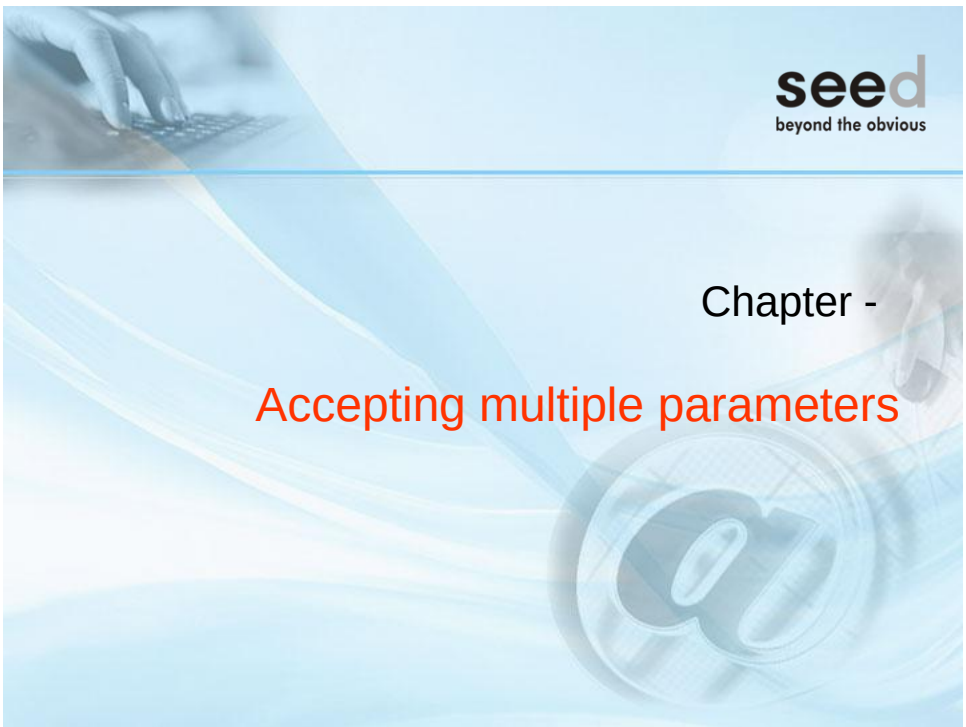


Objectives

- At the end of this chapter you will be able to understand
  - The Wrapped service implementation
  - Interoperability perspective of the wrapped service implementation

Accepting multiple parameters

- Consider the *SimpleServiceImpl* class:

```
public class SimpleServiceImpl implements SimpleServiceSkeletonInterface {
    public ConcatResponse concat(ConcatRequest concatRequest0) {
        String result = concatRequest0.getS1() + concatRequest0.getS2();
        ConcatResponse response = new ConcatResponse();
        response.setConcatResponse(result);
        return response;
    }
}
```

- Because it's a document style web service, we can have a single part in the input message. Therefore, we have a single parameter only. The same is true for the output message.

Accepting multiple parameters

- It would be nice if we could write:

```
public class SimpleServiceImpl implements SimpleServiceSkeletonInterface {
    public String concat(String s1, String s2) {
      return s1+s2;
    }
}
```

- while still accepting a single part (<concatRequest>) in the message. To do that, we just need to make two changes to the WSDL file:

Accepting multiple parameters- WSDL file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ...>
  <wsdl:types>
    <xsd:schema ...>
      <xsd:element name="concatRequest concat">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="s1" type="xsd:string" />
            <xsd:element name="s2" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="concatResponse" type="xsd:string" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="concatRequest">
    <wsdl:part name="parameters" element="tns:concatRequest concat" />
  </wsdl:message>
  <wsdl:message name="concatResponse">
    <wsdl:part name="parameters" element="tns:concatResponse" />
  </wsdl:message>
  <wsdl:portType name="SimpleService">
    <wsdl:operation name="concat">
      <wsdl:input message="tns:concatRequest" />
      <wsdl:output message="tns:concatResponse" />
    </wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definitions>
```

The element must be a sequence, which is indeed the case here.

Make sure the element name of that single part in the input message is the same as that of the operation.

seed
beyond the obvious

Accepting multiple parameters parameters - WSDL file

- Similarly, for the output message, the element name must be the name of the *operation* with the word "*Response*" appended and it must be a sequence (containing a single child element):

seed
beyond the obvious

Accepting multiple parameters parameters- WSDL file

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ...>
   <wsdl:types>
      <xsd:schema ...>
         <xsd:element name="concat">
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="s1" type="xsd:string" />
                  <xsd:element name="s2" type="xsd:string" />
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name="concatResponse" type="xsd:string" >
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name="r" type="xsd:string" />
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:schema>
   </wsdl:types>
   <wsdl:message name="concatRequest">
      <wsdl:part name="parameters" element="tns:concat" />
   </wsdl:message>
   <wsdl:message name="concatResponse">
      <wsdl:part name="parameters" element="tns:concatResponse" />
   </wsdl:message>
   <wsdl:portType name="SimpleService">
      <wsdl:operation name="concat">
         <wsdl:input message="tns:concatRequest" />
         <wsdl:output message="tns:concatResponse" />
      </wsdl:operation>
   </wsdl:portType>
   ...
</wsdl:definitions>
```

It must not be a simple type such as string. It must be a sequence.

The sequence must contain a single element. The element name (<r> here) is unimportant.

The element name must be "concat" + "Response", which happens to be the case already.

seed
beyond the obvious

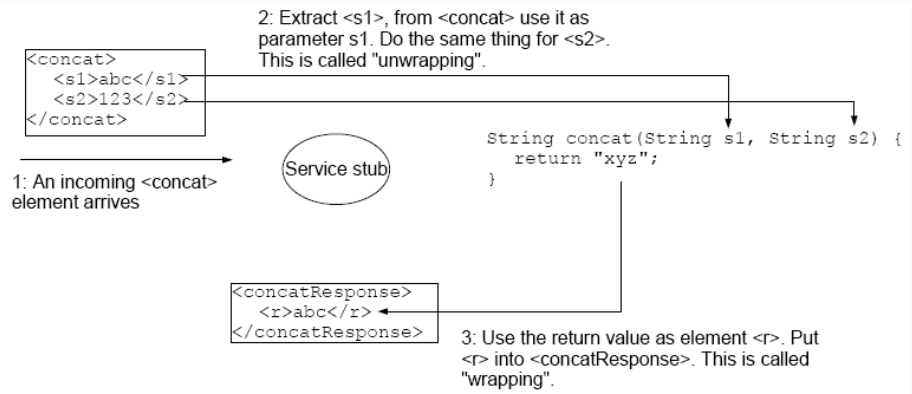
Accepting multiple parameters parameters- WSDL file

- Save the file *SimpleService.wsdl* as *WrappedService.wsdl* and modify it accordingly.

seed
beyond the obvious

Accepting multiple parameters

- We need a service stub that performs some special processing as shown:



seed
beyond the obvious

Accepting multiple parameters

- Note that this service is still a 100% document style service.
- The clients can still call it the same way
- The difference is how the service stub calls our implementation and how it handles our return value.
- There is no difference seen by the client.
- To generate such a service stub, add an option to the <wsdl2code> Ant task:

seed
beyond the obvious

Accepting multiple parameters – build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="jar.server">
   ...
   <target name="generate-service">
      <wsdl2code
         wsdlfilename="${name}.wsdl"
         serverside="true"
         generateservicexml="true"
         skipbuildxml="true"
         serversideinterface="true"
         namespacetopackages="http://ttdev.com/ss=com.ttdev.wrap"
         targetsourcefolderlocation="src"
         targetresourcesfolderlocation="src/META-INF"
         overwrite="true"
         unwrap="true" />
      <replaceregexp
         file="src/META-INF/services.xml"
         match="${name}Skeleton"
         replace="${name}Impl" />
   </target>
   <target name="generate-client">
      <wsdl2code
         wsdlfilename="${name}.wsdl"
         skipbuildxml="true"
         namespacetopackages="http://ttdev.com/ss=com.ttdev.wrap.client"
         targetsourcefolderlocation="src"
         overwrite="true"
         unwrap="true" />
   </target>
</project>
```

Generate a service stub that performs wrapping and unwrapping

Generate a client stub that performs wrapping and unwrapping

seed
beyond the obvious

Accepting multiple parameters- Interoperability

- The wrapped convention is a good idea. It is the only kind of web service supported by the .NET framework. Obviously Axis has also implemented this convention.
- From the viewpoint of the caller, it is just a document+literal style service. So if the caller doesn't understand the wrapped convention, it can still access it as a regular document style service.

seed
beyond the obvious

Quick Recap . . .

- We can use the wrapped convention so that our back end Java method can have multiple parameters. The clients understanding this convention can also call it using multiple parameters. For those not understanding it, they can still call it as a regular document style service.
- To ensure interoperability with .NET, we should use this convention