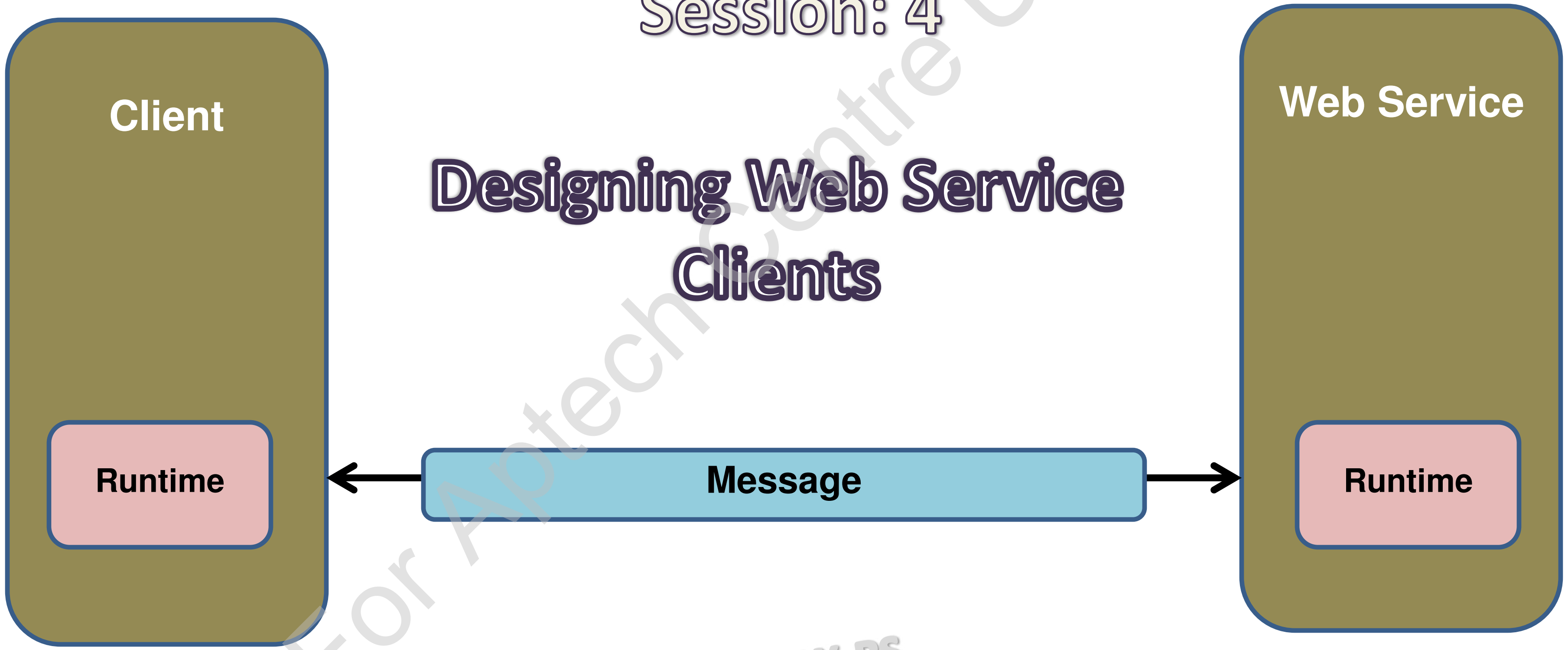
Developing Java Web Services

Session: 4

Designing Web Service Clients



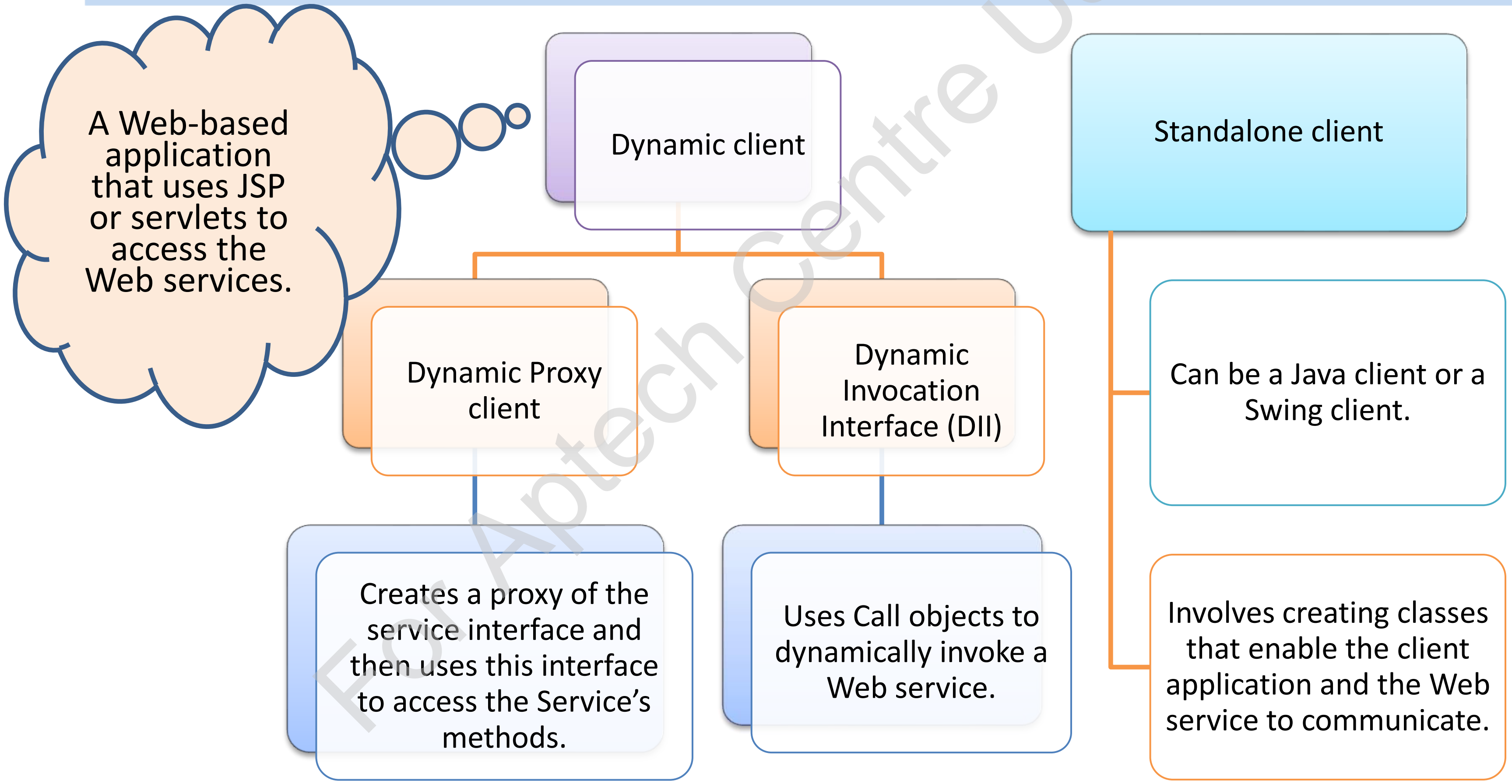
Objectives

- Explain Web Service Clients
- Describe the Modes of Communication
- Explain how to Locate and Access the Web Services
- Explain how to handle Web Service Exceptions

Web Services Clients

Web Service

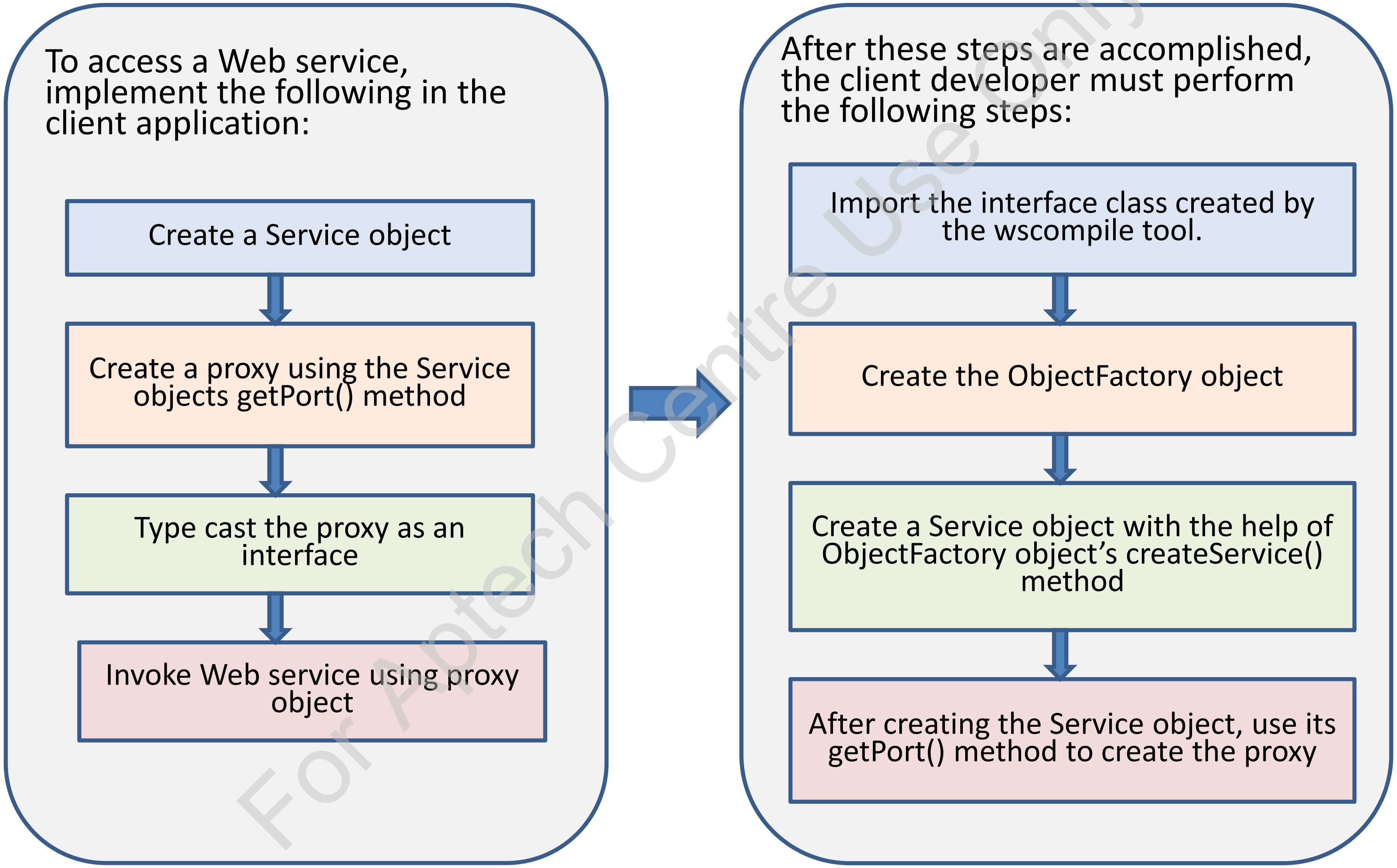
- ❑ Is invoked when the client accesses the Web service.
- ❑ Allows users to access it through standalone clients or dynamic clients.


A Web-based application that uses JSP or servlets to access the Web services.
Dynamic client
Dynamic Proxy client
Dynamic Invocation Interface (DII)
Creates a proxy of the service interface and then uses this interface to access the Service's methods.
Uses Call objects to dynamically invoke a Web service.
Standalone client
Can be a Java client or a Swing client.
Involves creating classes that enable the client application and the Web service to communicate.

Features of Dynamic Proxy Clients

- Allows accessing the Web services by using their proxy interface
- Makes the client applications more portable
- Provides the ease of development and increased performance rate as compared to a standalone client
- Is best suited to the applications that require being portable

Creating Dynamic Proxy Clients 1-4



To access a Web service, implement the following in the client application:

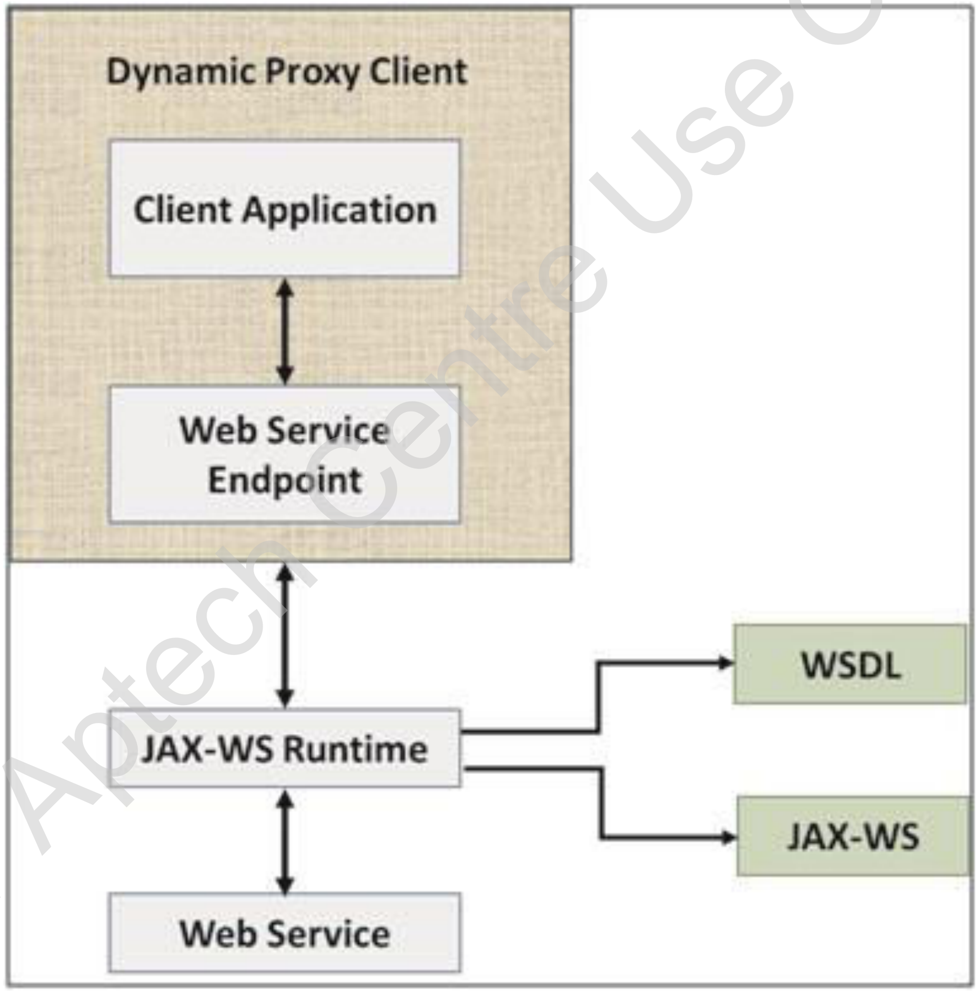
1. Create a Service object
2. Create a proxy using the Service objects getPort() method
3. Type cast the proxy as an interface
4. Invoke Web service using proxy object

After these steps are accomplished, the client developer must perform the following steps:

1. Import the interface class created by the wscompile tool.
2. Create the ObjectFactory object
3. Create a Service object with the help of ObjectFactory object's createService() method
4. After creating the Service object, use its getPort() method to create the proxy

Creating Dynamic Proxy Clients 2-4

- Following figure shows the use of Dynamic Proxy Client:



Creating Dynamic Proxy Clients 3-4

- Following code snippet demonstrates the use of ObjectFactory object:

```
@XmlRegistry
public class ObjectFactory {
private final static QName _Add_QNAME = new
QName("http://ws.soap.djws.com/", "add");
private final static QName _AddResponse_QNAME = new
QName("http://ws.soap.djws.com/", "addResponse");

/*** Create a new ObjectFactory that can be used to create new instances
of schema derived classes for package: com.djws.soap.ws **/
public ObjectFactory() {
}
/*** Create an instance of {@link Add } **/
public Add createAdd() {
return new Add();
}
 /*** Create an instance of {@link AddResponse } **/
```

Creating Dynamic Proxy Clients 4-4

```
public AddResponse createAddResponse() {
return new AddResponse();
}
/*** Create an instance of {@link JAXBElement }{@code <}{@link Add }{@code
>}} **/
@XmlElementDecl(namespace = "http://ws.soap.djws.com/", name =
"add")
public JAXBElement<Add> createAdd(Add value) {
return new JAXBElement<Add>(_Add_QNAME, Add.class, null, value);
}
/*** Create an instance of {@link JAXBElement }{@code <}{@link AddResponse
}{@code >}}**/
@XmlElementDecl(namespace = "http://ws.soap.djws.com/", name =
"addResponse")
public JAXBElement<AddResponse> createAddResponse(AddResponse value){
return new JAXBElement<AddResponse>(_AddResponse_QNAME,
AddResponse.class, null, value);
}
 }
```

Features of DII

Enables clients to invoke the methods of Web services, which are unknown at the compile time.

Allows the client application to lookup methods dynamically and access them through Call objects.

Provides the client developer with complete control over the client application.

Is the only method that supports one-way invocation.

Drawbacks

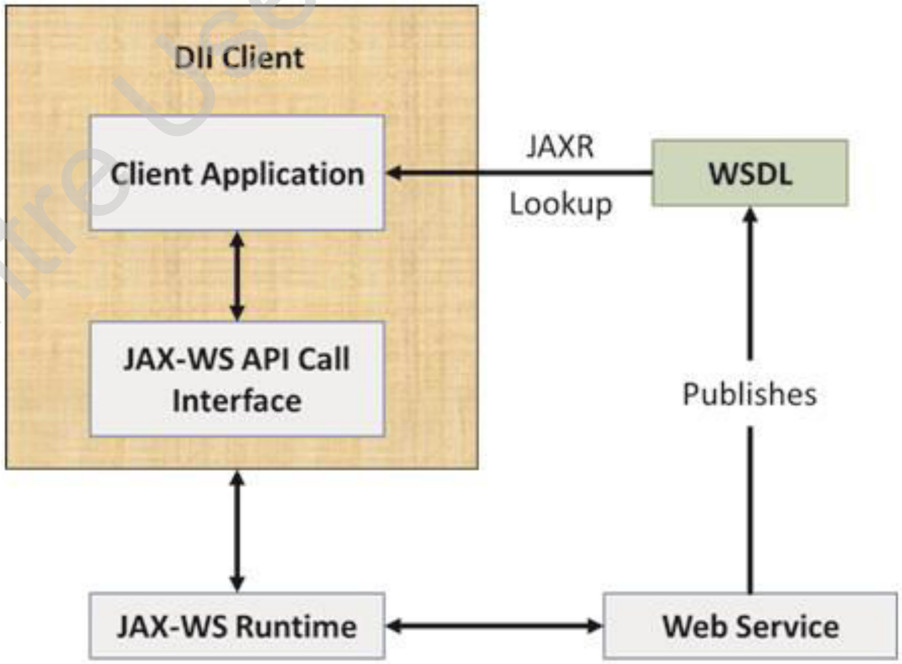
- The client developer needs to develop very complicated code.
- The effort for the same is much greater than the Static Stub and Dynamic Proxy methods.

Creating DII Clients 1-4

- To access a Web service using the DII method, perform the following tasks:

1. Create a Service object.
2. Create a QName class instance using the generated interface class.
3. Create the Call object and set its address.
4. Set the Call object properties.
5. Set Web service operation name.
6. Invoke Web service method.

- Following figure shows the use of DII client:



Creating DII Clients 2-4

- Following code snippet shows creation of a DII client:

```
WebServiceClient(name = "CalculatorWS", targetNamespace =
"http://ws.soap.syskan.com/", wsdlLocation =
"http://localhost:8080/SOAPWebService/CalculatorWS?wsdl")

public class CalculatorWS_Service extends Service{

private final static URL CalculatorWS_WSDL_LOCATION;
private final static WebServiceException CalculatorWS_EXCEPTION;
private final static QName CalculatorWS_QNAME = new
QName("http://ws.soap.djws.com/", "CalculatorWS");

static {
URL url = null;
WebServiceException e = null;
try {
  url = new URL("http://localhost:8080/SOAPWebService/CalculatorWS?wsdl");
}
catch (MalformedURLException ex) {
   e = new WebServiceException(ex);
}
```

Creating DII Clients 3-4

```
CalculatorWS_WSDL_LOCATION = url;
CalculatorWS_EXCEPTION = e;
}
public CalculatorWS_Service(){
super(__getWsdlLocation(),CalculatorWS_QNAME);
}
public CalculatorWS_Service(WebServiceFeature... features) {
super(__getWsdlLocation(),CalculatorWS_QNAME, features);
}
public CalculatorWS_Service(URL wsdlLocation) {
super(wsdlLocation, CalculatorWS_QNAME);
}
public CalculatorWS_Service(URL wsdlLocation, WebServiceFeature...
features) {
super(wsdlLocation, CalculatorWS_QNAME, features);
}
public CalculatorWS_Service(URL wsdlLocation, QName serviceName) {
super(wsdlLocation, serviceName);
}
public CalculatorWS_Service(URL wsdlLocation, QName serviceName,
WebServiceFeature... features) {
```

Creating DII Clients 4-4

```
super(wsdlLocation, serviceName, features);
}
/**** @return returns CalculatorWS**/
@WebEndpoint(name = "CalculatorWSPort")
public SOAPWS getCalculatorWSPort() {
return super.getPort(new QName("http://ws.soap.djws.com/",
"CalculatorWSPort"), CalculatorWS.class);
}
/**** @param features - A list of {@link javax.xml.ws.WebServiceFeature} to
configure on the proxy. Supported features not in the <code>features</code>
parameter will have their default values. * @return returns CalculatorWS**/
@WebEndpoint(name = "CalculatorWSPort")
public SOAPWS getCalculatorWSPort(WebServiceFeature... features) {
return super.getPort(new QName("http://ws.soap.syskan.com/",
"CalculatorWSPort"), CalculatorWS.class, features);
}
private static URL __getWsdlLocation(){
if (CalculatorWS_EXCEPTION!= null) {
throw CalculatorWS_EXCEPTION;
}
return CalculatorWS_WSDL_LOCATION;
}
```

Features of Standalone Clients

Provides the best performance compared to the other two dynamic approaches of accessing Web services

Allows developers to directly code their clients against the class

Is the easy to code and implement

Drawbacks

- ❑Major changes in the service interface could result in the development of the entire client application all over again.
- ❑Using the standalone client approach in applications where the service interface frequently changes could heavily cost client developers in terms of development effort.

Creating Standalone Clients 1-2

- To access a Web service using the standalone client, a client developer needs to implement the following in the client application:

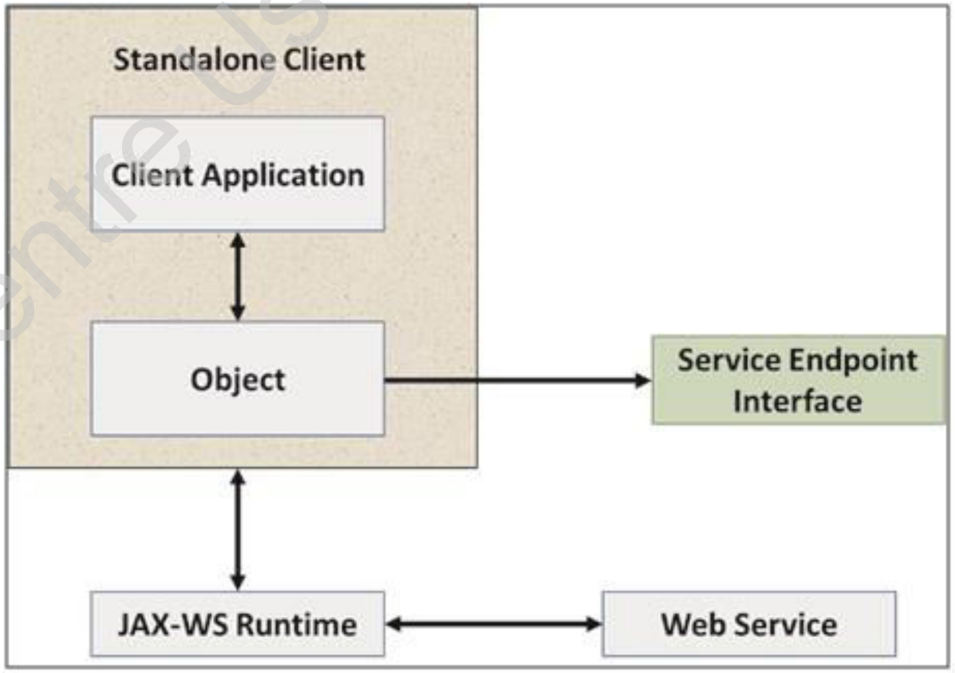
1
- Create a Java class to invoke the Web service method.

2
- Typecast the object as an interface.

3
- Set the endpoint address for the generated object.

4
- Access Web service methods using generated object

- Following figure shows a standalone client:

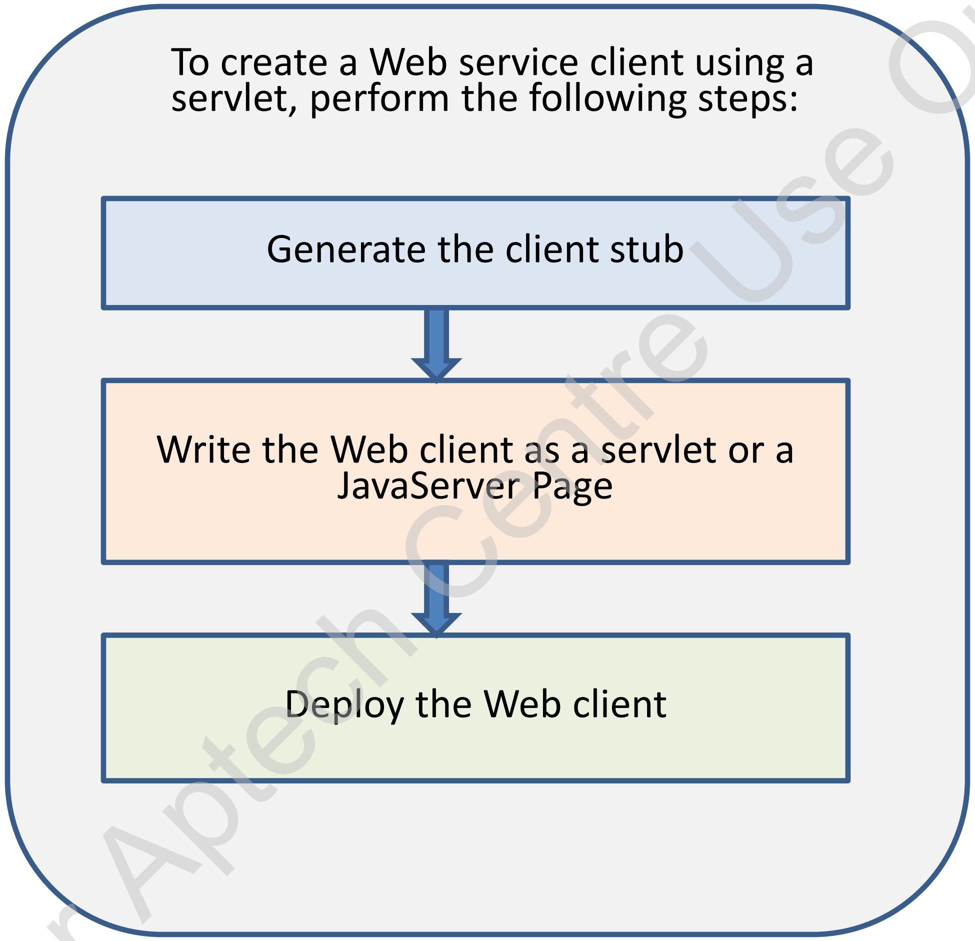


Creating Standalone Clients 2-2

- Following code snippet shows creation of a static stub client:

```
//stand alone client
...
public static void main(String[] args) {
try {
int num1 = 5;
int num2 = 7;
int result = addNum(num1, num2);
System.out.println("Result = " + result);
}
 catch (Exception ex){ System.out.println("Exception: " + ex);
}
}
private static int addNum(int i, int j)
{
com.djws.soap.ws.CalculatorWS_Service service = new
com.djws.soap.ws. CalculatorWS_Service();
com.djws.soap.ws.CalculatorWS port = service.getCalculatorWSPort();
return port.add(i, j);
}
…
```

Creating Web Tier Clients 1-5



To create a Web service client using a servlet, perform the following steps:

Generate the client stub

Write the Web client as a servlet or a JavaServer Page

Deploy the Web client

Creating Web Tier Clients 2-5

- Following code snippet shows creation of Web client as a servlet:

```
// Servlet Web client


...
@WebServlet(name = "JAXWSServlet", urlPatterns = {"/JAXWSServlet"})
public class JAXWSServlet extends HttpServlet {
    @WebServiceRef(wsdlLocation = "WEB-
INF/wsdl/localhost_8080/CalculatorWS/CalculatorWS.wsdl")
    private CalculatorWS_Service service;


    /**
     * Processes requests for both HTTP <code>GET</code> and
<code>POST</code>
     * methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
* @throws IOException if an I/O error occurs
     */
```

Creating Web Tier Clients 3-5

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
  /* TODO output your page here. You may use following sample code. */
    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet JAXWSServlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<form method=\"post\" action=\"\">");
    out.println("<h2>Consuming CalculatorWS Web Service Using a
Servlet Client</h2>");
out.println("First number: <input type=\"text\" id=\"txtbox1\"
name=\"num1\" size=\"6\" maxlength=\"6\" value=\"\">");
     out.println("<br></br>");
out.println("Second number: <input type=\"text\" id=\"txtbox2\"
name=\"num2\" size=\"6\" maxlength=\"6\"value=\"\" >");
out.println("<br></br>");
  out.println("<input type=\"submit\" name=\"btn\" value=\"Add\">");
```

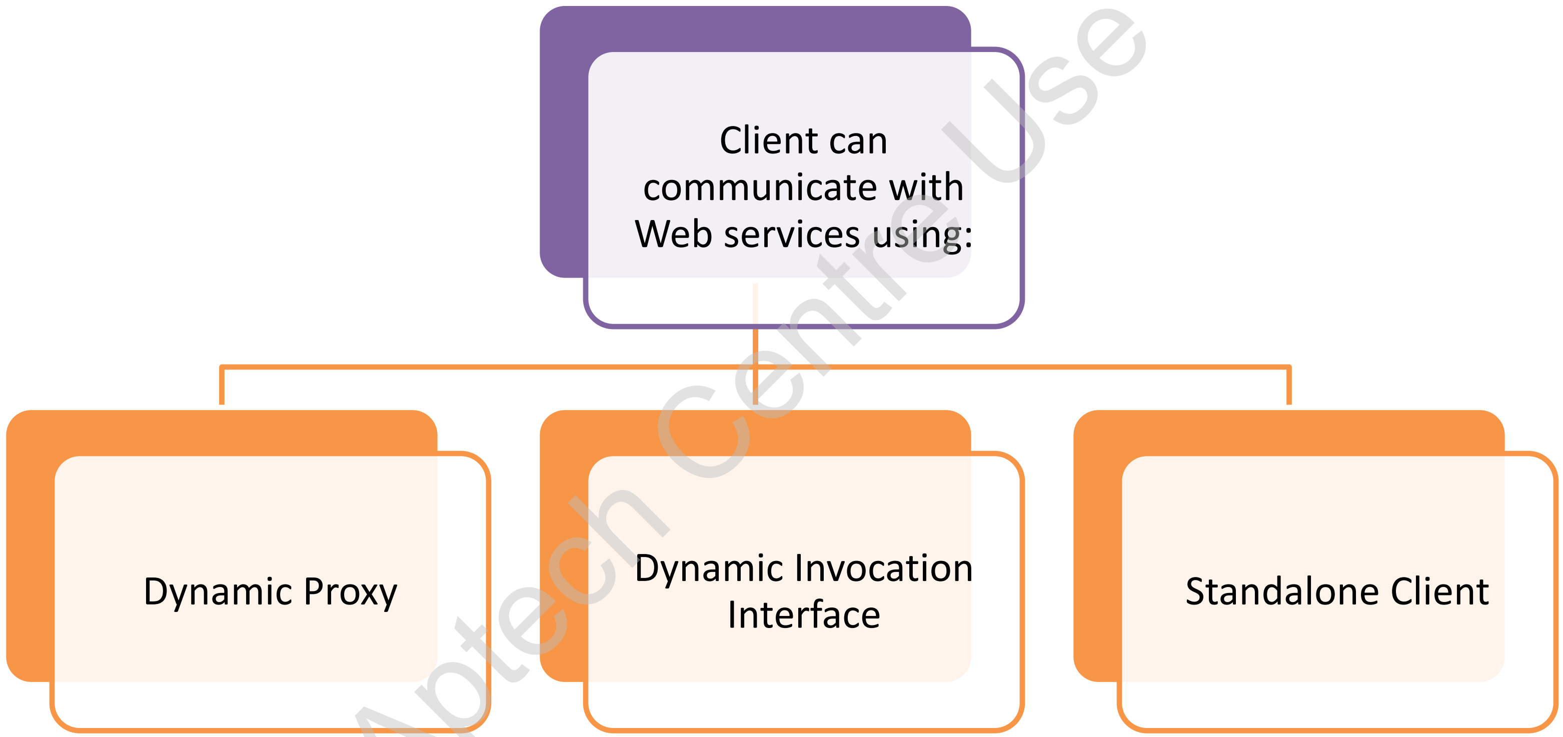
Creating Web Tier Clients 4-5

```
out.println("<br></br>");
   int i=0;
  int j=0;
  if (request.getParameter("btn").equalsIgnoreCase("Add")){

    if(request.getParameter("num1").isEmpty() ||
request.getParameter("num2").isEmpty()){
    out.println("Specify the two numbers to be added");
   }else {
          i = Integer.parseInt(request.getParameter("num1"));
         j = Integer.parseInt(request.getParameter("num2"));

         int result = addNum(i,j);
         out.println("<h3><u>Result</u></h3>" );
         out.println("Sum of the two numbers is " + result);
         out.println("</form>");
         out.println("</body>");
         out.println("</html>");
          }
      }
```

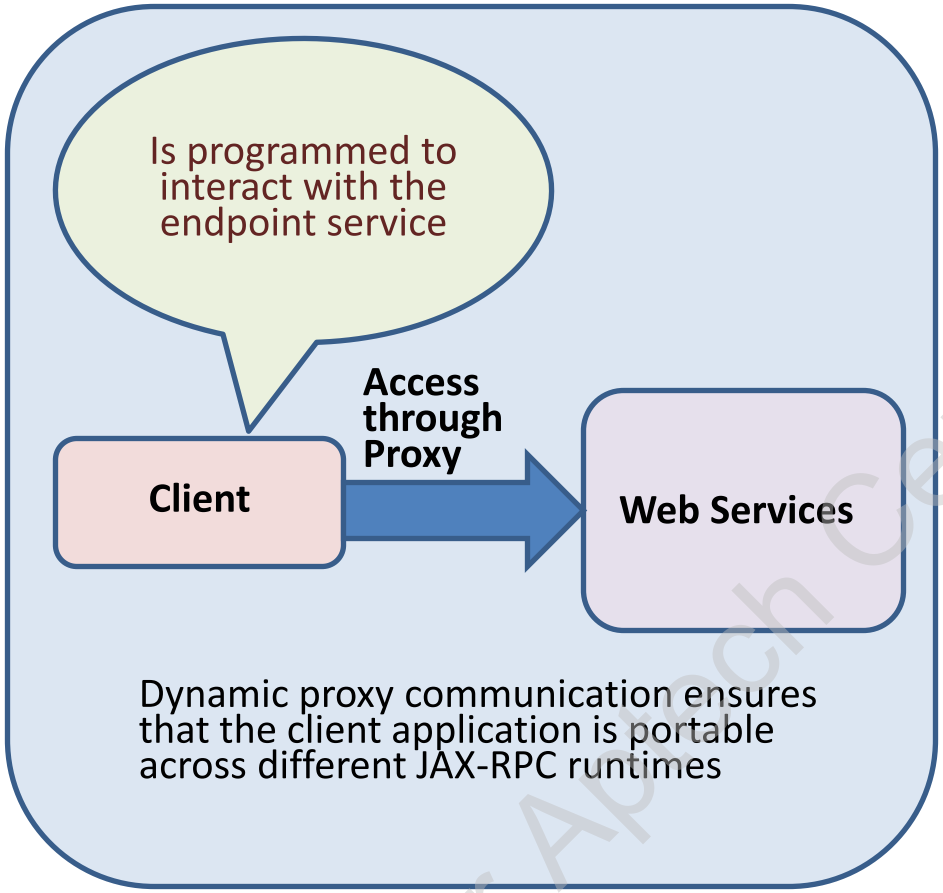
Creating Web Tier Clients 5-5

```
      }
    }
...
  private int addNum(int num1, int num2) {
        // Note that the injected javax.xml.ws.Service reference as well
as port objects are not thread safe.
// If the calling of port operations may lead to race condition some
synchronization is required.
com.djws.CalculatorWS port = service.getCalculatorWSPort();
return port.add(num1, num2);
}
}
```

Modes of Communication


Client can communicate with Web services using:
Dynamic Proxy
Dynamic Invocation Interface
Standalone Client

Using Dynamic Proxy Communication



Dynamic proxy communication ensures that the client application is portable across different JAX-RPC runtimes

JAX-RPC runtime

- ❑Is responsible for all communications between the service endpoint interface and the Web service
- ❑Uses WSDL document and the JAX-RPC mapping files

Using DII Communication

DII Approach

- Enables clients to dynamically access Web services through programmatic invocation of JAX-WS requests
- Uses a tool to create the service endpoint interface and other necessary class files based on the Web service WSDL document
- Uses JAX-RPC APIs
- Uses JAXR to search the registry for services
- Uses JAX-RPC for its communication with the service endpoint interface

Using Standalone Communication

JAX-WS Runtime

- Generates a local object that acts as a proxy for the service endpoint
- Enables communication between the client and the service
- Converts client requests to SOAP messages and passes them onto the service
- Reconverts the SOAP messages it receives from the service and passes them back as responses to the client

Locating and Accessing Web Services

- Following code snippet demonstrates how to locate and access a Web service:

```
@WebMethod
@WebResult(targetNamespace = "")
@RequestWrapper(localName = "add", targetNamespace =
"http://ws.soap.djws.com/", className =
"com.djws.soap.ws.Add")
@ResponseWrapper(localName = "addResponse", targetNamespace =
"http://ws.soap.djws.com/", className =
"com.djws.soap.ws.AddResponse")
@Action(input = "http://ws.soap.djws.com/CalculatorWS/addRequest", output
=
"http://ws.soap.djws.com/CalculatorWS/addResponse")
public int add(
@WebParam(name = "num1", targetNamespace = "")
int num1;
@WebParam(name = "num2", targetNamespace = "")
int num2;
);
```

Web Service Exceptions 1-2

Exceptions are errors or unexpected events encountered by an executing program.

When an exception occurs during the execution of Web service, the Web service is expected to capture not only the exception but also inform the client about the exception.

Web Service Endpoint Exceptions

System

Service-Specific

Are all exceptions thrown by errors beyond the control of the application

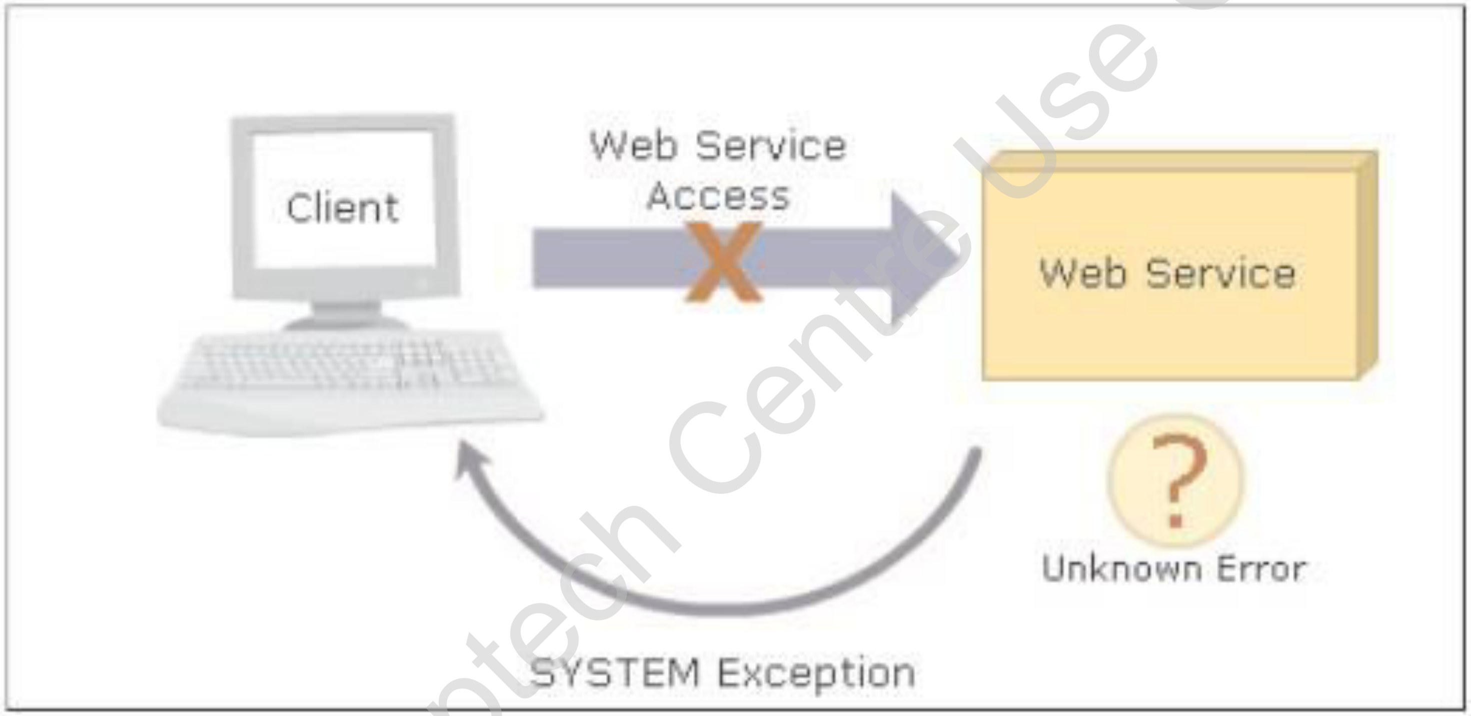
Are all unanticipated errors that can occur at the time of service method invocation

Can be handled by the client applications by:

- ❑prompting the user to retry
- ❑displaying the kind of exception occurred
- ❑translating the system exception to an unchecked exception

Web Service Exceptions 2-2

- Following figure shows the types of Web service exceptions:



System Exceptions in Web Service 1-2

- Different types of system exceptions are as follows:

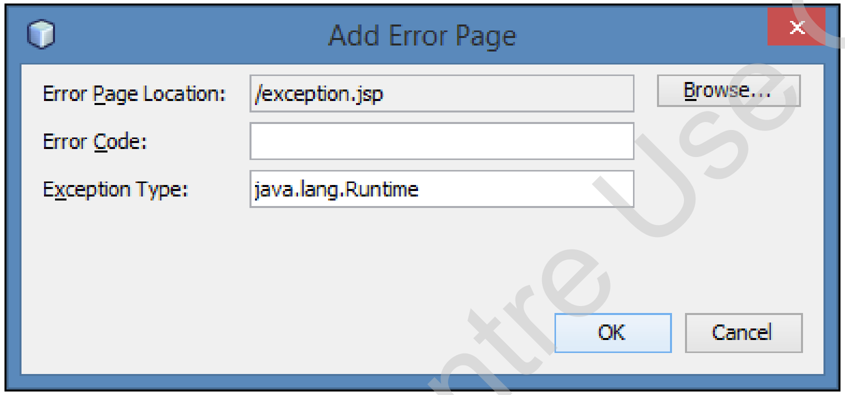
javax.xml.ws.WebServiceException	• Are thrown by the getPort() method due to insufficient data to create the proxy
javax.xml.ws.IOException	• Results due to unavailability of service, network failures, and so on give rise to IOException
javax.xml.ws.WebServiceException	• Results due to use of invalid property names and setting them with invalid values
javax.xml.ws.WebServiceException	• Occurs due to configuration errors such as invalid property names, invalid property values, type mismatch

- Following code snippet shows how runtime exceptions are handled in deployment descriptor of a Web application:

```
<error-page>
<exception-type>java.lang.Runtime</exception-type>
<location>/exception.jsp</location>
</error-page>
```

System Exceptions in Web Service 2-2

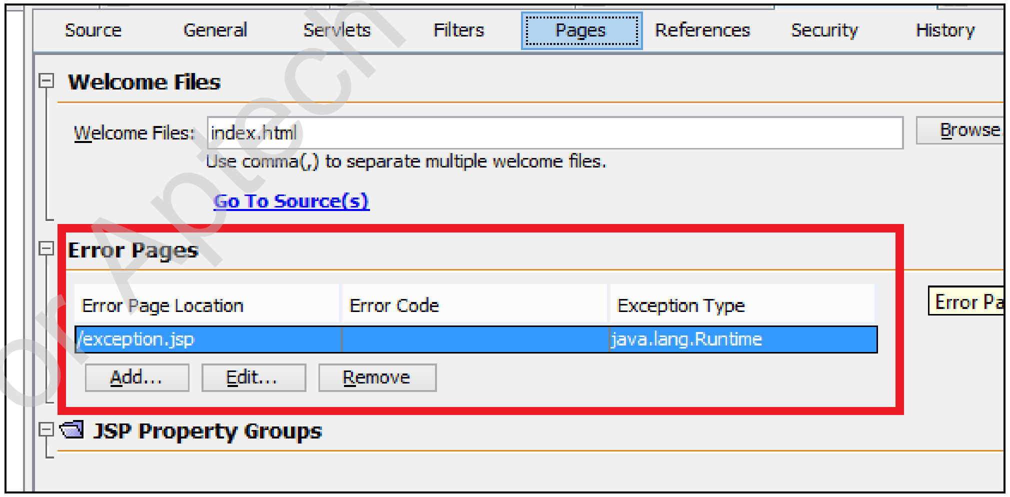
- Following figure shows a dialog box to add exception page in the Deployment Descriptor (Web.xml) using NetBeans IDE:



Add Error Page

Error Page Location:	/exception.jsp
Error Code:	
Exception Type:	java.lang.Runtime

- Following figure shows the result after setting up Error Page for Web.xml in NetBeans:



Welcome Files

Welcome Files: index.html

Use comma(,) to separate multiple welcome files.

Go To Source(s)

Error Pages

Error Page Location	Error Code	Exception Type
/exception.jsp		java.lang.Runtime

JSP Property Groups

Service Exceptions During Web Service Calls 1-2

Service Exceptions

- Are thrown by faults or errors generated by the client application itself
- Are also called checked exceptions in client applications
- Are a result of improper data passed to the Web service
- Are easy to determine as they are generated by the application itself
- Are generally listed as operation elements in a WSDL file and are known as wsdl:fault elements

Service Exceptions During Web Service Calls 2-2

JAX-WS Tools

- ❑ Can be used to map faults or errors to Java objects.
- ❑ Generate necessary exception classes and parameters to handle.

- Following code snippet demonstrates the use of illegalArgumentException:

```
<fault name=" illegalArgumentException" message="tns:
illegalArgumentException"/>
```

Exception Handling Mechanism 1-4

To handle service exceptions, developers can:

- Provide appropriate mechanisms to recover from exceptions
- Can resort to boundary checking for input values
- Can use JavaScript to validate the boundaries before sending requests to a service

In case of Java EE 7 Web Components, clients may handle service exceptions as unchecked applications, such as javax.servlet.ServletException or may divert it to an error page.

Exception Handling Mechanism 2-4

- Following code snippet demonstrates how illegalArgumentException is handled in case of non-numeral character entry:

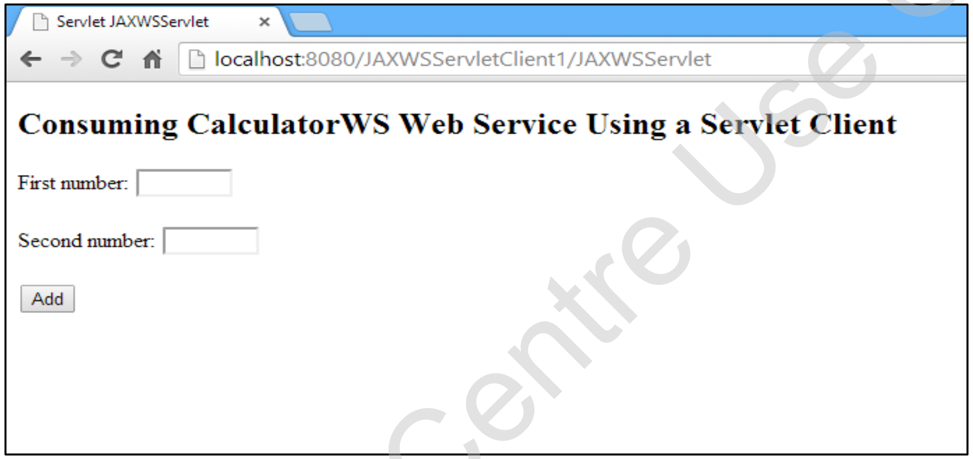
```
try {
com.djws.soap.ws.CalculatorWS_Service service = new
com.djws.soap.ws.CalculatorWS_Service();
com.djws.soap.ws.CalculatorWS port = service.getCalculatorWSPort();
     int result = port.add(num1, num2);
}
catch (illegalArgumentException cc){
System.out.println(cc);
}
```

- Following code snippet shows an error page:

```
<error-page>
<exception-type> illegalArgumentException</exception-type>
<location>/exception.jsp</location>
</error-page>
```

Exception Handling Mechanism 3-4

- Following figure shows the output which the Servlet Web service client code will produce when the page is first loaded:



Servlet JAXWSServlet

localhost:8080/JAXWSServletClient1/JAXWSServlet

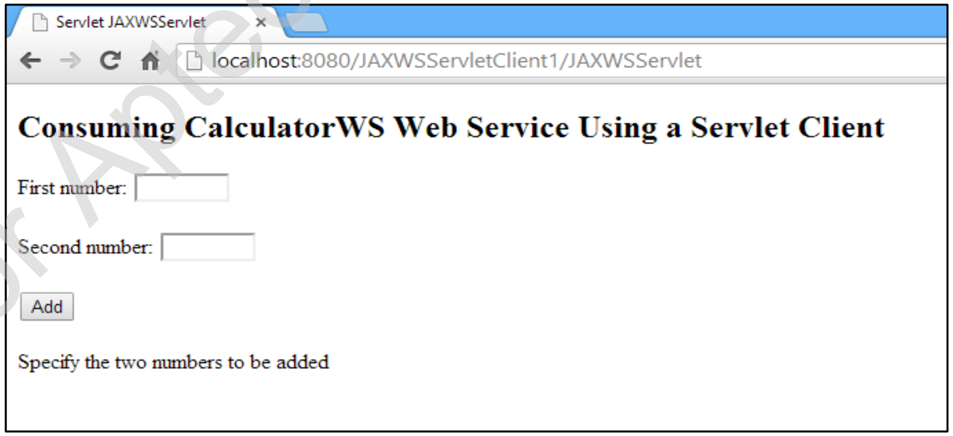
Consuming CalculatorWS Web Service Using a Servlet Client

First number:

Second number:

Add

- Following figure shows an error message if the **Add** button is clicked without entering any numbers in the text boxes:



Servlet JAXWSServlet

localhost:8080/JAXWSServletClient1/JAXWSServlet

Consuming CalculatorWS Web Service Using a Servlet Client

First number:

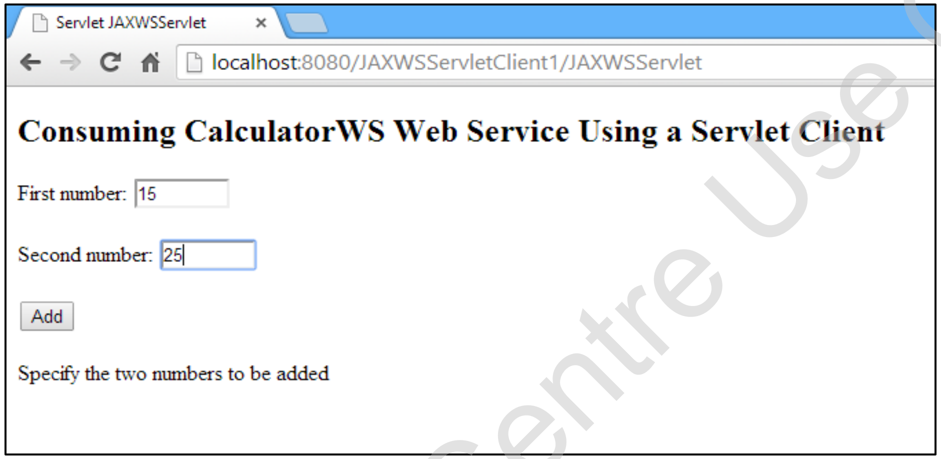
Second number:

Add

Specify the two numbers to be added

Exception Handling Mechanism 4-4

- Following figure shows the Web page after the numbers to be added are entered in the text boxes:


Servlet JAXWSServlet

localhost:8080/JAXWSServletClient1/JAXWSServlet

Consuming CalculatorWS Web Service Using a Servlet Client

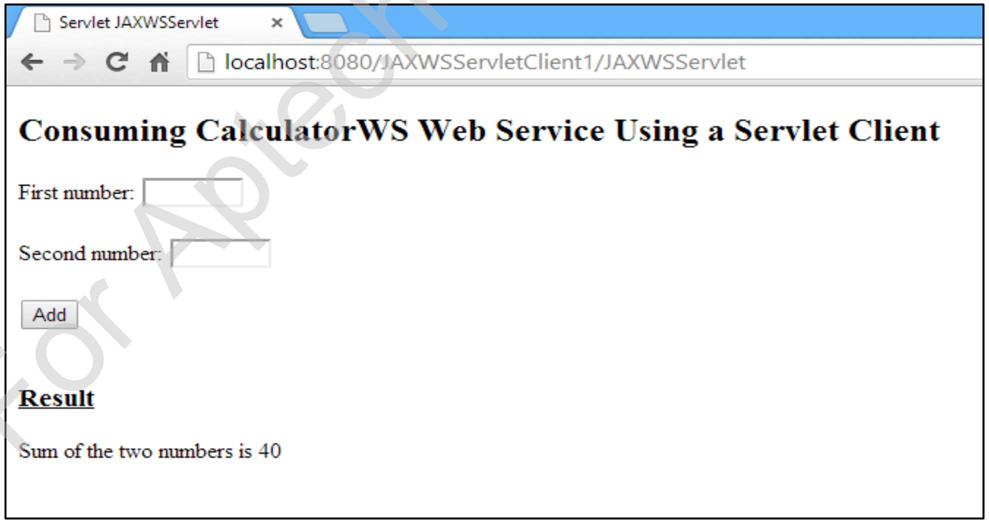
First number: 15

Second number: 25

Add

Specify the two numbers to be added

- Following figure shows the sum of two numbers after the **Add** button is clicked:


Servlet JAXWSServlet

localhost:8080/JAXWSServletClient1/JAXWSServlet

Consuming CalculatorWS Web Service Using a Servlet Client

First number:

Second number:

Add

Result

Sum of the two numbers is 40

Summary

- A Web service is invoked when the client accesses the Web service. Users can access a Web service through standalone clients or dynamic clients.
- A Dynamic client is a Web-based application that uses JSP or servlets to access the Web services. This type of client can be of two types – Dynamic Proxy client and Dynamic Invocation Interface (DII).
- A Dynamic Proxy client creates a proxy of the service interface and then uses this interface to access the Service's methods.
- A DII uses Call objects to dynamically invoke a Web service. This enables the client to invoke methods of a service even without knowing their endpoint addresses until runtime.
- A Standalone client can be a Java client or a Swing client.
- There are two types of exceptions that can be thrown by a Web service endpoint, System and Service-specific.
- All exceptions thrown by errors beyond the control of the application can be categorized as System exceptions.
- Service-specific exceptions are thrown by faults or errors generated by the client application itself.