIME part contains a header that identifies the type of content.
  - Comprises two parts:
    - The ***MIME header*** identifies the version of MIME and content type that as Multipart/Mixed indicates that the email contains multiple parts and of varying types.
    - The ***MIME package*** can contain one or more MIME parts. An email with attachment contains two or more MIME parts. One part contains the text message while the other parts contain the attachment files.
- SOAP with attachement puts the SOAP message along with binary data interface the MIME package (that is putted on the HTTP header).

SOAP WITH ATTACHMENT

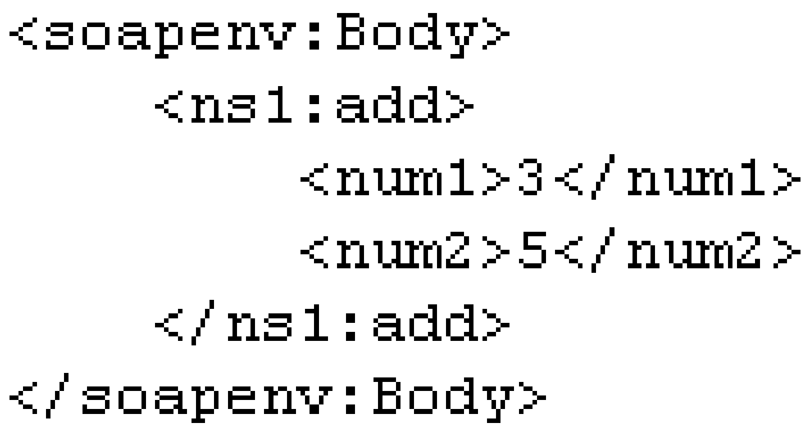


SOAP MESSAGING MODES

- Are classified in terms of:
  - Messaging style are RPC and Document.
  - Encoding type are SOAP encoding and Literal encoding.
- There are four messaging modes:
  - ***Document/Literal*** is used to send XML fragments (such as movie details, purchase order …) in a SOAP message.
  - ***RPC/Literal*** is based on sending request messages (represents a method invocation with zero or more parameters) and receiving response messages (represents a result or value returned by a method).
  - ***Document/Encoded***.
  - ***RPC/Encoded***.



```
<SOAP-ENV:Body>
  <d:dvd
    xmlns:d="http://www.flamingo.com/dvds/">
    <d:title>Spider-Man</d:title>
    <d:language>English</d:language>
    <d:discs>2</d:discs>
    <d:runtime>
      <d:minutes>121</d:minutes>
    </d:runtime>
    <d:price>9.49</d:price>
  </d:dvd>
</SOAP-ENV:Body>
```



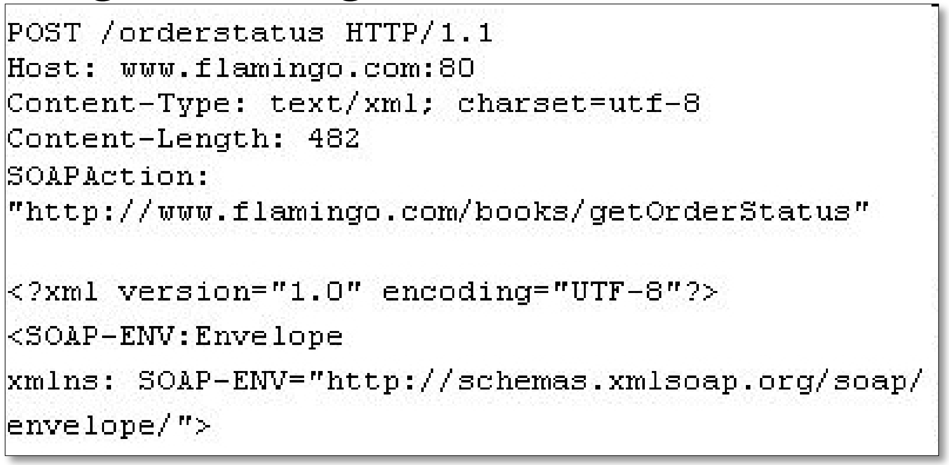
```
<soapenv:Body>
    <ns1:add>
        <num1>3</num1>
        <num2>5</num2>
    </ns1:add>
</soapenv:Body>
```

TRANSPORT PROTOCOL

- HTTP
  - A standard protocol used worldwide to transfer data over the Web.
  - Has a feature called HTTP tunneling.
  - SOAP was designed keeping in mind the HTTP protocol.
  - SOAP messages are transmitted as a payload of an HTTP message that usually contains form data such as username, password, credit card number, and so on. HTTP is a request-response protocol.
  - HTTP GET message does not have a payload area and hence it is not used to transmit SOAP message. A SOAP message is carried as a payload in an HTTP POST message.
- SOAP
  - A SOAP message sent using HTTP POST message comprises two parts:
    - HTTP Header.
    - SOAP Message.

HTTP HEADER of SOAP REQUEST

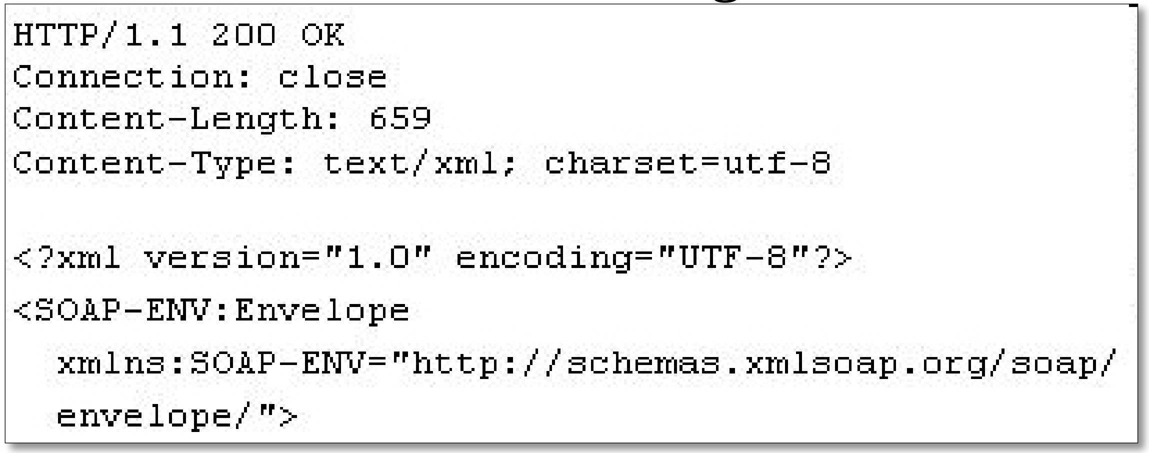
- Indicates that it is an HTTP version 1.0 POST message targeted for the Web Service ***orderstatus***.
- Provides host location of the Web Service. The value ***text/xml*** of Content-Type field indicates that the POST message contains XML content. The Content-Length field provides the number of characters in the entire message.
- The SOAPAction field:
  - Is a mandatory field for HTTP messages used for sending SOAP messages.
  - Indicates to the HTTP server what it should do before processing the message.
  - Can contain arbitrary data, a URI, an empty string or nothing at all. In the image, the SOAPAction field contains the namespace of the Web Service and the name of the method responsible for processing the message.



```
POST /orderstatus HTTP/1.1
Host: www.flamingo.com:80
Content-Type: text/xml; charset=utf-8
Content-Length: 482
SOAPAction:
"http://www.flamingo.com/books/getOrderStatus"

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

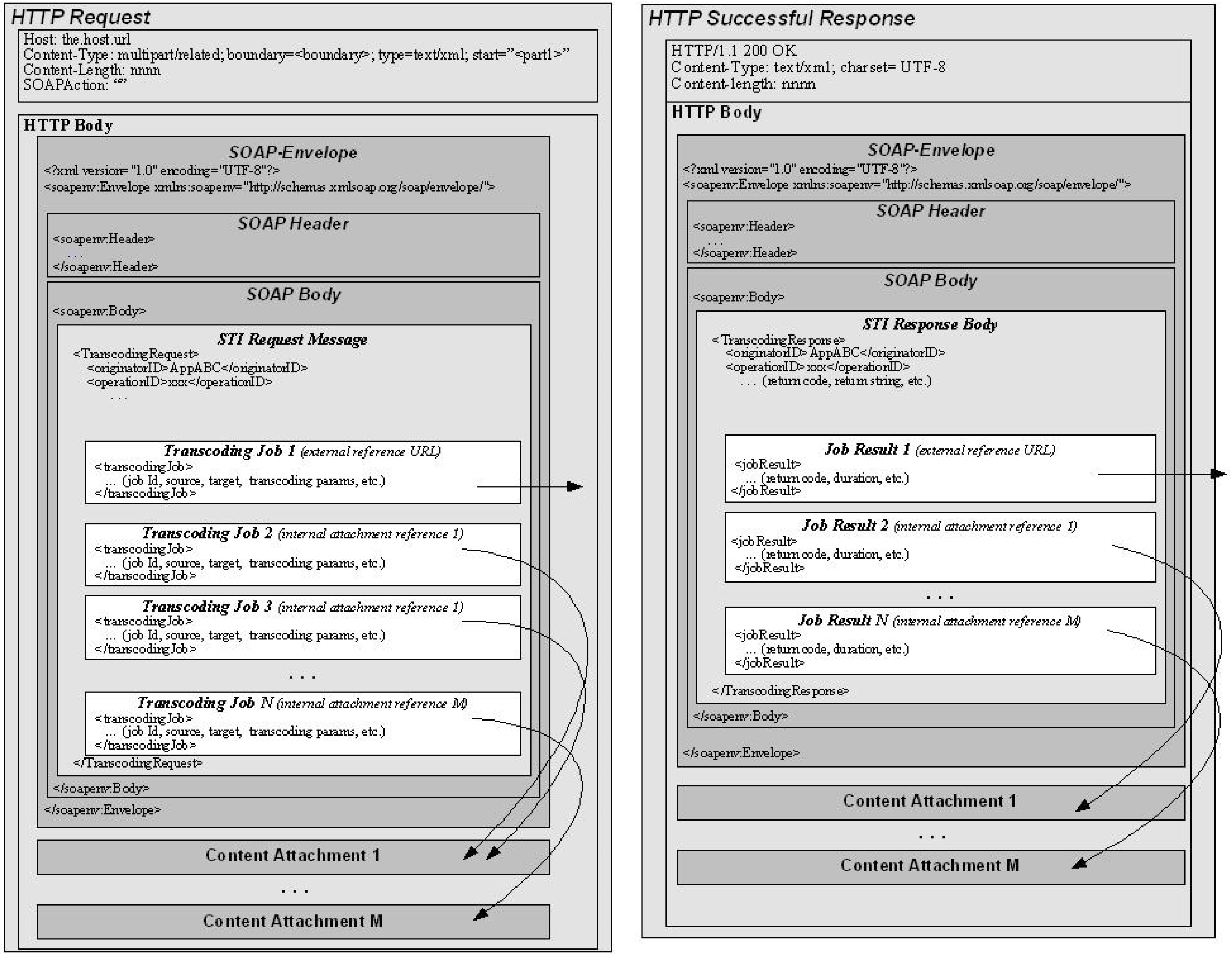
HTTP HEADER of SOAP RESPONSE

- Indicates that it is HTTP version 1.1 message.
- The value ***200 OK*** is an HTTP response success code indicating that message was received and processed successfully.
- Indicates that the connection between the HTTP client and server is closed. The Content-Length field provides the number of characters in the message. The Content-Type field indicates that the message content is XML.
- In case the HTTP server fails to process the SOAP message, an HTTP response message embedded with a SOAP message is sent to the sender. This SOAP error information is included in the ***Fault*** element of Body element of the SOAP message.



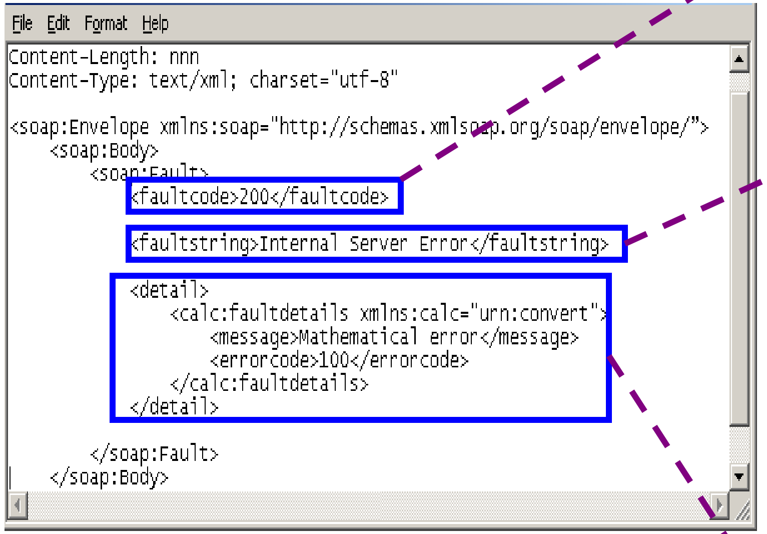
```
HTTP/1.1 200 OK
Connection: close
Content-Length: 659
Content-Type: text/xml; charset=utf-8

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

HTTP HEADER of SOAP RESPONSE


HTTP Request
Host: the.host.url
Content-Type: multipart/related; boundary=<boundary>; type=text/xml; start="<part1>"
Content-Length: nnnn
SOAPAction: ""
HTTP Body
SOAP-Envelope
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
SOAP Header
<soapenv:Header>
. . .
</soapenv:Header>
SOAP Body
<soapenv:Body>
STI Request Message
<TranscodingRequest>
<originatorID>AppABC</originatorID>
<operationID>xxx</operationID>
. . .
Transcoding Job 1 (external reference URL)
<transcodingJob>
... (job Id, source, target, transcoding params, etc.)
</transcodingJob>
Transcoding Job 2 (internal attachment reference 1)
<transcodingJob>
... (job Id, source, target, transcoding params, etc.)
</transcodingJob>
Transcoding Job 3 (internal attachment reference 1)
<transcodingJob>
... (job Id, source, target, transcoding params, etc.)
</transcodingJob>
. . .
Transcoding Job N (internal attachment reference M)
<transcodingJob>
... (job Id, source, target, transcoding params, etc.)
</transcodingJob>
</TranscodingRequest>
</soapenv:Body>
</soapenv:Envelope>
Content Attachment 1
. . .
Content Attachment M
HTTP Successful Response
HTTP/1.1 200 OK
Content-Type: text/xml; charset= UTF-8
Content-length: nnnn
HTTP Body
SOAP-Envelope
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
SOAP Header
<soapenv:Header>
. . .
</soapenv:Header>
SOAP Body
<soapenv:Body>
STI Response Body
<TranscodingResponse>
<originatorID>AppABC</originatorID>
<operationID>xxx</operationID>
. . . (return code, return string, etc.)
Job Result 1 (external reference URL)
<jobResult>
... (return code, duration, etc.)
</jobResult>
Job Result 2 (internal attachment reference 1)
<jobResult>
... (return code, duration, etc.)
</jobResult>
. . .
Job Result N (internal attachment reference M)
<jobResult>
... (return code, duration, etc.)
</jobResult>
</TranscodingResponse>
</soapenv:Body>
</soapenv:Envelope>
Content Attachment 1
. . .
Content Attachment M

FAULT ELEMENT

- Used to transport error and status information.
- Presence is optional.
- Must be a body entry; it can appear only once.
- Has sub four elements as this image:



```
Content-Length: nnn
Content-Type: text/xml; charset="utf-8"

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <soap:Fault>
            <faultcode>200</faultcode>
            <faultstring>Internal Server Error</faultstring>
            <detail>
                <calc:faultdetails xmlns:calc="urn:convert">
                    <message>Mathematical error</message>
                    <errorcode>100</errorcode>
                </calc:faultdetails>
            </detail>

        </soap:Fault>
    </soap:Body>
```

***faultcode*: Used by Web service consumer to identify the fault**

***faultstring*: Used to provide description of the fault**

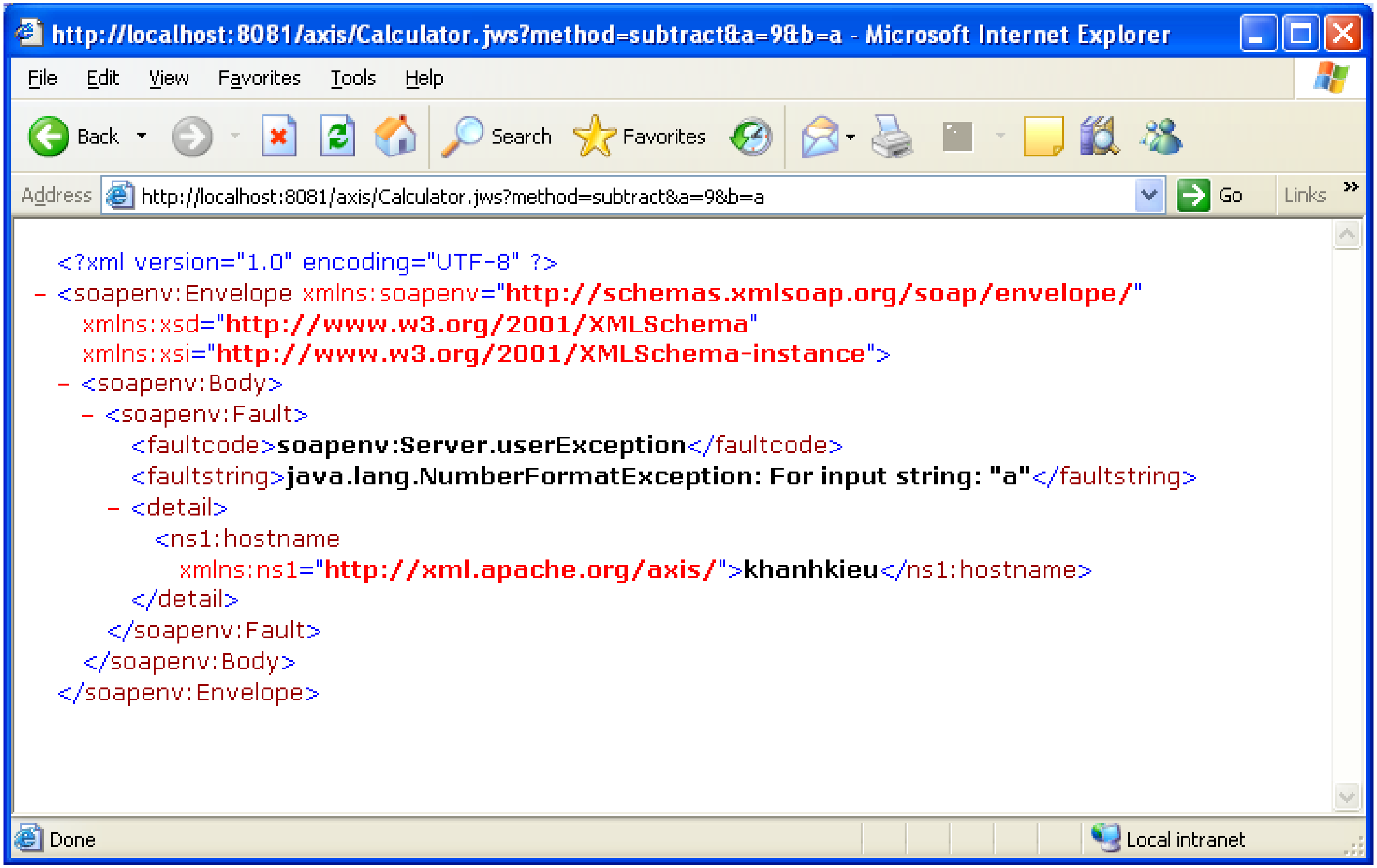
***faultactor*: Used to identify the source of the fault through URI**

***detail*: Holds application-specific error information**

FAULT ELEMENT (cont)

- Fault Code:
  - The faultcode subelement provides error information in the form of fault codes
    - ***VersionMismatch***: the receiving application uses the VersionMismatch fault code to indicate that it received a different version of SOAP message.
    - ***MustUnderstand***: the receiving node or application uses the MustUnderstand fault code to indicate that it was unable to process the header entry targeted for it. This is applicable only if the MustUnderstand attribute in the message received was set 1. In such a case, the receiving node sends a fault message as shown in the image.
    - ***Client***: the client fault code is used to indicate an error in the message or its data. In other words, it indicates an error on the message sender’s end.
    - ***Server***: the Server fault code is used to indicate that the intermediary node or the ultimate receiver was enable to process the message. The reason could be that the receiver was unable to access the database to determine the status of purchase order.
  - SOAP Fault over HTTP:
    - The SOAP fault message is sent using HTTP by embedding it in an HTTP response message.
    - The SOAP error information is included in a fault element. The fault element is enclosed in the Body element.

EXAMPLE


http://localhost:8081/axis/Calculator.jws?method=subtract&a=9&b=a - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://localhost:8081/axis/Calculator.jws?method=subtract&a=9&b=a
Go
Links
Done
Local intranet

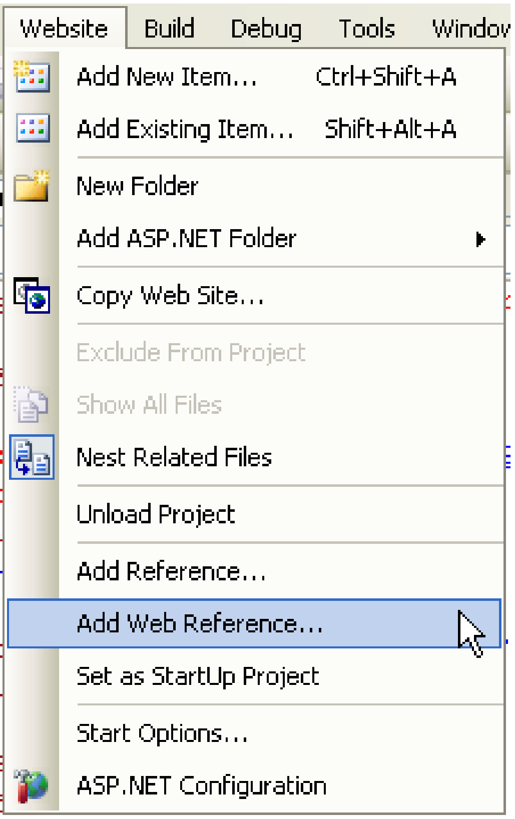
```
  <?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <soapenv:Body>
    - <soapenv:Fault>
        <faultcode>soapenv:Server.userException</faultcode>
        <faultstring>java.lang.NumberFormatException: For input string: "a"</faultstring>
      - <detail>
          <ns1:hostname
            xmlns:ns1="http://xml.apache.org/axis/">khanhkieu</ns1:hostname>
        </detail>
      </soapenv:Fault>
    </soapenv:Body>
  </soapenv:Envelope>
```

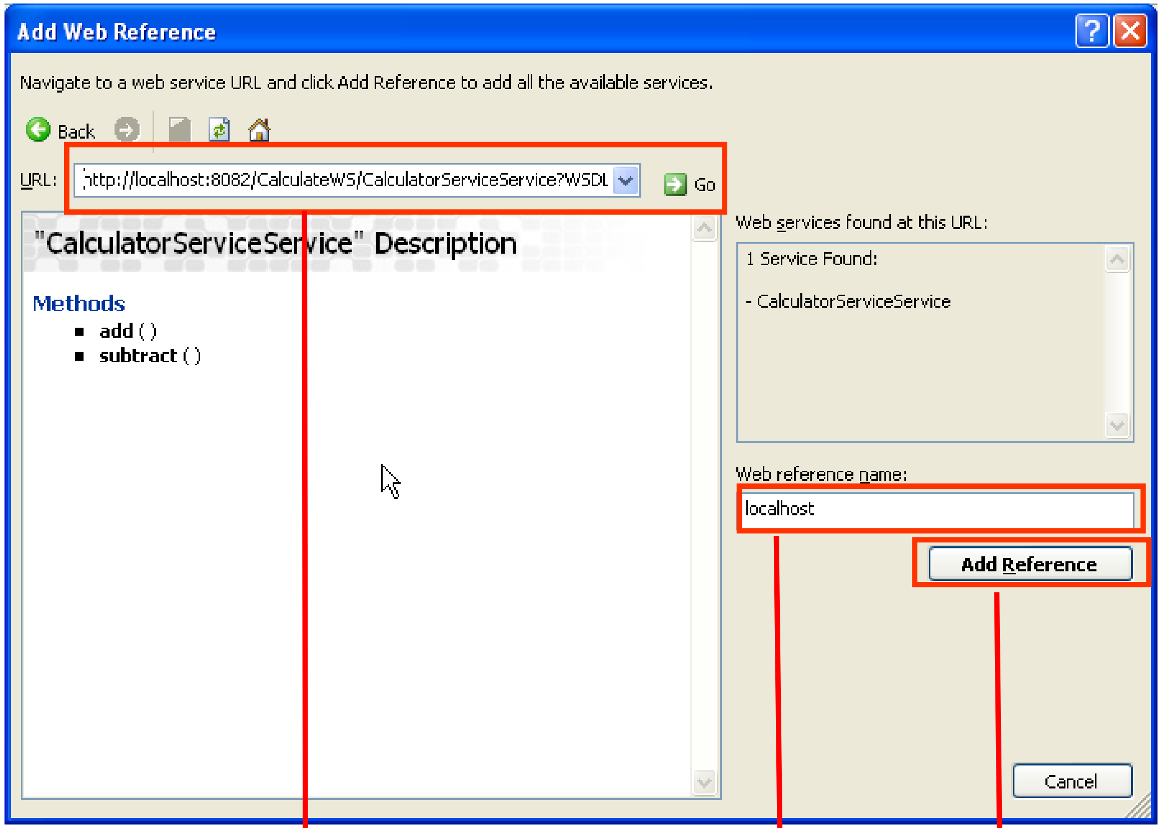
STEPS TO USE .NET CONSUMING WS OF JAVA

- **Step 1**: Creating Web Services using Java, then hosting on the Web Service **(see first slide)**
- **Step 2**: Create Web Application with .NET that is used to consume Web Service’s Java
- **Step 3**: Adding WS to Web Application
- **Step 4**: Running the application

STEPS TO USE .NET CONSUMING WS OF JAVA

- **Step 3**: Adding WS to Web Application
  - Click menu Website, choose "Add Web Reference …"





Navigate to a web service URL and click Add Reference to add all the available services.

URL: http://localhost:8082/CalculateWS/CalculatorServiceService?WSDL

"CalculatorServiceService" Description

Methods
- add ( )
- subtract ( )

Web services found at this URL:

1 Service Found:

- CalculatorServiceService

Web reference name: localhost

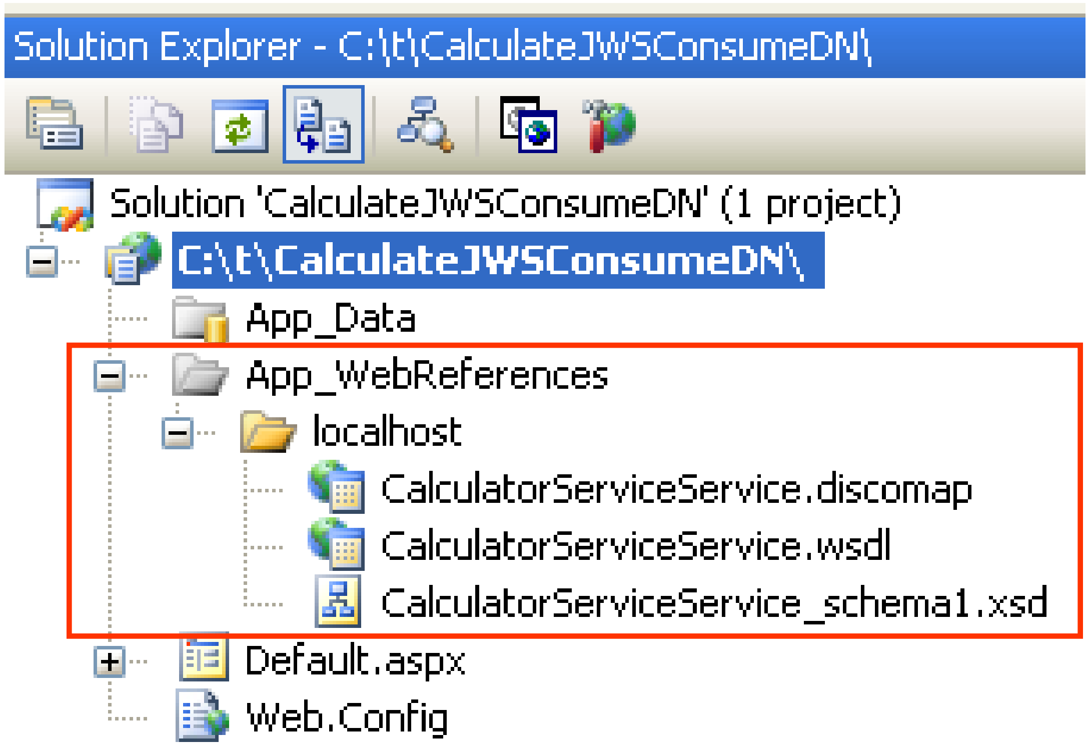
Pass the url endpoint/ address that is deployed WS. Then, click Go

Typing the reference name

Click Add Reference

STEPS TO USE .NET CONSUMING WS OF JAVA

- **Step 3**: Adding WS to Web Application (cont)

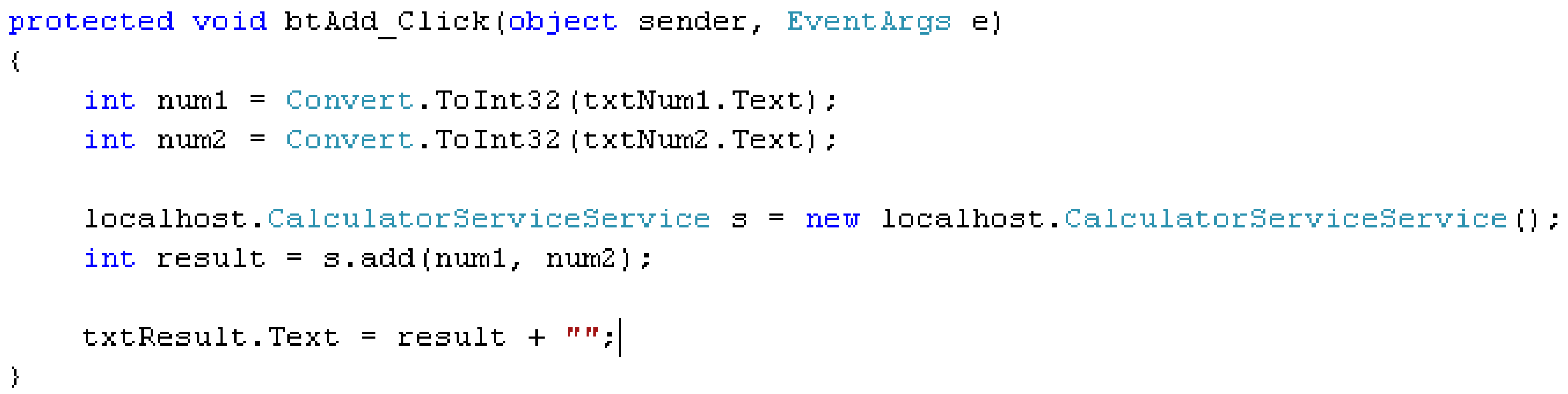


Solution Explorer - C:\t\CalculateJWSConsumeDN\

```
Solution 'CalculateJWSConsumeDN' (1 project)
C:\t\CalculateJWSConsumeDN\
    App_Data
    App_WebReferences
        localhost
            CalculatorServiceService.discomap
            CalculatorServiceService.wsdl
            CalculatorServiceService_schema1.xsd
    Default.aspx
    Web.Config
```

STEPS TO USE .NET CONSUMING WS OF JAVA

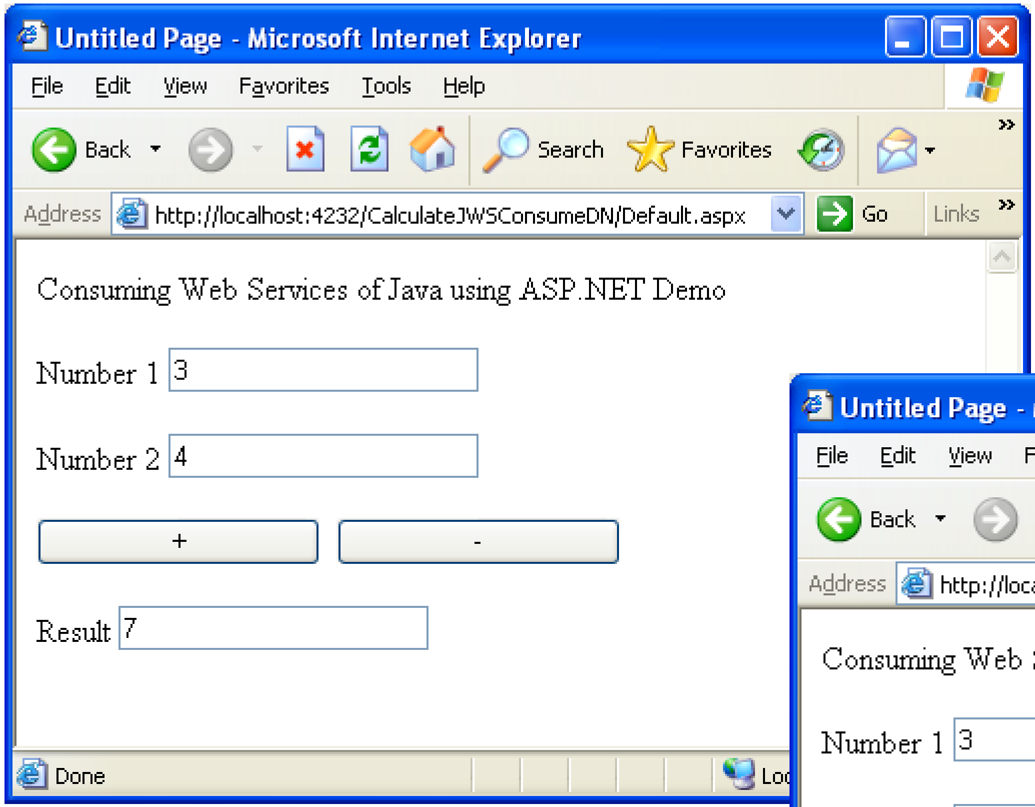
- **Step 3**: Adding WS to Web Application (cont)
  - Designing the GUI, then coding the web application with events with following example as



```
protected void btAdd_Click(object sender, EventArgs e)
{
    int num1 = Convert.ToInt32(txtNum1.Text);
    int num2 = Convert.ToInt32(txtNum2.Text);

    localhost.CalculatorServiceService s = new localhost.CalculatorServiceService();
    int result = s.add(num1, num2);

    txtResult.Text = result + "";
}
```

STEPS TO USE .NET CONSUMING WS OF JAVA

- **Step 4**: Running the application



Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address
Go
Links
+
-
Done

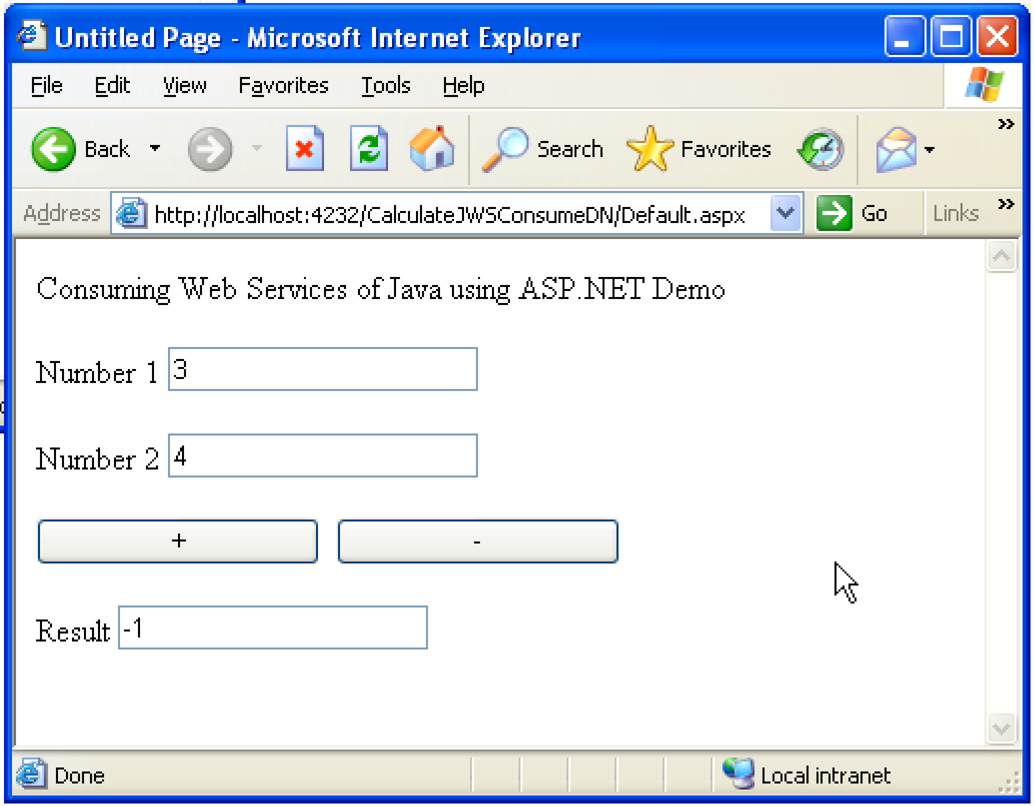
http://localhost:4232/CalculateJWSConsumeDN/Default.aspx

Consuming Web Services of Java using ASP.NET Demo

Number 1: 3

Number 2: 4

Result: 7



Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address
Go
Links
+
-
Done
Local intranet

http://localhost:4232/CalculateJWSConsumeDN/Default.aspx

Consuming Web Services of Java using ASP.NET Demo

Number 1: 3

Number 2: 4

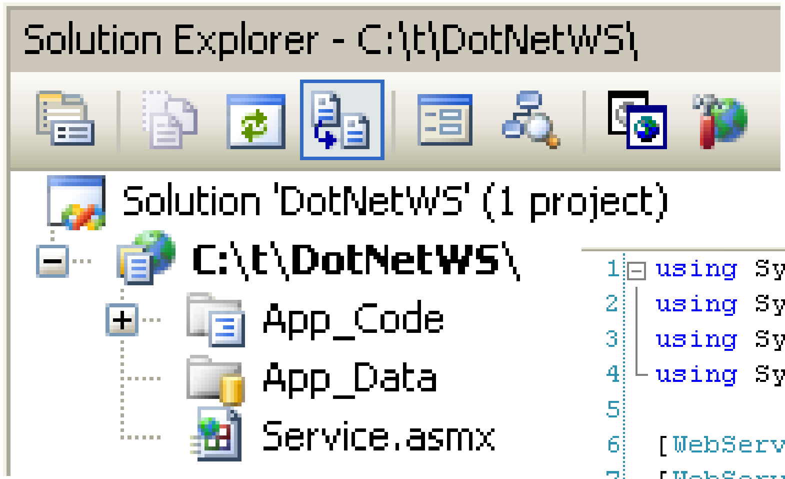
Result: -1

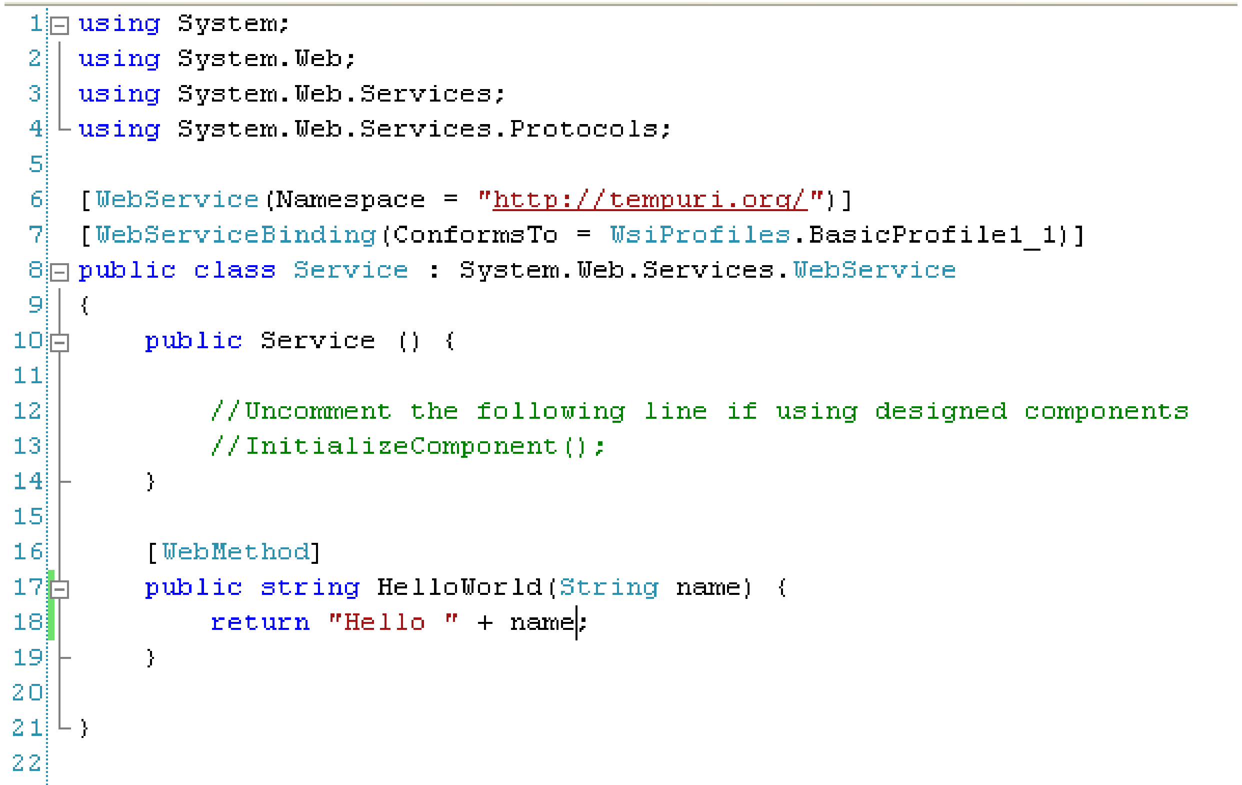
STEPS TO USE JAVA CONSUMING WS OF .NET

- **Step 1**: Creating Web Services using .NET, then hosting on the Web Service on IIS
- **Step 2**: Create Application with Java Technology (JSP, JSF, …) that is used to consume Web Service’s .NET
- **Step 3**: Adding WS to Web Application
- **Step 4**: Running the application

STEPS TO USE JAVA CONSUMING WS OF .NET

- **Step 1**: Creating Web Services using .NET, then hosting on the Web Service on IIS
  - Creating the ASP.NET Web services project

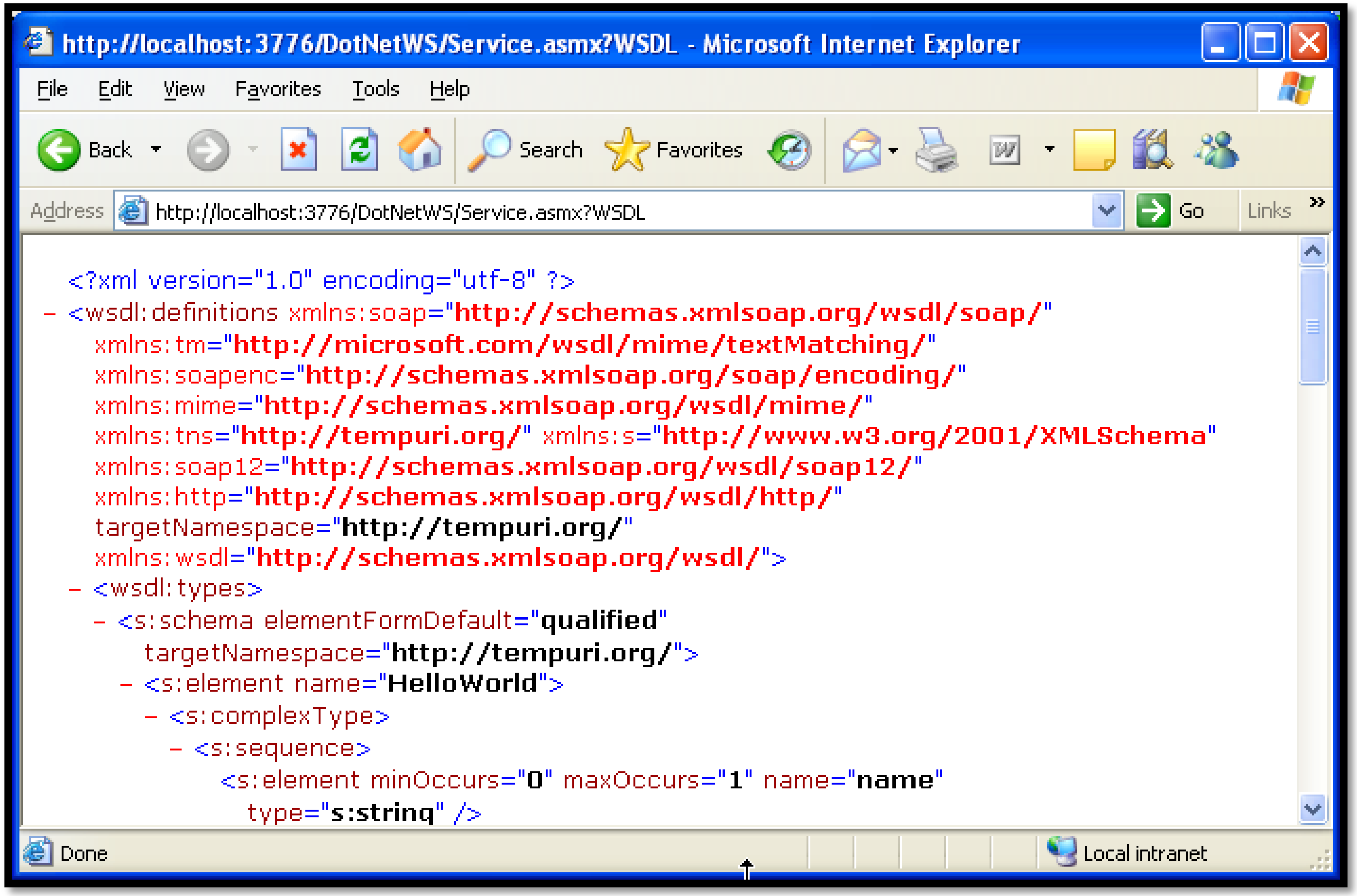

Solution Explorer - C:\t\DotNetWS\
Solution 'DotNetWS' (1 project)
C:\t\DotNetWS\
App_Code
App_Data
Service.asmx



```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {

        //Uncomment the following line if using designed components
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld(String name) {
        return "Hello " + name;
    }

}
```

STEPS TO USE JAVA CONSUMING WS OF .NET

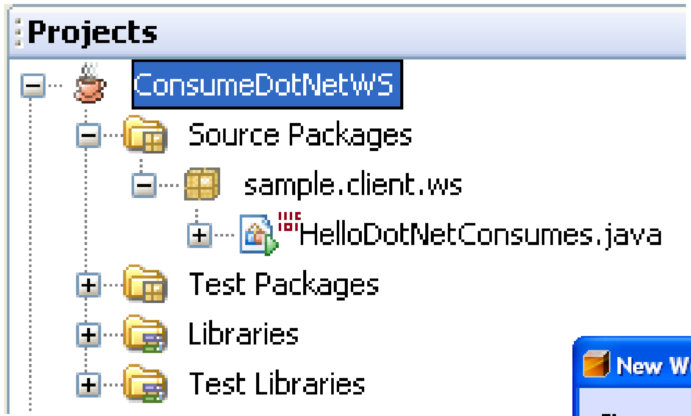
- **Step 1**: Creating Web Services using .NET, then hosting on the Web Service on IIS (cont)
  - Hosting the Web Services on IIS
  - Then typing addition the query string url with ?WSDL

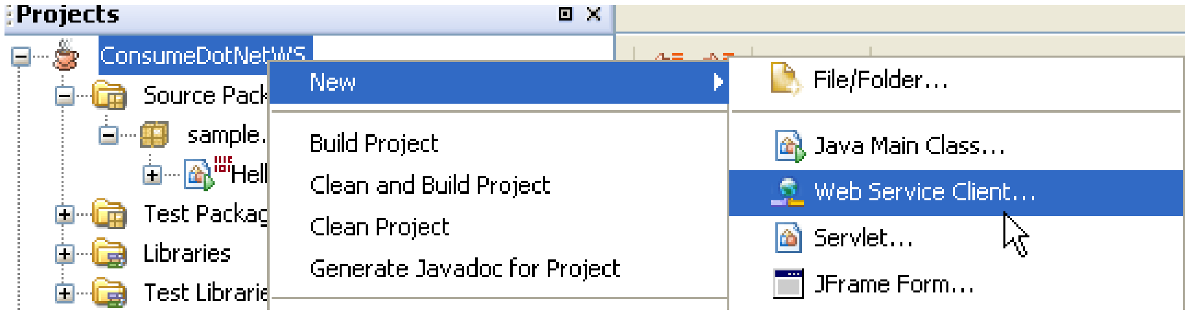

http://localhost:3776/DotNetWS/Service.asmx?WSDL - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:3776/DotNetWS/Service.asmx?WSDL Go Links
Done Local intranet

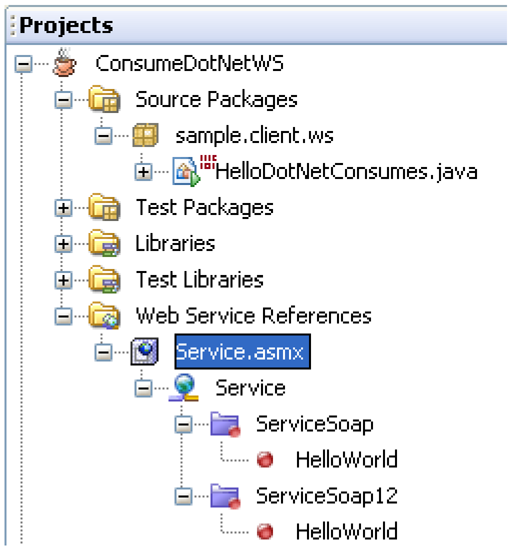
```
  <?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:tns="http://tempuri.org/" xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    targetNamespace="http://tempuri.org/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified"
        targetNamespace="http://tempuri.org/">
      - <s:element name="HelloWorld">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="name"
                type="s:string" />
```

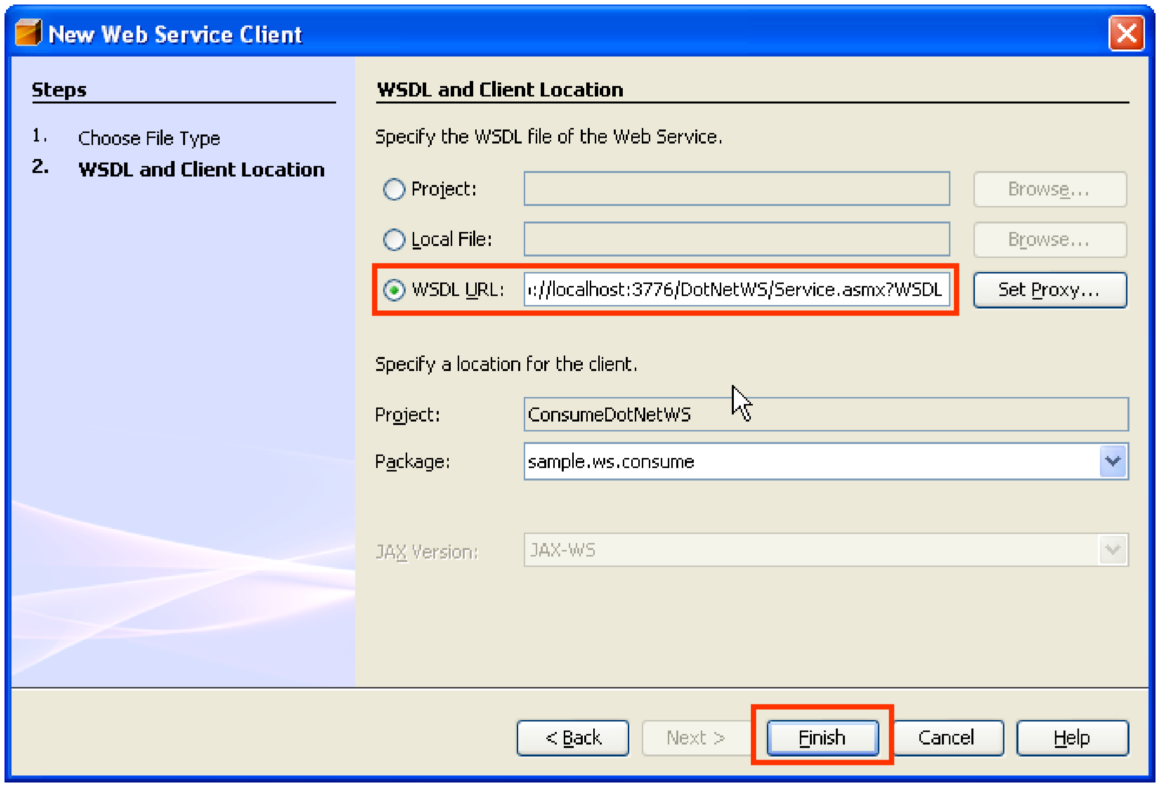
STEPS TO USE JAVA CONSUMING WS OF .NET

- **Step 2, 3:** Creating Application to consume Web Services (see second lecture slide)








New Web Service Client

Steps
1. Choose File Type
2. **WSDL and Client Location**

WSDL and Client Location

Specify the WSDL file of the Web Service.

( ) Project:
( ) Local File:
(•) WSDL URL: ://localhost:3776/DotNetWS/Service.asmx?WSDL

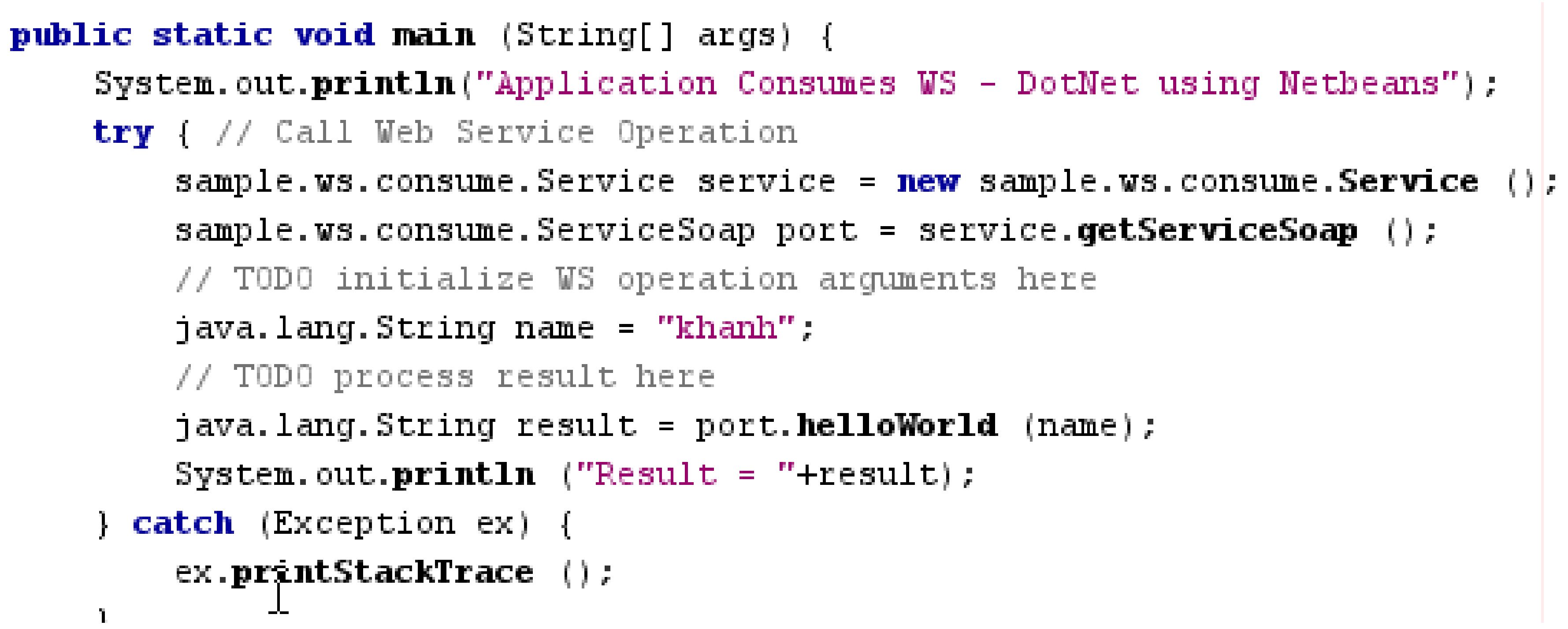
Specify a location for the client.

Project: ConsumeDotNetWS
Package: sample.ws.consume

JAX Version: JAX-WS

STEPS TO CREATE WS USING NETBEANS

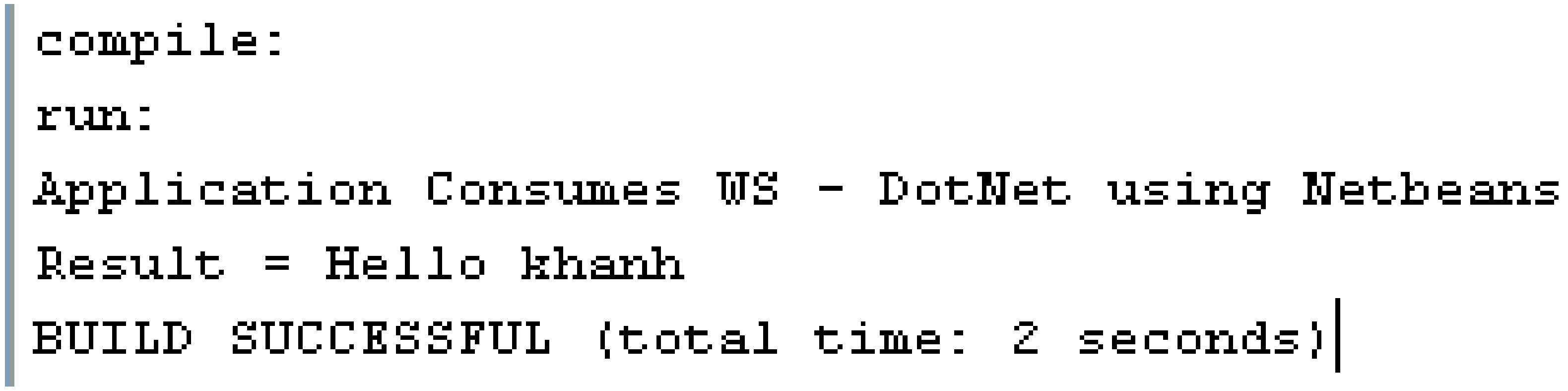
- **Step 3**: Add and consume WS as following example



```
public static void main (String[] args) {
    System.out.println("Application Consumes WS - DotNet using Netbeans");
    try { // Call Web Service Operation
        sample.ws.consume.Service service = new sample.ws.consume.Service ();
        sample.ws.consume.ServiceSoap port = service.getServiceSoap ();
        // TODO initialize WS operation arguments here
        java.lang.String name = "khanh";
        // TODO process result here
        java.lang.String result = port.helloWorld (name);
        System.out.println ("Result = "+result);
    } catch (Exception ex) {
        ex.printStackTrace ();
```

STEPS TO CREATE WS USING NETBEANS

- **Step 4**: Running application



```
compile:
run:
Application Consumes WS - DotNet using Netbeans
Result = Hello khanh
BUILD SUCCESSFUL (total time: 2 seconds)
```

EXERCISES

- Using .NET to consume all exercises in first lesson
- Writing all exercises in first lesson in .NET and using Java application or Web application to consume those