Web Service Example

The first web service in Java, can be compiled and deployed using core Java SE 6 (Java Standard Edition 6) or greater without any additional software. All of the libraries required to compile, execute, and consume web services are available in core Java 6, which supports *JAX-WS* (Java API for XML-Web Services). JAX-WS supports SOAP-based and REST-style services. JAX-WS is commonly shortened to *JWS* for Java Web Services. The current version of JAX-WS is 2.x.

A SOAP-based web service could be implemented as a single Java class but, following best practices, there should be an interface that declares the methods, which are the web service operations, and an implementation, which defines the methods declared in the interface. The interface is called the SEI: Service Endpoint Interface. The implementation is called the SIB: Service Implementation Bean.

The Interface

```java
package com.naveen;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

/**
 *  The annotation @WebService signals that this is the
 *  SEI (Service Endpoint Interface). @WebMethod signals
 *  that each method is a service operation.
 *
 *  The @SOAPBinding annotation impacts the under-the-hood
 *  construction of the service contract, the WSDL
 *  (Web Services Definition Language) document. Style.RPC
 *  simplifies the contract and makes deployment easier.
*/
@WebService
@SOAPBinding(style = Style.RPC)
public interface TimeServer {
```

Web Service Example

```
        public @WebMethod String getTimeAsString();
        public @WebMethod String getServerName(String name);
        public @WebMethod String getCompanyName();

}
```

Create a class which implements the interface

```
package com.naveen;

import java.util.Date;

import javax.jws.WebService;

/**
 *  The @WebService property endpointInterface links the
 *  SIB (this class) to the SEI (com.naveen.TimeServer).
 *  Note that the method implementations are not annotated
 *  as @WebMethods.
*/
@WebService(endpointInterface = "com.naveen.TimeServer")
public class TimeServerImpl implements TimeServer {
    public String getTimeAsString() { return new Date().toString(); }

    public String getServerName(String name){
        return "The Server Name is " + name;
    }

        @Override
        public String getCompanyName() {
                return "Naveen Air Travels";
        }

}
```

Web Service Example

The above two files can be compiled and generate class to be places in com.naveen folder(if eclipse, implicitly done);

```
package com.naveen;

import javax.xml.ws.Endpoint;

/**
 * This application publishes the web service whose
 * SIB is com.naveen.TimeServerImpl. For now, the
 * service is published at network address 127.0.0.1.,
 * which is localhost, and at port number 9876, as this
 * port is likely available on any desktop machine. The
 * publication path is /nav, an arbitrary name.
 *
 * Note: can be any number, not from the reserved(80,23,21,
3306,1521 etc)
 *
 * The Endpoint class has an overloaded publish method.
 * In this two-argument version, the first argument is the
 * publication URL as a string and the second argument is
 * an instance of the service SIB, in this case
 * com.naveen.TimeServerImpl.
 *
 * The application runs indefinitely, awaiting service requests.
 * It needs to be terminated at the command prompt with control-C
 * or the equivalent.
 *
 * Once the applicatation is started, open a browser to the URL
 *
 *     http://127.0.0.1:9876/nav?wsdl
 *
 * to view the service contract, the WSDL document. This is an
 * easy test to determine whether the service has deployed
```

Web Service Example

```
 * successfully. If the test succeeds, a client then can be
 * executed against the service.
*/
public class TimeServerPublisher {
   public static void main(String[ ] args) {
     // 1st argument is the publication URL
     // 2nd argument is an SIB instance
     Endpoint.publish("http://127.0.0.1:9876/nav", new
TimeServerImpl());
   }
}
```

Testing the Web Service with a Browser

We can test the deployed service by opening a browser and viewing the WSDL (Web Service Definition Language) document, which is an automatically generated service contract. (WSDL is pronounced "whiz dull.") The browser is opened to a URL that has two parts. The first part is the URL published in the Java TimeServerPublisher application: http://127.0.0.1:9876/ts. Appended to this URL is the query string ?wsdl

The result is shown below

```
<!--
 Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
JAX-WS RI 2.1.6 in JDK 6.
-->
–
```

Web Service Example

```
<!--
 Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is
JAX-WS RI 2.1.6 in JDK 6.
-->
–
<definitions targetNamespace="http://naveen.com/"
name="TimeServerImplService">
<types/>
<message name="getTimeAsString"/>
–
<message name="getTimeAsStringResponse">
<part name="return" type="xsd:string"/>
</message>
–
<message name="getServerName">
<part name="arg0" type="xsd:string"/>
</message>
–
<message name="getServerNameResponse">
<part name="return" type="xsd:string"/>
</message>
<message name="getCompanyName"/>
–
<message name="getCompanyNameResponse">
<part name="return" type="xsd:string"/>
</message>
–
<portType name="TimeServer">
–
      <operation name="getTimeAsString">
            <input message="tns:getTimeAsString"/>
            <output message="tns:getTimeAsStringResponse"/>
      </operation>
      –
      <operation name="getServerName">
            <input message="tns:getServerName"/>
            <output message="tns:getServerNameResponse"/>
      </operation>
      –
      <operation name="getCompanyName">
            <input message="tns:getCompanyName"/>
            <output message="tns:getCompanyNameResponse"/>
      </operation>
</portType>
–
<binding name="TimeServerImplPortBinding" type="tns:TimeServer">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="rpc"/>
–
<operation name="getTimeAsString">
<soap:operation soapAction=""/>
–
```

Web Service Example

```
<input>
<soap:body use="literal" namespace="http://naveen.com/"/>
</input>
–
<output>
<soap:body use="literal" namespace="http://naveen.com/"/>
</output>
</operation>
–
<operation name="getServerName">
<soap:operation soapAction=""/>
–
<input>
<soap:body use="literal" namespace="http://naveen.com/"/>
</input>
–
<output>
<soap:body use="literal" namespace="http://naveen.com/"/>
</output>
</operation>
–
<operation name="getCompanyName">
<soap:operation soapAction=""/>
–
<input>
<soap:body use="literal" namespace="http://naveen.com/"/>
</input>
–
<output>
<soap:body use="literal" namespace="http://naveen.com/"/>
</output>
</operation>
</binding>
–
<service name="TimeServerImplService">
–
<port name="TimeServerImplPort" binding="tns:TimeServerImplPortBinding">
      <soap:address location="http://127.0.0.1:9876/nav"/>
</port>
</service>
</definitions>
```

Then I've the java client which call the implement class(the client program can be in any platform and language like .Net, C, C++, PHP, etc).

Below is the sample Java program

package com.naveen;

Web Service Example

```
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import java.net.URL;
class TimeClient {
   public static void main(String args[ ]) throws Exception {
      URL url = new URL("http://localhost:9876/nav?wsdl");

      // Qualified name of the service:
      //   1st arg is the service URI
      //   2nd is the service name published in the WSDL
      QName qname = new QName("http://naveen.com/",
"TimeServerImplService");

      // Create, in effect, a factory for the service.
      Service service = Service.create(url, qname);

      // Extract the endpoint interface, the service "port".
      TimeServer eif = service.getPort(TimeServer.class);

      System.out.println(eif.getTimeAsString());
      System.out.println(eif.getServerName("Naveen"));
      System.out.println(eif.getCompanyName());

   }
}
```