Lesson 18C

Android

Consuming Web Services Using KSOAP (on IIS) and REST (on Apache Tomcat)

Victor Matos
Cleveland State University

Notes are based on:
http://developer.android.com/index.html
http://www.w3.org/TR/NOTE-CDFsubmit.html
http://code.google.com/p/ksoap2-android/

Consuming Web Services

Web Services

- Support **machine-to-machine** collaboration**.**

- They can be described, published, located, and invoked over a data network.

- **Web services** are used to implement the notion of a Service-Oriented Architecture (SOA).

  SOA applications are independent of specific programming languages or operating systems.

- Web services rely on existing transport technologies, such as HTTP, and XML, for invoking the implementation.

Consuming Web Services

Web Services

The interface describing the format of services can be done using the **W**eb **S**ervices **D**escription **L**anguage (**WSDL**).

According to **W3C** there are two major types of web services

- **REST**-compliant which use XML to represent its Web resources, and offers a "state-less" set of operations; and
- **Arbitrary** solutions, in which the service may expose a heterogeneous set of operations.

Consuming Web Services

Web Services - Implementations

Two widely used architectures supporting Web services are

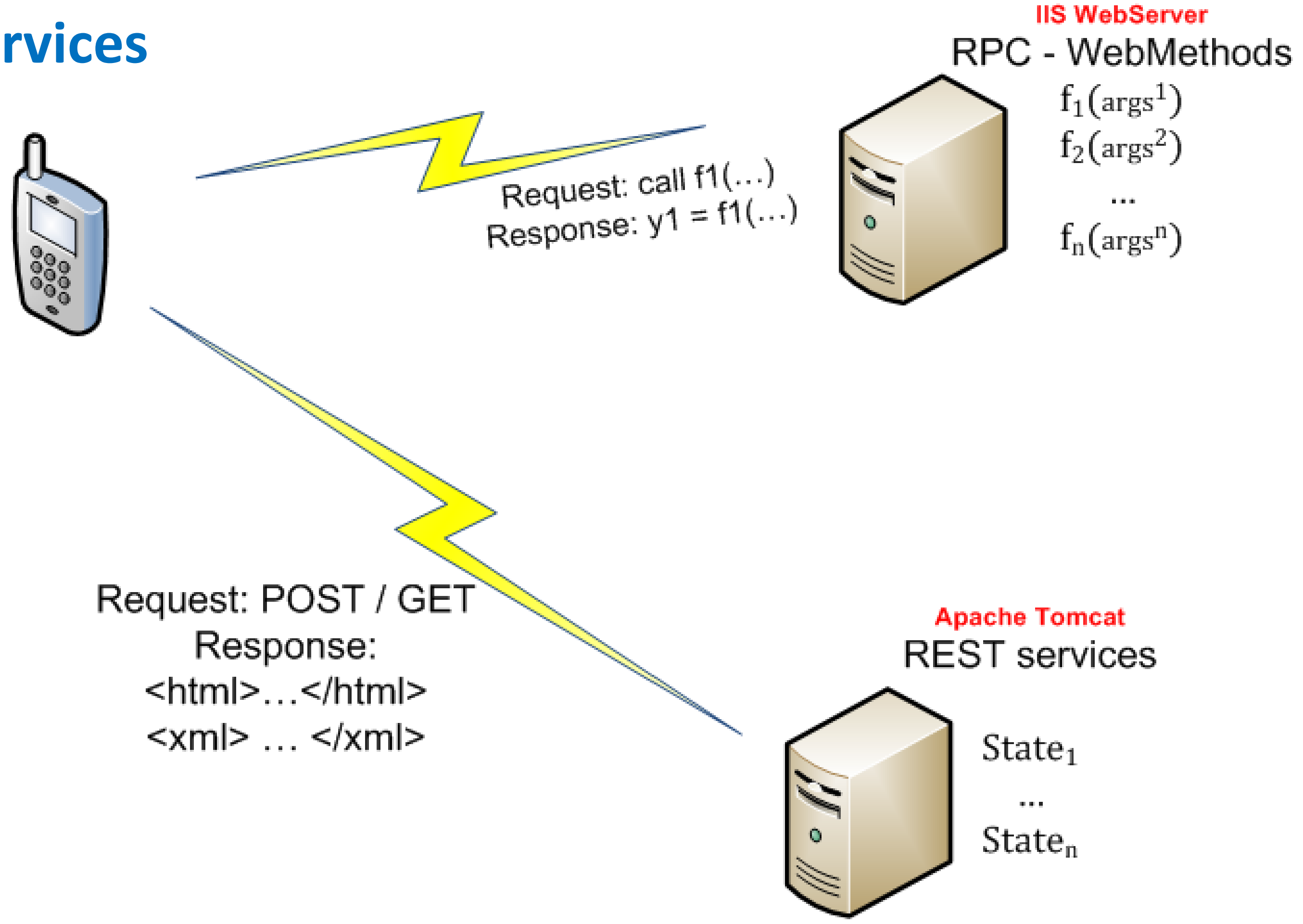
Representational State Transfer (REST) Closely tie to the HTTP protocol by associating its operation to the common GET, POST, PUT, DELETE for HTTP.

Remote Procedure Call (RPC). Web services are directly implemented as language-specific functions or method calls. In this category we find

1. Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA),
2. Microsoft's Distributed Component Object Model (DCOM) and
3. Sun Microsystems's Java/Remote Method Invocation (RMI).

Consuming Web Services

Web Services



Consuming Web Services

How Android Applications Consume WebServices?

We will present two examples of how an Android application can request services hosted in a

(1) IIS webserver (RPC discrete function oriented approach)

(2) Apache-Tomcat 7.0 (REST state oriented approach)

The examples are exhaustive and include details of how the server-side methods are constructed (you may skip this portion based on your previous experience)

Consuming Web Services

Example 1 - How .NET Web Services Are Called?

Services are dormant server-side pieces of code. They wait for incoming messages to be awaken and do some work. Clients initiate the interaction by sending a message to server-services requesting action.

Services expose one or more **endpoints** where messages can be sent.

Each endpoint consists of

- **address** (where to send messages)
- **binding** (how to send messages)
- **contract** (what messages contain)

Clients can use **WSDL** to know this information before accessing a service.

Consuming Web Services

Example 1 - How .NET Web Services Are Called?

Windows Communication Foundation (WCF) uses the information found in the service contract to perform dispatching and serialization.

Dispatching is the process of deciding which method to call for an incoming SOAP message.

Serialization is the process of mapping between the data found in a SOAP message and the corresponding .NET objects used in the method

Consuming Web Services

Example 1 - How .NET Web Services Are Called?

Our example code consists of two fragments which implement the server and client side of the application.

Server Side:

- The document http://support.microsoft.com/kb/301273 describes how to create a simple Web service running on a Windows IIS-Server.

Client Side:

- We use the KSOAP 2.0 platform to request a sequence of remote procedure calls to the IIS server hosting our service code.
- The methods include functions taking zero, or more arguments.
- Arguments send/received can be simple data types or complex objects.

http://code.google.com/p/ksoap2-android/
http://www.java2s.com/Code/Jar/k/Downloadksoap2base254jar.htm

Consuming Web Services

Example 1 - How .NET Web Services Are Called?

KSOAP2 Documentation & Download

http://code.google.com/p/ksoap2-android/

http://code.google.com/p/ksoap2-android/source/browse/m2-repo/com/google/code/ksoap2-android/ksoap2-android-assembly/2.5.5/

http://www.java2s.com/Code/Jar/k/Downloadksoap2base254jar.htm

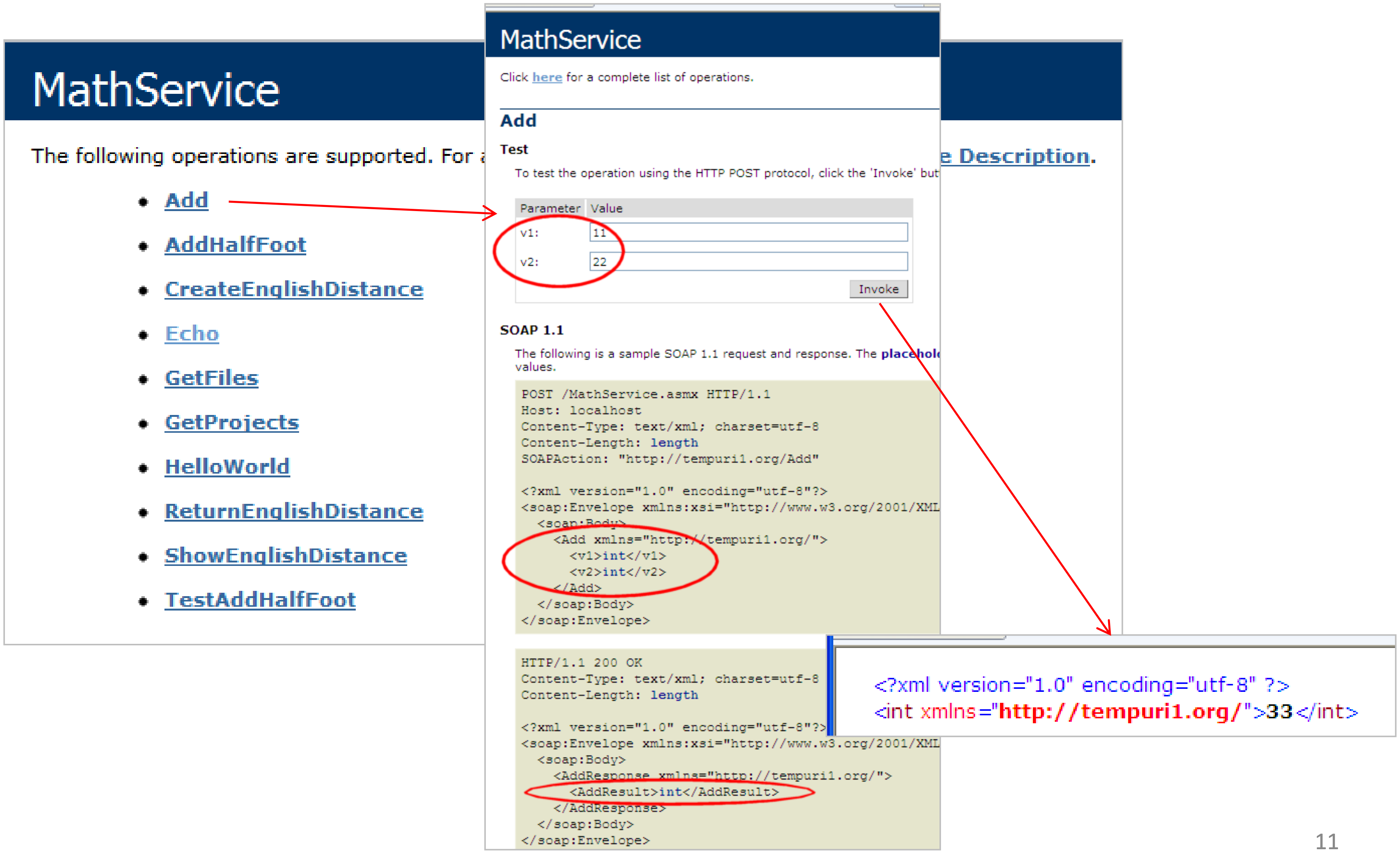
Hosting Solution

There are several options for externally hosting your .NET solution, registration strategy varies. Some Providers are:

1. http://www.asp.net/hosting
2. www.somee.com

Consuming Web Services
Example 1 – TUTORIAL – IIS Server Side Code

Services Available at the IIS Server



MathService

The following operations are supported. For a … e Description.

- Add
- AddHalfFoot
- CreateEnglishDistance
- Echo
- GetFiles
- GetProjects
- HelloWorld
- ReturnEnglishDistance
- ShowEnglishDistance
- TestAddHalfFoot

MathService

Click here for a complete list of operations.

Add

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' but

Parameter	Value
v1:	11
v2:	22

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placehol** values.

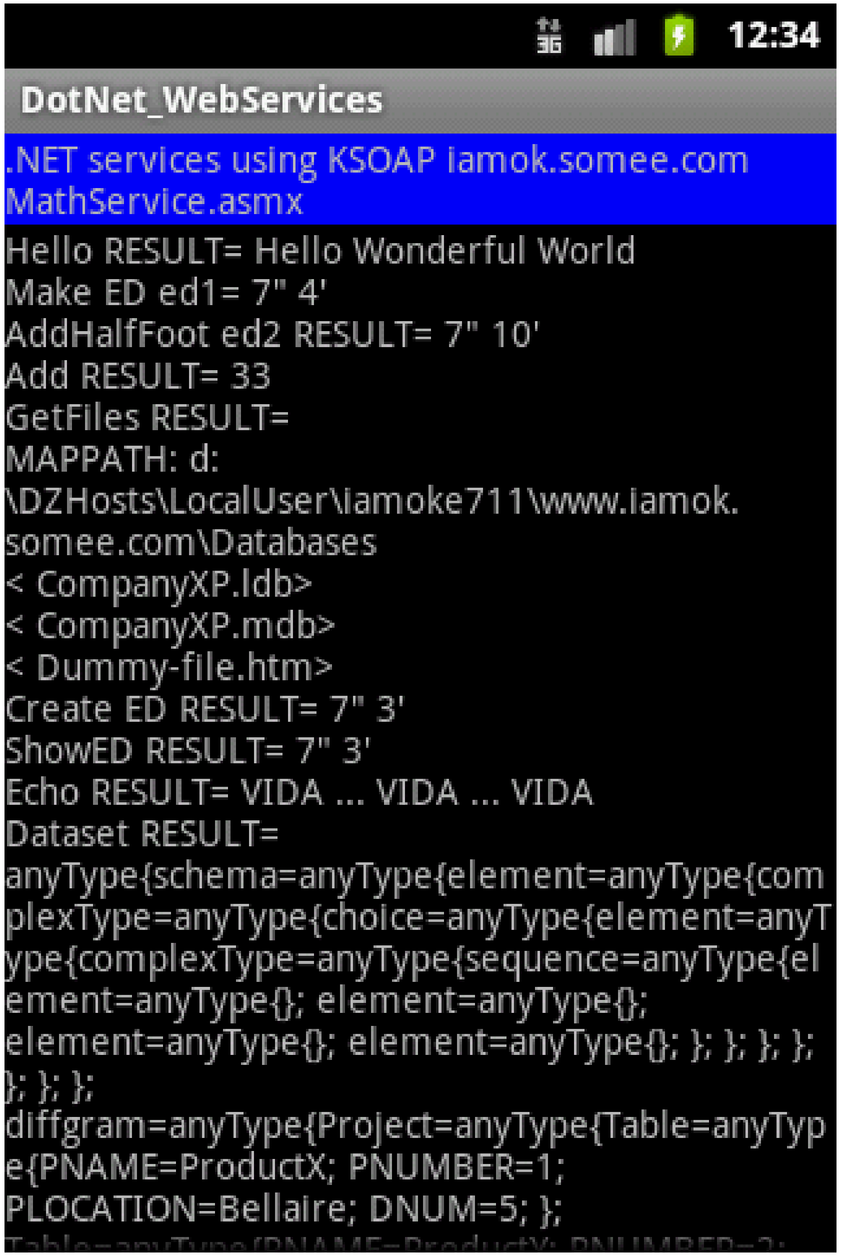
```
POST /MathService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri1.org/Add"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XML
  <soap:Body>
    <Add xmlns="http://tempuri1.org/">
      <v1>int</v1>
      <v2>int</v2>
    </Add>
  </soap:Body>
</soap:Envelope>
```

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

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XML
  <soap:Body>
    <AddResponse xmlns="http://tempuri1.org/">
      <AddResult>int</AddResult>
    </AddResponse>
  </soap:Body>
</soap:Envelope>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://tempuri1.org/">33</int>
```

Consuming Web Services
Example 1 – TUTORIAL – IIS Server Side Code

Android App accessing all services available at the IIS server



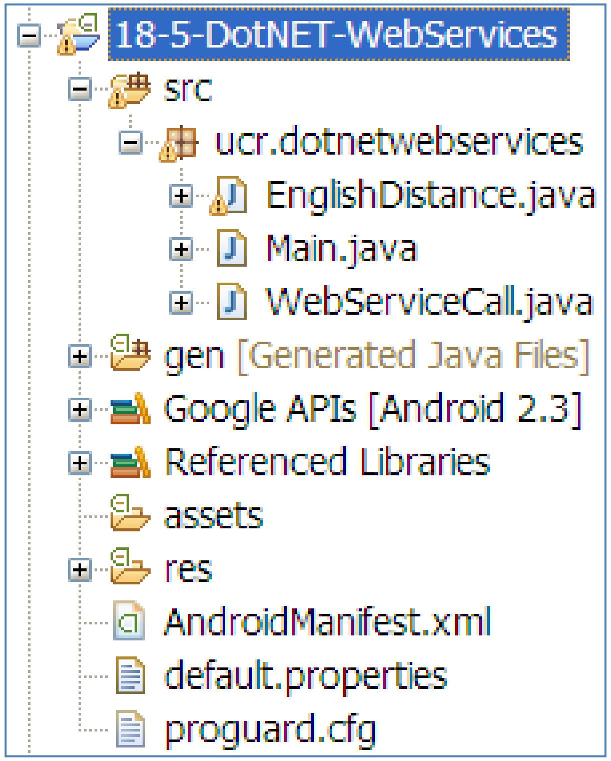
```
.NET services using KSOAP iamok.somee.com
MathService.asmx
Hello RESULT= Hello Wonderful World
Make ED ed1= 7" 4'
AddHalfFoot ed2 RESULT= 7" 10'
Add RESULT= 33
GetFiles RESULT=
MAPPATH: d:
\DZHosts\LocalUser\iamoke711\www.iamok.
somee.com\Databases
< CompanyXP.ldb>
< CompanyXP.mdb>
< Dummy-file.htm>
Create ED RESULT= 7" 3'
ShowED RESULT= 7" 3'
Echo RESULT= VIDA ... VIDA ... VIDA
Dataset RESULT=
anyType{schema=anyType{element=anyType{com
plexType=anyType{choice=anyType{element=anyT
ype{complexType=anyType{sequence=anyType{el
ement=anyType{}; element=anyType{};
element=anyType{}; element=anyType{}; }; }; }; };
}; }; };
diffgram=anyType{Project=anyType{Table=anyTyp
e{PNAME=ProductX; PNUMBER=1;
PLOCATION=Bellaire; DNUM=5; };
```

Consuming Web Services
Example 1 – TUTORIAL – Android Application

Our Android app uses KSOAP 2 API. KSOAP is a webservice client library for *constrained* Java environments. SOAP protocol is widely used for machine-to-machine interaction, it is strong-typed and supports synchronous, asynchronous, and complex-routing communication schemes.

Our implementation includes three classes

1. **Main** webcalls are assembled
2. **EnglishDistance** (Serialized Class)
3. **WebServiceCall** deals with HTTP transporting of the request/response and envelope objects



```
18-5-DotNET-WebServices
  src
    ucr.dotnetwebservices
      EnglishDistance.java
      Main.java
      WebServiceCall.java
  gen [Generated Java Files]
  Google APIs [Android 2.3]
  Referenced Libraries
  assets
  res
  AndroidManifest.xml
  default.properties
  proguard.cfg
```

Consuming Web Services
Example 1 – TUTORIAL – Android Application

A fragment of the Android Main class follows

```
public class Main extends Activity {
public TextView txtMsg;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txtMsg = (TextView)findViewById(R.id.txtMsg);

        WebServiceCall webServiceCall = new WebServiceCall();

        // add two numbers - get int result
        int intResult = webServiceCall.AddService(11, 22);
        txtMsg.append( "\nAdd RESULT= " + intResult );
}
```

All the intelligent work is done by the **WebServiceCall** class.

Consuming Web Services
Example 1 – TUTORIAL – Android Application

The **WebServiceCall** class negotiates the transporting of request/response objects as well as preparation of serialization of complex data elements (1/4)

```
import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;

import android.util.Log;

public class WebServiceCall
{
private static final String SOAP_ACTION = "http://tempuri1.org/";
private static final String NAMESPACE = "http://tempuri1.org/";
private static final String URL =
                    "http://iamok.somee.com/MathService.asmx";

protected Object call( String soapAction,
                      SoapSerializationEnvelope envelope)
{
```

Consuming Web Services
Example 1 – TUTORIAL – Android Application

WebServiceCall class (continuation 2/4)

```
Object result = null;

final HttpTransportSE transportSE = new HttpTransportSE(URL);

transportSE.debug = false;

// call and Parse Result.
try
{
    transportSE.call(soapAction, envelope);
    result = envelope.getResponse();

} catch (final Exception e)
{
    Log.e("<<Exception>>",e.getMessage());
}
return result;
}
```

Consuming Web Services
Example 1 – TUTORIAL – Android Application

Fragment of the **WebServiceCall** class called to add two numbers. Here input parameters and output values resulting from the webservice are sent and received. (cont. 3/4)

```
public int AddService(int v1, int v2){

int intResult = 0;
// indicate webservice (endpoint) to be called
final String webMethod = "Add";

// Create the outgoing request message
final SoapObject requestObject = new SoapObject(NAMESPACE,
                                                webMethod);
// add outgoing parameters (name-type must agree with endpoint)
requestObject.addProperty("v1", v1);
requestObject.addProperty("v2", v2);

// Create soap envelope for .NET server
final SoapSerializationEnvelope envelope =
      new SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope.dotNet = true;

// place in envelope the outgoing request object
envelope.setOutputSoapObject(requestObject);
```

Consuming Web Services
Example 1 – TUTORIAL – Android Application

The **WebServiceCall** class (continuation 4/4)

Web service is called here

```
try
{
    // call webmethod and parse returning response object
    final Object response = (Object) this.call(
                                      SOAP_ACTION + webMethod,
                                      envelope);
    if (response != null)
        intResult = Integer.parseInt(response.toString());

} catch (Exception e)
{
    Log.e("<<Exception-Add>>",e.getMessage());
}

return intResult;
}


}
```

Transferring Complex Data Objects to/from Server

- Previous fragment illustrates the sending of simple pieces of primitive data (int, strings, …) between client and server.
- When complex objects are exchanged the sender and receiver must deal with the serialization of the objects in transit.
- Serializing requires the client's Java definition of the class to implement the **KvmSerializable** interface. The server must also mark the corresponding class as [**Serializable**].

Transferring Complex Data Objects to/from Server (1 of 5)

In the following fragment an **EnglishDistance** class is defined in the client (Java) and server (C#). The following fragments of Android code illustrates the defining of the class and the calling mechanism to send/receive this complex data type.

```
// Calling DotNET Webservices running on an IIS server
// This class represents a KvmSerializable version of an
// EnglishDistance(fett, inches). Observe the required
// methods: getProperty, setProperty, getPropertyCount.
// Author: Victor Matos - April 2011

package ucr.dotnetwebservices;

import java.util.Hashtable;

import org.ksoap2.serialization.KvmSerializable;
import org.ksoap2.serialization.PropertyInfo;
import org.ksoap2.serialization.SoapObject;

public class EnglishDistance implements KvmSerializable {
private int _feet;
private int _inches;
```

Consuming Web Services
Example 1 – TUTORIAL – IIS Server Side Code

Transferring Complex Data Objects to/from Server (2 of 5)

```
public EnglishDistance(int feetValue, int inchesValue){
     _feet = feetValue;
     if (inchesValue >= 12) {
          _feet += (int)(inchesValue / 12);
          _inches = inchesValue % 12;
     }
}

public  EnglishDistance (SoapObject obj)
{
     this._feet = Integer.parseInt(obj.getProperty("feet").toString());
     this._inches = Integer.parseInt(obj.getProperty("inches").toString());
}

public String showDistance(){
     return _feet + "\" " + _inches + "\' ";
}

@Override
public int getPropertyCount() {
     return 2;
}
```

Needed by interface

Transferring Complex Data Objects to/from Server (3 of 5)

```
@Override
public Object getProperty(int index) {
     Object object = null;
     switch (index)
     {
          case 0:
          {
               object = this._feet;
               break;
          }
          case 1:
          {
               object = this._inches;
               break;
          }
     }
     return object;
}
```

Needed by interface

Consuming Web Services
Example 1 – TUTORIAL – IIS Server Side Code

Transferring Complex Data Objects to/from Server (4 of 5)

```
@Override
public void getPropertyInfo(int index, Hashtable hashTable,
                                          PropertyInfo propertyInfo) {
      switch (index)
      {
      case 0:
      {
            propertyInfo.name = "feet";
            propertyInfo.type = PropertyInfo.INTEGER_CLASS;
            break;
      }
      case 1:
      {
            propertyInfo.name = "inches";
            propertyInfo.type = PropertyInfo.INTEGER_CLASS;
            break;
      }

}
```

Needed by interface

Transferring Complex Data Objects to/from Server (5 of 5)

```
}//getProperty
@Override
public void setProperty(int index, Object obj) {
     switch (index)
     {
     case 0:
     {
          this._feet = Integer.parseInt(obj.toString());
          break;
     }
     case 1:
     {
          this._inches = Integer.parseInt(obj.toString());
          break;
     }

     }

}//setProperty

}//EnglishDistance
```

Needed by interface

Transferring Data Objects - .NET Code – IIS Server Definition 1/2

The following fragment shows the Server side definition of the **EnglishDistance** class. Notice the server needs to agree with the client's definition on attributes (name, type); however it does not need to include the same methods.

```
namespace MathService
{
    [Serializable]
    public class EnglishDistance     {
        private int _feet = 0;
        private int _inches = 0;

        public EnglishDistance()    {
            _feet = 0;     _inches = 0;
        }

        public EnglishDistance(int feet, int inches)
        {
            this._feet = feet;     this._inches = inches;
            if (inches >= 12)  {
                this._feet += (int)(inches / 12);
                this._inches += (inches % 12);
            }
        }
```

Consuming Web Services
Example 1 – TUTORIAL – IIS Server Side Code

Transferring Data Objects. Server Definition 2/2

```
        public String ShowEnglishDistance()     {
              return _feet + "\" " + _inches + "\' ";
         }

         public int feet
         {
             get { return this._feet; }
             set { _feet = value; }
         }

         public int inches
         {
             get { return this._inches; }
             set {
                         if (value >= 12)
                         {
                             this._feet += (int)(value / 12);
                         }
                         this._inches = (value % 12);
                     }
         }

     }//EnglishDistance
}
```

Consuming Web Services
Example 1 – TUTORIAL – Android Code

Android – Sending / Receiving Objects from Web Services (1 of 3)

This fragment is from the Android's app Main class. It creates an instance of an EnglishDistance and sends it to the server to be increased by half a foot. Transferring of the object requires its serialization.

```
// make a local EnglishDistance object & pass it to webservice
// get the modified object back from service (6 more inches)

EnglishDistance ed1 = new EnglishDistance(6, 16);
txtMsg.append( "\nMake ED ed1= " + ed1.showDistance() );

EnglishDistance ed2  = (EnglishDistance)webServiceCall.AddHalfFoot(ed1);

if (ed2 != null)
    txtMsg.append( "\nAddHalfFoot ed2 RESULT= " + ed2.showDistance() );
else
    txtMsg.append( "\nAddHalfFoot ed2 RESULT= NULL");
```

Android – Sending / Receiving Objects from Web Services (2 of 3)

This fragment is from the Android's app WebServiceCall class. It prepares the request object and its EnglishDistance object parameter, calls the service and picks the returned object.

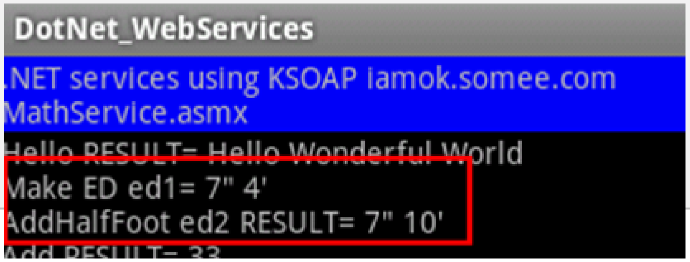
```
public EnglishDistance AddHalfFoot(EnglishDistance inputEd){

// indicate webservice (endpoint) to be called
final String webMethod = "AddHalfFoot";

// Create the outgoing request object
final SoapObject requestObject = new SoapObject(NAMESPACE, webMethod);

// add outgoing parameter (name-type) must agree with endpoint
requestObject.addProperty("inputEd", inputEd);

// Create soap envelope for IIS server
final SoapSerializationEnvelope envelope =
      new SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope.dotNet = true;
```

Consuming Web Services
Example 1 – TUTORIAL – Android Code

Android – Sending / Receiving Objects from Web Services (3 of 3)

This fragment is from the Android's app WebServiceCall class. It prepares the request object and its EnglishDistance object parameter, calls the service and picks the returned object.

```java
// place in the envelope the outgoing request object
envelope.setOutputSoapObject(requestObject);
envelope.addMapping(NAMESPACE,
                    EnglishDistance.class.getSimpleName(),
                    EnglishDistance.class);

// call webservice and parse returning response object
EnglishDistance outputEd = null;
final Object response = this.call( SOAP_ACTION + webMethod, envelope );
if (response != null)
{
   outputEd = new EnglishDistance((SoapObject) response);
}

return outputEd;

}//AddHalfFoot
```



```
DotNet_WebServices
.NET services using KSOAP iamok.somee.com
MathService.asmx
Hello RESULT= Hello Wonderful World
Make ED ed1= 7" 4'
AddHalfFoot ed2 RESULT= 7" 10'
Add RESULT= 33
```

REST Protocol – Android & Apache's Tomcat Server

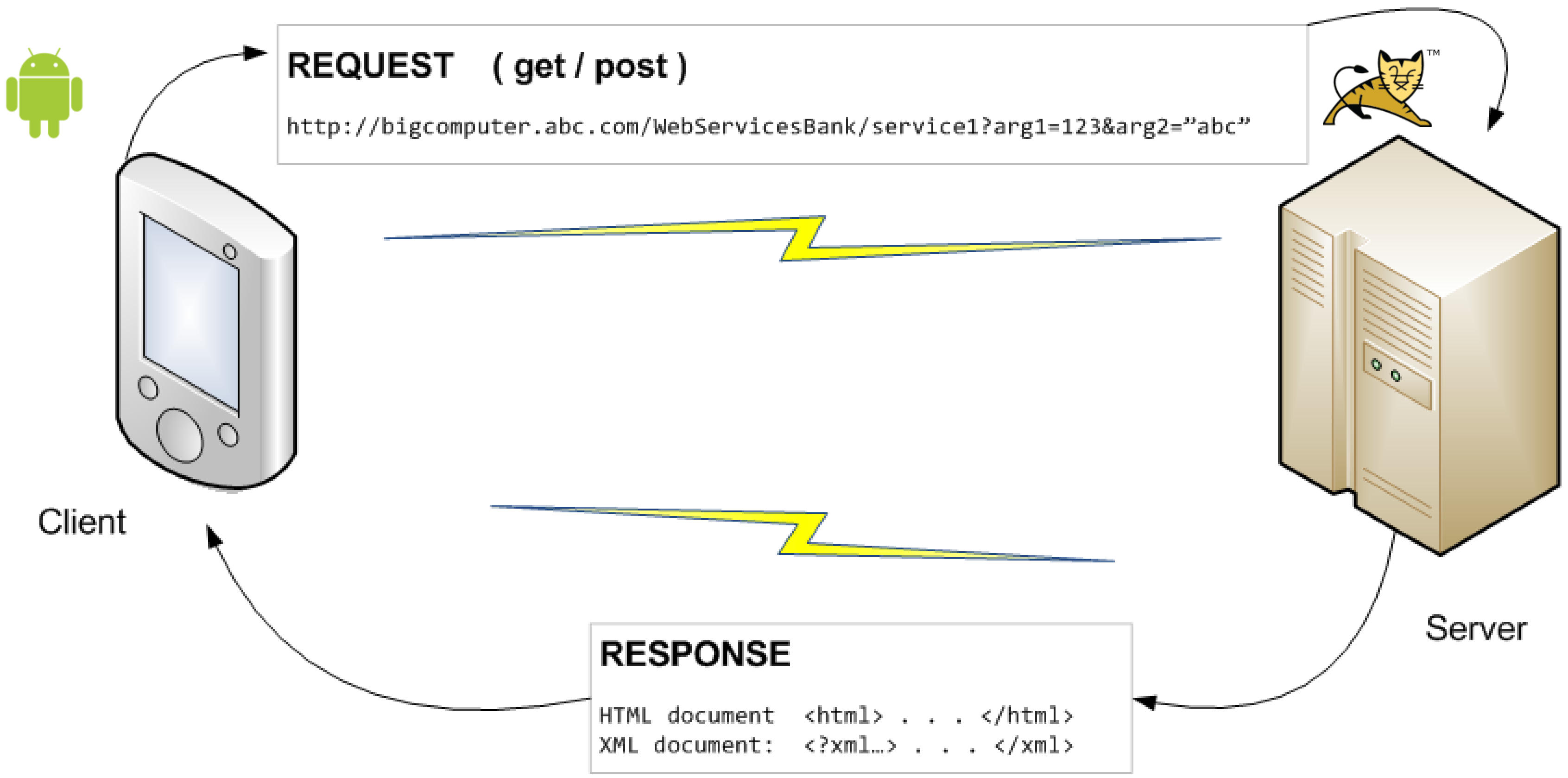
Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

- This networking option imposes less overhead than KSOAP
- A Java **servlet** communicates with the HTTP server and the client application through the following objects.

  HttpServletRequest: the request object
  Represents a client's request. This object gives a servlet access to incoming information such as HTML form data and HTTP request headers.

  HttpServletResponse: the response object
  Represents the servlet's response. The servlet uses this object to return data to the client such as HTTP errors (200, 404, and others), response headers (Content-Type, Set-Cookie, and others), and output data by writing to the response's output stream or output writer.

REST Protocol – Android & Apache's Tomcat Server


REQUEST (get / post)
http://bigcomputer.abc.com/WebServicesBank/service1?arg1=123&arg2=”abc”
Client
Server
RESPONSE
HTML document <html> . . . </html>
XML document: <?xml…> . . . </xml>

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

HttpServletRequest Methods

authenticate(HttpServletResponse response)
Use the container login mechanism configured for the ServletContext to authenticate the user making this request.

getAuthType()
Returns the name of the authentication scheme used to protect the servlet.

getContextPath()
Returns the portion of the request URI that indicates the context of the request.

getCookies()
Returns an array containing all of the Cookie objects the client sent with this request.

getDateHeader(java.lang.String name)
Returns the value of the specified request header as a long value that represents a Date object.

getHeader(java.lang.String name)
Returns the value of the specified request header as a String.

getHeaderNames()
Returns an enumeration of all the header names this request contains.

getHeaders(java.lang.String name)
Returns all the values of the specified request header as an Enumeration of String objects.

getIntHeader(java.lang.String name)
Returns the value of the specified request header as an int.

getMethod()
Returns the name of the HTTP method with which this request was made, for example GET, POST, or PUT.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

HttpServletRequest Methods

getPart(java.lang.String name)
    Gets the Part with the given name.
getParts()
    Gets all the Part components of this request, provided that it is of type multipart/form-data.
getPathInfo()
    Returns any extra path information associated with the URL the client sent when it made this request.
getPathTranslated()
    Returns any extra path information after the servlet name but before the query string, and translates it to a real path.
getQueryString()
    Returns the query string that is contained in the request URL after the path.
getRemoteUser()
    Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.
getRequestedSessionId()
    Returns the session ID specified by the client.
getAsyncContext()
    Gets the AsyncContext that was created or reinitialized by the most recent invocation of startAsync() or startAsync(ServletRequest,ServletResponse) on this request.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

Interface ServletRequest

getAttribute(java.lang.String name)
Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.
getAttributeNames()
Returns an Enumeration containing the names of the attributes available to this request.
getCharacterEncoding()
Returns the name of the character encoding used in the body of this request.
getContentLength()
Returns the length, in bytes, of the request body and made available by the input stream, or -1 if the length is not known.
getContentType()
Returns the MIME type of the body of the request, or null if the type is not known.
getDispatcherType()
Gets the dispatcher type of this request.
getInputStream()
Retrieves the body of the request as binary data using a ServletInputStream.
getLocalAddr()
Returns the Internet Protocol (IP) address of the interface on which the request was received.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

Interface ServletRequest

getLocalAddr()
Returns the Internet Protocol (IP) address of the interface on which the request was received.
getLocale()
Returns the preferred Locale that the client will accept content in, based on the Accept-Language header.
getLocales()
Returns an Enumeration of Locale objects indicating, in decreasing order starting with the preferred locale, the locales that are acceptable to the client based on the Accept-Language header.
getLocalName()
Returns the host name of the Internet Protocol (IP) interface on which the request was received.
getLocalPort()
Returns the Internet Protocol (IP) port number of the interface on which the request was received.
getParameter(java.lang.String name)
Returns the value of a request parameter as a String, or null if the parameter does not exist.
getParameterMap()
Returns a java.util.Map of the parameters of this request.
getParameterNames()
Returns an Enumeration of String objects containing the names of the parameters contained in this request.
getParameterValues(java.lang.String name)
Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

Interface ServletRequest

getProtocol()
Returns the name and version of the protocol the request uses in the form protocol/majorVersion.minorVersion, for example, HTTP/1.1.

getReader()
Retrieves the body of the request as character data using a BufferedReader.

getRealPath(java.lang.String path)
Deprecated. As of Version 2.1 of the Java Servlet API, use ServletContext#getRealPath instead.

getRemoteAddr()
Returns the Internet Protocol (IP) address of the client or last proxy that sent the request.

getRemoteHost()
Returns the fully qualified name of the client or the last proxy that sent the request.

getRemotePort()
Returns the Internet Protocol (IP) source port of the client or last proxy that sent the request.

getRequestDispatcher(java.lang.String path)
Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path.

getScheme()
Returns the name of the scheme used to make this request, for example, http, https, or ftp.

getServerName()
Returns the host name of the server to which the request was sent.

getServerPort()
Returns the port number to which the request was sent.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

Interface ServletRequest

getServletContext()
    Gets the servlet context to which this ServletRequest was last dispatched.
isAsyncStarted()
    Checks if this request has been put into asynchronous mode.
isAsyncSupported()
    Checks if this request supports asynchronous operation.
isSecure()
    Returns a boolean indicating whether this request was made using a secure channel, such as HTTPS.
removeAttribute(java.lang.String name)
    Removes an attribute from this request.
setAttribute(java.lang.String name, java.lang.Object o)
    Stores an attribute in this request.
setCharacterEncoding(java.lang.String env)
    Overrides the name of the character encoding used in the body of this request.
startAsync()
    Puts this request into asynchronous mode, and initializes its AsyncContext with the original (unwrapped) ServletRequest and ServletResponse objects.
startAsync(ServletRequest servletRequest, ServletResponse servletResponse)
    Puts this request into asynchronous mode, and initializes its AsyncContext with the given request and response objects.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

Interface ServletResponse

flushBuffer()
    Forces any content in the buffer to be written to the client.
getBufferSize()
    Returns the actual buffer size used for the response.
getCharacterEncoding()
    Returns the name of the character encoding (MIME charset) used for the body sent in this response.
getContentType()
    Returns the content type used for the MIME body sent in this response.
getLocale()
    Returns the locale specified for this response using the setLocale(java.util.Locale) method.
getOutputStream()
    Returns a ServletOutputStream suitable for writing binary data in the response.
getWriter()
    Returns a PrintWriter object that can send character text to the client.
isCommitted()
    Returns a boolean indicating if the response has been committed.
reset()
    Clears any data that exists in the buffer as well as the status code and headers.
resetBuffer()
    Clears the content of the underlying buffer in the response without clearing headers or status code.
setBufferSize(int size)
    Sets the preferred buffer size for the body of the response.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

Interface HttpServletResponse

setCharacterEncoding(java.lang.String charset)
Sets the character encoding (MIME charset) of the response being sent to the client, for example, to UTF-8.

setContentLength(int len)
Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.

setContentType(java.lang.String type)
Sets the content type of the response being sent to the client, if the response has not been committed yet.

setLocale(java.util.Locale loc)
Sets the locale of the response, if the response has not been committed yet.

addCookie(Cookie cookie)
Adds the specified cookie to the response.

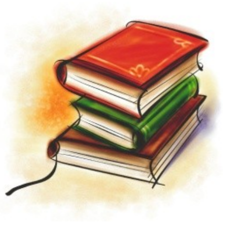
addDateHeader(java.lang.String name, long date)
Adds a response header with the given name and date-value.

addHeader(java.lang.String name, java.lang.String value)
Adds a response header with the given name and value.

addIntHeader(java.lang.String name, int value)
Adds a response header with the given name and integer value.

containsHeader(java.lang.String name)
Returns a boolean indicating whether the named response header has already been set.

Consuming Web Services
Example 2 – TUTORIAL – Android Code



REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

Interface HttpServletResponse

encodeRedirectUrl(java.lang.String url)
Deprecated. As of version 2.1, use encodeRedirectURL(String url) instead
encodeRedirectURL(java.lang.String url)
Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
encodeUrl(java.lang.String url)
Deprecated. As of version 2.1, use encodeURL(String url) instead
encodeURL(java.lang.String url)
Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
getHeader(java.lang.String name)
Gets the value of the response header with the given name.
getHeaderNames()
Gets the names of the headers of this response.
getHeaders(java.lang.String name)
Gets the values of the response header with the given name.
getStatus()
Gets the current status code of this response.
sendError(int sc, java.lang.String msg)
Sends an error response to the client using the specified status and clears the buffer.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Reference: http://download.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html

Interface HttpServletResponse

sendError(int sc, java.lang.String msg)
Sends an error response to the client using the specified status and clears the buffer.
sendRedirect(java.lang.String location)
Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer.
setDateHeader(java.lang.String name, long date)
Sets a response header with the given name and date-value.
setHeader(java.lang.String name, java.lang.String value)
Sets a response header with the given name and value.
setIntHeader(java.lang.String name, int value)
Sets a response header with the given name and integer value.
setStatus(int sc)
Sets the status code for this response.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2A.

Server-Side Code - Servlet gets /echoes the REQUEST Object

This example illustrates the use of the **REQUEST** object. We assume you are using **Eclipse EE** and Apache **Tomcat 7.0** or newer.

1. In Eclipse - create a new *Dynamic Web project*. Set project name to: **ServerServices**, and allow target to be **Tomcat Server 7.0.**

2. Add the package **csu.yourlastname** to the folder **ServerServices/JavaResources/src**

3. Create inside the previous package a new **Servlet**. Name it: **MyFirstServlet**.

4. Set the servlet code as indicated in the class. Save it and execute (make sure the Tomcat service is **NOT** started – Instead you will use a local server instance (2)).

5. In the Eclipse's browser pane, enter the following URL:
   http://localhost:8080/ServerServices/MyFirstServlet?arg1=123&arg2="abc"

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2A.

Server-Side Code - Servlet gets /echoes the REQUEST Object

Our first servlet accepts any number of arguments in the form <**key=value**>. It uses the REQUEST object to extract and manage the incoming data and as a Java Map.

Example:
Assume the users calls the servlet with the following URL

```
http://localhost:8080/ServerServices/MyFirstServlet?arg1=123&arg2="abc"
```

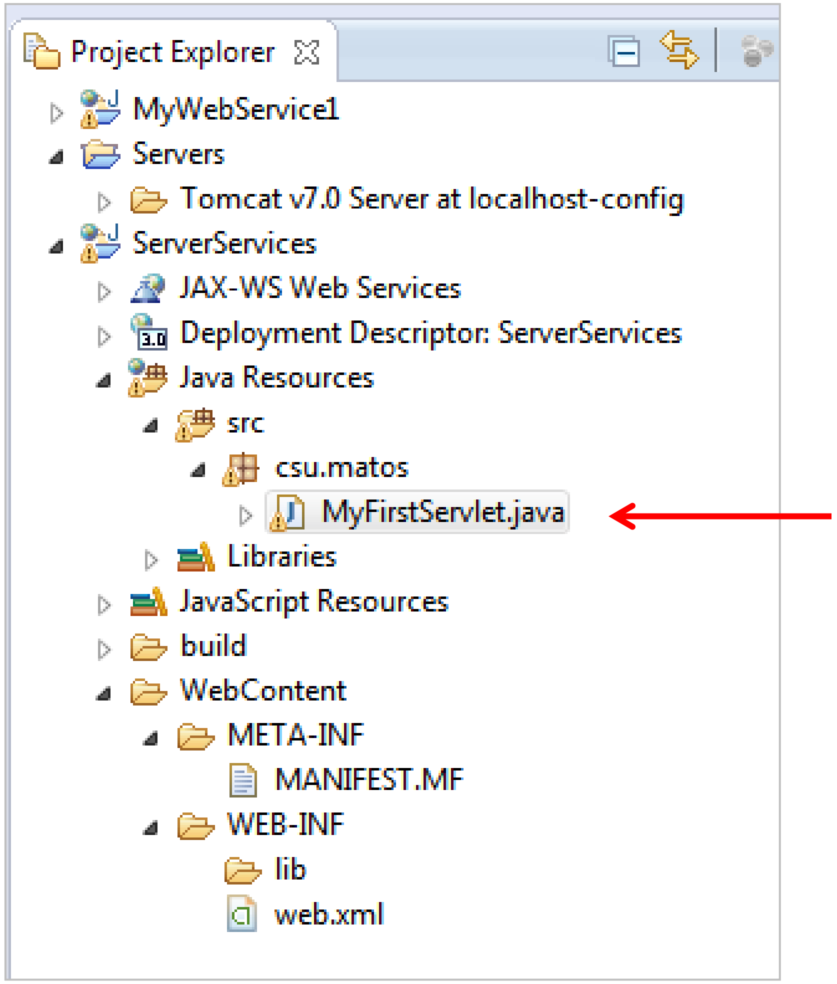
arg1	123
arg2	“abc”

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2A.

Server-Side Code - Servlet gets /echoes the REQUEST Object



```
Project Explorer
▷ MyWebService1
◢ Servers
    ▷ Tomcat v7.0 Server at localhost-config
◢ ServerServices
    ▷ JAX-WS Web Services
    ▷ Deployment Descriptor: ServerServices
    ◢ Java Resources
        ◢ src
            ◢ csu.matos
                ▷ MyFirstServlet.java
        ▷ Libraries
    ▷ JavaScript Resources
    ▷ build
    ◢ WebContent
        ◢ META-INF
            MANIFEST.MF
        ◢ WEB-INF
            lib
            web.xml
```

REST Protocol – Android & Apache's Tomcat Server

Example 2A. Server-Side Code - Servlet gets the REQUEST Object

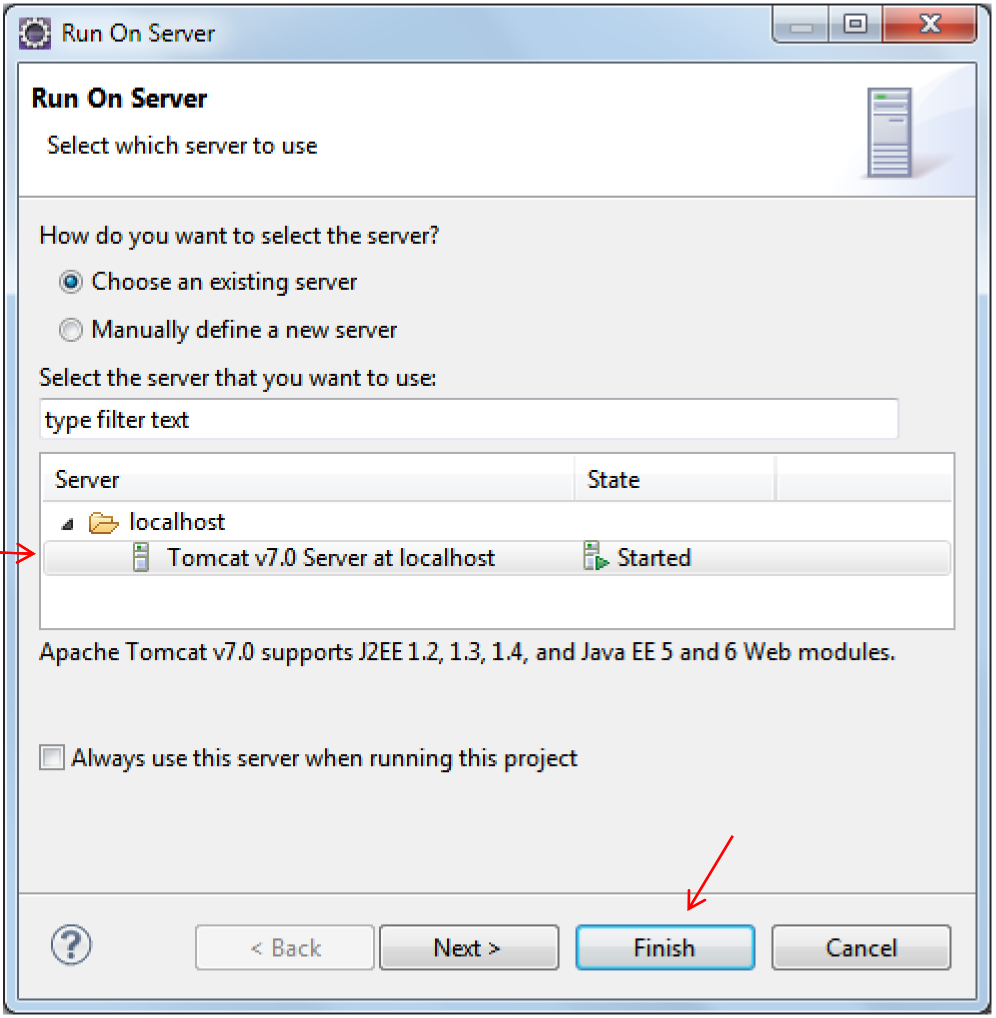
```
package csu.matos;
import . . .
@WebServlet("/MyFirstServlet")
public class MyFirstServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   public void doGet(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {

      Map<String, String[]> requestMap = request.getParameterMap();

      if (requestMap.isEmpty() ){
         System.out.println("<<<< SERVLET CALLED - Empty Request >>>> ");
         return;
      }

      for (String key : requestMap.keySet() ){
         String[] value = requestMap.get(key);
         System.out.println(">>>> " + key + "   " + value[0]);
         if (key.equalsIgnoreCase("myarg1"))
              System.out.println("got myArg1");
      }
   }//doGet
}//class
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2A. Server-Side Code - Servlet gets /echoes the REQUEST Object

Run the servlet



Run On Server

Run On Server

Select which server to use

How do you want to select the server?

[● Choose an existing server]

○ Manually define a new server

Select the server that you want to use:

type filter text

Server	State
localhost	
Tomcat v7.0 Server at localhost	Started

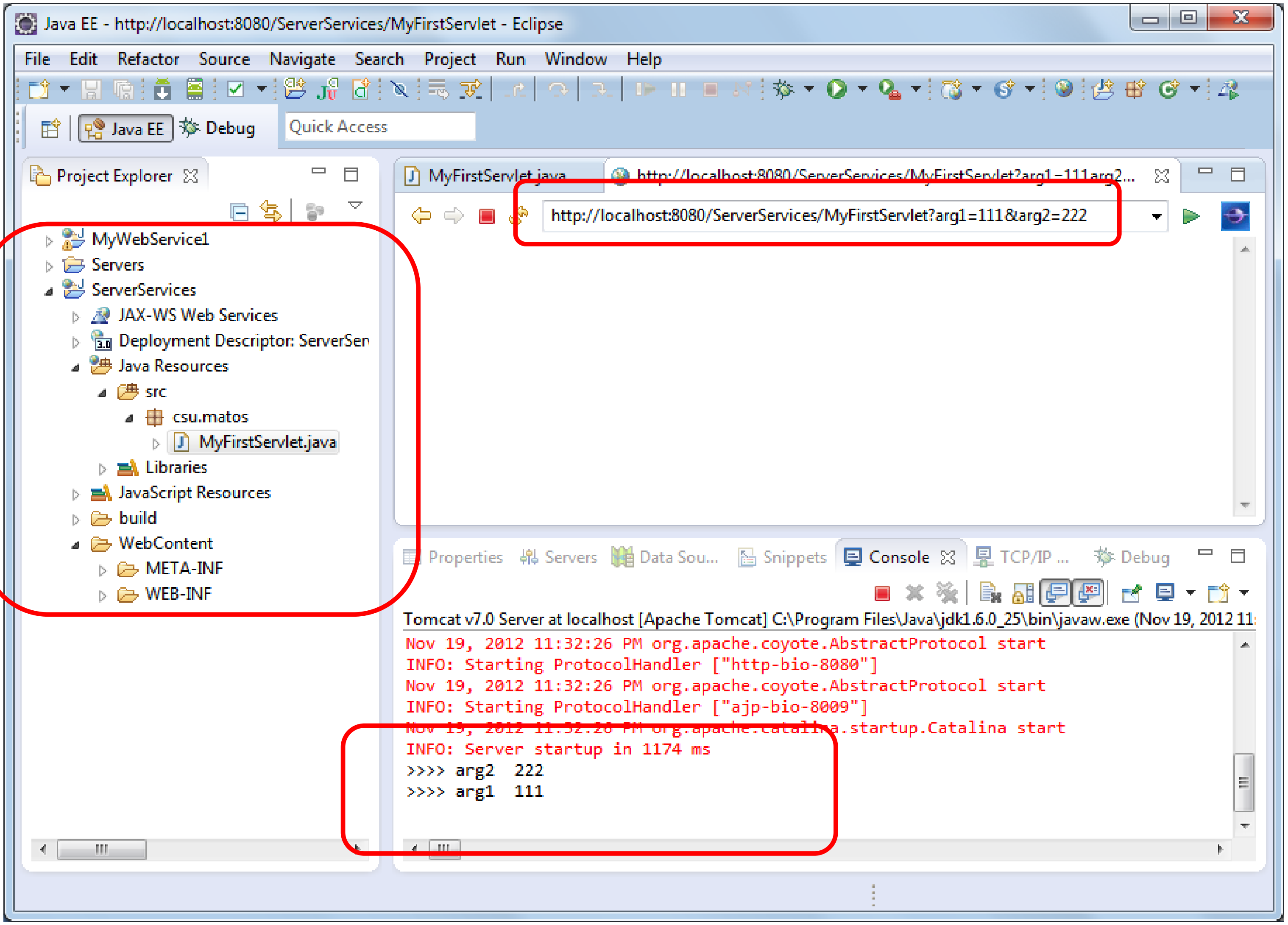
Apache Tomcat v7.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 and 6 Web modules.

☐ Always use this server when running this project

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2A. Server-Side Code - Servlet gets the REQUEST Object



Java EE - http://localhost:8080/ServerServices/MyFirstServlet - Eclipse
File Edit Refactor Source Navigate Search Project Run Window Help
Java EE
Debug
Quick Access
Project Explorer
MyWebService1
Servers
ServerServices
JAX-WS Web Services
Deployment Descriptor: ServerSer
Java Resources
src
csu.matos
MyFirstServlet.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
MyFirstServlet.java
http://localhost:8080/ServerServices/MyFirstServlet?arg1=111arg2...
Properties Servers Data Sou... Snippets Console TCP/IP ... Debug

http://localhost:8080/ServerServices/MyFirstServlet?arg1=111&arg2=222

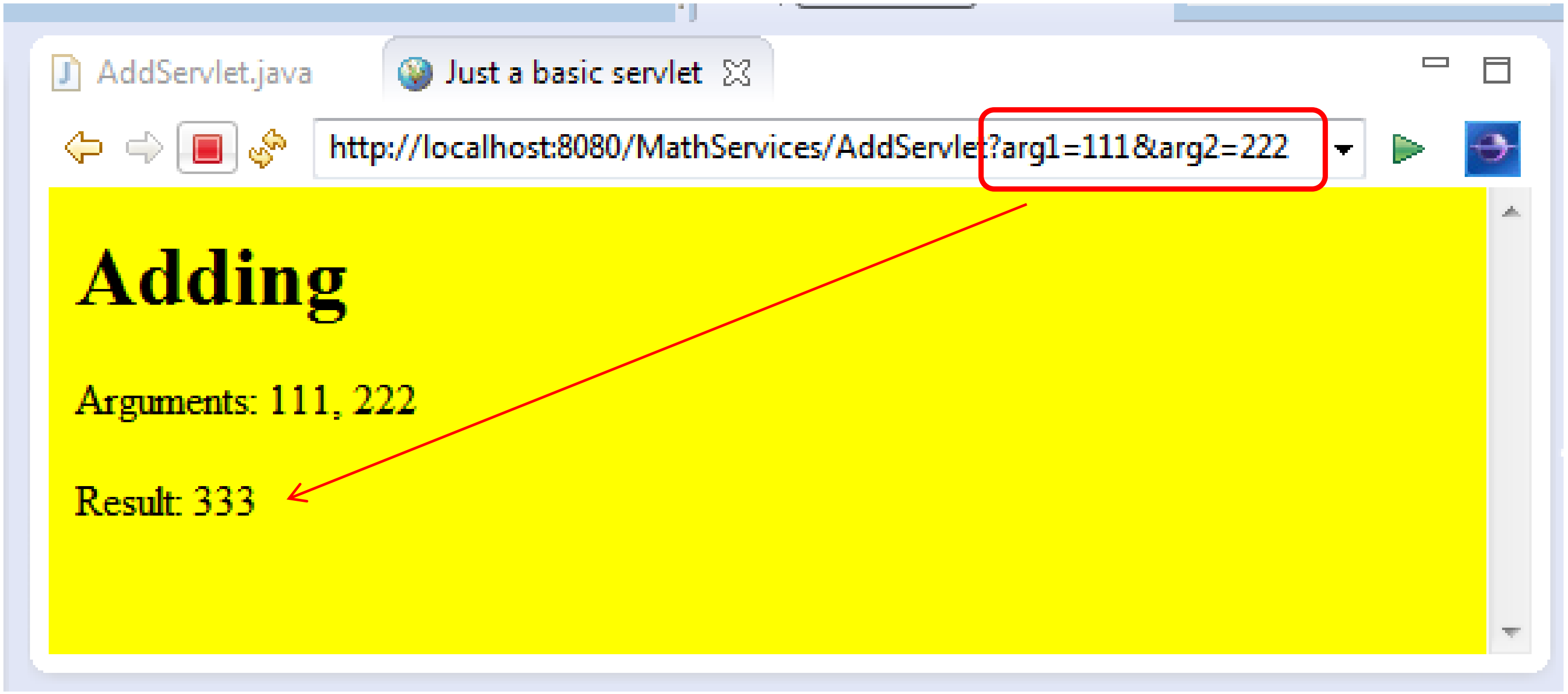
```
Tomcat v7.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk1.6.0_25\bin\javaw.exe (Nov 19, 2012 11:
Nov 19, 2012 11:32:26 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Nov 19, 2012 11:32:26 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
Nov 19, 2012 11:32:26 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1174 ms
>>>> arg2  222
>>>> arg1  111
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2B. A servlet adds two numbers sent in the REQUEST Object
The answer is sent back into the RESPONSE object as an HTML page.

1. Create new 'Web Dynamic Project'. Name it: `MathServices`.
2. Add a new Servlet . Call it: `AddServlet` .
3. The servlet accepts two numbers and adds them.
4. The result is sent back to the user in a dynamically created HTML page



http://localhost:8080/MathServices/AddServlet?arg1=111&arg2=222

```
Adding

Arguments: 111, 222

Result: 333
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2B. Servlet adds two numbers sent in the REQUEST Object

```
package csu.matos;

import . . .

@WebServlet("/AddServlet")
public class AddServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

  public AddServlet() {
  }

  protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
    PrintWriter out = response.getWriter();

    out.println("<html >");
    out.println("<head><title>Just a basic servlet</title></head>");
    out.println("<body bgColor=\"Yellow\" >");
    out.println("<h1>Adding  </h1>");
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2B. Servlet adds two numbers sent in the REQUEST Object

```
    try {
      int arg1 = Integer.parseInt(request.getParameter("arg1"));
      int arg2 = Integer.parseInt(request.getParameter("arg2"));

      out.println("<p> Arguments: " + arg1 + ", " + arg2);

      out.println("<p> Result: " + (arg1 + arg2));

    } catch (Exception e) {
      out.println("<p> Problem with arguments...");
    }
    out.println("</body></html>");
    out.flush();
  }

  protected void doPost(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
  }

}
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2B. Servlet adds two numbers sent in the REQUEST Object

Installing the Servlet in the Production Tomcat Server

1. After successfully testing the service in the Eclipse environment ***Stop*** the instance of Tomcat running in Eclipse.

2. Start the Windows Tomcat service. In Windows do: Start > Control Panel > Administrative Tools > Services > Look for Apache Tomcat 7.0 > Start

3. Locate `Program Files/Apache Software Foundation/Tomcat 7.0/webapps` folder. Create there the subdirectory `/MathServices.`

4. Transfer from the Eclipse's application workspace the two subdirectories in /WebContent (META-INF and WEB-INF) to the folder created in step 3.

5. Transfer from Eclipse's /build folder the subdirectory /classes into **WEB-INF**.

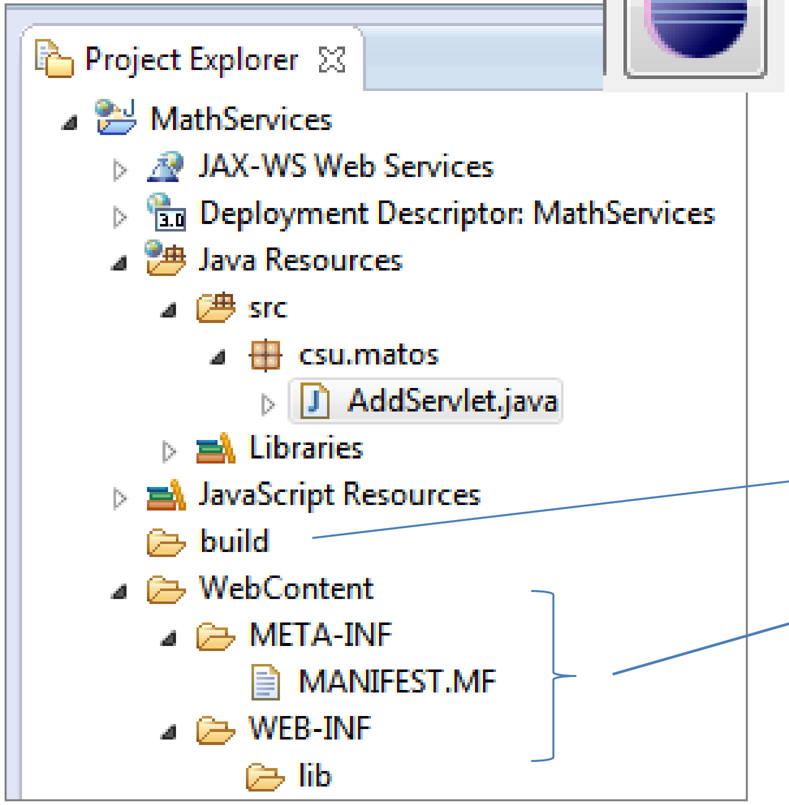
6. In a web browser enter the URL:
   http://localhost:8080/MathServices/AddServlet?arg1=111&arg2=222

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

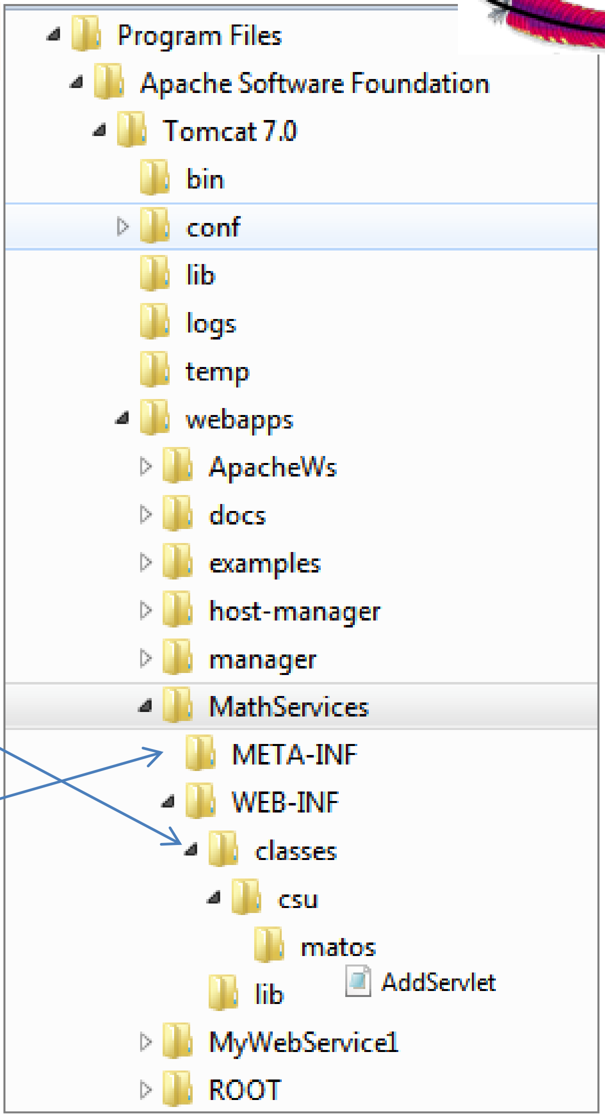
Example 2B. Servlet adds two numbers sent in the REQUEST Object

Eclipse Workspace



Project Explorer
- MathServices
  - JAX-WS Web Services
  - Deployment Descriptor: MathServices
  - Java Resources
    - src
      - csu.matos
        - AddServlet.java
    - Libraries
  - JavaScript Resources
  - build
  - WebContent
    - META-INF
      - MANIFEST.MF
    - WEB-INF
      - lib

Tomcat 7.0

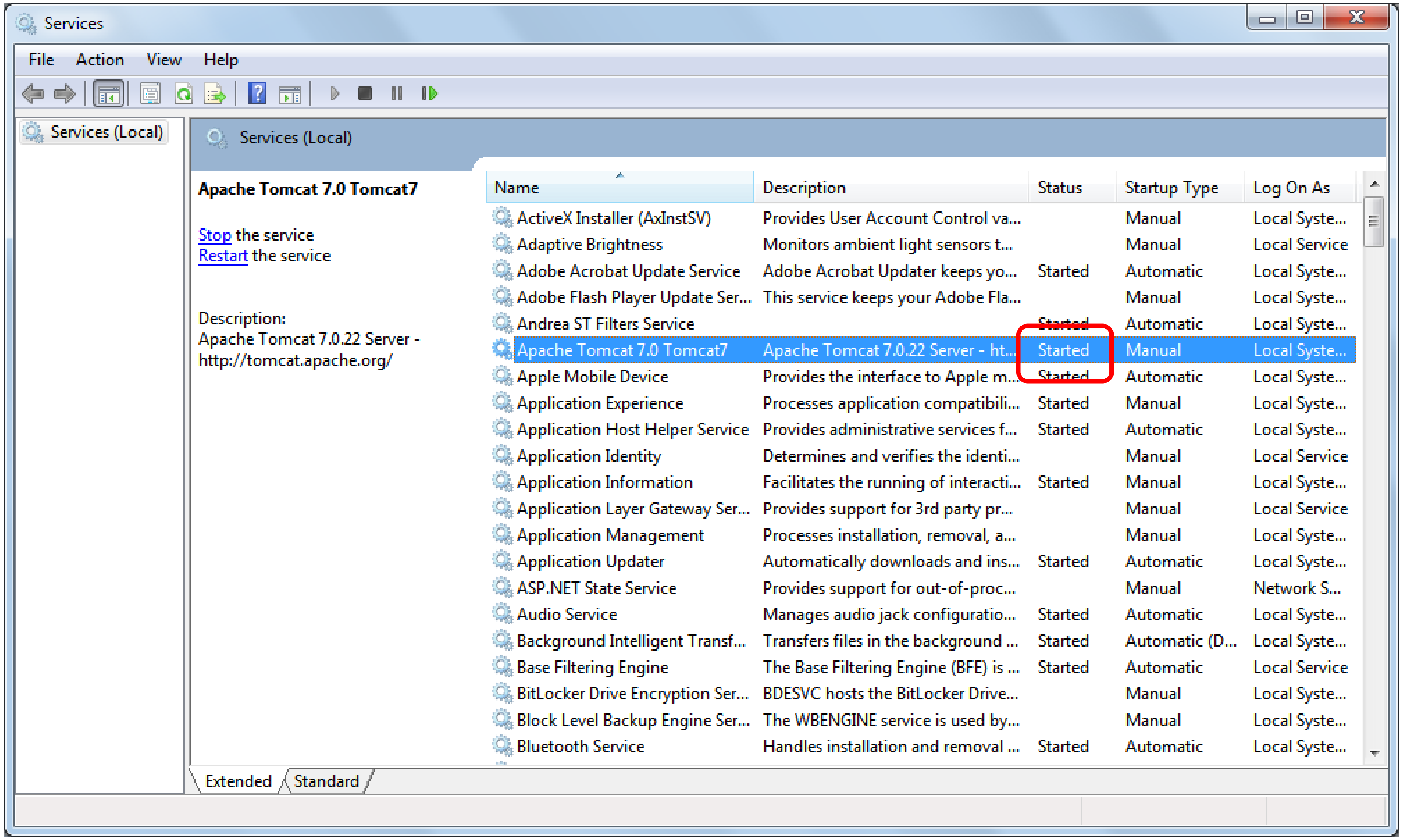


- Program Files
  - Apache Software Foundation
    - Tomcat 7.0
      - bin
      - conf
      - lib
      - logs
      - temp
      - webapps
        - ApacheWs
        - docs
        - examples
        - host-manager
        - manager
        - MathServices
          - META-INF
          - WEB-INF
            - classes
              - csu
                - matos
                  - AddServlet
            - lib
        - MyWebService1
        - ROOT

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2B. Servlet adds two numbers sent in the REQUEST Object



Apache Tomcat 7.0 Tomcat7

Stop the service
Restart the service

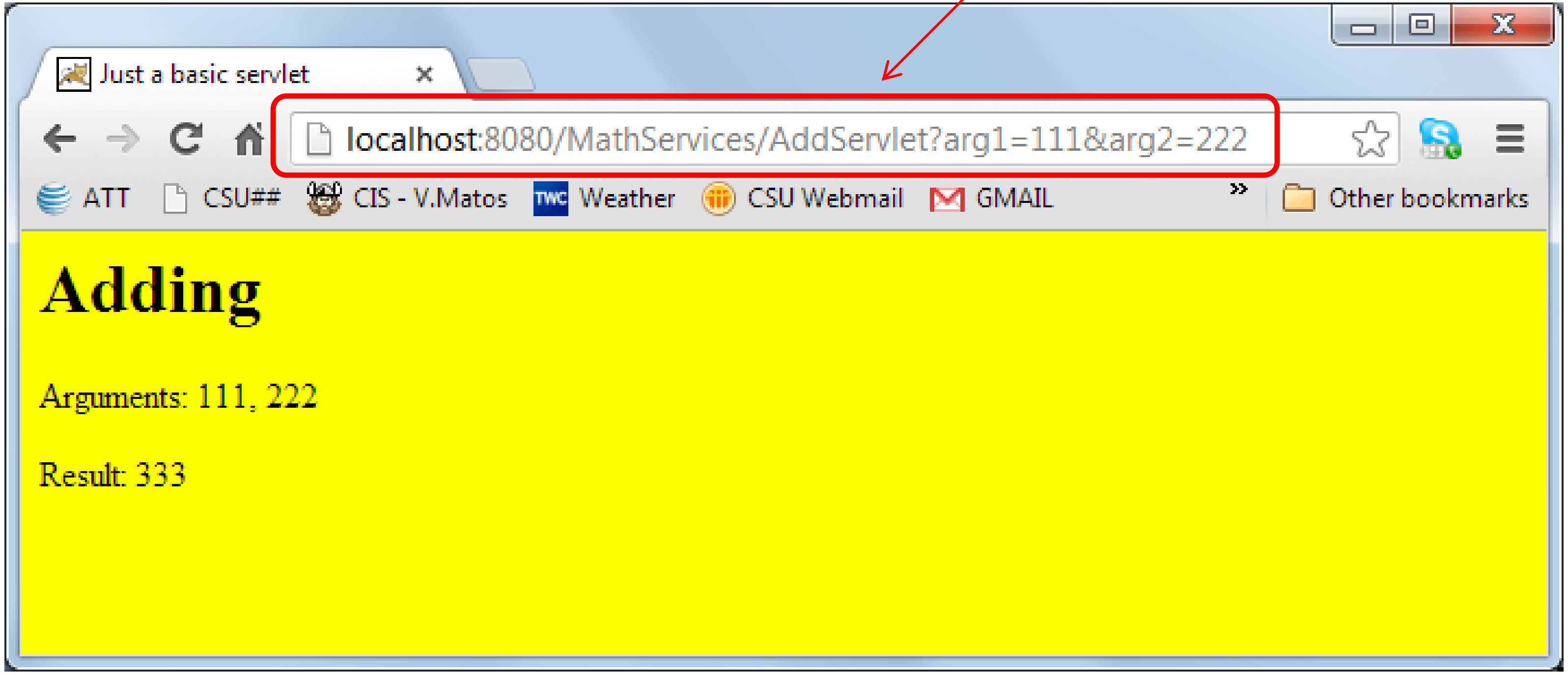
Description:
Apache Tomcat 7.0.22 Server - http://tomcat.apache.org/

Name	Description	Status	Startup Type	Log On As
ActiveX Installer (AxInstSV)	Provides User Account Control va...		Manual	Local Syste...
Adaptive Brightness	Monitors ambient light sensors t...		Manual	Local Service
Adobe Acrobat Update Service	Adobe Acrobat Updater keeps yo...	Started	Automatic	Local Syste...
Adobe Flash Player Update Ser...	This service keeps your Adobe Fla...		Manual	Local Syste...
Andrea ST Filters Service		Started	Automatic	Local Syste...
Apache Tomcat 7.0 Tomcat7	Apache Tomcat 7.0.22 Server - ht...	Started	Manual	Local Syste...
Apple Mobile Device	Provides the interface to Apple m...	Started	Automatic	Local Syste...
Application Experience	Processes application compatibili...	Started	Manual	Local Syste...
Application Host Helper Service	Provides administrative services f...	Started	Automatic	Local Syste...
Application Identity	Determines and verifies the identi...		Manual	Local Service
Application Information	Facilitates the running of interacti...	Started	Manual	Local Syste...
Application Layer Gateway Ser...	Provides support for 3rd party pr...		Manual	Local Service
Application Management	Processes installation, removal, a...		Manual	Local Syste...
Application Updater	Automatically downloads and ins...	Started	Automatic	Local Syste...
ASP.NET State Service	Provides support for out-of-proc...		Manual	Network S...
Audio Service	Manages audio jack configuratio...	Started	Automatic	Local Syste...
Background Intelligent Transf...	Transfers files in the background ...	Started	Automatic (D...	Local Syste...
Base Filtering Engine	The Base Filtering Engine (BFE) is ...	Started	Automatic	Local Service
BitLocker Drive Encryption Ser...	BDESVC hosts the BitLocker Drive...		Manual	Local Syste...
Block Level Backup Engine Ser...	The WBENGINE service is used by...		Manual	Local Syste...
Bluetooth Service	Handles installation and removal ...	Started	Automatic	Local Syste...

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2B. Servlet adds two numbers sent in the REQUEST Object



Adding

Arguments: 111, 222

Result: 333

Note:
Later in this notes, we will learn how to consume this web-service from an Android App

REST Protocol – Android & Apache's Tomcat Server

Example 2C. Multitier Application:
Client ⟷ WebServer ⟷ Database Server

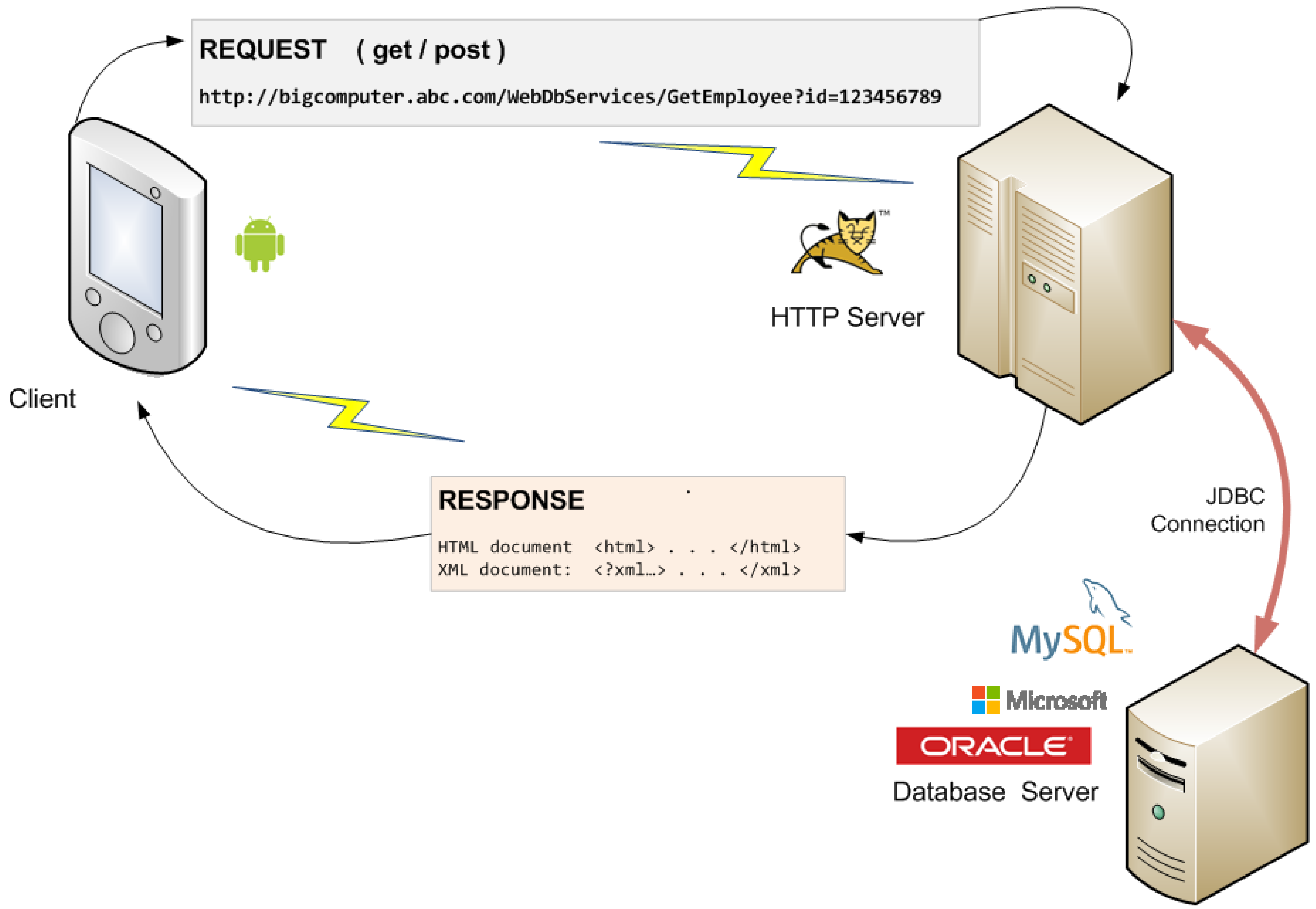
In this example we will create a multi-tier application on which several computers cooperate in serving the client's needs.

1. A servlet residing in a web-server will take from the client's application an employee's id number.

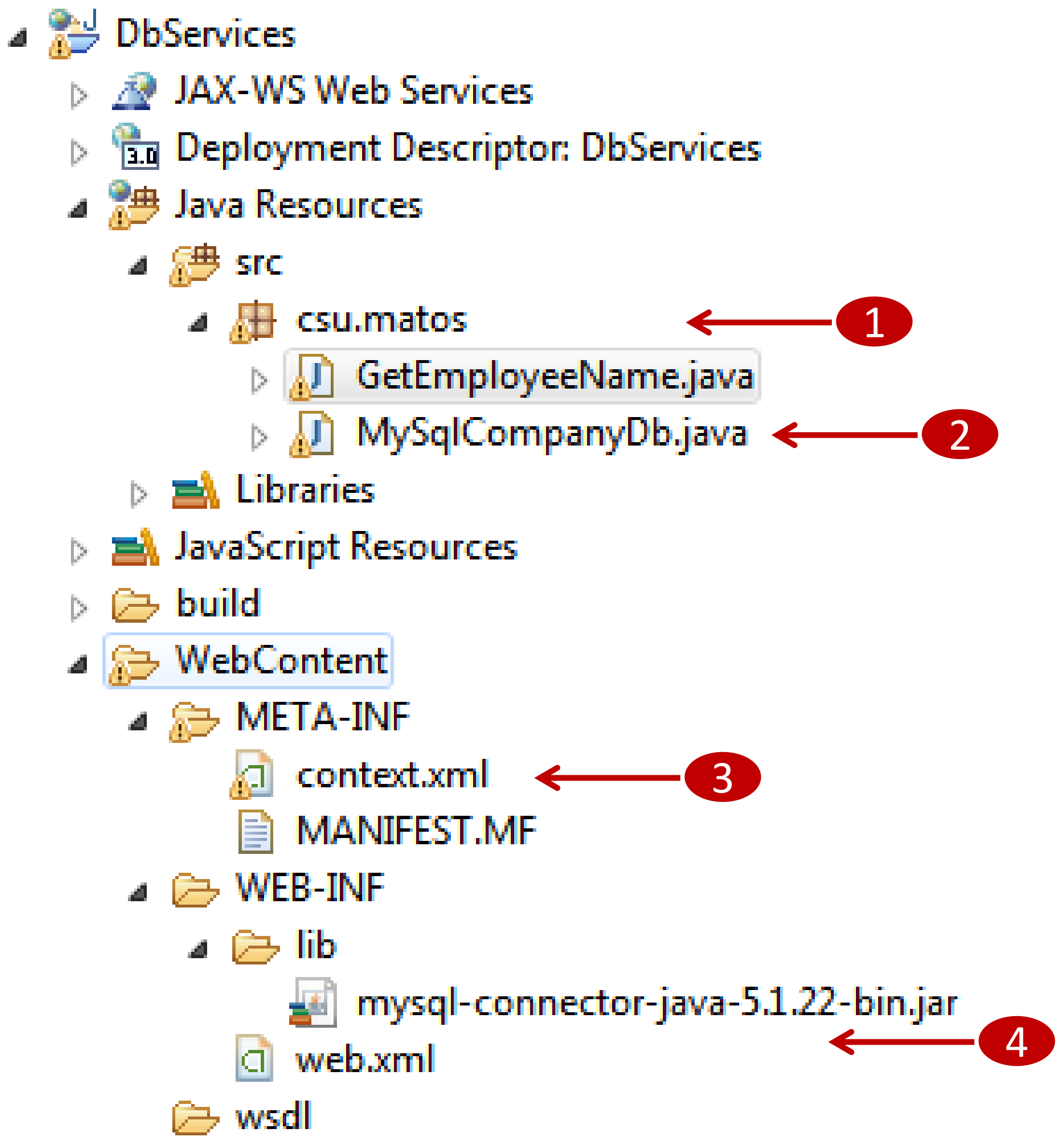
2. The servlet uses JDBC technology to forward the id number to a second server computer where **MySQL** database operations are performed.

3. The front-end web-server collects results from the database computer and delivers the response to the client's device.

Example 2C. Multitier Application – WebServer – Database Server


REQUEST (get / post)
http://bigcomputer.abc.com/WebDbServices/GetEmployee?id=123456789
Client
HTTP Server
RESPONSE
HTML document <html> . . . </html>
XML document: <?xml…> . . . </xml>
JDBC Connection
MySQL
Microsoft
ORACLE
Database Server

Database Access Using Servlet



1. The servlet **GetEmployeeName.java** accepts the user's request containing an employee **id**.
2. The class **MySqlCompabyDb** creates a JDBC connection to the database, sends a SQL statement, and accepts its ResultSet (employee's name) which is returned to the servlet.
3. The **context.xml** file describes the DataSource (driver type, username, password, url) needed to access the database.
4. MySQL JDBC driver is added to **WEB-INF/lib**. The **web.xml** maps the servlet's package organization and its calling name.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

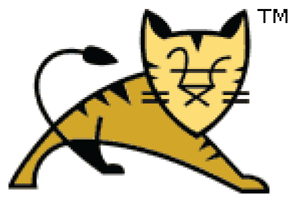
REST Protocol – Android & Apache's Tomcat Server

Example 2C. Multitier Application – WebServer – Database Server

FIRST STEPS – Create the HTTP Server Components

1. Use Eclipse to create a new *Web Dynamic Project*. The project will be called: **DbServices** .
2. Add a new servlet named: **GetEmployeeName**.
3. The servlet awaits for the argument “**id**” holding an employee identification number.
4. Create the class **MySqlCompanyDb** to deal with database operations using a JDBC connection to a remote MySQL database server (see code ahead).
5. Create a server-wide “DataSource” specification (details follow) and add it to the Tomcat’s **/conf/context.xml** file.
6. Create a **web.xml** file (as indicated later) describing deployment information. Add it to the Java application’s **/WebContent** folder.
7. Test the application using the Eclipse environment. When ready to deploy in the production server, follow the six steps suggested in Example 2B.

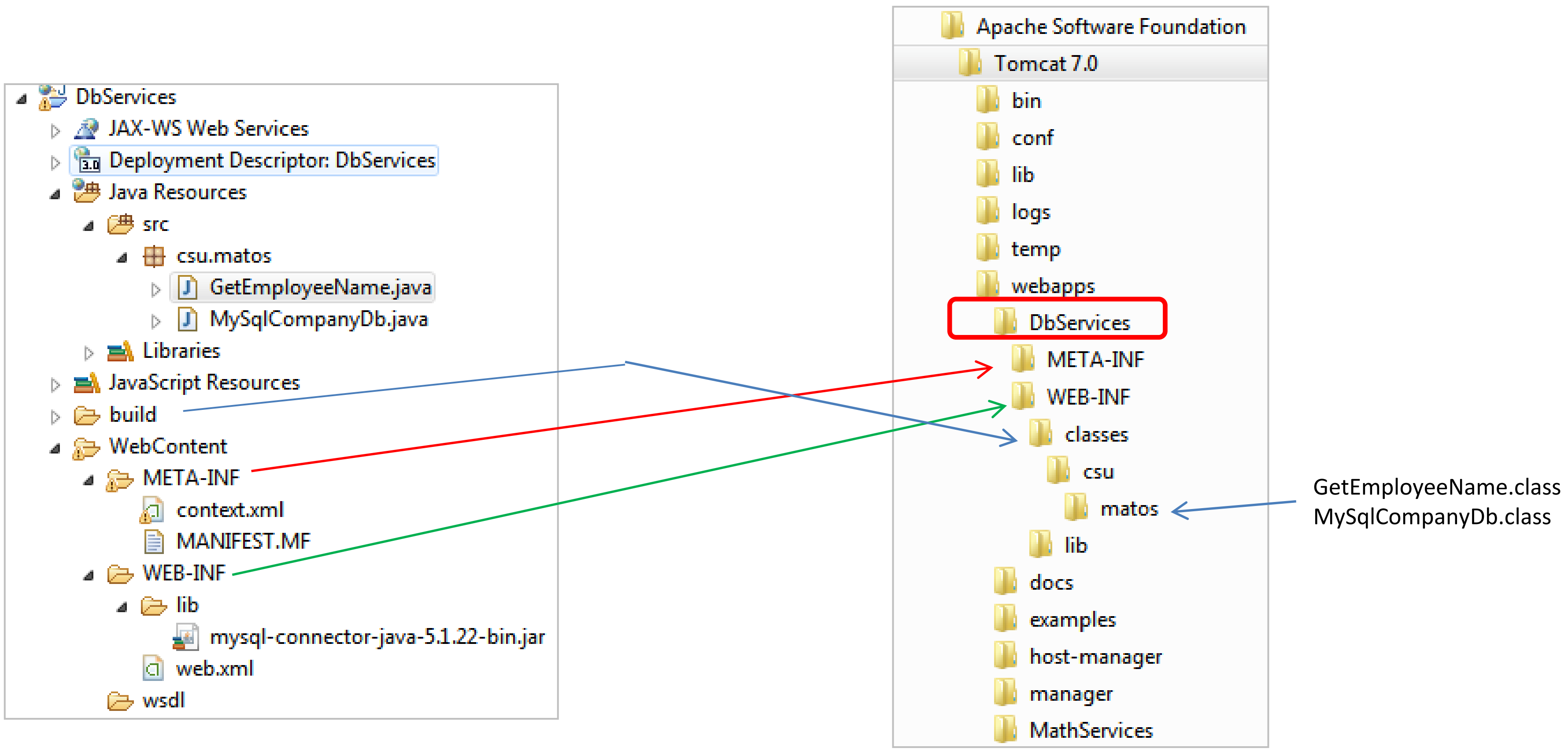
Consuming Web Services
Example 2 – TUTORIAL – Android Code



REST Protocol – Android & Apache's Tomcat Server

Example 2C. Multitier Application – WebServer – Database Server

Tranfering the WebService to the Production Tomcat Server



```
DbServices
  JAX-WS Web Services
  Deployment Descriptor: DbServices
  Java Resources
    src
      csu.matos
        GetEmployeeName.java
        MySqlCompanyDb.java
    Libraries
  JavaScript Resources
  build
  WebContent
    META-INF
      context.xml
      MANIFEST.MF
    WEB-INF
      lib
        mysql-connector-java-5.1.22-bin.jar
      web.xml
    wsdl
```

```
Apache Software Foundation
  Tomcat 7.0
    bin
    conf
    lib
    logs
    temp
    webapps
      DbServices
        META-INF
        WEB-INF
          classes
            csu
              matos
          lib
      docs
      examples
      host-manager
      manager
      MathServices
```

GetEmployeeName.class
MySqlCompanyDb.class

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – WebServer – Database Server

5. Put the following code into the GetEmployee.java servlet (1 of 2)

```
package csu.matos;
import . . .

@WebServlet("/GetEmployeeName")
public class GetEmployeeName extends HttpServlet {
  private static final long serialVersionUID = 1L;

  public GetEmployeeName() {
  }

  protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
    try {
      PrintWriter out = response.getWriter();

      out.println("<html>");
      out.println("<head>");
      out.println("<title>Multi-tier WebService Example</title>");
      out.println("</head>");
      out.println("<body  bgcolor=\"Yellow\">");
      out.println("<h3>Web-Database-Service Testing</h3>");
      HttpSession session = request.getSession(true);
      out.println("<br>Session ID " + session.getId());
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – WebServer – Database Server

5. Put the following code into the servlet (2 of 2)

```
      out.print("<form >");
      String empId = request.getParameter("id");
      out.println("<br>You provided ID:  " + empId);

      MySqlCompanyDb db = new MySqlCompanyDb();
      String strEmployeeName = db.getDbRecord(empId);
      out.println("<br> Employee name from Database: " + strEmployeeName);

      out.println("</form>");
      out.println("</body>");
      out.println("</html>");

    } catch (IOException e) {
      e.printStackTrace();
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
  protected void doPost(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – WebServer – Database Server

6. Add to the package a new class to implement **MySqlCompanyDb** 1 of 2

```
package csu.matos;
import . . .

public class MySqlCompanyDb {
  // Goal: connect to remote MySQL (Company) DB using JDBC.
  public String getDbRecord(String id) throws  ClassNotFoundException,
                                               SQLException {
    String result = "***";
    try
    {
      Context initContext = new InitialContext();

      Context envContext  = (Context)initContext.lookup("java:/comp/env");

      DataSource ds = (DataSource)envContext.lookup("jdbc/mysqlresource");

      Connection cnn = ds.getConnection();

      String mySQL = " select fname, lname  "
                   + " from employee where ssn = " + id;
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

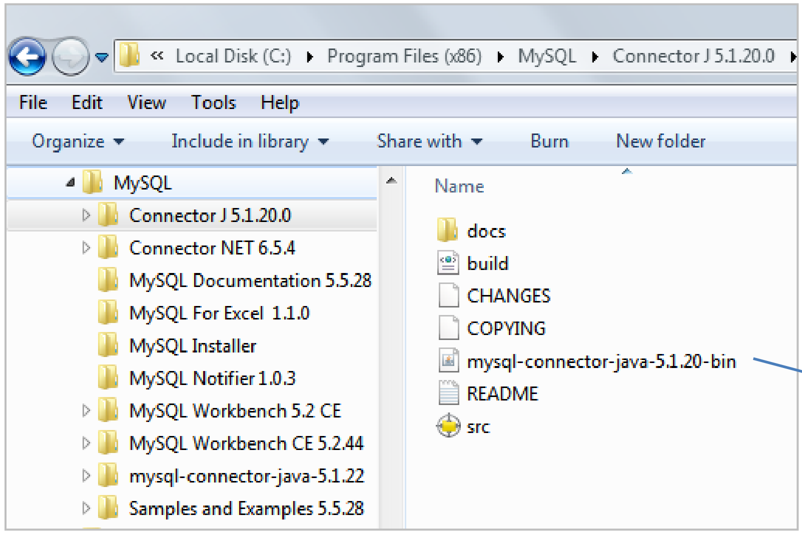
Example 2C. Multitier Application – WebServer – Database Server

6. Add to the package a new class to implement **MySqlCompanyDb** 2 of 2

```
      Statement stm = cnn.createStatement();

      ResultSet rs = stm.executeQuery(mySQL);

      while ( rs.next() ){
        result = rs.getString("fname") + " " + rs.getString("lname");
      }

    }
      catch (SQLException e) {
      result = " [*[Problems1 ]*] " + e.getMessage();

      } catch (NamingException e) {
        result = " [*[Problems2 ]*] " + e.getMessage();
      e.printStackTrace();
    }
      return result;
  }//getDbRecord

}// class
```

Example 2C. Multitier Application – WebServer – Database Server

8. In order for the JDBC component to work you must add the corresponding database driver to the app (this example uses MySQL, other common drivers are for Oracle, SQL-Server, MS-Access, etc).
9. Locate the "mysql-connector-java-bin.jar" driver (it is usually stored in c:\Program Files\MySQL\Connector J\ folder).
10. Add a copy of it to the WEB-INF/lib folder

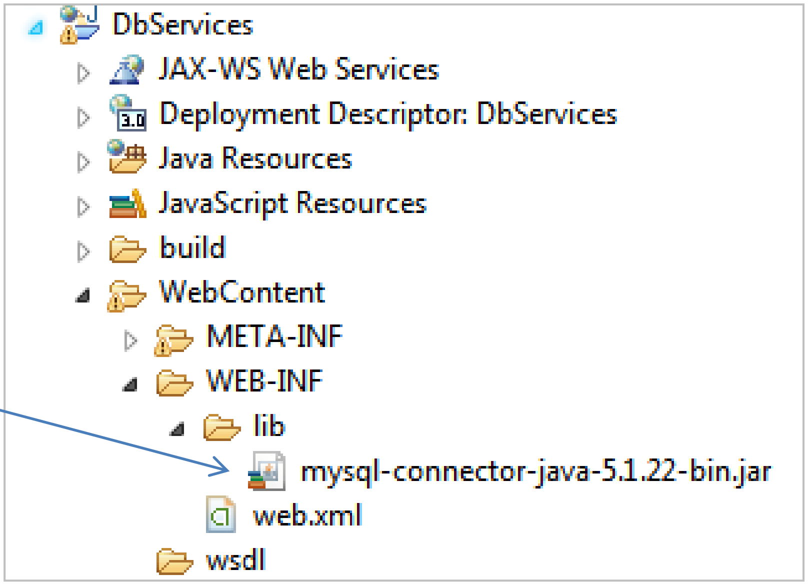

« Local Disk (C:) ▸ Program Files (x86) ▸ MySQL ▸ Connector J 5.1.20.0 ▸

File Edit View Tools Help

Organize ▾ Include in library ▾ Share with ▾ Burn New folder

- MySQL
  - Connector J 5.1.20.0
  - Connector NET 6.5.4
  - MySQL Documentation 5.5.28
  - MySQL For Excel 1.1.0
  - MySQL Installer
  - MySQL Notifier 1.0.3
  - MySQL Workbench 5.2 CE
  - MySQL Workbench CE 5.2.44
  - mysql-connector-java-5.1.22
  - Samples and Examples 5.5.28

Name
- docs
- build
- CHANGES
- COPYING
- mysql-connector-java-5.1.20-bin
- README
- src

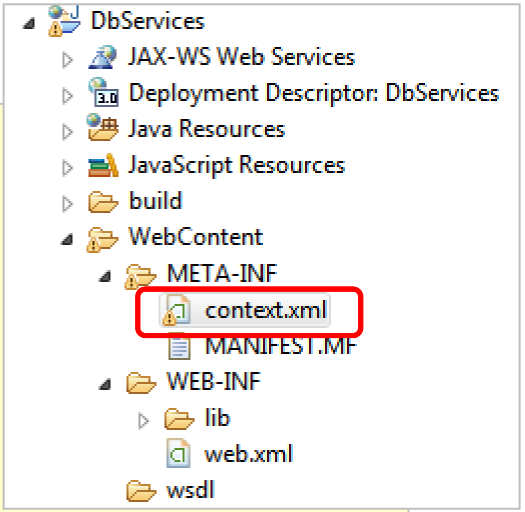

- DbServices
  - JAX-WS Web Services
  - Deployment Descriptor: DbServices
  - Java Resources
  - JavaScript Resources
  - build
  - WebContent
    - META-INF
    - WEB-INF
      - lib
        - mysql-connector-java-5.1.22-bin.jar
      - web.xml
    - wsdl

Download driver from: www.mysql.org

REST Protocol – Android & Apache's Tomcat Server

Example 2C. Multitier Application – WebServer – Database Server

11. Add the following **DataSource** specification to the application's **context.xml** file.
12. Place the file into the **/WebControl/META-INF/** folder of your Eclipse workspace solution (later, this fragment will be copied to the Tomcat's **/conf/context.xml file**)



```
DbServices
  JAX-WS Web Services
  Deployment Descriptor: DbServices
  Java Resources
  JavaScript Resources
  build
  WebContent
    META-INF
      context.xml
      MANIFEST.MF
    WEB-INF
      lib
      web.xml
    wsdl
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <!-- comments removed -->
  <Resource   name="jdbc/mysqlresource"
              auth="Container"
              type="javax.sql.DataSource"
              maxActive="100" maxIdle="30" maxWait="10000"
              username="csuperson"
              password="euclid"
              driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://192.168.1.66:3306/company"/>
</Context>
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

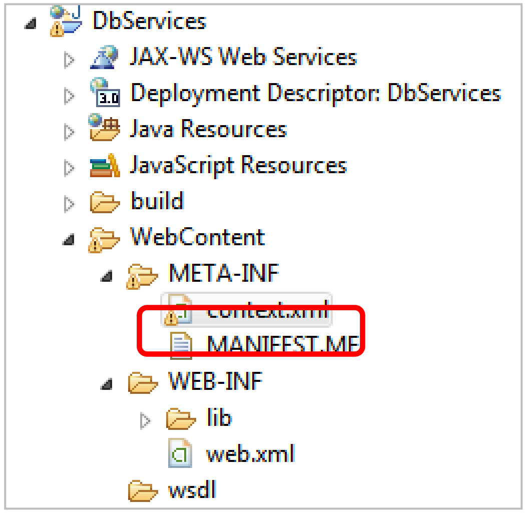
Example 2C. Multitier Application – WebServer – Database Server

12 *cont.* The previous DataSource assumes the existence of a database user identified as **csuperson** (password: **euclid**). The user has access to a database (called **Company**) in which **Employee** data will be found.

A *DataSource* helps the JDBC connection identify the involved server and user.

Other users could reuse the data-source and provide individual credentials.

Reference:
http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html
http://tomcat.apache.org/tomcat-5.5-doc/jndi-resources-howto.html



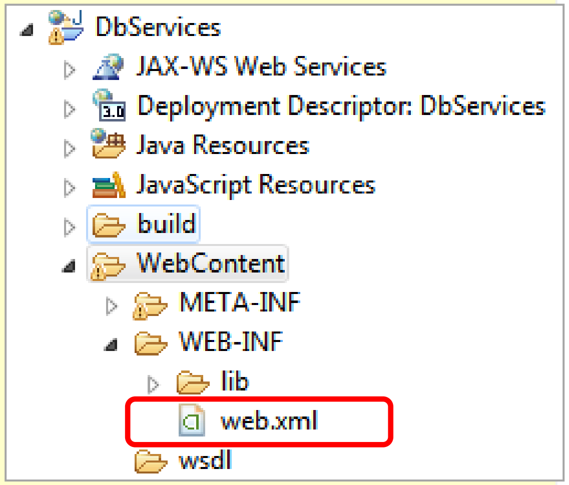
```
DbServices
  JAX-WS Web Services
  Deployment Descriptor: DbServices
  Java Resources
  JavaScript Resources
  build
  WebContent
    META-INF
      context.xml
      MANIFEST.MF
    WEB-INF
      lib
      web.xml
    wsdl
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – WebServer – Database Server

13-1. Add the following **web.xml** file to the **/WebControl/WEB-INF/** Eclipse's folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">

  <display-name>DbServices</display-name>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
  </welcome-file-list>

  <servlet>
    <description></description>
    <display-name>GetEmployeeName</display-name>
    <servlet-name>GetEmployeeName</servlet-name>
    <servlet-class>csu.matos.GetEmployeeName</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>GetEmployeeName</servlet-name>
    <url-pattern>/GetEmployeeName</url-pattern>
  </servlet-mapping>
```



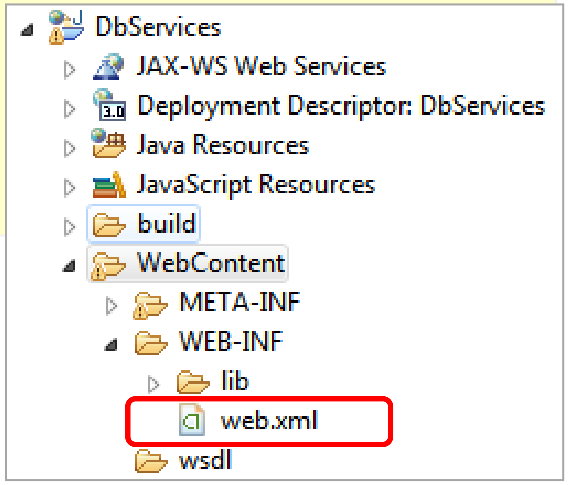
```
DbServices
  JAX-WS Web Services
  Deployment Descriptor: DbServices
  Java Resources
  JavaScript Resources
  build
  WebContent
    META-INF
    WEB-INF
      lib
      web.xml
    wsdl
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – WebServer – Database Server

13-2. cont. Add the following **web.xml** file to the **/WebControl/WEB-INF/** Eclipse's folder.

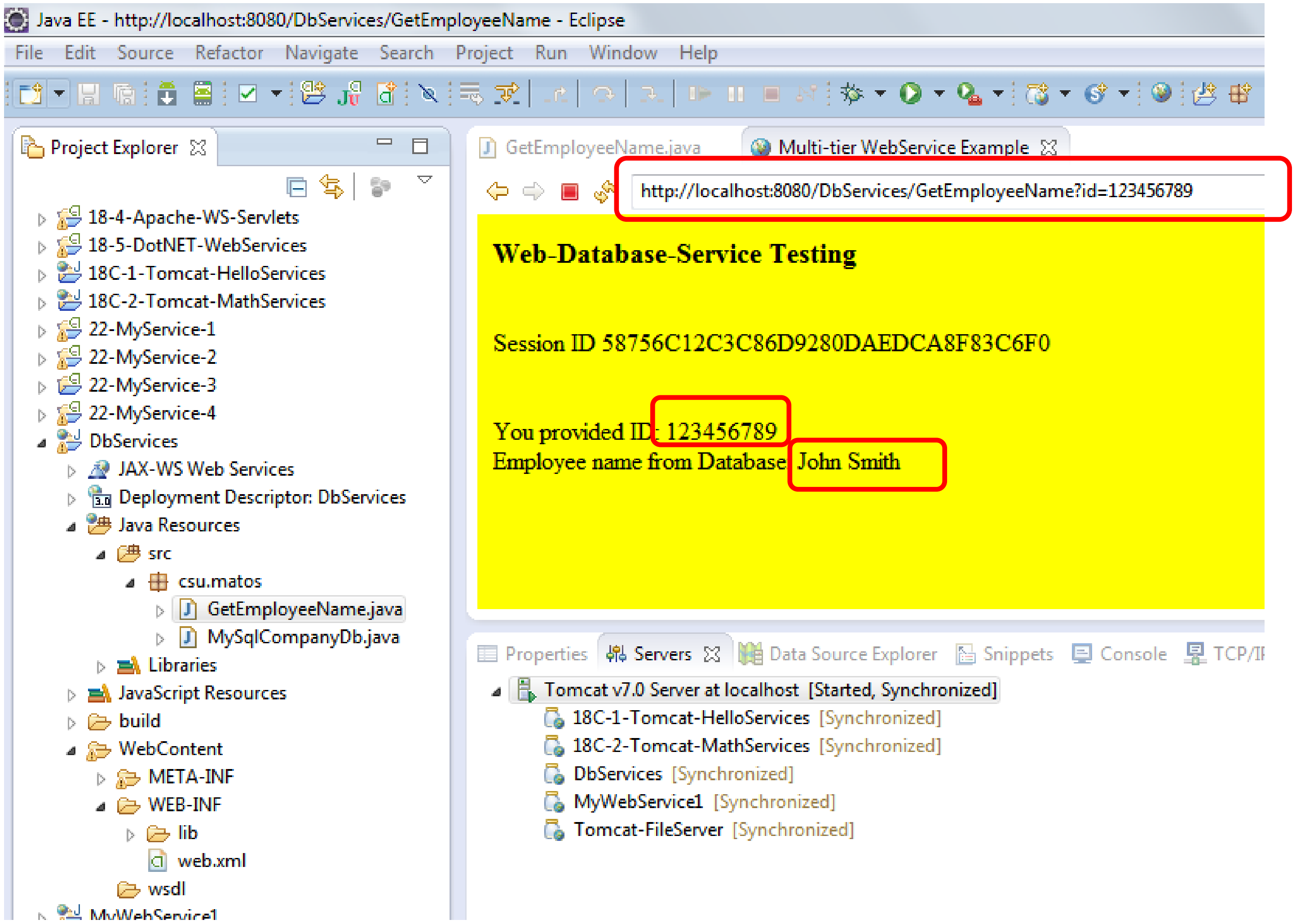
```
  <resource-ref>
    <description>MySQL Datasource - CompanyDB Example</description>
    <res-ref-name>jdbc/mysqlresource</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>

</web-app>
```



Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – WebServer – Database Server

14. Test your application from Eclipse. In the browser window enter the URL:
`http//localhost:8080/WebDbServices/GetEmployee?id=123456789`



Java EE - http://localhost:8080/DbServices/GetEmployeeName - Eclipse

http://localhost:8080/DbServices/GetEmployeeName?id=123456789

Web-Database-Service Testing

Session ID 58756C12C3C86D9280DAEDCA8F83C6F0

You provided ID: 123456789
Employee name from Database: John Smith

Tomcat v7.0 Server at localhost [Started, Synchronized]
- 18C-1-Tomcat-HelloServices [Synchronized]
- 18C-2-Tomcat-MathServices [Synchronized]
- DbServices [Synchronized]
- MyWebService1 [Synchronized]
- Tomcat-FileServer [Synchronized]

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – WebServer – Database Server

Migrate your Eclipse solution to the Apache Tomcat Server. Do this…

15. Locate your Tomcat directory. Create the folder: **/webappps/DbServices**

16. Copy there the contents of your Eclipse's directory **/WebContents** (it contains **META-INF** and **WEB-INF**).

17. Copy to the Tomcat's **/webapps/DbServices/WEB-INF** folder the contents of your Eclipse's **/DbServices/build** directory (it contains: /classes/...)

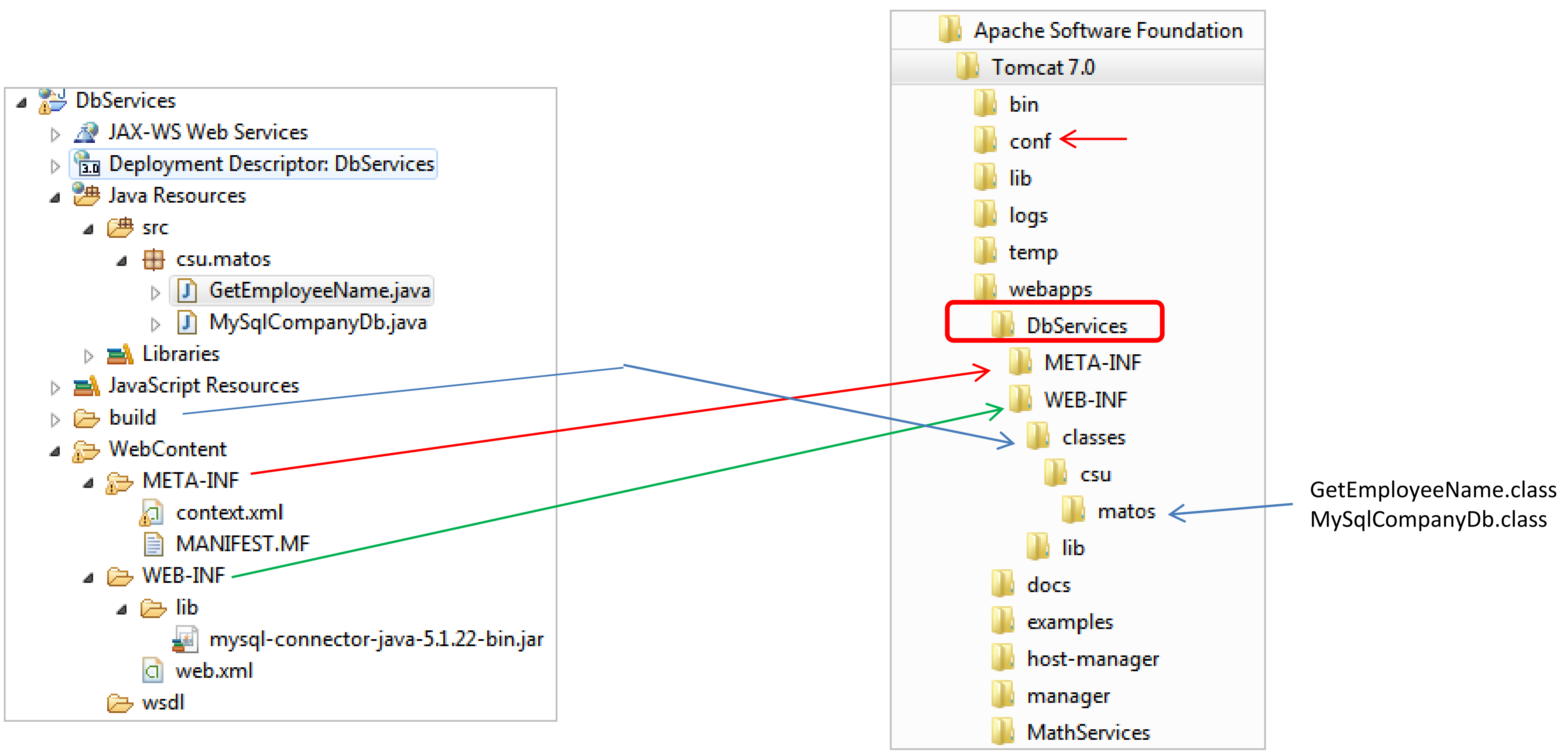
18. Modify the Tomcat's **/conf/context.xml** file. Add the data source tag <Resource . . . /> introduced in step 8 (place it just before </Context>)

19. Test the application. Use a browser and enter the URL `http://localhost:8080/DbServices/GetEmployeeName?id=123456789`

REST Protocol – Android & Apache's Tomcat Server

Example 2C. Multitier Application – WebServer – Database Server

Tranfering the WebService to the Production Tomcat Server



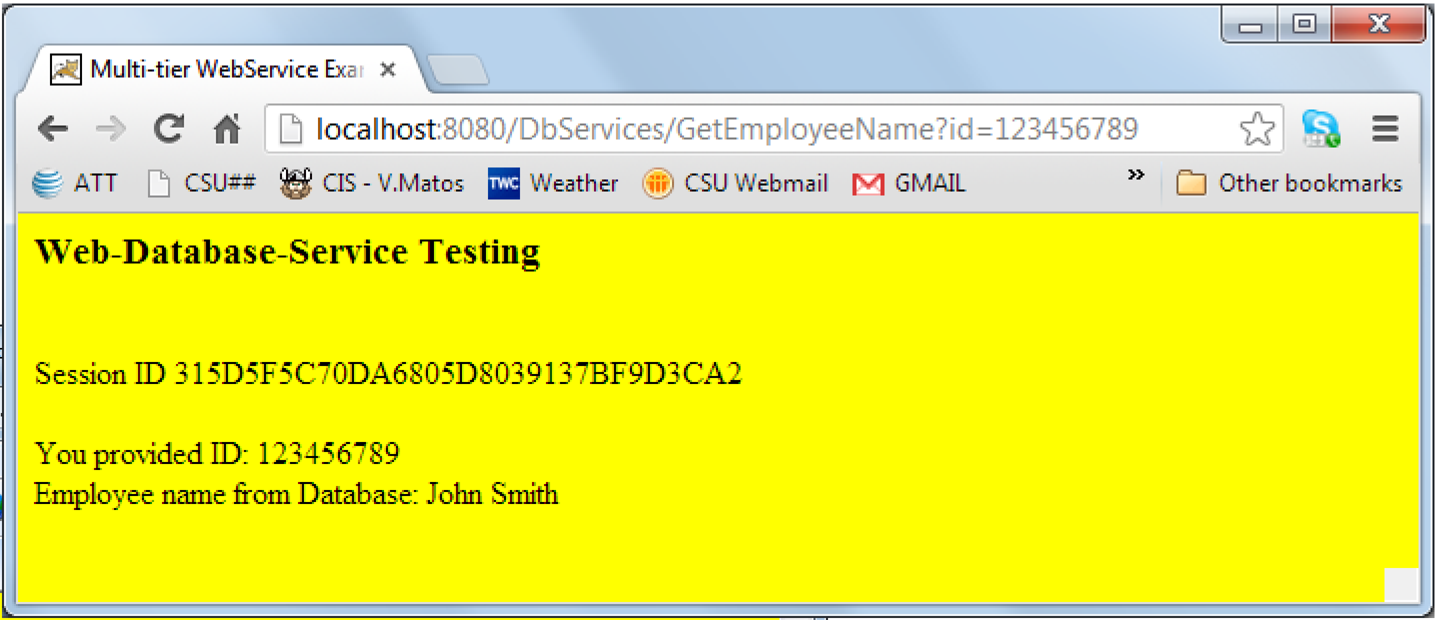
Consuming Web Services
Example 2 – TUTORIAL – Android Code

REST Protocol – Android & Apache's Tomcat Server

Example 2C. Multitier Application – WebServer – Database Server

Testing Production Application with a Browser

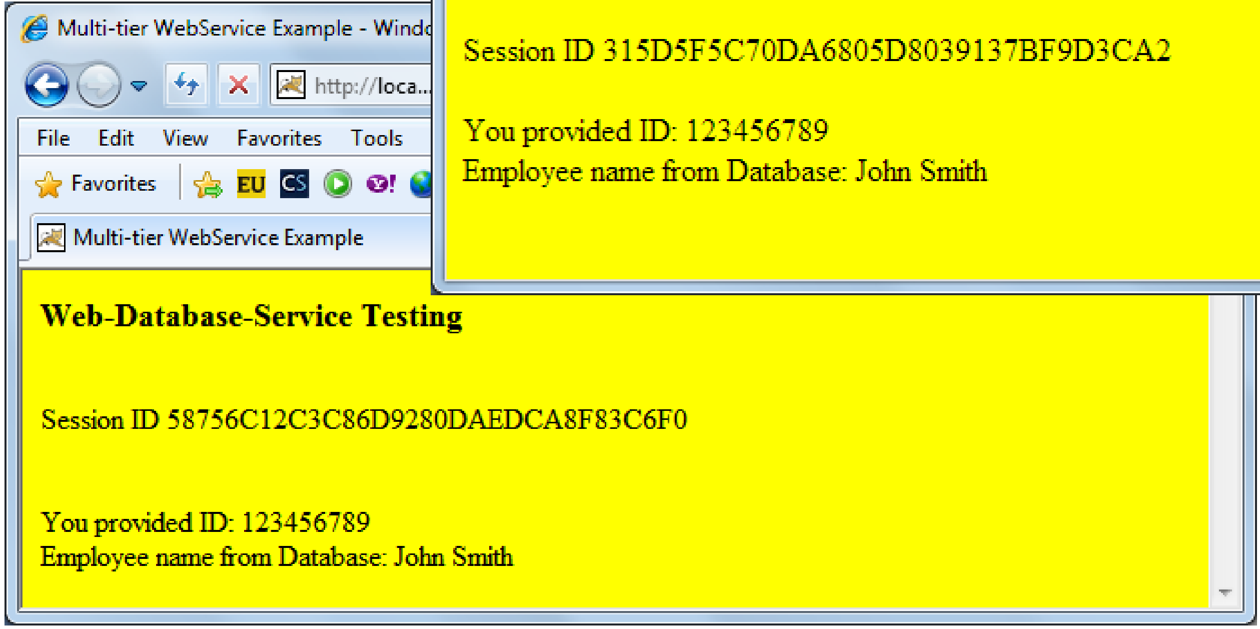
Google Chrome


Multi-tier WebService Exar
localhost:8080/DbServices/GetEmployeeName?id=123456789
ATT
CSU##
CIS - V.Matos
Weather
CSU Webmail
GMAIL
Other bookmarks

Web-Database-Service Testing

Session ID 315D5F5C70DA6805D8039137BF9D3CA2

You provided ID: 123456789
Employee name from Database: John Smith


Multi-tier WebService Example - Wind
http://loca...
File
Edit
View
Favorites
Tools
Favorites
Multi-tier WebService Example

Web-Database-Service Testing

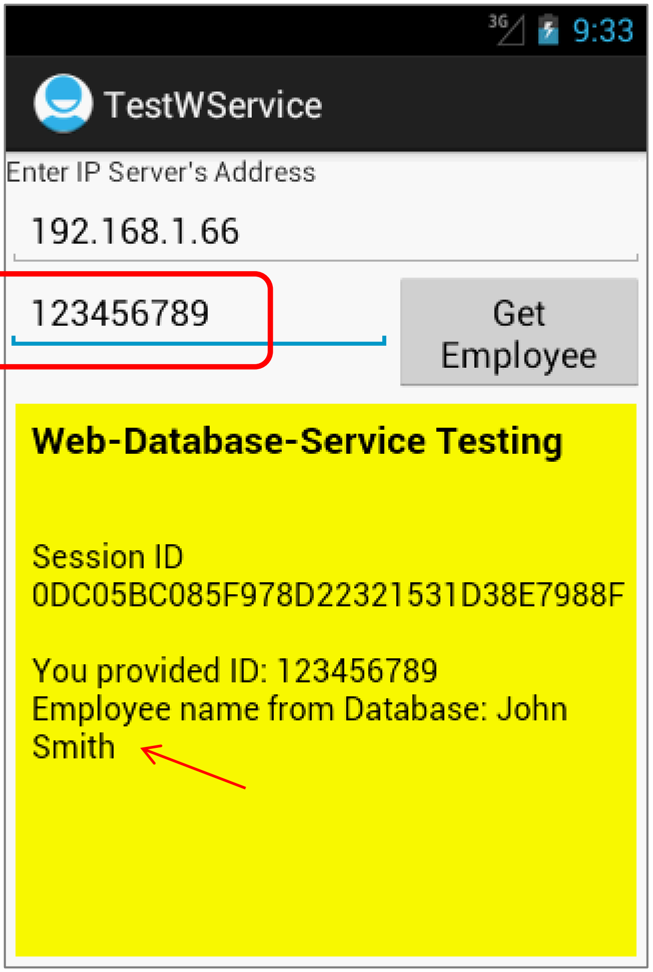
Session ID 58756C12C3C86D9280DAEDCA8F83C6F0

You provided ID: 123456789
Employee name from Database: John Smith

Internet Explorer

Example 2C. Multitier Application – Client Android App

14. On this phase we will develop the **ClientDb** Android App to reach the Tomcat Server



Enter IP Server's Address

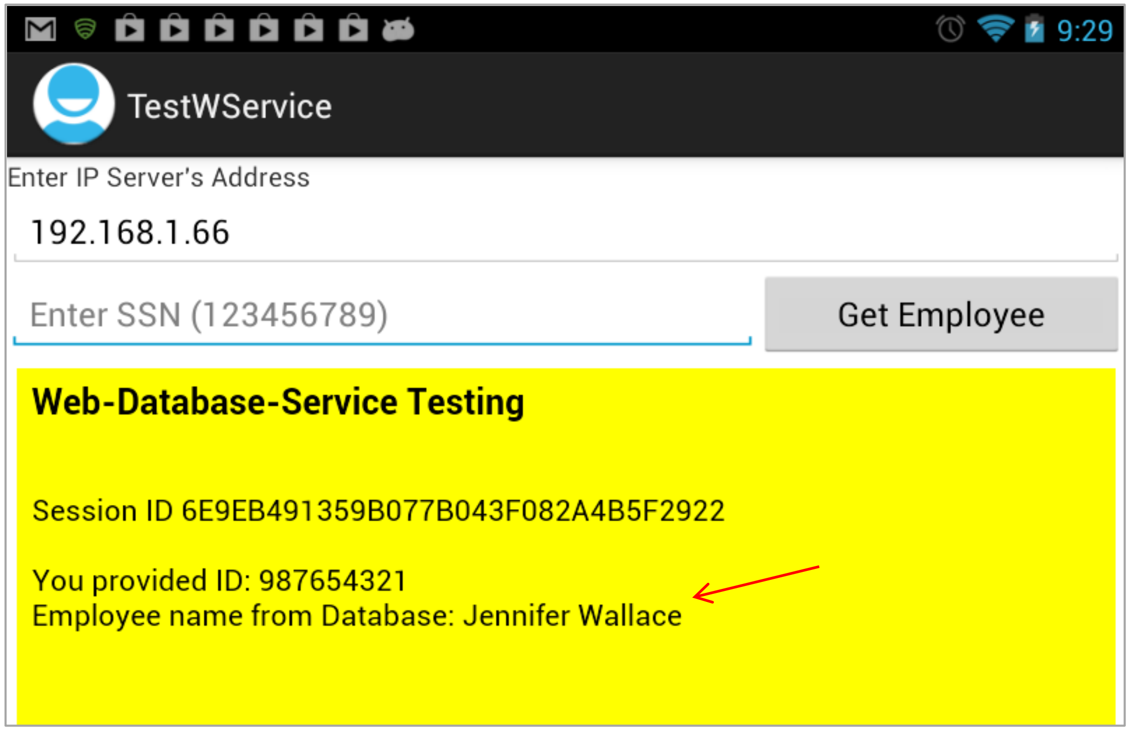
192.168.1.66

123456789

Web-Database-Service Testing

Session ID 0DC05BC085F978D22321531D38E7988F

You provided ID: 123456789
Employee name from Database: John Smith



Enter IP Server's Address

192.168.1.66

Enter SSN (123456789)

Web-Database-Service Testing

Session ID 6E9EB491359B077B043F082A4B5F2922

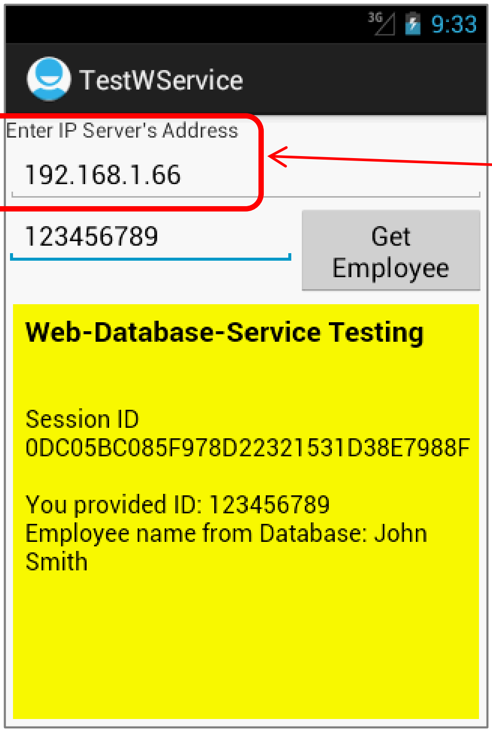
You provided ID: 987654321
Employee name from Database: Jennifer Wallace

User supplies the server's IP address and employee's ID number. The system returns her full name.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – Client Android App

14. On this phase we will develop the **ClientDb** Android App to reach the Tomcat Server



192.168.1.66

123456789

Web-Database-Service Testing

Session ID
0DC05BC085F978D22321531D38E7988F

You provided ID: 123456789
Employee name from Database: John Smith

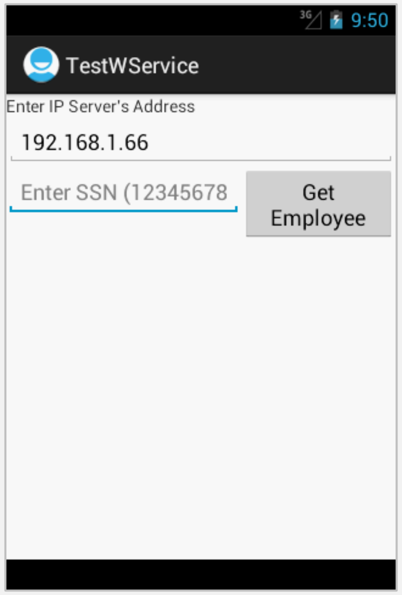
- You need to provide the real IP address. "**Localhost**" is not recognized.
- Since SDK4.0 slow tasks cannot be performed in the main UI. Therefore we will use an **AsynTask** to interact with the remote server.

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – Client Android App

14. Developing the **ClientDb** Android App to reach the Tomcat Server

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Enter IP Server&apos;s Address" />

    <EditText
        android:id="@+id/txtServerIP"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="192.168.1.66" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
```

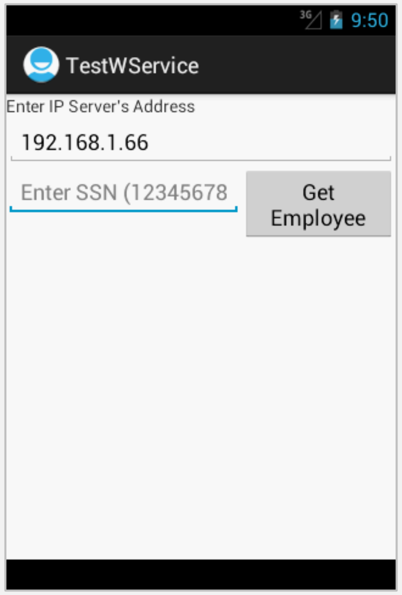


Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – Client Android App

14. Developing the **ClientDb** Android App to reach the Tomcat Server

```
        <EditText
            android:id="@+id/txtId"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.94"
            android:hint="Enter SSN (123456789)"
            android:inputType="number" >
            <requestFocus />
        </EditText>

        <Button
            android:id="@+id/btnGetEmployee"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.31"
            android:text="Get Employee" />
    </LinearLayout>

    <WebView
        android:id="@+id/webView1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="5dip" />
</LinearLayout>
```



Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – Client Android App

15. Developing the **ClientDb** Android App to reach the Tomcat Server (1 of 4)

```
package csu.matos;

import . . .

public class TestWService extends Activity {

    WebView  webview;
    EditText txtId;
    EditText txtServerIP;
    Button btnGetEmployee;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txtServerIP = (EditText)findViewById(R.id.txtServerIP);
        txtId = (EditText)findViewById(R.id.txtId);
        webview = (WebView) findViewById(R.id.webView1);
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – Client Android App

15. Developing the **ClientDb** Android App to reach the Tomcat Server (2 of 4)

```
        btnGetEmployee = (Button)findViewById(R.id.btnGetEmployee);
        btnGetEmployee.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
          // Call Tomcat server
          new SlowInternetTask().execute(null, null, null);
      }
    });
    }//onCreate

    protected void callServiceGetEmployee() {
      try {
        // use actual tcp-ip address, do not enter "localhost:8080"
        // http://192.168.1.66:8080/DbServices/GetEmployeeName?id=123456789

        String url = "http://"
               + txtServerIP.getText().toString()
               + ":8080/DbServices/GetEmployeeName?id="
               + txtId.getText().toString();

        DefaultHttpClient client = new DefaultHttpClient();
        HttpGet methodGet = new HttpGet(new URI(url));
        HttpResponse response = client.execute(methodGet);
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – Client Android App

15. Developing the **ClientDb** Android App to reach the Tomcat Server (3 of 4)

```
        InputStream isResponseData = response.getEntity().getContent();
        String strResponseData = generateString(isResponseData);

        // responseData is an HTML document.
        // Just dump it into the Android's WebView
        webview.loadDataWithBaseURL(null, strResponseData,
                                     "text/html", "UTF-8", null);
      } catch (Exception e) {
        webview.loadDataWithBaseURL(null,
                                    "<html>Problems...</html>",
                                          "text/html", "UTF-8", null);

       }

    }
```

Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – Client Android App

15. Developing the **ClientDb** Android App to reach the Tomcat Server (3 of 4)

```
public class SlowInternetTask extends AsyncTask<String, String, String> {
   private final ProgressDialog dialog = new
                                          ProgressDialog(TestWService.this)
   @Override
   protected void onPreExecute() {
      super.onPreExecute();
      dialog.setMessage("Please wait\n Getting data from remote server");
      dialog.show();
   }

   @Override
   protected String doInBackground(String... params) {
      callServiceGetEmployee();
      dialog.dismiss();
      return null;
   }

}// SlowInternetTask
```

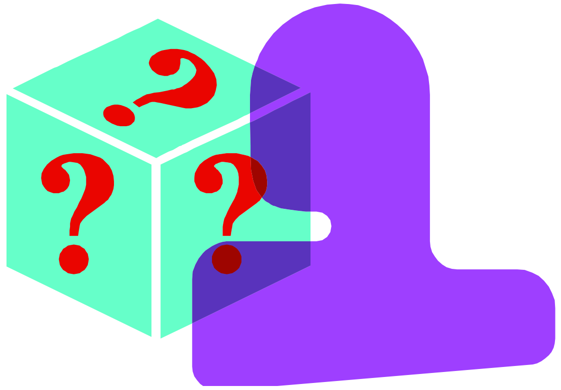
Consuming Web Services
Example 2 – TUTORIAL – Android Code

Example 2C. Multitier Application – Client Android App

15. Developing the **ClientDb** Android App to reach the Tomcat Server (4 of 4)

```
  public String generateString(InputStream stream) {
    // convert input byte stream into a buffered character stream
    BufferedReader buffer = new BufferedReader(new InputStreamReader(stream));
    // extract lines from buffer, put then in string builder
    StringBuilder stringbuilder = new StringBuilder();
    // get input lines (HTML material in this example)
    try {
      String currentLine = "";
      while ((currentLine = buffer.readLine()) != null) {
        Log.d("currentLine", currentLine);
        stringbuilder.append(currentLine + "\n");
      }
    } catch (IOException e) {
      Log.e("<<ERROR2_1>>", e.getMessage());
    }
    try {
      stream.close();
    } catch (IOException e) {
      Log.e("<<ERROR2_2>>", e.getMessage());
    }
    return stringbuilder.toString();
  }
}
```

Producing & Consuming Web Services



< Questions />

Appendix A. Connecting to Oracle DBMS

REST Protocol – Android & Apache's Tomcat Server

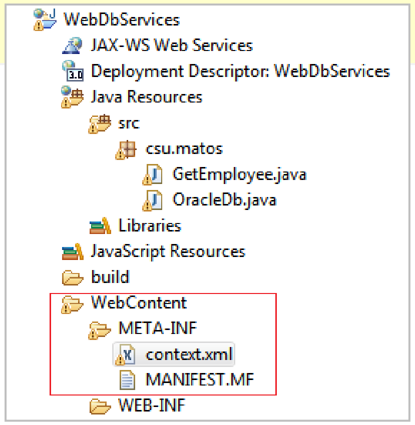
This relaces Step 8 in Example 2C. Multitier Application

Add the following **DataSource** to the application's **context.xml** file. Add the file to the **/WebControl/META-INF/** folder of your Eclipse workspace solution (later, this fragment will be copied to the Tomcat's **/conf/context.xml file**)

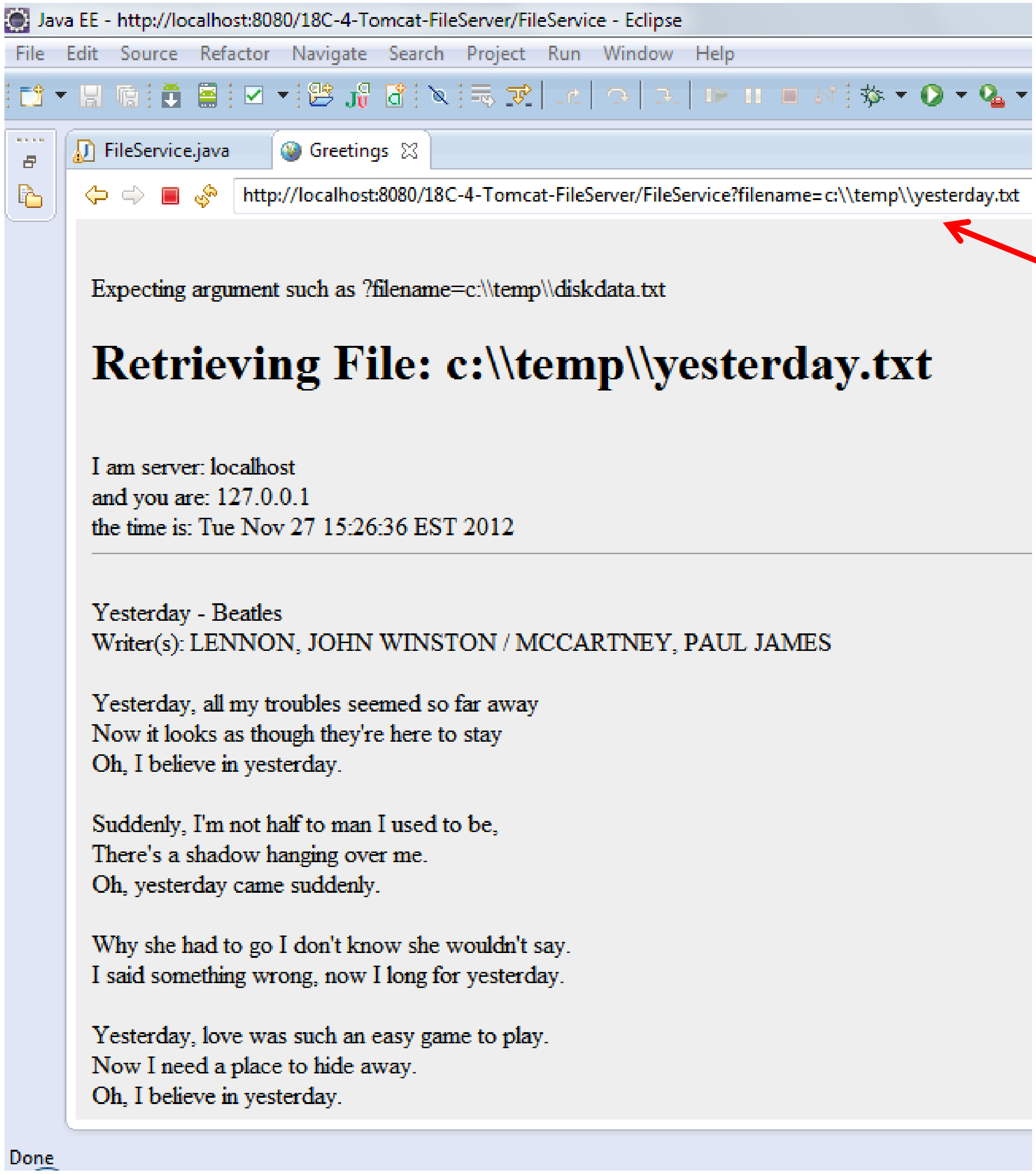
```xml
<?xml version="1.0" encoding="UTF-8"?>
<context>
            <Resource name="jdbc/myoracle" auth="Container"
               type="javax.sql.DataSource" driverClassName="oracle.jdbc.OracleDriver"
               url="jdbc:oracle:thin:@sancho.csuohio.edu:1521:ORCL"
               username="CSUPERSON" password="EUCLID" maxActive="20" maxIdle="10"
               maxWait="-1"/>

</context>
```

The above *DataSource* helps the JDBC connection identify the involved server and user. Other users could reuse the data-source and provide individual credentials.

Reference:
http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html
http://tomcat.apache.org/tomcat-5.5-doc/jndi-resources-howto.html


```
WebDbServices
  JAX-WS Web Services
  Deployment Descriptor: WebDbServices
  Java Resources
    src
      csu.matos
        GetEmployee.java
        OracleDb.java
    Libraries
  JavaScript Resources
  build
  WebContent
    META-INF
      context.xml
      MANIFEST.MF
    WEB-INF
```

Appendix B. Connecting to Server’s File System


Java EE - http://localhost:8080/18C-4-Tomcat-FileServer/FileService - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
FileService.java
Greetings
Done

http://localhost:8080/18C-4-Tomcat-FileServer/FileService?filename=c:\\temp\\yesterday.txt

```
Expecting argument such as ?filename=c:\\temp\\diskdata.txt

Retrieving File: c:\\temp\\yesterday.txt

I am server: localhost
and you are: 127.0.0.1
the time is: Tue Nov 27 15:26:36 EST 2012

Yesterday - Beatles
Writer(s): LENNON, JOHN WINSTON / MCCARTNEY, PAUL JAMES

Yesterday, all my troubles seemed so far away
Now it looks as though they're here to stay
Oh, I believe in yesterday.

Suddenly, I'm not half to man I used to be,
There's a shadow hanging over me.
Oh, yesterday came suddenly.

Why she had to go I don't know she wouldn't say.
I said something wrong, now I long for yesterday.

Yesterday, love was such an easy game to play.
Now I need a place to hide away.
Oh, I believe in yesterday.
```

In this example the user supplies a file name. The servlet retrieves from the server’s file system the requested file and returns its contents to the user

Appendix B. Connecting to Server's File System

```
@WebServlet("/FileService")
public class FileService extends HttpServlet {
private static final long serialVersionUID = 1L;

public FileService() {
}

protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {

System.out.println("DEBUGGING " + request.getRequestURL());
// pair <filename,value> should be arriving in the REQUEST object
String[] values = request.getParameterValues("filename");

String filename = " [no '?filename=xxxx' supplied in the call]";
if (values != null)
filename = values[0];

// prepare a RESPONSE page to show in the client
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head>");
out.println("<title>Greetings</title>");
out.println("</head>");
out.println("<body bgcolor=\"#7777ff\">");
out.println("<br> Expecting argument such as ?filename=c:\\\\temp\\\\diskdata.txt");
out.println("<p> ");
```

Appendix B. Connecting to Server’s File System

```
out.println("<h1>Retrieving File: " + filename + " </h1>");
out.println("<br> I am server: " + request.getServerName());
out.println("<br> and you are: " + request.getRemoteHost());
out.println("<br> the time is: " + new Date().toString());

readLocalDisk(response, values);

out.println("</body>");
out.println("</html>");

}
// ///////////////////////////////////////////////////////////////////////////////
private void readLocalDisk(HttpServletResponse response, String[] values) {
try {
File filename = new File("c:\\temp\\diskdata.txt");
if ( values != null) {
filename = new File(values[0]);
}

InputStream is = new FileInputStream(filename);

if (is != null) {
    InputStreamReader isr = new InputStreamReader(is);
    BufferedReader bufreader = new BufferedReader(isr);
    PrintWriter respwriter = response.getWriter();
    respwriter.println("<hr>");
    String text = "";
    while ((text = bufreader.readLine()) != null) {
        respwriter.println("<br>" + text);
    }
}
```

Appendix B. Connecting to Server’s File System

```
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

}//readLocalDisk


protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
// TODO Auto-generated method stub
}

}
```

Appendix C. JDBC Architecture

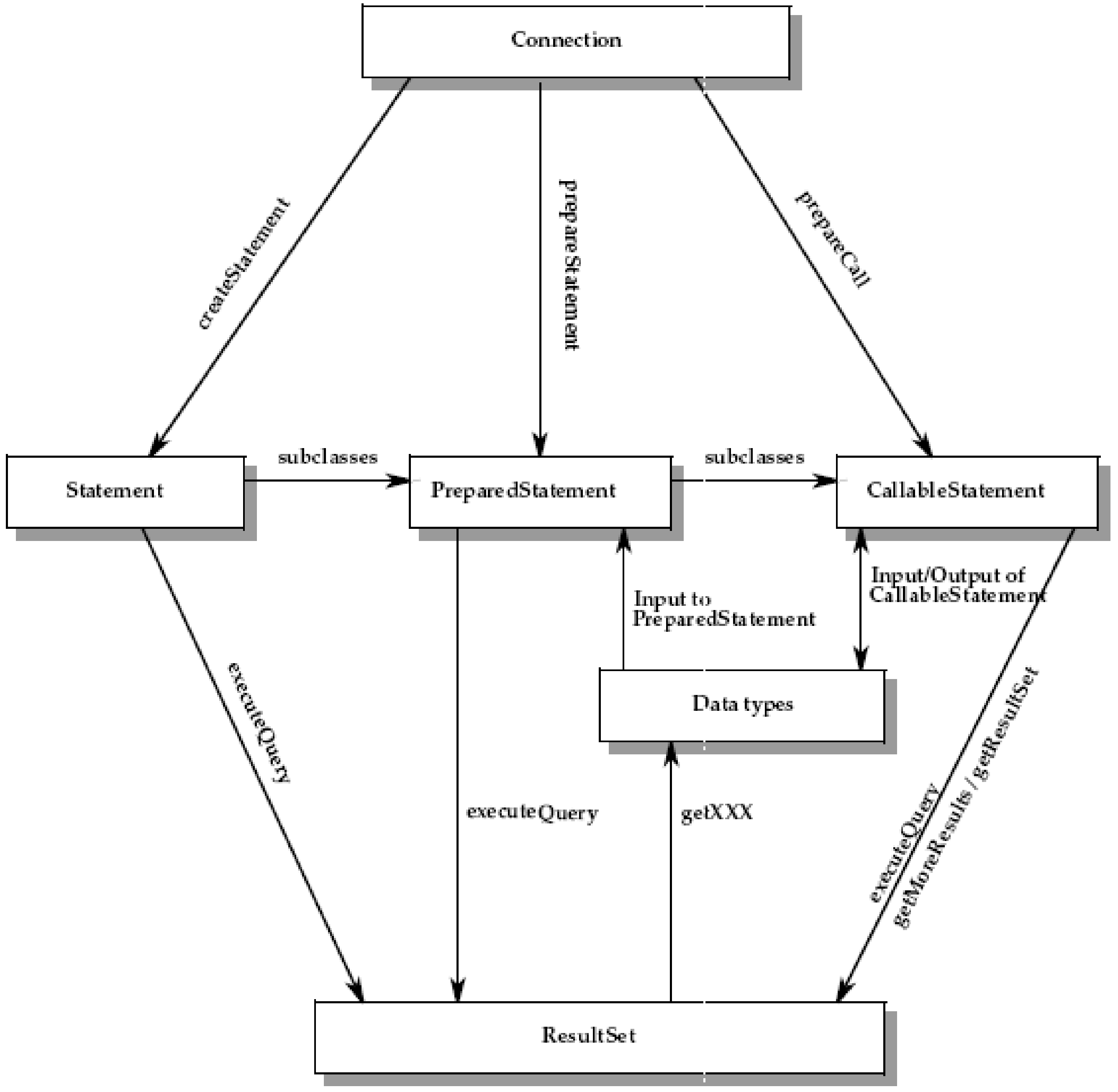


FIGURE 5-1 Relationships between major classes and interface in the `java.sql` package

JDBC™ 4.0 Specification. JSR 221
Sun Microsystems
Lance Andersen, Specification Lead
November 7, 2006