Spring Web Service, Spring JMS, Eclipse & Maven tutorials

Organizations are increasingly communicating between disparate software components in a loosely coupled and often asynchronous manner. These tutorials will help you understand two of the popular integration technologies **Web Services** & messaging (i.e. **JMS** – Java Messaging Service).

By

K. Arulkumaran

&

A. Sivayini

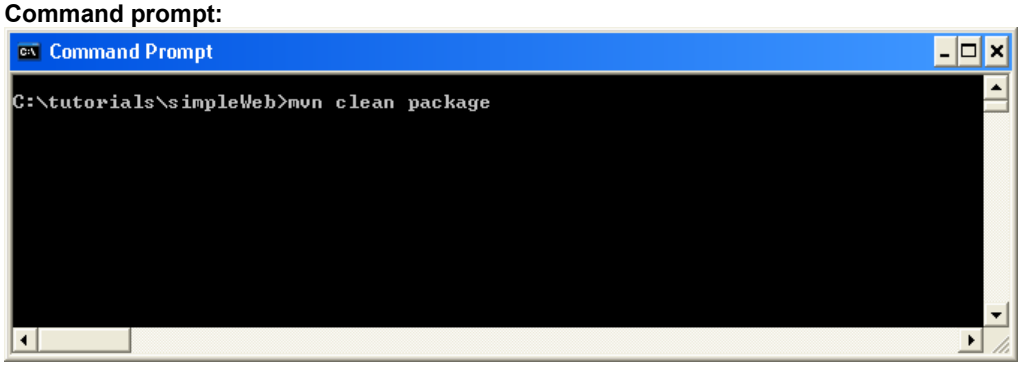
Website: http://www.lulu.com/java-success

Feedback email: java-interview@hotmail.com

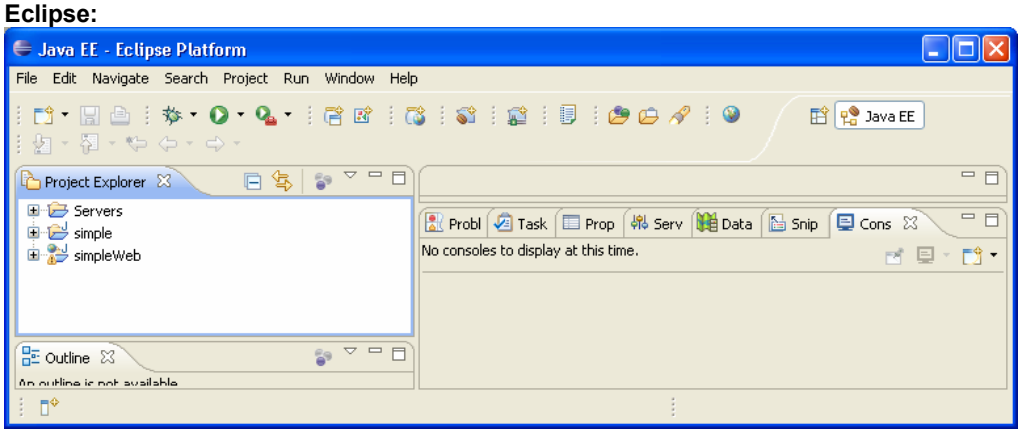
Table Of Contents

Notations

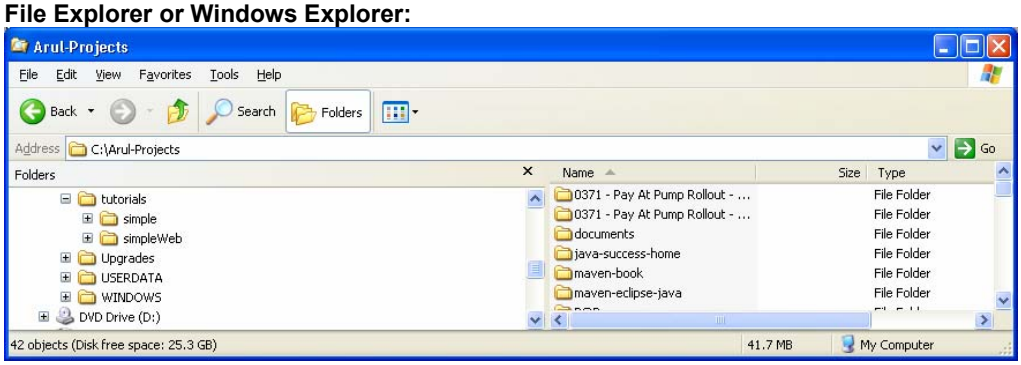
Command prompt:



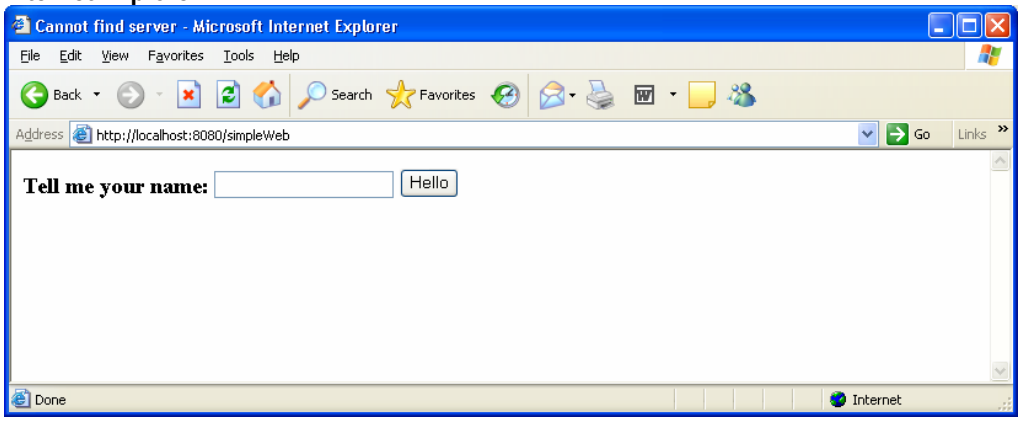
Eclipse:



File Explorer or Windows Explorer:



Internet Explorer:

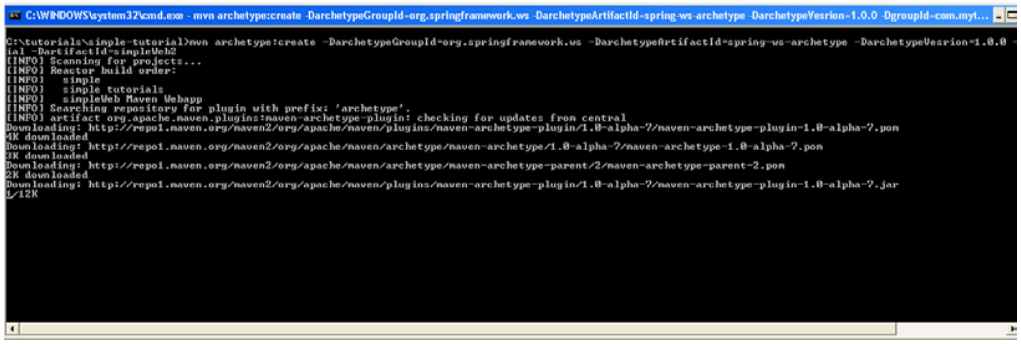


Tutorial 11 – Spring Web Service

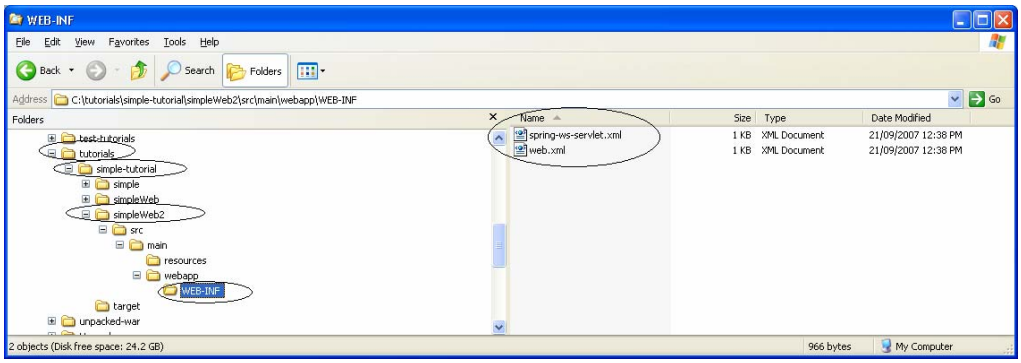
This tutorial will guide you through building a simple web service using spring framework. This tutorial assumes that you have gone through **Tutorials 1-10** & the **source code** for tutorials 1-10. Also refer to **spring-ws-reference.pdf** (http://static.springframework.org/spring-ws/site/reference/pdf/spring-ws-reference.pdf) for further information.

Step 1: Create a project named **simpleWeb2.** Run the following command in a command window:

```
C:\tutorials\simple-tutorial>mvn archetype:create   \
-DarchetypeGroupId=org.springframework.ws -DarchetypeArtifactId=spring-ws-archetype \
-DarchetypeVesrion=1.0.0 -DgroupId=com.mytutorial -DartifactId=simpleWeb2
```



Step 2: Now you should have the **simpleWeb2** project with some basic files like **pom.xml**, **web.xml**, **spring-ws-servlet.xml** etc.



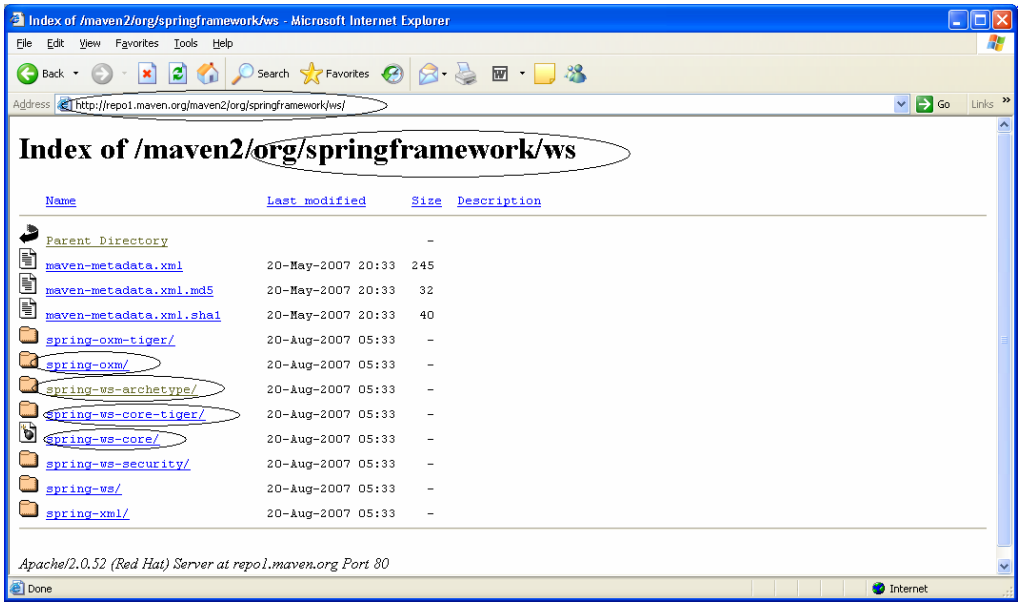
Step 3: Open the **pom.xml** file under "**C:\tutorials\simple-tutorial\simpleWeb2**" in a text pad and modify it to look as shown below.

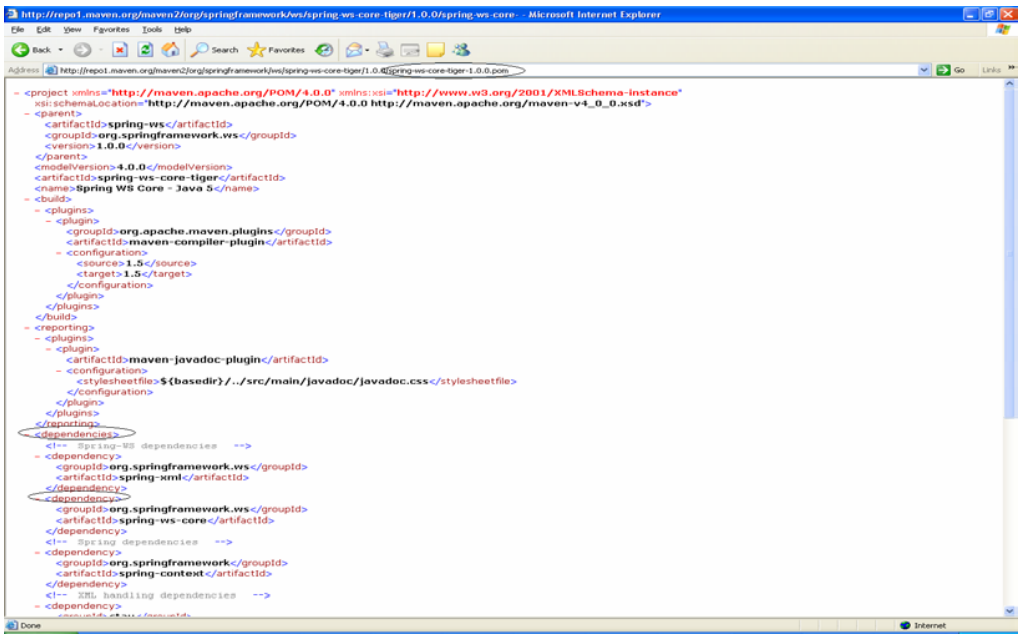
```
<?xml version="1.0" encoding="UTF-8"?>
<project>
        <parent>
                <artifactId>simple-tutorial</artifactId>
                <groupId>com.mytutorial</groupId>
                <version>1.0</version>
        </parent>
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.mytutorial</groupId>
```

```
<artifactId>simpleWeb2</artifactId>
<packaging>war</packaging>
<name>simpleWeb2 Spring-WS Application</name>
<version>1.0-SNAPSHOT</version>
<url>http://www.springframework.org/spring-ws</url>
<build>
	<finalName>simpleWeb2</finalName>
</build>
<dependencies>
	<!--  Spring web services -->
	<dependency>
		<artifactId>spring-xml</artifactId>
		<groupId>org.springframework.ws</groupId>
		<version>1.0.0</version>
		<exclusions>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-core</artifactId>
			</exclusion>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-beans</artifactId>
			</exclusion>
			<exclusion>
				<groupId>commons-logging</groupId>
				<artifactId>commons-logging</artifactId>
			</exclusion>
			<exclusion>
				<groupId>jdom</groupId>
				<artifactId>jdom</artifactId>
			</exclusion>
			<exclusion>
				<groupId>dom4j</groupId>
				<artifactId>dom4j</artifactId>
			</exclusion>
		</exclusions>
	</dependency>
	<dependency>
		<artifactId>spring-oxm-tiger</artifactId>
		<groupId>org.springframework.ws</groupId>
		<version>1.0.0</version>
		<exclusions>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-core</artifactId>
			</exclusion>
			<exclusion>
				<groupId>org.springframework</groupId>
				<artifactId>spring-beans</artifactId>
			</exclusion>
			<exclusion>
				<groupId>commons-logging</groupId>
				<artifactId>commons-logging</artifactId>
			</exclusion>
			<exclusion>
				<groupId>com.thoughtworks.xstream</groupId>
				<artifactId>xstream</artifactId>
			</exclusion>
		</exclusions>
	</dependency>
	<dependency>
		<artifactId>spring-ws-core-tiger</artifactId>
		<groupId>org.springframework.ws</groupId>
```

```
            <version>1.0.0</version>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-core</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-beans</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-context</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-web</artifactId>
                </exclusion>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-webmvc</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>com.sun.xml.messaging.saaj</groupId>
            <artifactId>saaj-impl</artifactId>
            <version>1.3</version>
            <scope>runtime</scope>
            <exclusions>
                <exclusion>
                    <groupId>javax.activation</groupId>
                    <artifactId>activation</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>javax.activation</groupId>
            <artifactId>activation</artifactId>
            <version>1.1</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.4</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>jdom</groupId>
            <artifactId>jdom</artifactId>
            <version>1.0</version>
        </dependency>

        <dependency>
            <groupId>jaxen</groupId>
            <artifactId>jaxen</artifactId>
            <version>1.1</version>
        </dependency>
        <dependency>
            <groupId>javax.xml.soap</groupId>
            <artifactId>saaj-api</artifactId>
            <version>1.3</version>
```

```
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>com.sun.xml.messaging.saaj</groupId>
                <artifactId>saaj-impl</artifactId>
                <version>1.3</version>
                <scope>runtime</scope>
            </dependency>

            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring</artifactId>
                <version>2.0.6</version>
            </dependency>
        </dependencies>

</project>
```

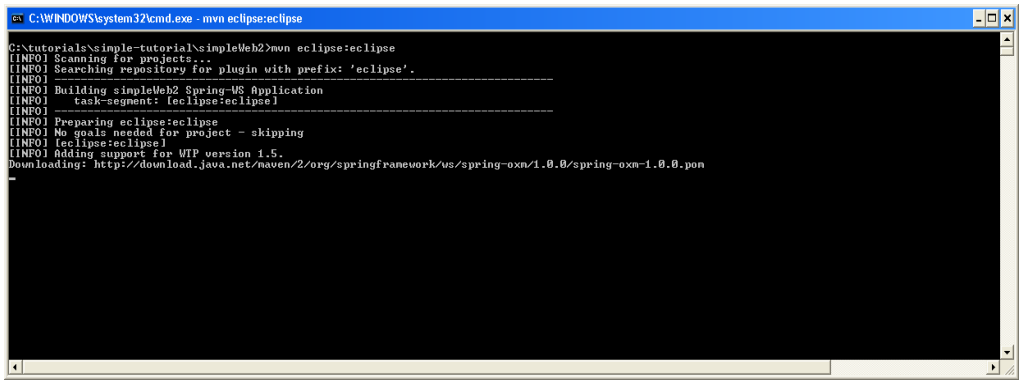




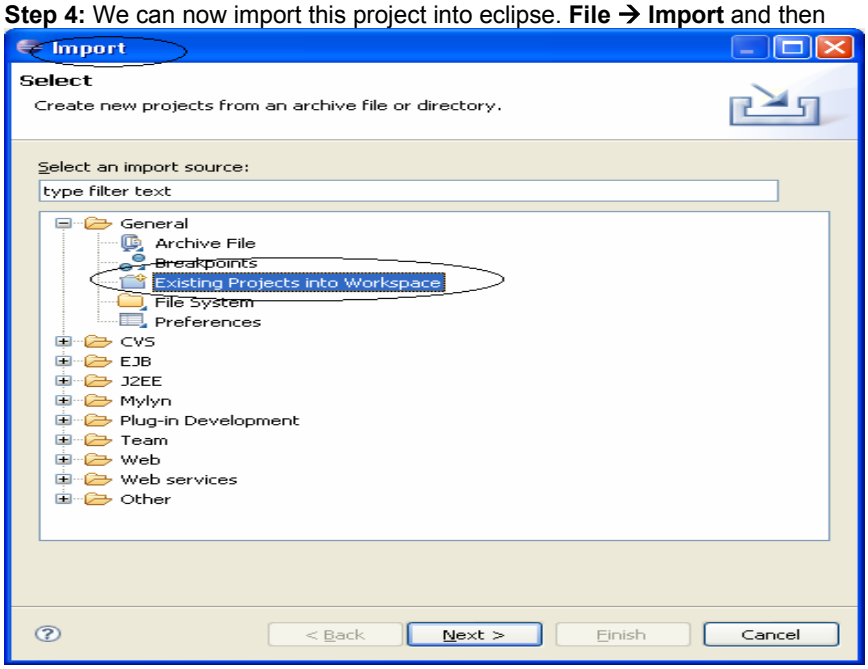
Note: The **<exclusion>** tags will remove any transitive dependencies you would like to exclude. Look at the **.pom** file as shown above for the dependencies defined and you can exclude them if you wish.

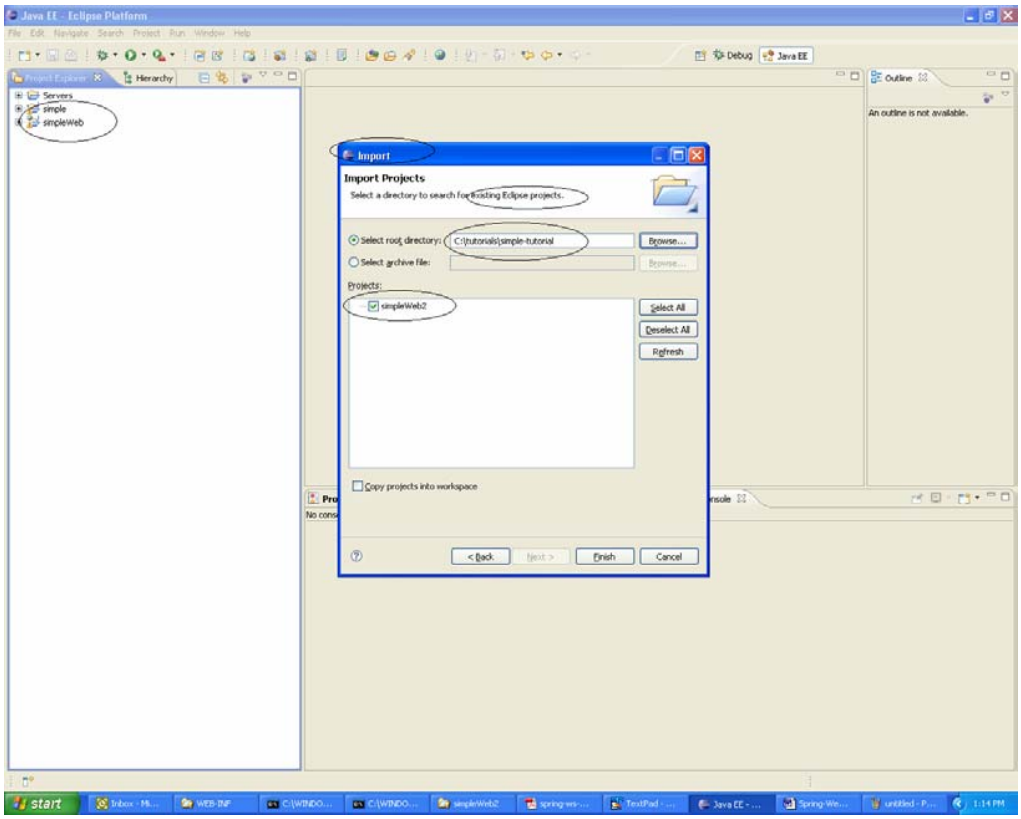
Step 3: Next step is to generate eclipse metadata using the **mvn** command as shown below:

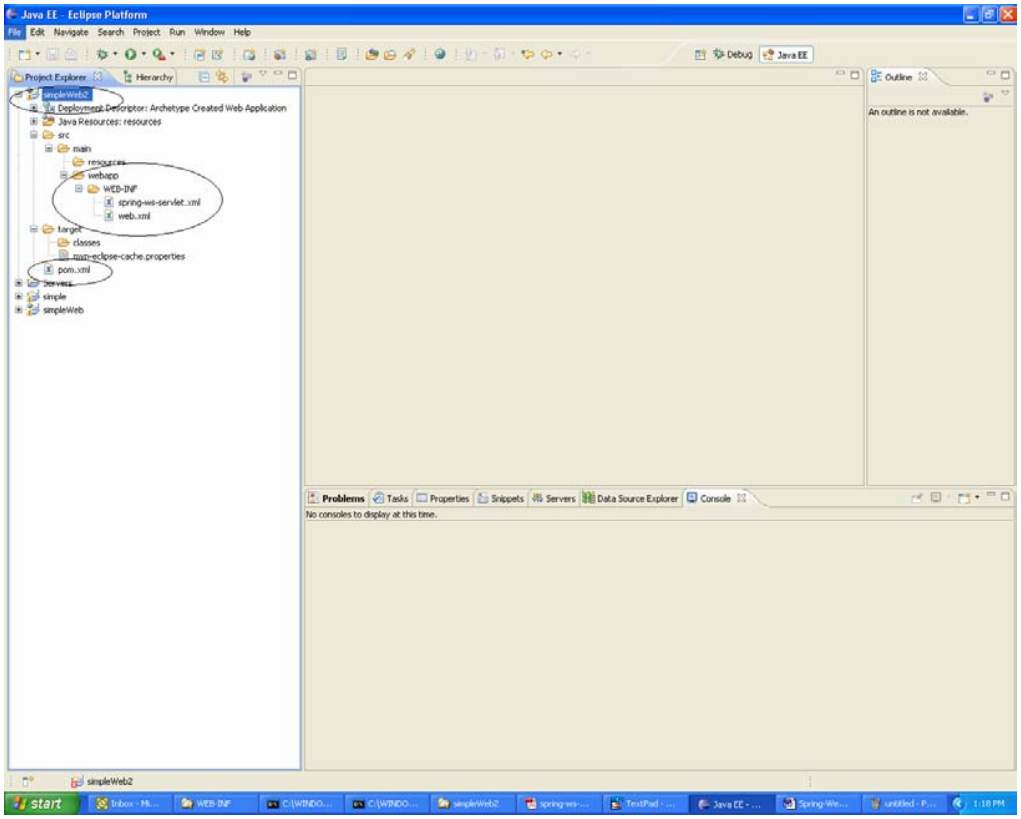
```
C:\tutorials\simple-tutorial\simpleWeb2>mvn eclipse:eclipse
```



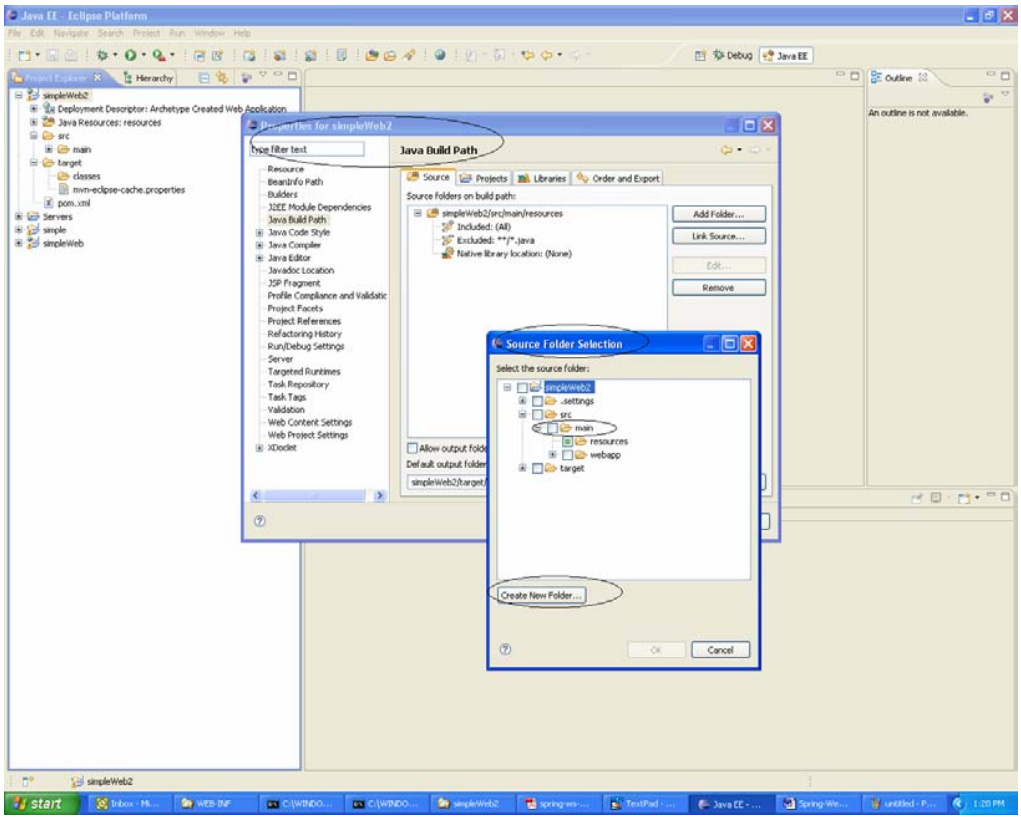
Step 4: We can now import this project into eclipse. **File → Import** and then

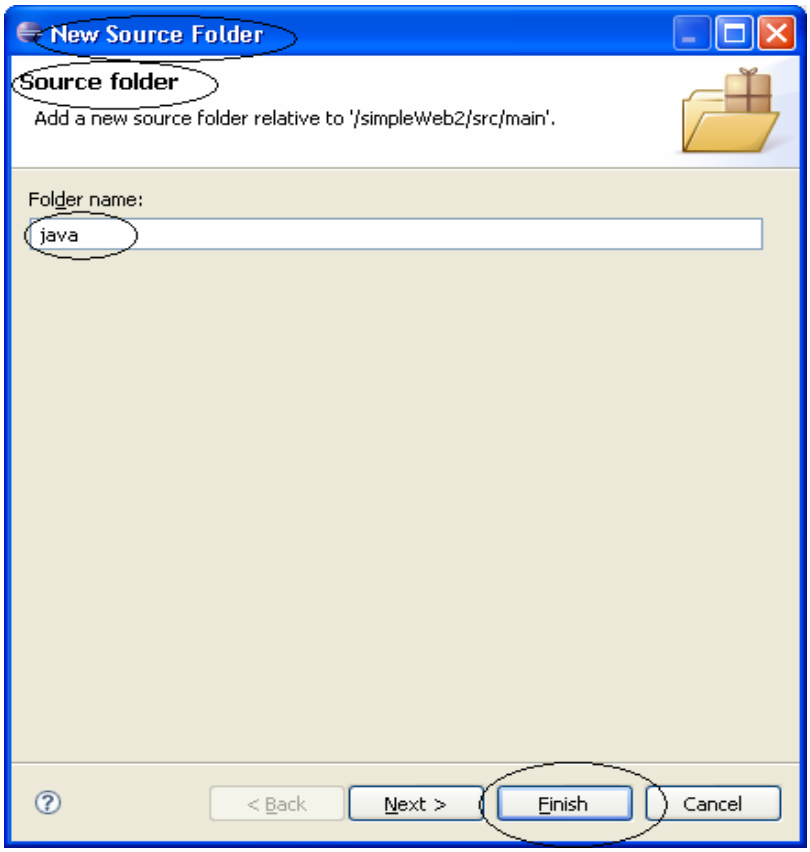






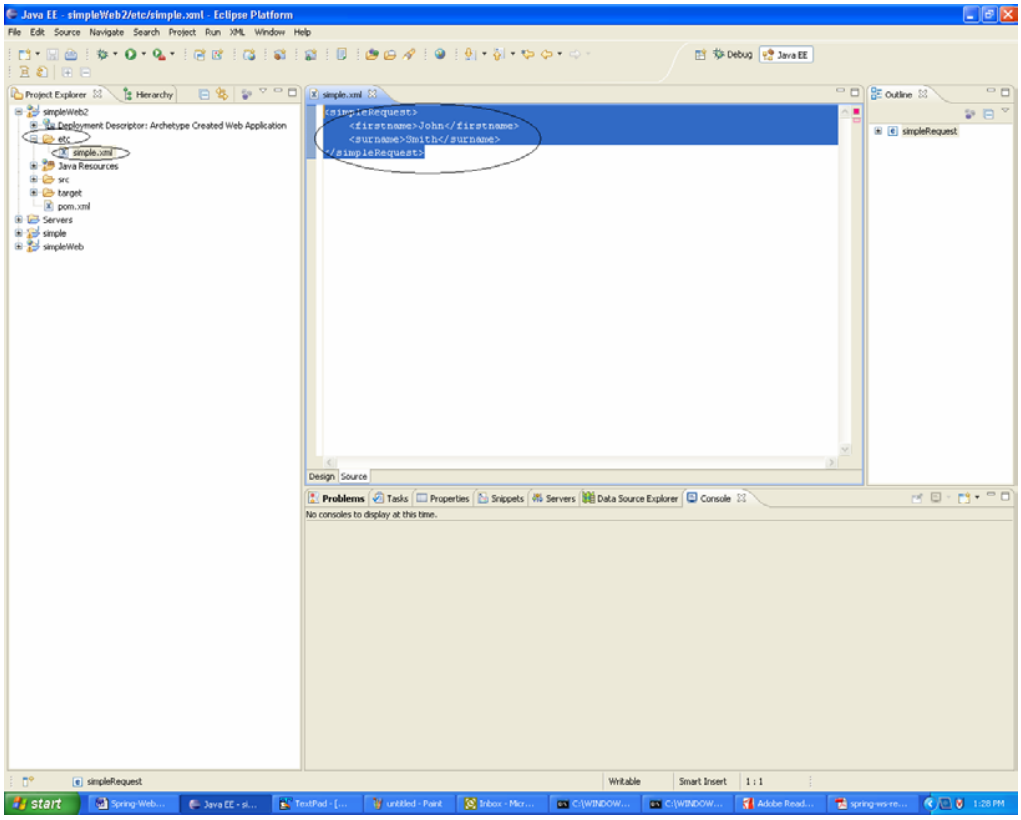
Step 5: Next step is to create a folder "**java**" for the java files under "**src/main**". Right click on **simpleWeb2** and select "**properties**"


Java EE - Eclipse Platform
Java Build Path
Source Folder Selection
Select the source folder:
Create New Folder...
OK
Cancel


New Source Folder
Source folder
Add a new source folder relative to '/simpleWeb2/src/main'.
Folder name:
java
< Back
Next >
Finish
Cancel

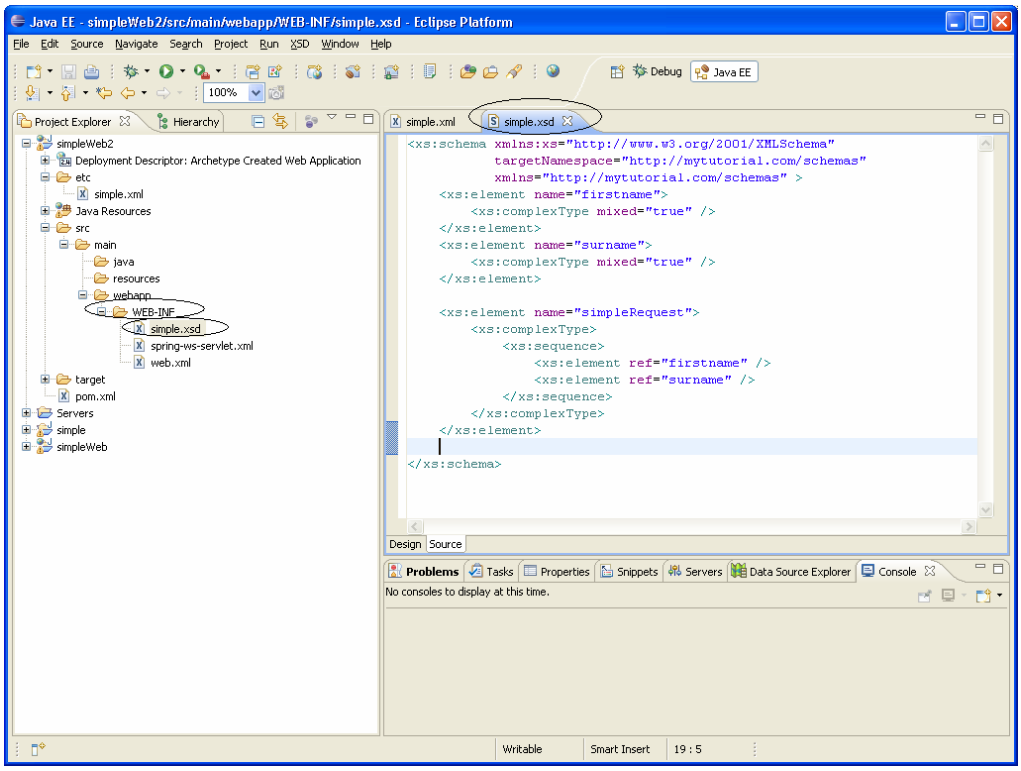
Step 6: Now define the input XML file and its contract. Create a new folder "**etc**" under "**simpleWeb2**" for any miscellaneous files like "**simple.xml**" that are not packaged.

```
<simpleRequest>
	<firstname>John</firstname>
	<surname>Smith</surname>
</simpleRequest>
```

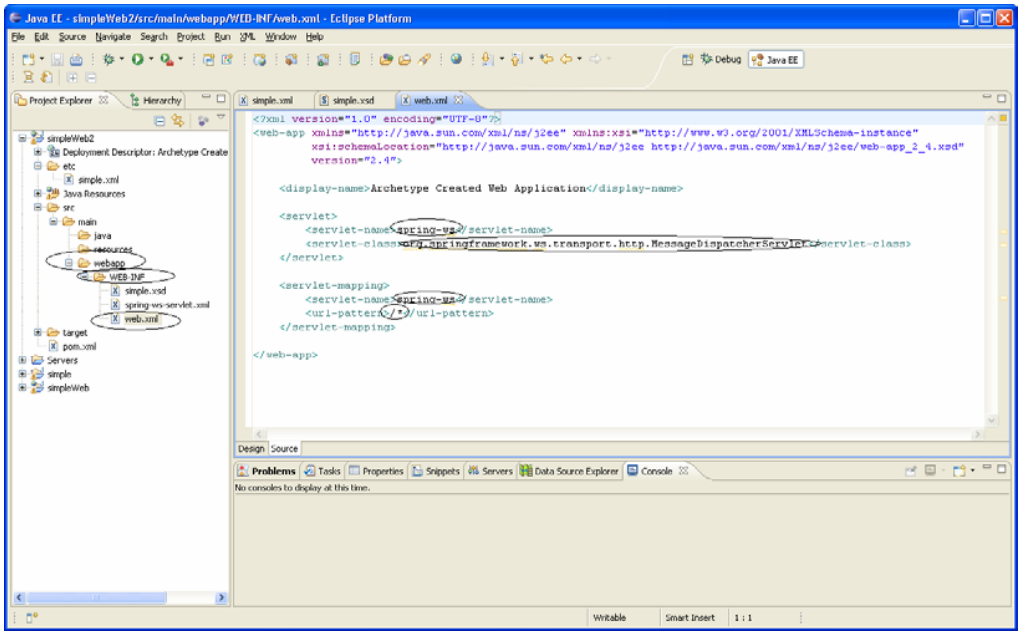


The next step is to define the schema definition for the above xml file. The "**simple.xsd"** should be created under "C:\tutorials\simple-tutorial\simpleWeb2**src\main\webapp\WEB-INF**" as shown below:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://mytutorial.com/schemas"
           xmlns="http://mytutorial.com/schemas" >
      <xs:element name="firstname">
            <xs:complexType mixed="true" />
      </xs:element>
      <xs:element name="surname">
            <xs:complexType mixed="true" />
      </xs:element>

      <xs:element name="simpleRequest">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="firstname" />
                        <xs:element ref="surname" />
                  </xs:sequence>
            </xs:complexType>
      </xs:element>

</xs:schema>
```



Step 7: The web.xml can remain as it is.

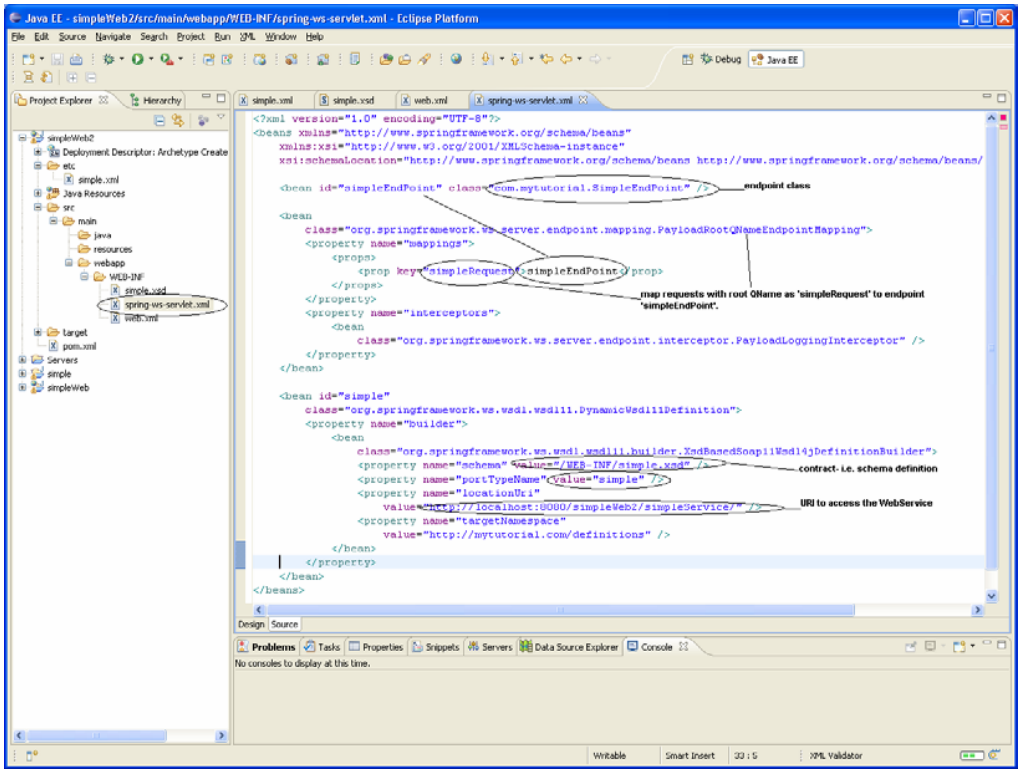


Step 8: With spring-ws, you do not have to define the **.wsdl** file. It will be automatically generated based on your **simple.xsd** file you just created and the **simple-ws-servlet.xml** (the file convention is <**servlet-name-defined-in-web.xml**>**-servlet.xml**) file you are about to define.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
```

```
        <bean id="simpleEndPoint" class="com.mytutorial.SimpleEndPoint" />

        <bean

        class="org.springframework.ws.server.endpoint.mapping.PayloadRootQNameEndpointMapping">
                <property name="mappings">
                        <props>
                                <prop key="simpleRequest">simpleEndPoint</prop>
                        </props>
                </property>
                <property name="interceptors">
                        <bean

        class="org.springframework.ws.server.endpoint.interceptor.PayloadLoggingInterceptor" />
                </property>
        </bean>

        <bean id="simple"
                class="org.springframework.ws.wsdl.wsdl11.DynamicWsdl11Definition">
                <property name="builder">
                        <bean

        class="org.springframework.ws.wsdl.wsdl11.builder.XsdBasedSoap11Wsdl4jDefinitionBuilder">
                                <property name="schema" value="/WEB-INF/simple.xsd" />
                                <property name="portTypeName" value="simple" />
                                <property name="locationUri"
                                        value="http://localhost:8080/simpleWeb2/simpleService/" />
                                <property name="targetNamespace"
                                        value="http://mytutorial.com/definitions" />
                        </bean>
                </property>
        </bean>
</beans>
```

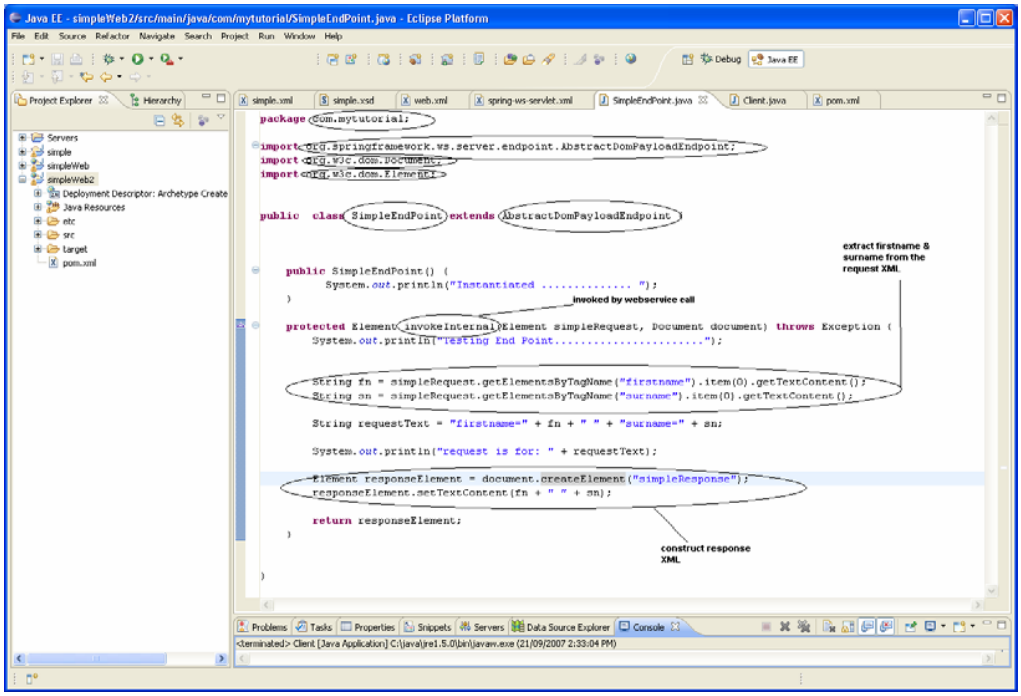
Generates **simple.wsdl**



Step 9: The next step is to define the end point class as shown below to read the incoming XML request and construct an XML response back to the caller. Create a new package **com.mytutorial** under "**java**" and then create the java file "**SimpleEndPoint.java**".

```
package com.mytutorial;
```

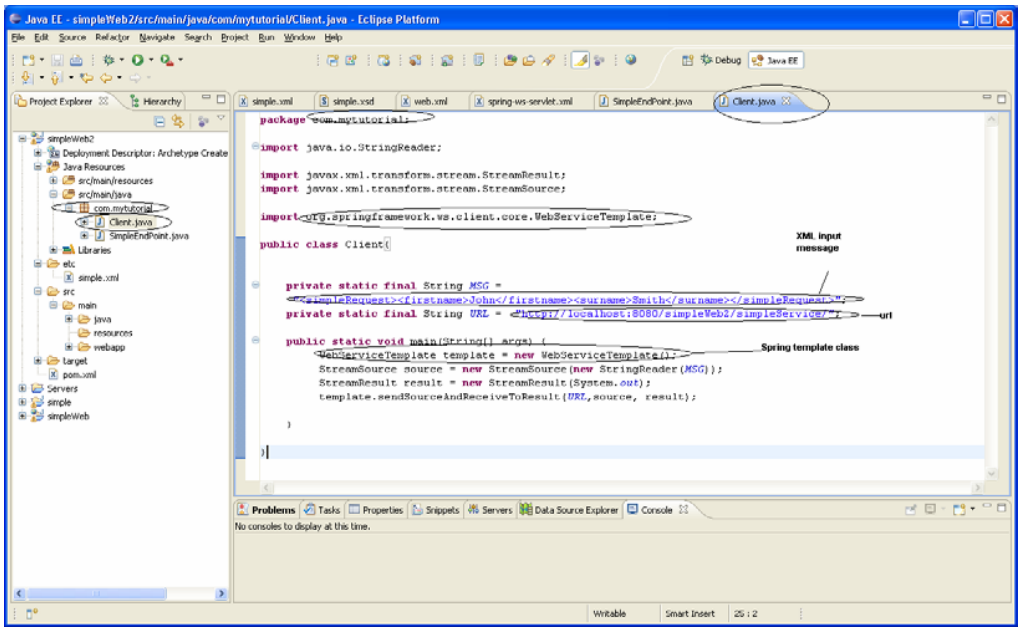
```
import org.springframework.ws.server.endpoint.AbstractDomPayloadEndpoint;
import org.w3c.dom.Document;
import org.w3c.dom.Element;


public  class SimpleEndPoint extends AbstractDomPayloadEndpoint {


        public SimpleEndPoint() {
                System.out.println("Instantiated ..... ");
        }

        protected Element invokeInternal(Element simpleRequest, Document document)
                                throws Exception {
     System.out.println("Testing End Point.....................");


     String fn =simpleRequest.getElementsByTagName("firstname")
                                .item(0).getTextContent();
     String sn =simpleRequest.getElementsByTagName("surname")
                                  .item(0).getTextContent();

     String requestText = "firstname=" + fn + " " + "surname=" + sn;

     System.out.println("request is for: " + requestText);

     Element responseElement = document.createElement
                                        ("simpleResponse");
     responseElement.setTextContent(fn + " " + sn);

     return responseElement;
   }


}
```



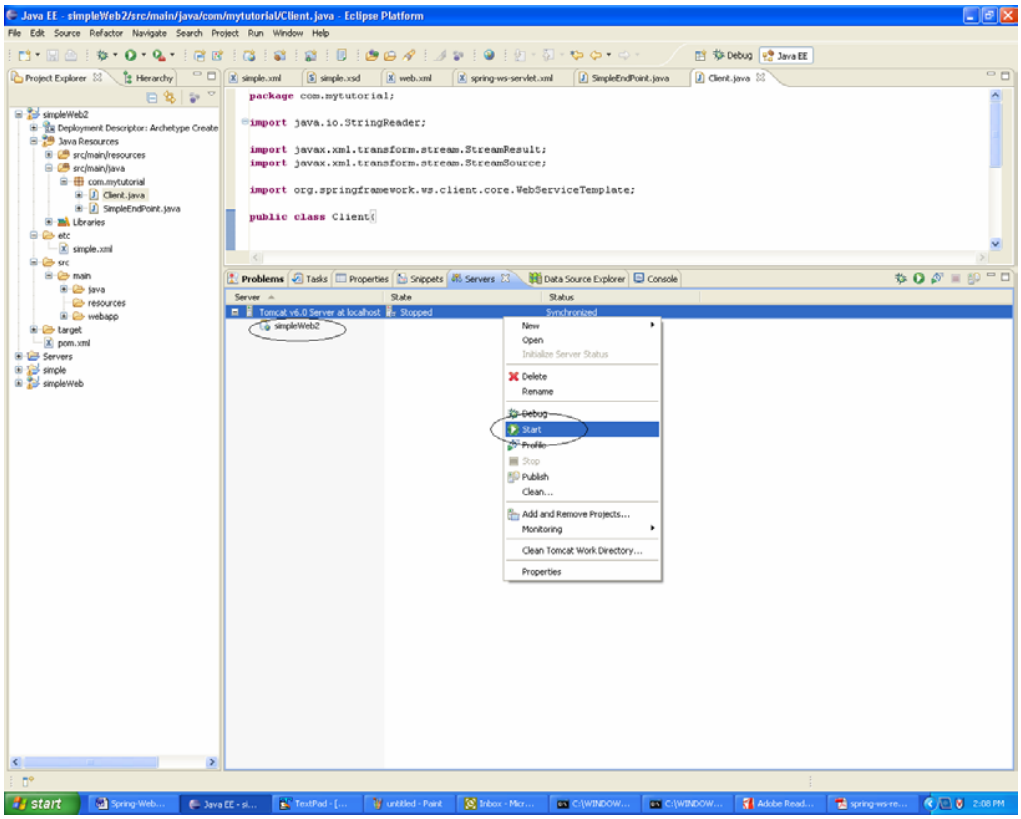
Step 10: Finally we need to create a Web Services client named "**Client.java**" to invoke our web service.

```
package com.mytutorial;

import java.io.StringReader;

import javax.xml.transform.stream.StreamResult;
```

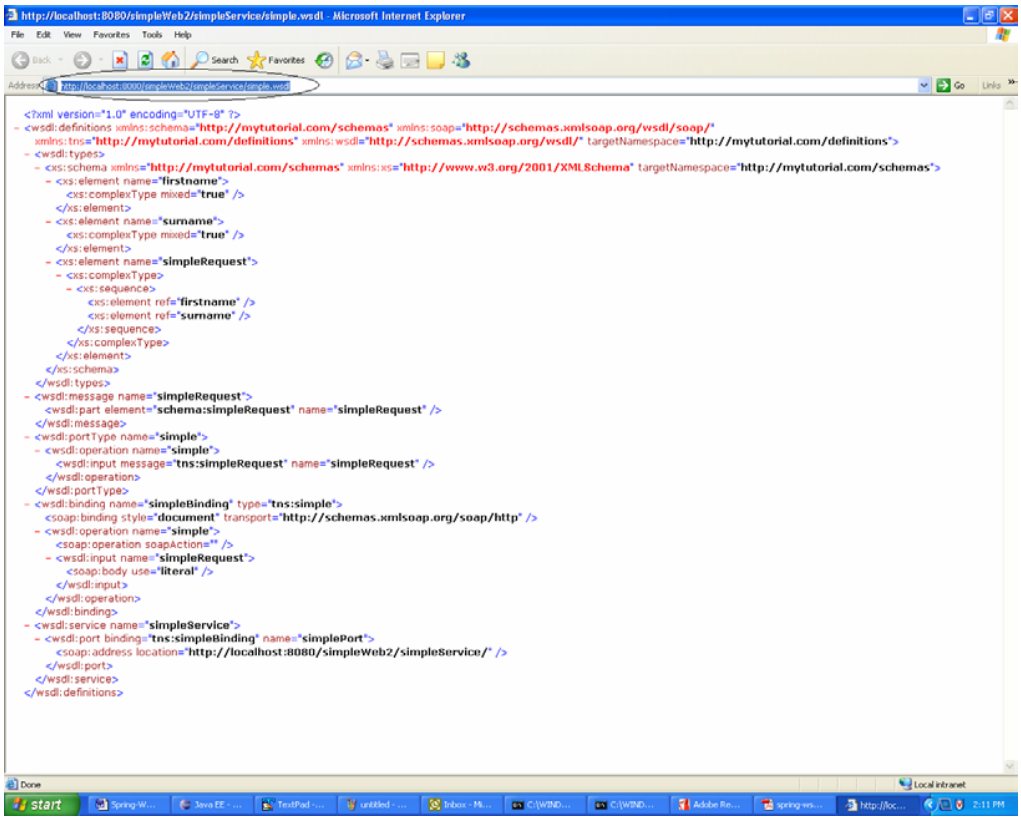
```
import javax.xml.transform.stream.StreamSource;

import org.springframework.ws.client.core.WebServiceTemplate;

public class Client{


	private static final String MSG =
	 "<simpleRequest><firstname>John</firstname><surname>Smith</surname></simpleRequest>";
	private static final String URL =  "http://localhost:8080/simpleWeb2/simpleService/";

	public static void main(String[] args) {
		WebServiceTemplate template = new WebServiceTemplate();
		StreamSource source = new StreamSource(new StringReader(MSG));
		StreamResult result = new StreamResult(System.out);
		template.sendSourceAndReceiveToResult(URL,source, result);

	}

}
```



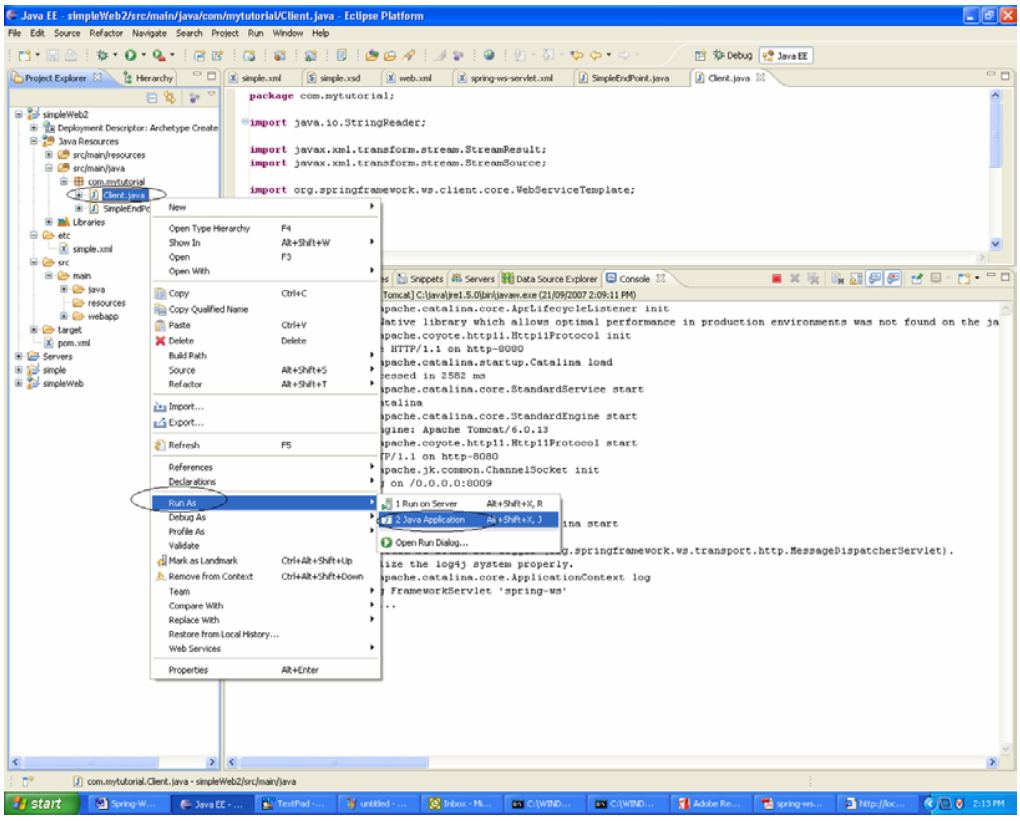
Step 11: Deploy the "**simpleWeb2**" war file into Tomcat under eclipse and start the Tomcat Server.



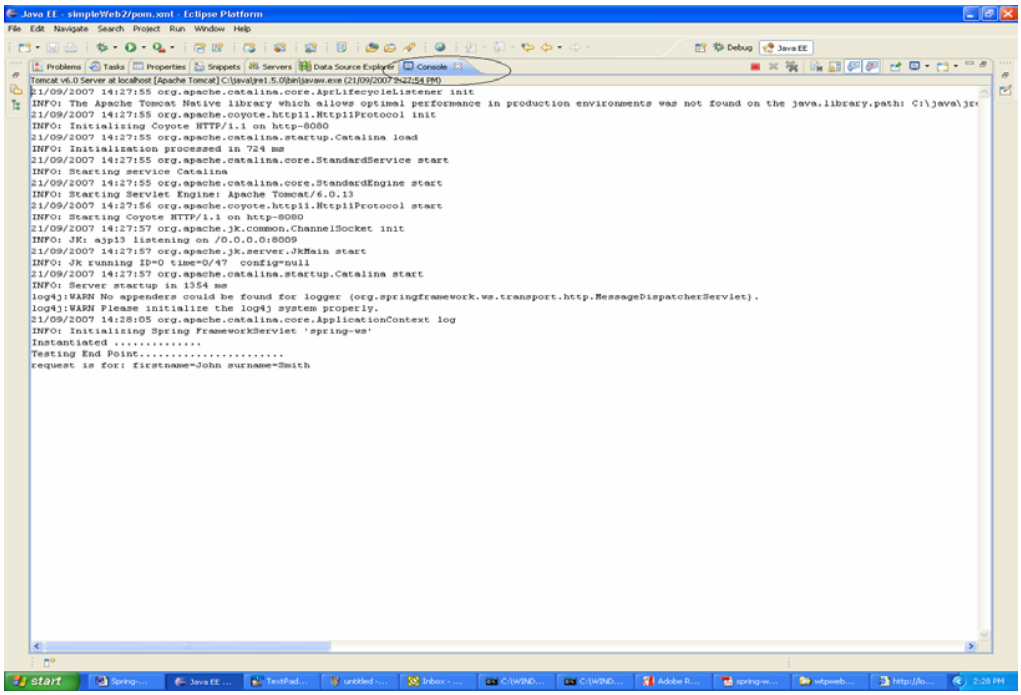
Open up a browser and type the following URL http://localhost:8080/simpleWeb2/simpleService/simple.wsdl to look at the generated **simple.wsdl** file as shown below:



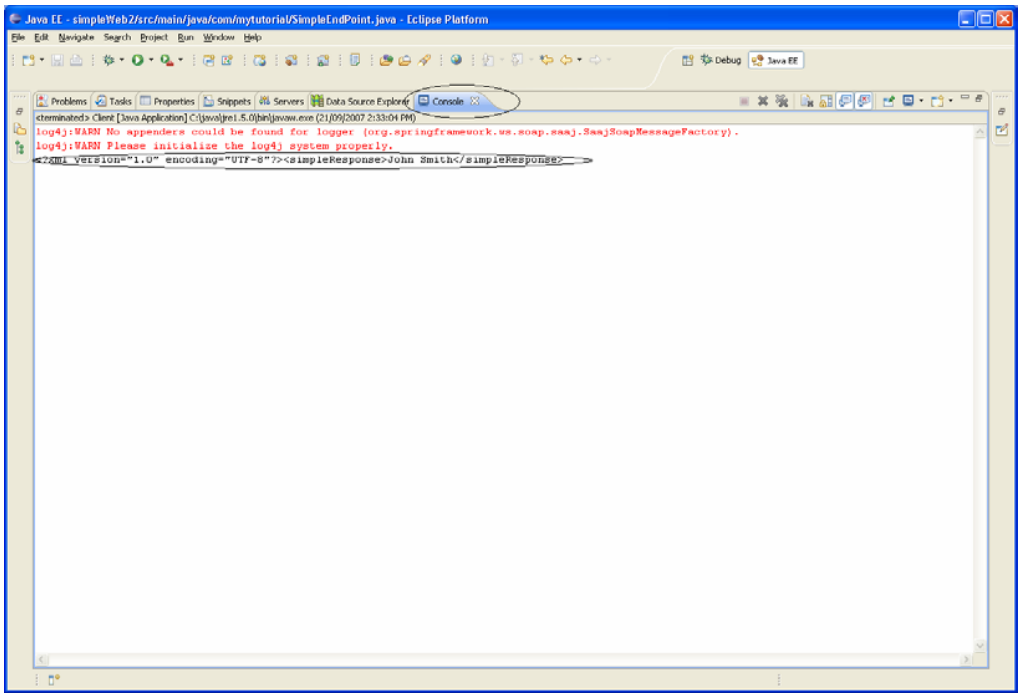
Step 11: Run the **Client.java** (Right click and **Run As → Java Application**) file and check the output printed on the console.



The server output is as follows:



The client output is as follows:



That's all to it.

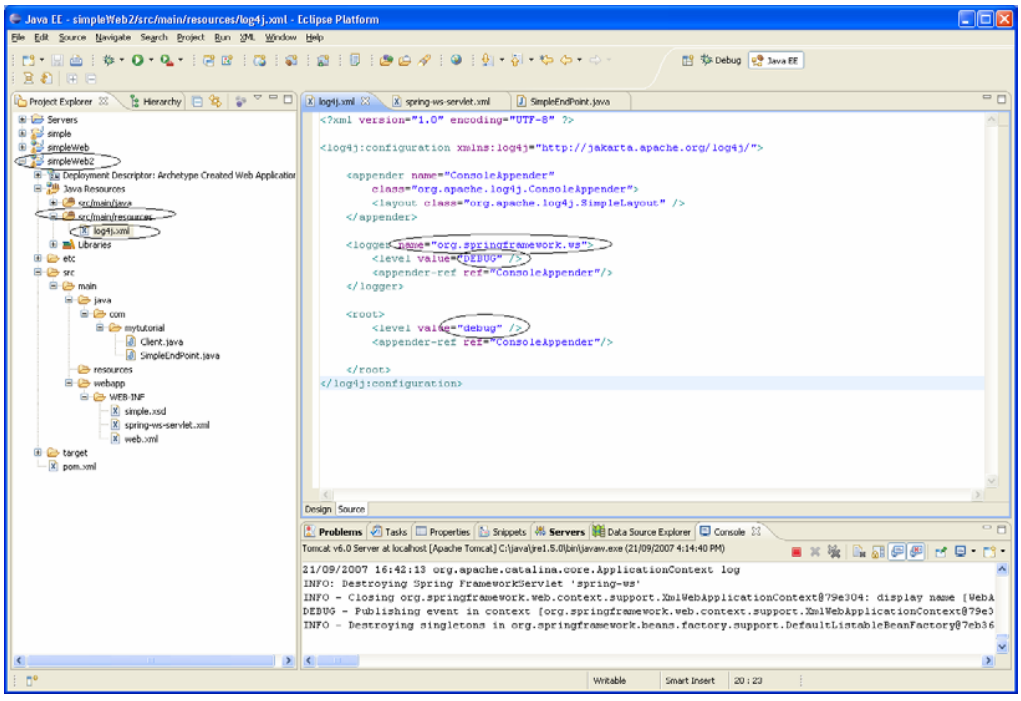
Also refer to **spring-ws-reference.pdf** (http://static.springframework.org/spring-ws/site/reference/pdf/spring-ws-reference.pdf) for further information.

Tutorial 12 – Spring Web Service with Logging

This tutorial is a continuation of **Tutorial 11**. In this tutorial I will be turning on the logging feature of spring-ws framework via **log4j** & interceptors provided by spring-ws so that soap messages can be printed.

Step 1: Add **log4j.xml** file under "**src/main/resources**"

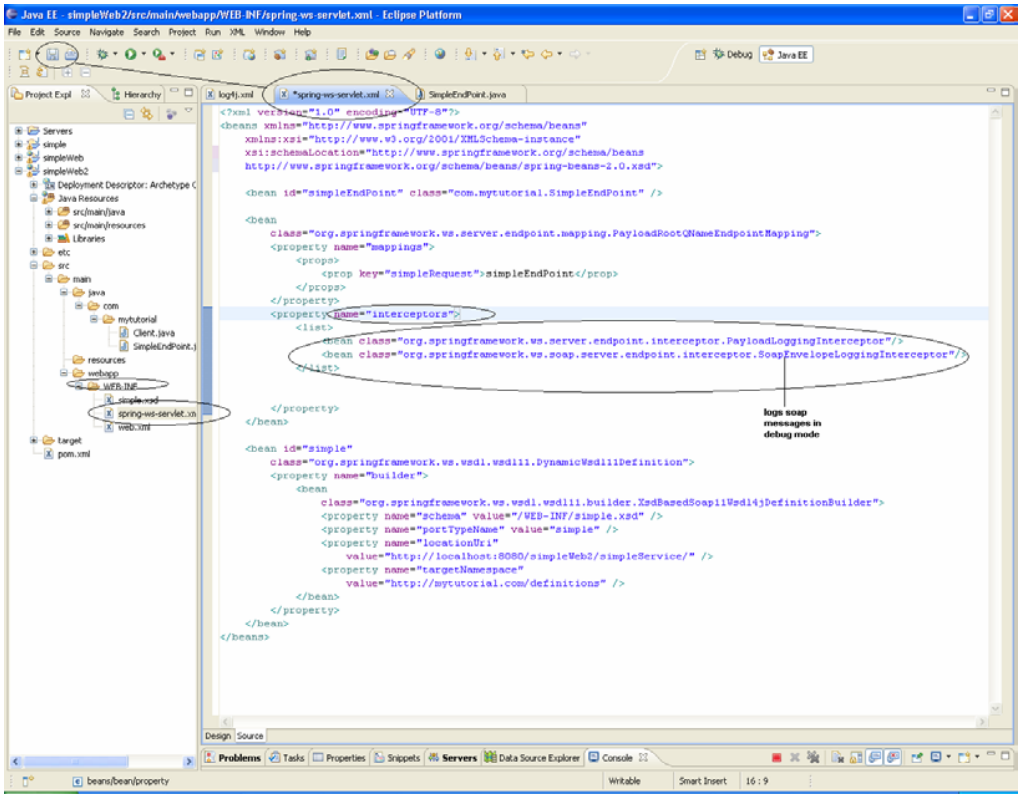
```
<?xml version="1.0" encoding="UTF-8" ?>

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

	<appender name="ConsoleAppender"
		class="org.apache.log4j.ConsoleAppender">
		<layout class="org.apache.log4j.SimpleLayout" />
	</appender>

	<logger name="org.springframework.ws">
		<level value="DEBUG" />
		<appender-ref ref="ConsoleAppender"/>
	</logger>

	<root>
		<level value="debug" />
		<appender-ref ref="ConsoleAppender"/>

	</root>
</log4j:configuration>
```



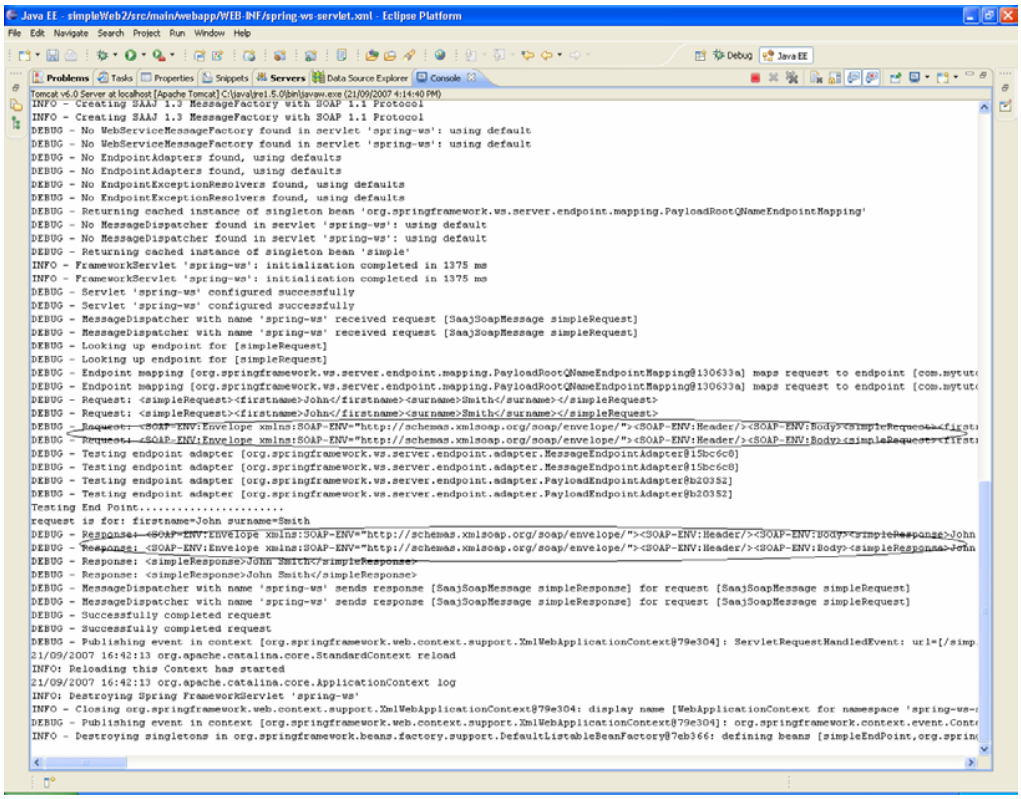
Step 2: Modify "**spring-ws-servlet.xml**" to add the soap interceptors as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

        <bean id="simpleEndPoint" class="com.mytutorial.SimpleEndPoint" />

        <bean

        class="org.springframework.ws.server.endpoint.mapping.PayloadRootQNameEndpointMapping">
                <property name="mappings">
                        <props>
                                <prop key="simpleRequest">simpleEndPoint</prop>
                        </props>
                </property>
                <property name="interceptors">          --> Interceptors
                        <list>
        <bean class="org.springframework.ws.server.endpoint.interceptor.PayloadLoggingInterceptor"/>
        <bean
class="org.springframework.ws.soap.server.endpoint.interceptor.SoapEnvelopeLoggingInterceptor"/>
       </list>


                </property>
        </bean>

        <bean id="simple"
                class="org.springframework.ws.wsdl.wsdl11.DynamicWsdl11Definition">
                <property name="builder">
                        <bean

        class="org.springframework.ws.wsdl.wsdl11.builder.XsdBasedSoap11Wsdl4jDefinitionBuilder">
                                <property name="schema" value="/WEB-INF/simple.xsd" />
                                <property name="portTypeName" value="simple" />
                                <property name="locationUri"
                                        value="http://localhost:8080/simpleWeb2/simpleService/" />
                                <property name="targetNamespace"
                                        value="http://mytutorial.com/definitions" />
                        </bean>
                </property>
        </bean>
</beans>
```



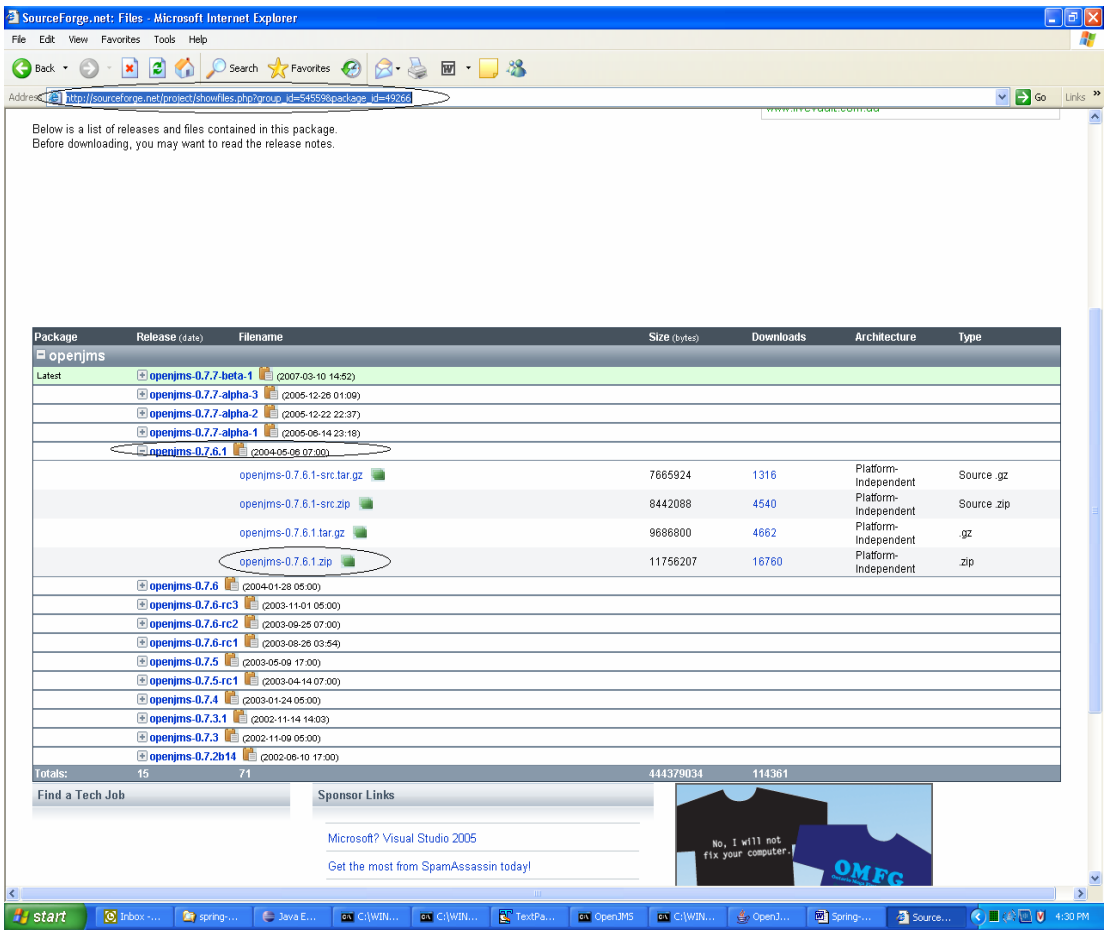
Step 3: Deploy the "**simpleWeb2**" to Tomcat and run the client again and you should now be able to see a number of debug messages. Take note of the SOAP messages.



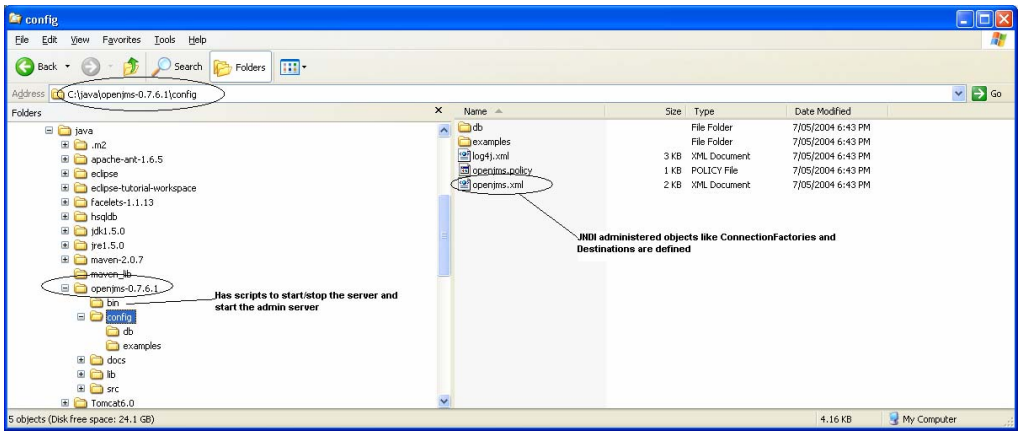
Tutorial 13 – Spring JMS

This tutorial will guide you through building a simple messaging (i.e. JMS based) application using spring framework. You need a message broker or a JMS provider for your JMS applications. There are number of brokers (aka providers) both commercial like Websphere MQ (used to be called MQSeries), Web Methods etc and open source providers like Active MQ, OpenJMS etc. To keep things simple, I will be using **OpnenJMS** (http://openjms.sourceforge.net/) in this tutorial as our JMS provider. This tutorial is based on http://www.devx.com/Java/Article/20903.

Step 1: Download and install **OpenJMS** from http://sourceforge.net/project/showfiles.php?group_id=54559&package_id=49266 . The installation is basically extracting the archive into your **c:\java** folder. You can either use a WinZip utility or "`jar xvf openjms-0.7.6.1.zip`" from a command prompt.



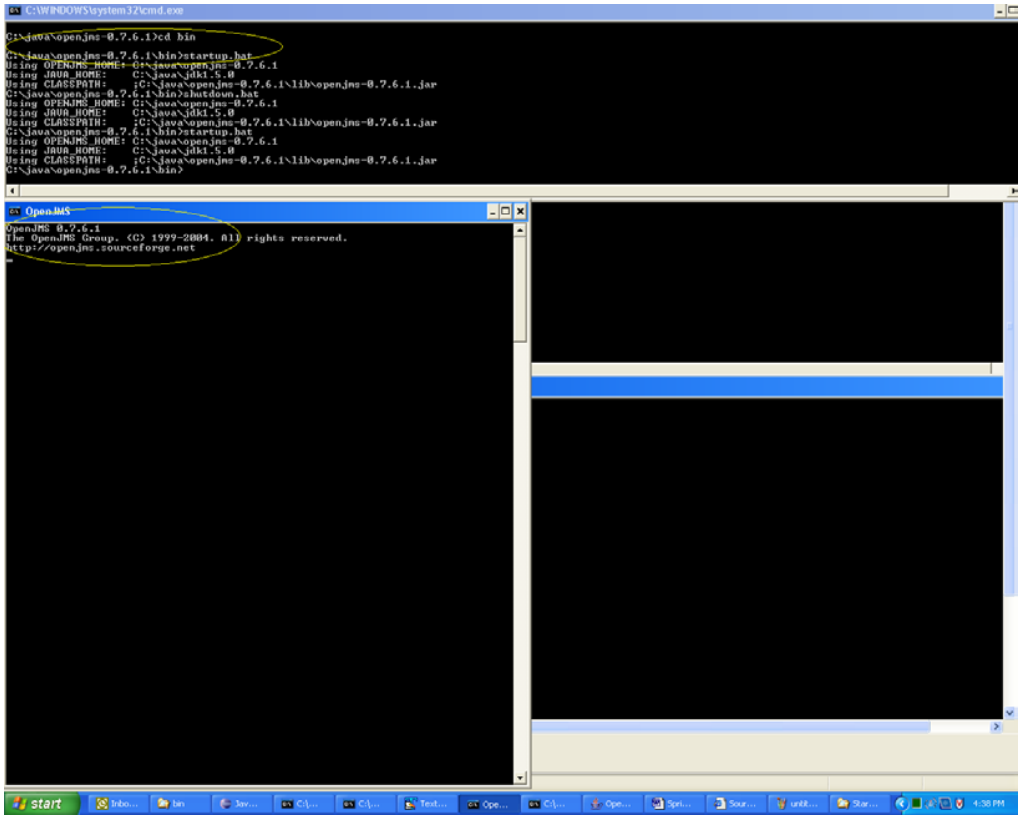
You can start & stop the server in a command prompt using batch scripts under "**bin**" directory. Also have a look at the **opnejms.xml** file under "**config**" folder where connection factories (e.g. Topic Connection Factories, Queue Connection Factories) & the destinations like queues and topics are configured. These are JNDI administered objects and loaded into a JNDI tree (i.e. integral part of OpenJMS) when the server starts up.



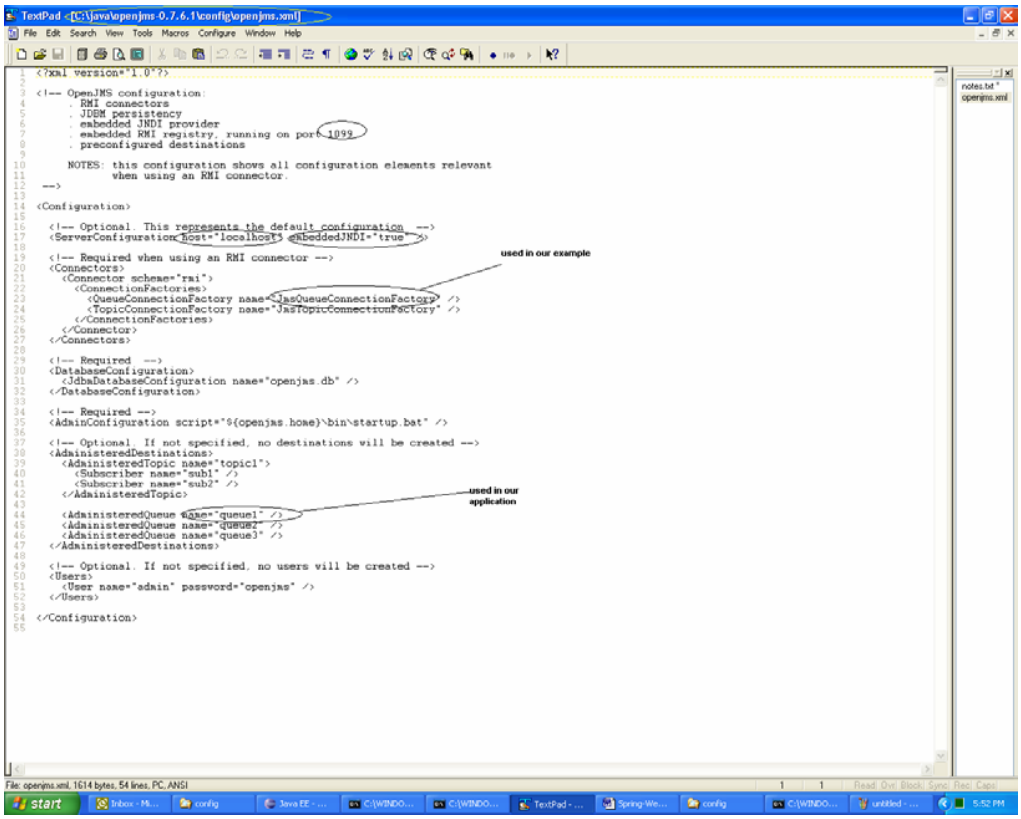
You can start the server by running the "**startup.bat**" under **C:\java\openjms-0.7.6.1\bin**.

You can shutdown the server by running the "**shutdown.bat**" under **C:\java\openjms-0.7.6.1\bin**.

Refer http://openjms.sourceforge.net/usersguide/index.html for documentation.

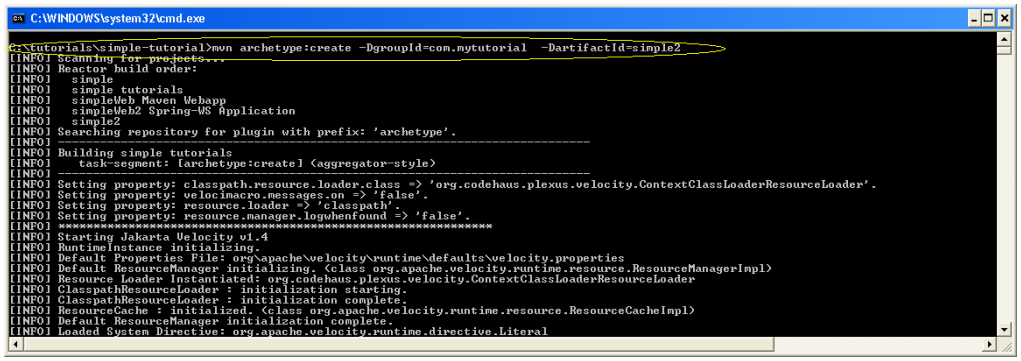


Also have a look at the **openjms.xml** file under **C:\java\openjms-0.7.6.1\config** where some default JNDI administered Connection Factories and destinations are defined.



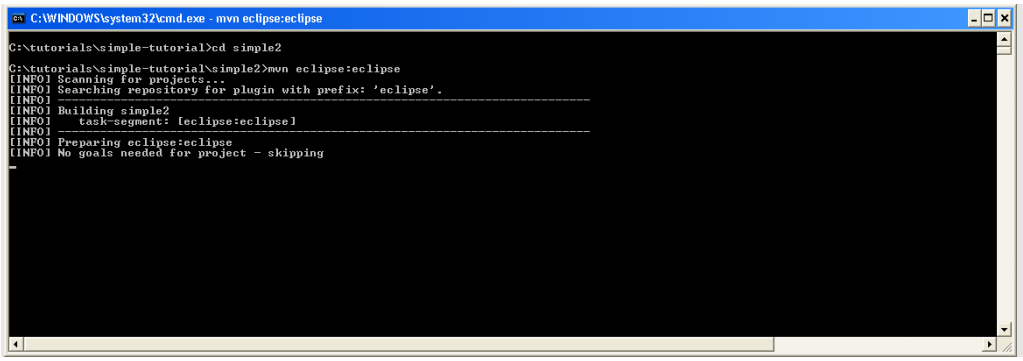
Step 2: Create a new **simple2** project by running the following command in a command prompt.

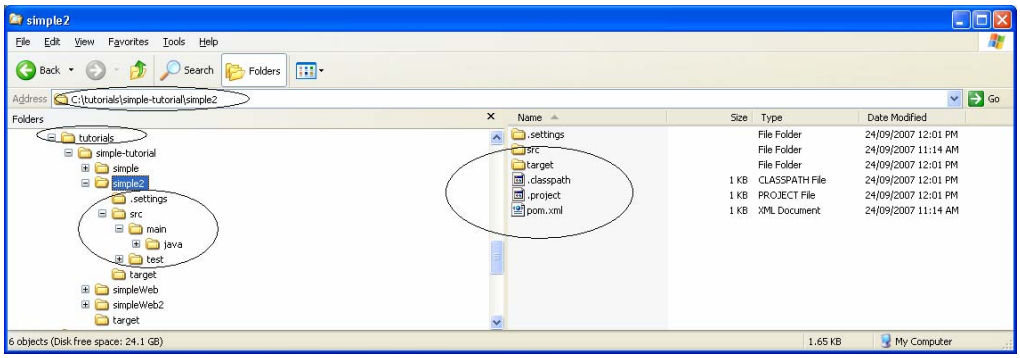
```
C:\tutorials\simple-tutorial>mvn archetype:create -DgroupId=com.mytutorial    \
                                                  -DartifactId=simple2
```



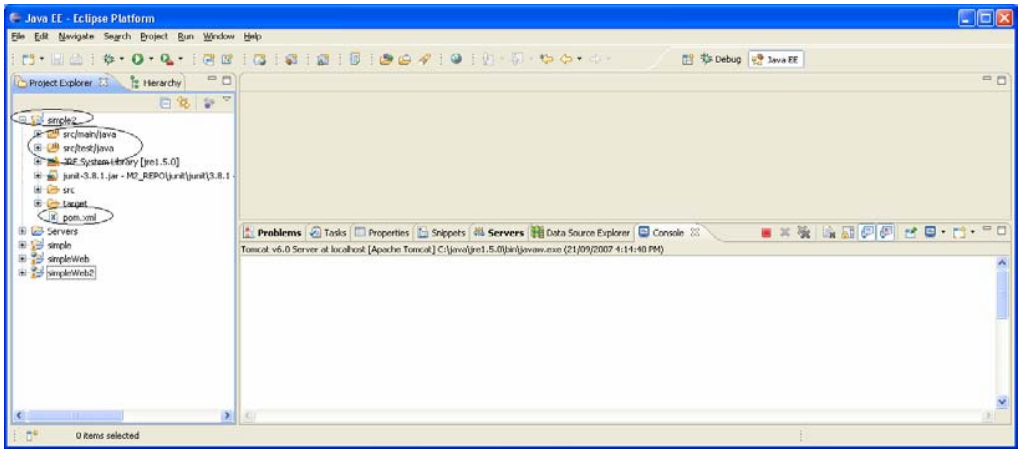
Step 3: Generate eclipse metadata files like **.project** & **.classpath** by running the following command.

```
C:\tutorials\simple-tutorial\simple2>mvn eclipse:eclipse
```

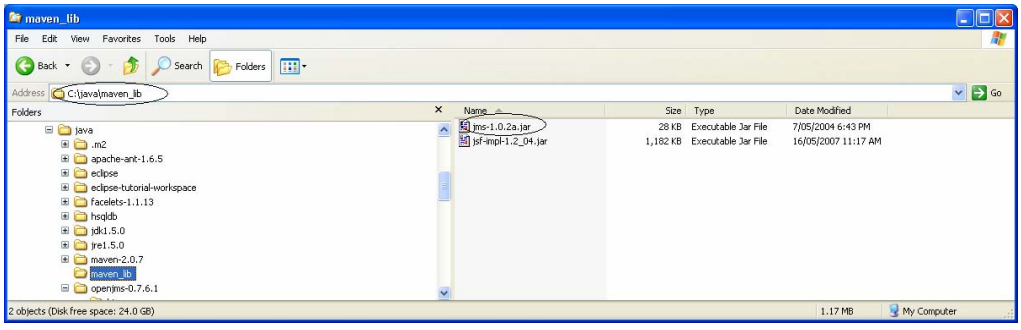




Step 4: Import this project into eclipse workspace as did in previous tutorials**.**

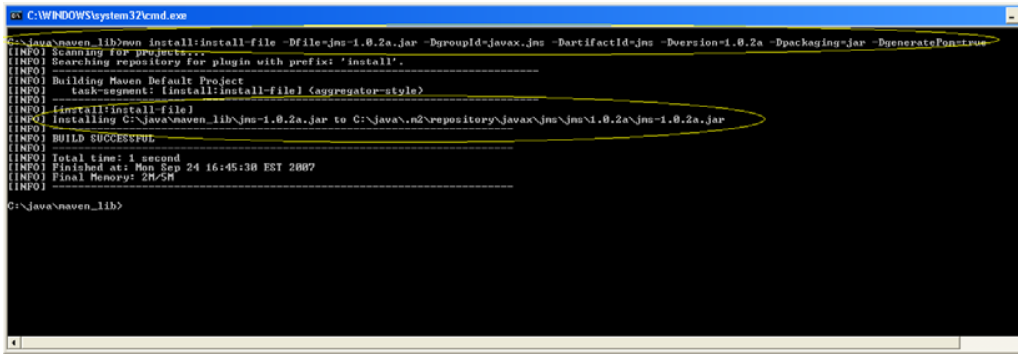


Step 5: Copy **jms-1.0.2a.jar** from "**C:\java\openjms-0.7.6.1\lib**" to "**C:\java\maven_lib**" from where we can install this jar file into our local repository.



Run the following command to install the **jms-1.0.2a.jar** into our local repository at "**C:\java\.m2\repository**".

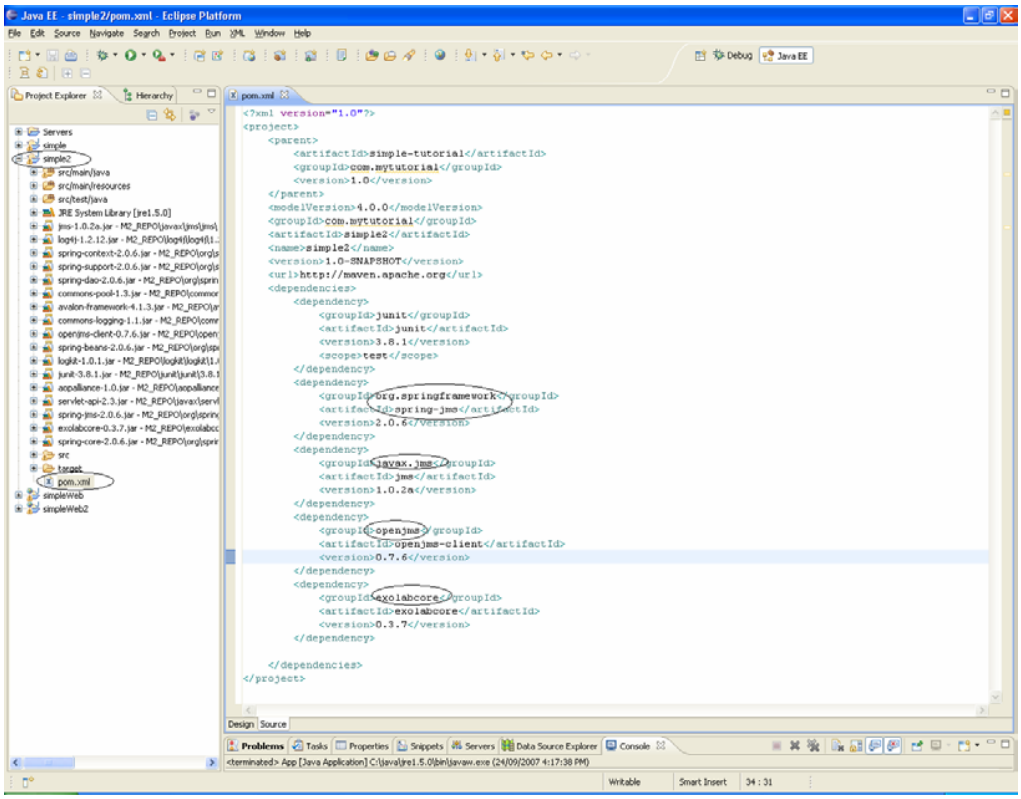
```
C:\java\maven_lib>mvn install:install-file -Dfile=jms-1.0.2a.jar -DgroupId=javax.jms  \
                -DartifactId=jms -Dversion=1.0.2a -Dpackaging=jar -DgeneratePom=true
```



Step 6: Now open up the **pom.xml** file under "**C:\tutorials\simple-tutorial\simple2**" to add the dependencies as shown below:

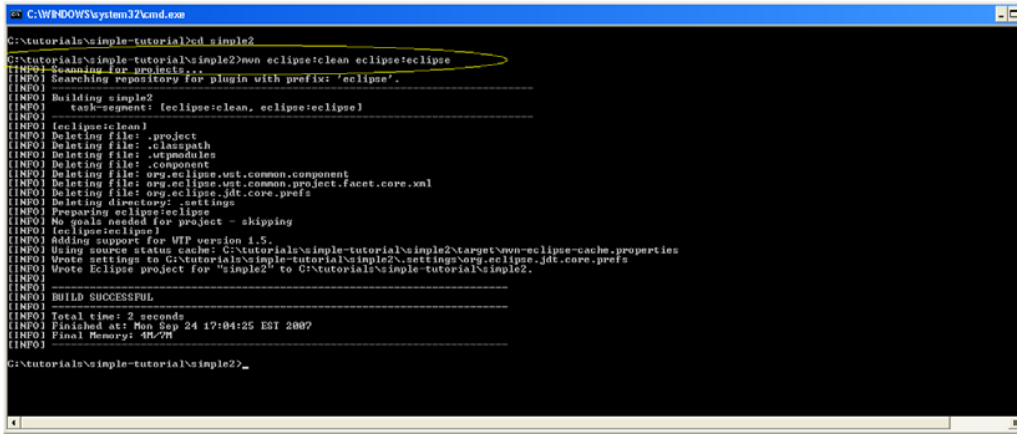
```
<?xml version="1.0"?>
<project>
      <parent>
            <artifactId>simple-tutorial</artifactId>
            <groupId>com.mytutorial</groupId>
            <version>1.0</version>
      </parent>
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.mytutorial</groupId>
      <artifactId>simple2</artifactId>
      <name>simple2</name>
      <version>1.0-SNAPSHOT</version>
      <url>http://maven.apache.org</url>
      <dependencies>
            <dependency>
                  <groupId>junit</groupId>
                  <artifactId>junit</artifactId>
                  <version>3.8.1</version>
                  <scope>test</scope>
            </dependency>
            <dependency>
                  <groupId>org.springframework</groupId>
                  <artifactId>spring-jms</artifactId>
                  <version>2.0.6</version>
            </dependency>
            <dependency>
                  <groupId>javax.jms</groupId>
                  <artifactId>jms</artifactId>
                  <version>1.0.2a</version>
            </dependency>
            <dependency>
                  <groupId>openjms</groupId>
                  <artifactId>openjms-client</artifactId>
                  <version>0.7.6</version>
            </dependency>
            <dependency>
                  <groupId>exolabcore</groupId>
                  <artifactId>exolabcore</artifactId>
                  <version>0.3.7</version>
            </dependency>

      </dependencies>
```

```
</project>
```

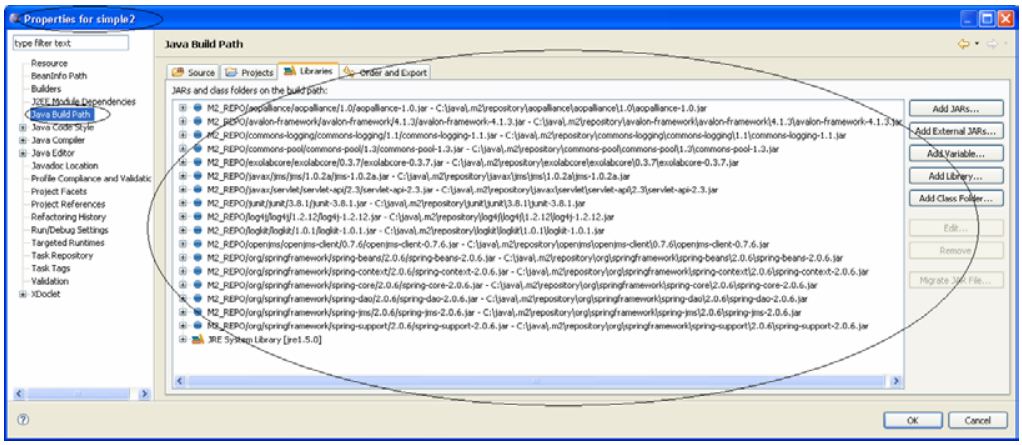


Step 7: Run the following command in a command prompt to update the project build path dependencies.

```
C:\tutorials\simple-tutorial\simple2>mvn eclipse:clean  eclipse:eclipse
```



Now inside eclipse **right click** on "**simple2**" and press "**F5**" to refresh. Check your build path for dependency files as shown below:



Step 8: Create the following **.java** files under **com.mytutorial** package.

MyJmsService.java

```
package com.mytutorial;

public interface MyJmsService {

	public abstract void process();

}
```

MyJmsServiceImpl.java

```
package com.mytutorial;

public class MyJmsServiceImpl implements MyJmsService {

	private JmsSender sender;
	private JmsReceiver receiver;

	public JmsSender getSender() {
		return sender;
	}

	public void setSender(JmsSender sender) {
		this.sender = sender;
	}

	public JmsReceiver getReceiver() {
		return receiver;
	}

	public void setReceiver(JmsReceiver receiver) {
		this.receiver = receiver;
	}

	public void process() {
		sender.sendMesage();

		//sleep for 10 seconds so that you can see the message in
		//the OpenJMS admin console
		try {
			Thread.sleep(10000);
		}
```

```
            catch (InterruptedException iex) {

            }
            receiver.processMessage();
      }

}
```

JmsSender.java

```
package com.mytutorial;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;

import org.springframework.jms.core.JmsTemplate102;
import org.springframework.jms.core.MessageCreator;

public class JmsSender {

      private JmsTemplate102 jmsTemplate102;

      public JmsTemplate102 getJmsTemplate102() {
            return jmsTemplate102;
      }

      public void setJmsTemplate102(JmsTemplate102 jmsTemplate102) {
            this.jmsTemplate102 = jmsTemplate102;
      }

      public void sendMesage() {
            jmsTemplate102.send("queue1", new MessageCreator() {
                  public Message createMessage(Session session)
throws JMSException {
                        return session.createTextMessage("This is a
sample message");
                  }
            });
      }

}
```

Defined in **openjms.xml** under C:\java\openjms-0.7.6.1\config

JmsReceiver.java

```
package com.mytutorial;

import javax.jms.Message;
import javax.jms.TextMessage;

import org.springframework.jms.core.JmsTemplate102;

public class JmsReceiver {
      private JmsTemplate102 jmsTemplate102;

      public JmsTemplate102 getJmsTemplate102() {
            return jmsTemplate102;
      }

      public void setJmsTemplate102(JmsTemplate102 jmsTemplate102) {
            this.jmsTemplate102 = jmsTemplate102;
      }

      public void processMessage() {
```

```
            Message msg = jmsTemplate102.receive("queue1");
            try {
                  TextMessage textMessage = (TextMessage) msg;
                  if (msg != null) {
                        System.out.println(" Message Received -->"
                                    + textMessage.getText());
                  }

            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

Step 9: Next, we need to modify the "**App.java**", which was created before when we ran the maven archetype command.

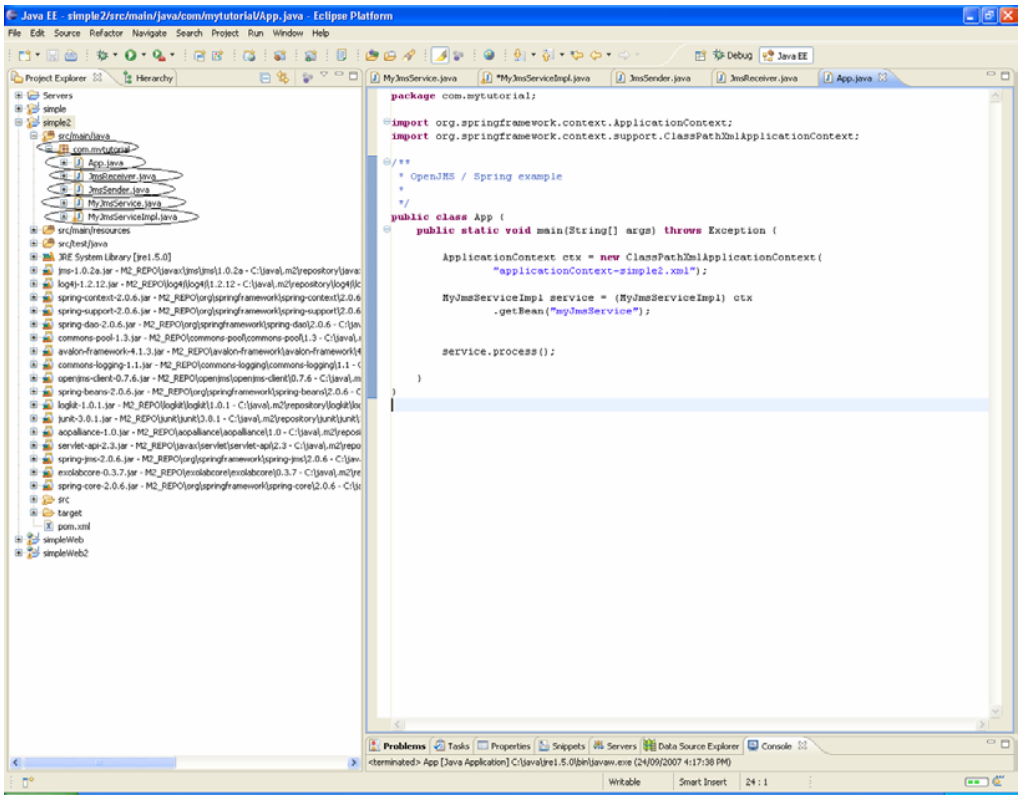
App.java

```
package com.mytutorial;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * OpenJMS / Spring example
 *
 */
public class App {
      public static void main(String[] args) throws Exception {

            ApplicationContext ctx = new
ClassPathXmlApplicationContext(
                        "applicationContext-simple2.xml");

            MyJmsServiceImpl service = (MyJmsServiceImpl) ctx
                        .getBean("myJmsService");


            service.process();

      }
}
```

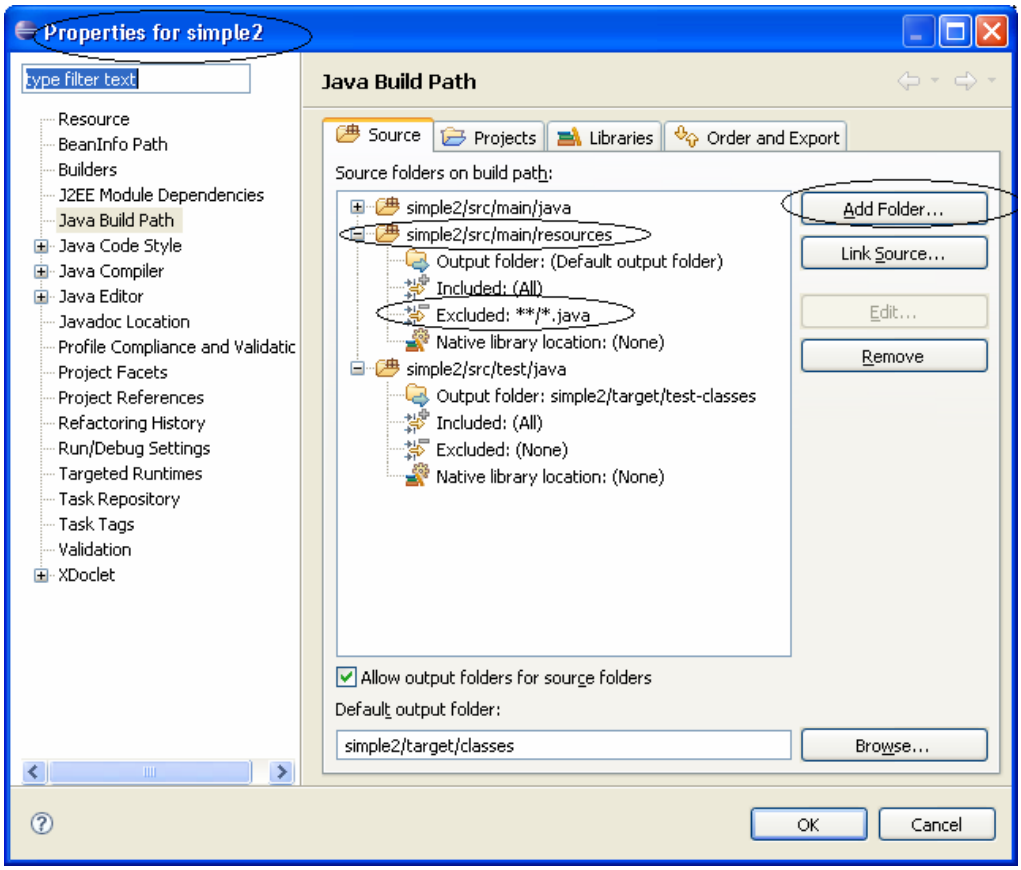
Spring context file

Defined in "**applicationContext-sample2.xml**"

Now you should have the following files



Step 10: Finally we need to wire all these using spring. We will declare this via **applicationContext-simple2.xml**. Firstly need to create the "**resources**" folder under "**simple2/src/main**". You can do this via right clicking on "**simple2**" and then selecting "**properties**"

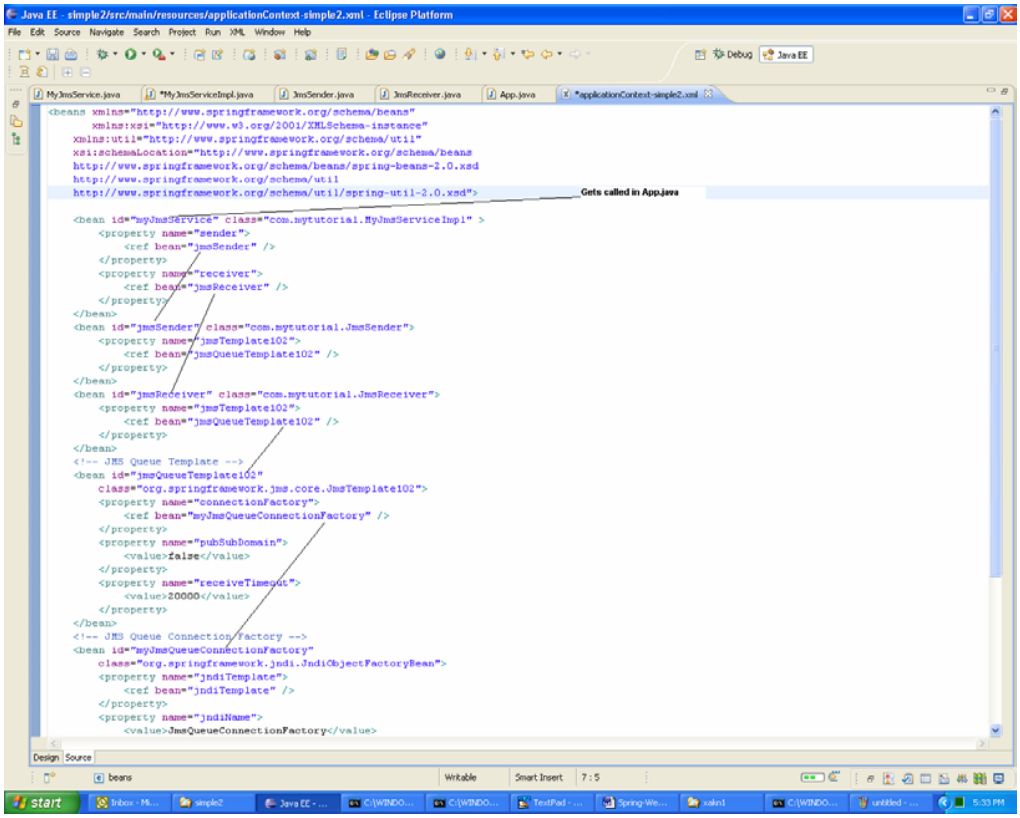


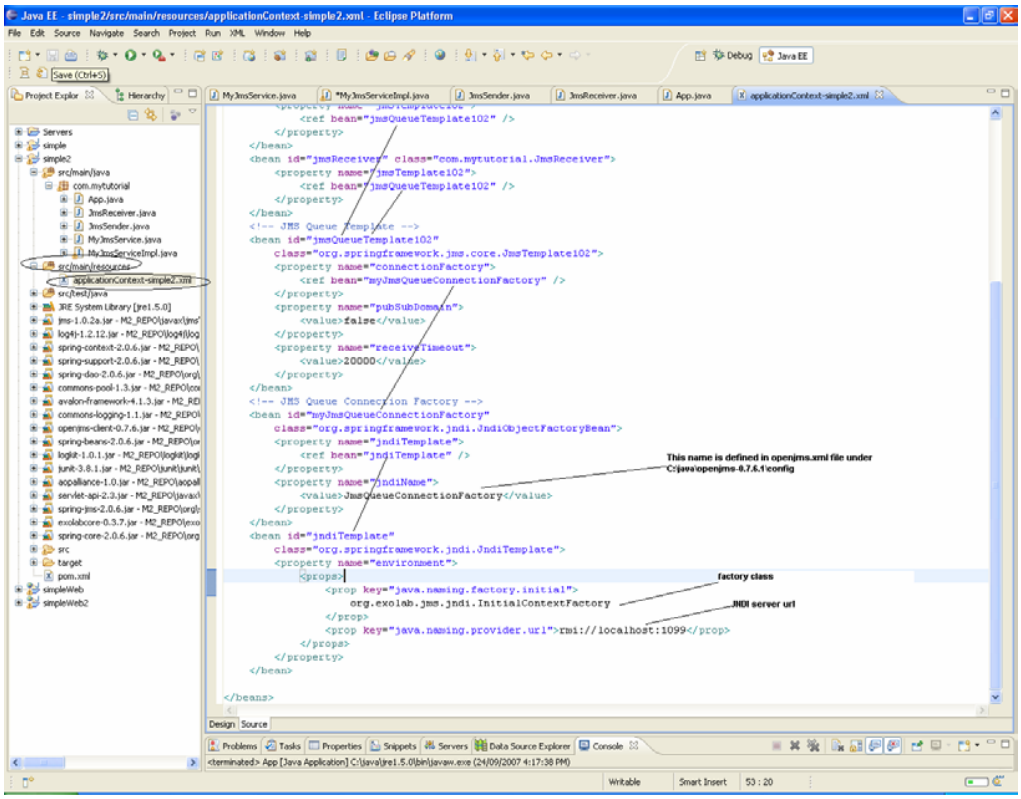
applicationContext-simple2.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:util="http://www.springframework.org/schema/util"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
	http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
	http://www.springframework.org/schema/util 	http://www.springframework.org/schema/util/spring-util-
2.0.xsd">

	<bean id="myJmsService" class="com.mytutorial.MyJmsServiceImpl" >
		<property name="sender">
			<ref bean="jmsSender" />
		</property>
		<property name="receiver">
			<ref bean="jmsReceiver" />
		</property>
	</bean>
	<bean id="jmsSender" class="com.mytutorial.JmsSender">
		<property name="jmsTemplate102">
			<ref bean="jmsQueueTemplate102" />
		</property>
	</bean>
	<bean id="jmsReceiver" class="com.mytutorial.JmsReceiver">
		<property name="jmsTemplate102">
			<ref bean="jmsQueueTemplate102" />
		</property>
	</bean>
	<!-- JMS Queue Template -->
	<bean id="jmsQueueTemplate102"
		class="org.springframework.jms.core.JmsTemplate102">
		<property name="connectionFactory">
			<ref bean="myJmsQueueConnectionFactory" />
		</property>
		<property name="pubSubDomain">
			<value>false</value>
		</property>
```

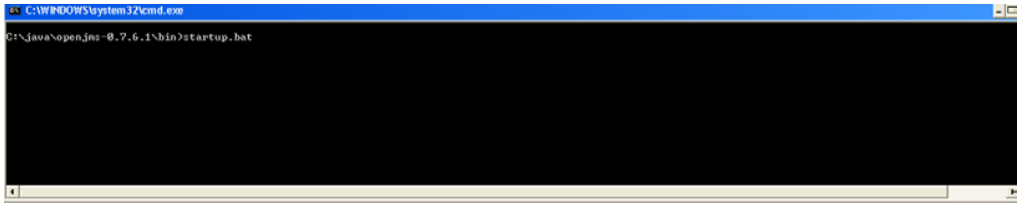
Used in **App.java**

```
                <property name="receiveTimeout">
                        <value>20000</value>
                </property>
        </bean>
        <!-- JMS Queue Connection Factory -->
        <bean id="myJmsQueueConnectionFactory"
                class="org.springframework.jndi.JndiObjectFactoryBean">
                <property name="jndiTemplate">
                        <ref bean="jndiTemplate" />
                </property>
                <property name="jndiName">
                        <value>JmsQueueConnectionFactory</value>
                </property>
        </bean>
        <bean id="jndiTemplate"
                class="org.springframework.jndi.JndiTemplate">
                <property name="environment">
                        <props>
                                <prop key="java.naming.factory.initial">
                                        org.exolab.jms.jndi.InitialContextFactory
                                </prop>
                                <prop key="java.naming.provider.url">rmi://localhost:1099</prop>
                        </props>
                </property>
        </bean>

</beans>
```

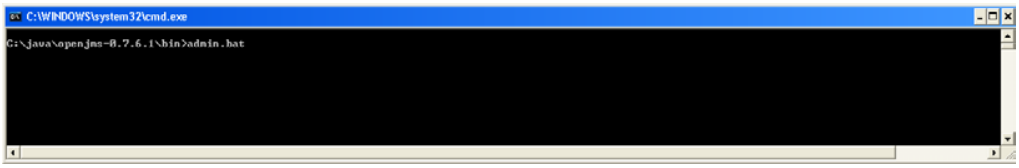




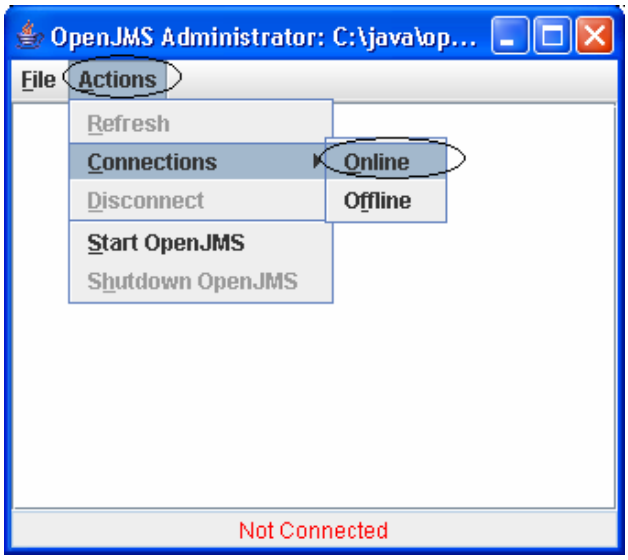
Step 11: The final step is to run this application. Firstly make sure that the "**OpenJMS**" server has started. If not start it by:

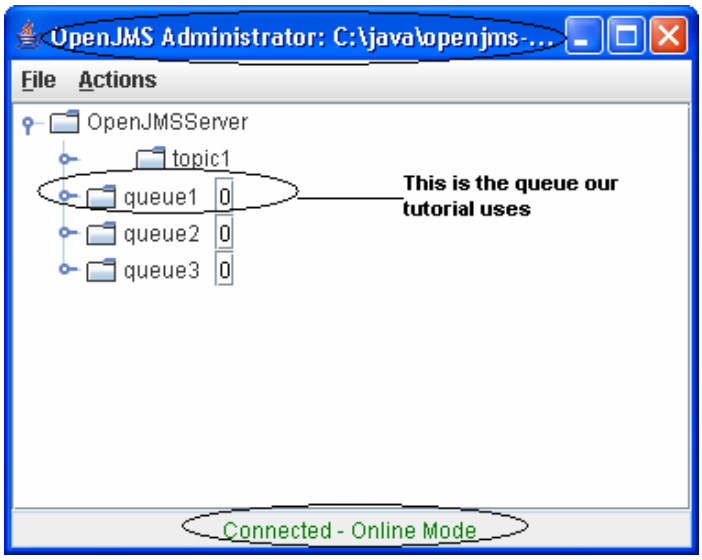


Next open up the "**OpenJMS administrator**" from a separate command prompt:

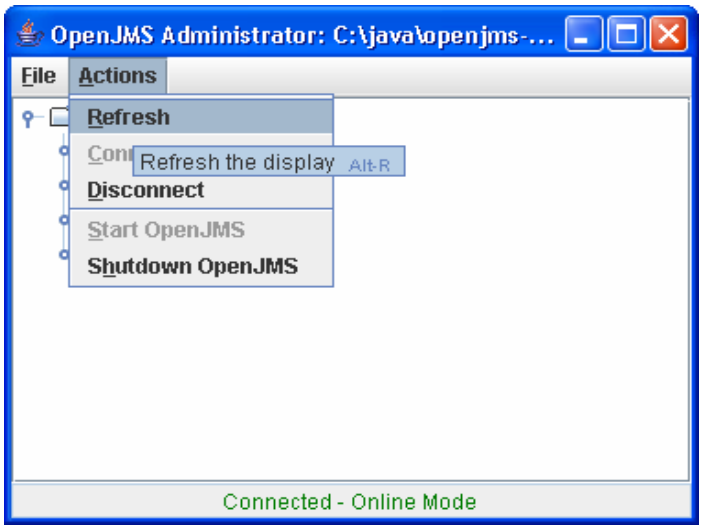


This will spawn a new window:

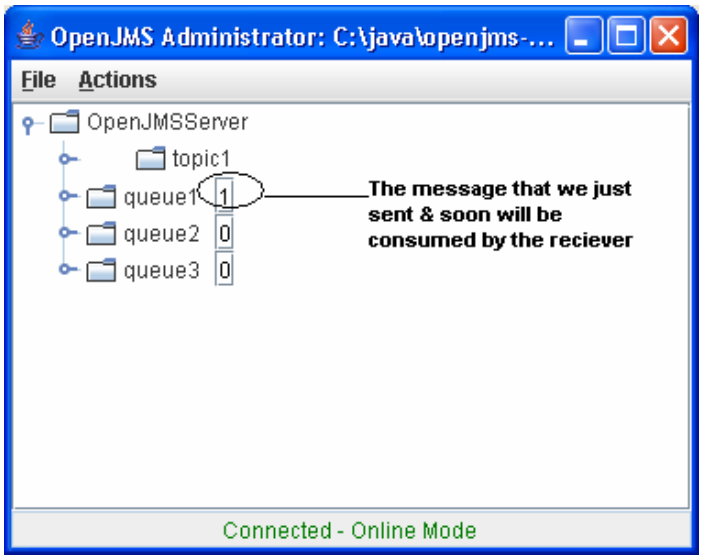




Finally run the **App.java** from the eclipse and also refresh the OpenJMS administrator screen to check the message count in the destination(s).



You can notice the presence of a sent message. This is the reason why a **Thread.sleep(10000)** was added in "**`MyJmsServiceImpl.java`**" file in the **`process(…)`** method.



After 10 seconds you should get an output as follows:

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
 Message Received -->This is a sample message
```

Once the message is consumed the **queue1** count will be back to "**0**".

Experiment with sending different messages by modifying the code. You could also try sending more than one messages and receive them in the receiver by changing the code appropriately (receiver may require a while loop to continuously monitor for messages). Also try the publish/subscribe model.

Interview Questions with Answers on **JMS** are discussed under "J2EE" section (subsection JMS) in **Java/J2EE Job Interview Companion** at http://www.lulu.com/content/192463

Please feel free to email any errors to java-interview@hotmail.com. Also stay tuned at http://www.lulu.com/java-success for more tutorials and Java/J2EE interview resources.

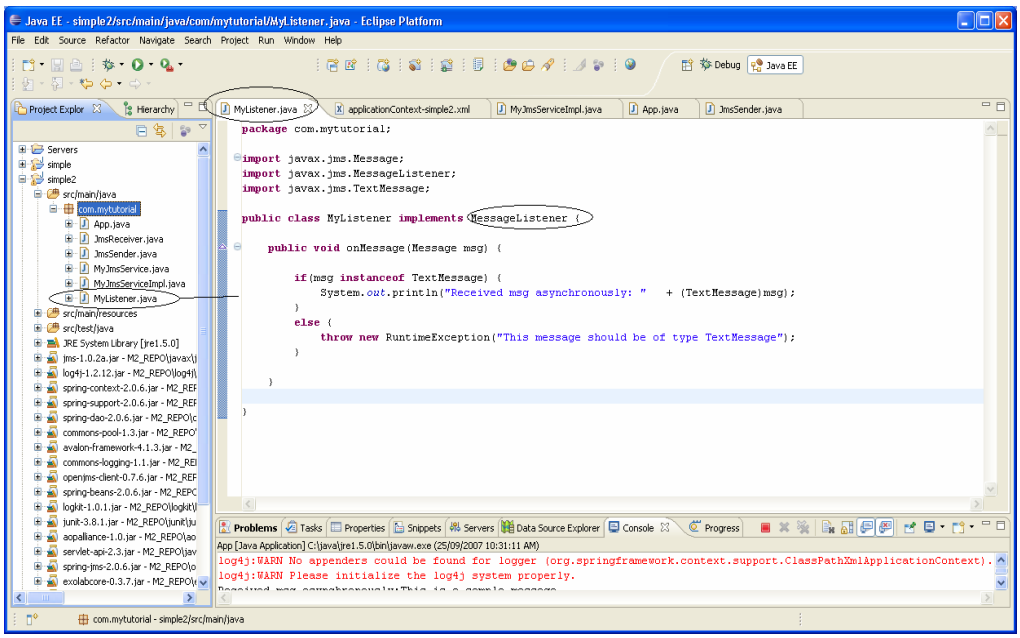
Tutorial 14 – Spring JMS Asynchronous

In the last tutorial (Tutorial 13), we looked at consuming messages synchronously. JMS is typically associated with **asynchronous** process. During a synchronous receiving process the calling thread blocks until a message becomes available or timeout occurs. Similar to MDBs (Message Driven Beans) in the EJB world a Message Driven Pojo (MDP) can act as a message receiver to receive messages asynchronously.

Step 1: Write the Message Driven Pojo (MDP) class.

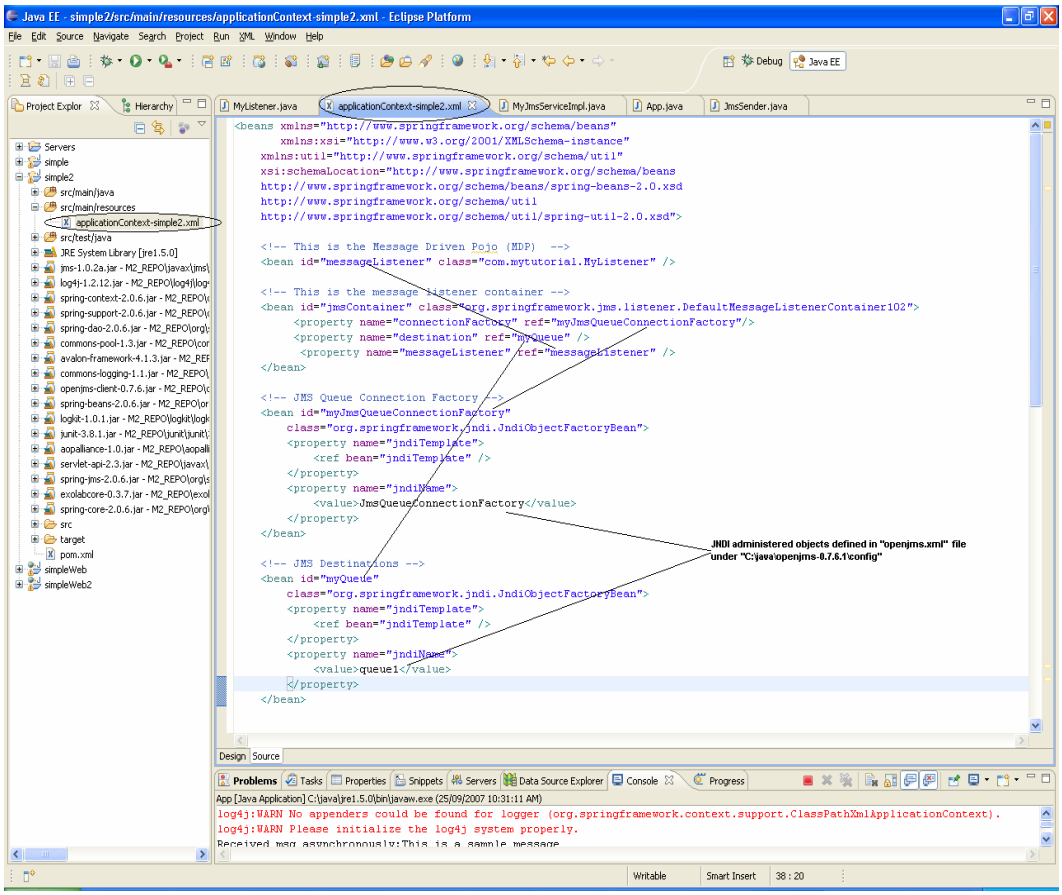
MyListener.java

```
package com.mytutorial;

import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

public class MyListener implements MessageListener {

	public void onMessage(Message msg) {

		if(msg instanceof TextMessage) {
			System.out.println("Received msg asynchronously: "  + (TextMessage)msg);
		}
		else {
			throw new RuntimeException("This message should be of type TextMessage");
		}

	}

}
```



Step 2: Modify the **"applicationContext-simple2.xml"** to add the "**messageListener**" and the "**jmsContainer**".

```
<!-- This is the Message Driven Pojo (MDP)  -->
<bean id="messageListener" class="com.mytutorial.MyListener" />

<!-- This is the message listener container -->
<bean id="jmsContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer102">
    <property name="connectionFactory" ref="myJmsQueueConnectionFactory"/>
    <property name="destination" ref="myQueue" />
     <property name="messageListener" ref="messageListener" />
</bean>

<!-- JMS Queue Connection Factory -->
<bean id="myJmsQueueConnectionFactory" class="org.springframework.jndi.JndiObjectFactoryBean">
            <property name="jndiTemplate">
                        <ref bean="jndiTemplate" />
            </property>
            <property name="jndiName">
                        <value>JmsQueueConnectionFactory</value>
            </property>
</bean>

<!-- JMS Destinations -->
<bean id="myQueue"  class="org.springframework.jndi.JndiObjectFactoryBean">
            <property name="jndiTemplate">
                        <ref bean="jndiTemplate" />
            </property>
            <property name="jndiName">
                        <value>queue1</value>
            </property>
</bean>
```



The complete "**applicationContext-simple2.xml**" should look like

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:util="http://www.springframework.org/schema/util"
            xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
        http://www.springframework.org/schema/util
        http://www.springframework.org/schema/util/spring-util-2.0.xsd">

        <!-- This is the Message Driven Pojo (MDP)  -->
        <bean id="messageListener" class="com.mytutorial.MyListener" />

        <!-- This is the message listener container -->
        <bean id="jmsContainer" class="org.springframework.jms.listener.DefaultMessageListenerContainer102">
            <property name="connectionFactory" ref="myJmsQueueConnectionFactory"/>
            <property name="destination" ref="myQueue" />
             <property name="messageListener" ref="messageListener" />
        </bean>

        <!-- JMS Queue Connection Factory -->
        <bean id="myJmsQueueConnectionFactory"
                    class="org.springframework.jndi.JndiObjectFactoryBean">
                    <property name="jndiTemplate">
                                <ref bean="jndiTemplate" />
                    </property>
                    <property name="jndiName">
                                <value>JmsQueueConnectionFactory</value>
                    </property>
        </bean>

        <!-- JMS Destinations -->
        <bean id="myQueue"
                    class="org.springframework.jndi.JndiObjectFactoryBean">
                    <property name="jndiTemplate">
                                <ref bean="jndiTemplate" />
                    </property>
                    <property name="jndiName">
                                <value>queue1</value>
                    </property>
        </bean>


        <bean id="myJmsService" class="com.mytutorial.MyJmsServiceImpl" >
                    <property name="sender">
                                <ref bean="jmsSender" />
                    </property>
                    <property name="receiver">
                                <ref bean="jmsReceiver" />
                    </property>
        </bean>
        <bean id="jmsSender" class="com.mytutorial.JmsSender">
                    <property name="jmsTemplate102">
                                <ref bean="jmsQueueTemplate102" />
                    </property>
        </bean>
        <bean id="jmsReceiver" class="com.mytutorial.JmsReceiver">
                    <property name="jmsTemplate102">
                                <ref bean="jmsQueueTemplate102" />
                    </property>
        </bean>

        <!-- JMS Queue Template -->
        <bean id="jmsQueueTemplate102"
                    class="org.springframework.jms.core.JmsTemplate102">
                    <property name="connectionFactory">
                                <ref bean="myJmsQueueConnectionFactory" />
                    </property>
                    <property name="pubSubDomain">
                                <value>false</value>
                    </property>
                    <property name="receiveTimeout">
                                <value>20000</value>
                    </property>
        </bean>

        <bean id="jndiTemplate"
                    class="org.springframework.jndi.JndiTemplate">
                    <property name="environment">
                                <props>
                                            <prop key="java.naming.factory.initial">
                                                        org.exolab.jms.jndi.InitialContextFactory
                                            </prop>
                                            <prop key="java.naming.provider.url">rmi://localhost:1099</prop>
                                </props>
```

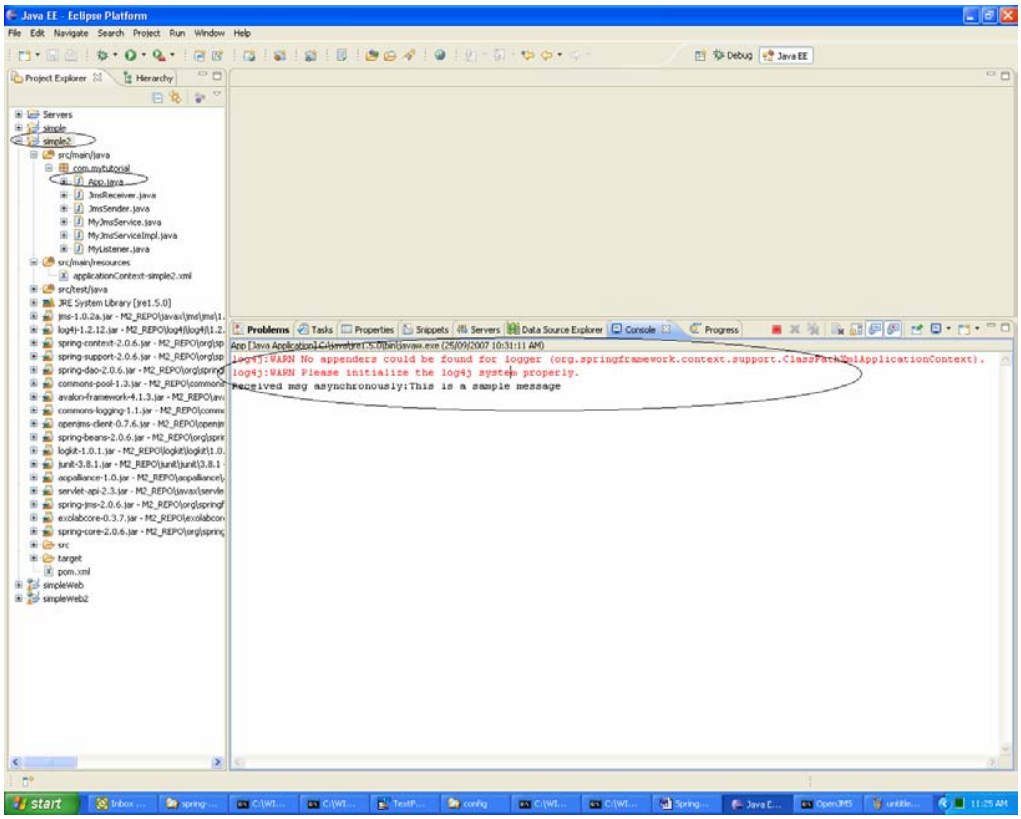
```
                </property>
        </bean>

</beans>
```

Step 3: Modify the "**MyJmsServiceImpl.java**" by commenting out the receiver section as shown below.

```
package com.mytutorial;

public class MyJmsServiceImpl implements MyJmsService {

      private JmsSender sender;
      private JmsReceiver receiver;

      public JmsSender getSender() {
            return sender;
      }

      public void setSender(JmsSender sender) {
            this.sender = sender;
      }

      public JmsReceiver getReceiver() {
            return receiver;
      }

      public void setReceiver(JmsReceiver receiver) {
            this.receiver = receiver;
      }

      public void process() {

            sender.sendMesage();

            //sleep for 10 seconds so that you can see the message in
            //the OpenJMS admin console

            /*try {
                  Thread.sleep(10000);
            }

            catch (InterruptedException iex) {

            }
            receiver.processMessage();*/
      }

}
```

Step 4: Make sure that the "**OpenJMS Server**" is running and then run the **App.java** and see asynchronous message receipts in action.

```
log4j:WARN No appenders could be found for logger
(org.springframework.context.support.ClassPathXmlApplicationContext).
log4j:WARN Please initialize the log4j system properly.
Received msg asynchronously:This is a sample message
```



That's all to it. Experiment with publish/subscribe, multiple messages etc. Also refer to spring reference documentation at http://www.springframework.org/docs/reference/index.html.
