



# Licence

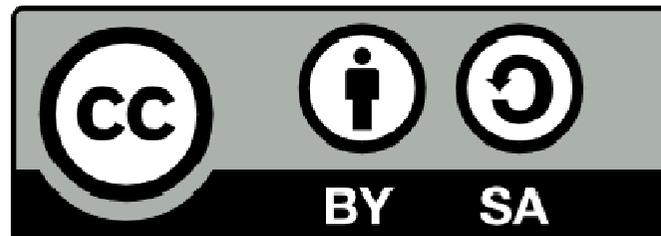
---

## Creative Commons

*Contrat Paternité*

*Partage des Conditions Initiales à l'Identique*

2.0 France

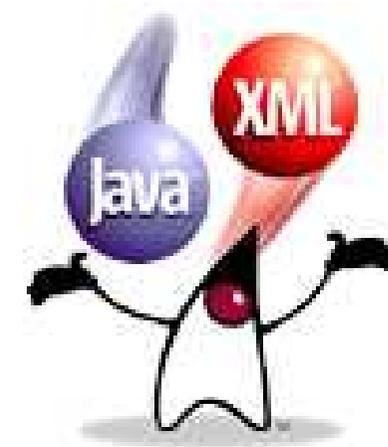


<http://creativecommons.org/licenses/by-sa/2.0/fr>

# Plan du cours

---

- Généralités JAX-WS
- Développement serveur
  - Bottom -> Up
  - Top -> Down
- Développement client
- Annotations
- Handler
- Java 6 et EJB (client et serveur)
- JAX-WS « in Actions » : Services Web eBay
- Tutoriels avec Netbeans 6.8



# Déroulement du cours

---

## ➤ Pédagogie du cours

- Illustration avec de nombreux exemples qui sont disponibles à l'adresse TODO
- Des bulles d'aide tout au long du cours
- Survol des principaux concepts en évitant une présentation exhaustive

## ➤ Logiciels utilisés

- Navigateur Web, Netbeans 6.8, Tomcat 6, Glassfish 3, Maven 2

## ➤ Pré-requis

- Schema XML, WSDL, SOAP, JAXB

## ➤ Remerciements

- HowHigh



# Ressources

---

## ➤ Billets issus de Blog

- [blogs.sun.com/alexismp/entry/metro\\_boulot\\_dodo](http://blogs.sun.com/alexismp/entry/metro_boulot_dodo)
- [www.dotmyself.net/documentation/5.html](http://www.dotmyself.net/documentation/5.html)
- [jee-bpel-soa.blogspot.com/search/label/jax-ws](http://jee-bpel-soa.blogspot.com/search/label/jax-ws)

## ➤ Articles

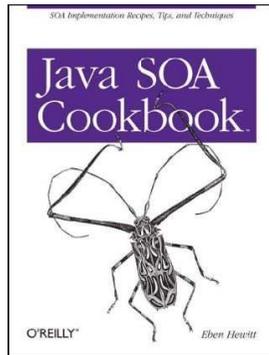
- [wiki.apache.org/ws/StackComparison](http://wiki.apache.org/ws/StackComparison)
- [www.ibm.com/developerworks/webservices/library/ws-jsrart](http://www.ibm.com/developerworks/webservices/library/ws-jsrart)
- [java.sun.com/developer/technicalArticles/J2SE/jax\\_ws\\_2/](http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/)
- [www.jroller.com/gmazza/entry/adding\\_jax\\_ws\\_handlers\\_to](http://www.jroller.com/gmazza/entry/adding_jax_ws_handlers_to)
- [blog.vinodsingh.com/2008/09/using-jax-ws-handlers\\_25.html](http://blog.vinodsingh.com/2008/09/using-jax-ws-handlers_25.html)

## ➤ Cours

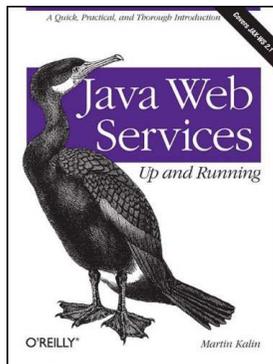
- [java.sun.com/webservices/docs/1.6/tutorial/doc/index.html](http://java.sun.com/webservices/docs/1.6/tutorial/doc/index.html)
- [www.javapassion.com/webservices/jaxwsbasics.pdf](http://www.javapassion.com/webservices/jaxwsbasics.pdf)

# Ressources : Bibliothèque

---



- **Java SOA Cookbook**
  - Auteur : Eben Hewitt
  - Éditeur : Oreilly
  - Edition : Mars 2009 - 752 pages - ISBN : 0596520727



- **Java Web Services : Up and Running**
  - Auteur : Martin Kalin
  - Éditeur : Oreilly
  - Edition : Février 2009 - 316 pages - ISBN : 059652112X



- **Les Cahiers du Programmeur : Java EE 5**
  - Auteur : Antonio Goncalves
  - Éditeur : Eyrolles
  - Edition : Août 2008 - 351 pages - ISBN : 2212123639

# Généralités - Développement de Services Web

---

- Nous nous intéressons dans ce cours au développement des services Web
  - Côté Serveur : code pour le traitement du service Web
  - Côté Client : code qui permet d'appeler un service Web
- La majorité des langages de programmation orientés Web supportent le développement de services Web
  - Java, PHP, C#, C++, ...
- Nous nous limitons au langage Java dans ce cours
- Différents *frameworks* de développement de Services Web
  - **JAX-WS** Specification Sun ([jax-ws.dev.java.net](http://jax-ws.dev.java.net))
  - **AXIS 1** et **2** Apache ([ws.apache.org/axis](http://ws.apache.org/axis) et [ws.apache.org/axis2](http://ws.apache.org/axis2))
  - **CXF** Apache ([cxf.apache.org](http://cxf.apache.org))
  - **XFire** Codehaus ([xfire.codehaus.org](http://xfire.codehaus.org))
  - **JBossWS** JBoss ([www.jboss.org/jbossws](http://www.jboss.org/jbossws))

# Généralités JAX-WS

---

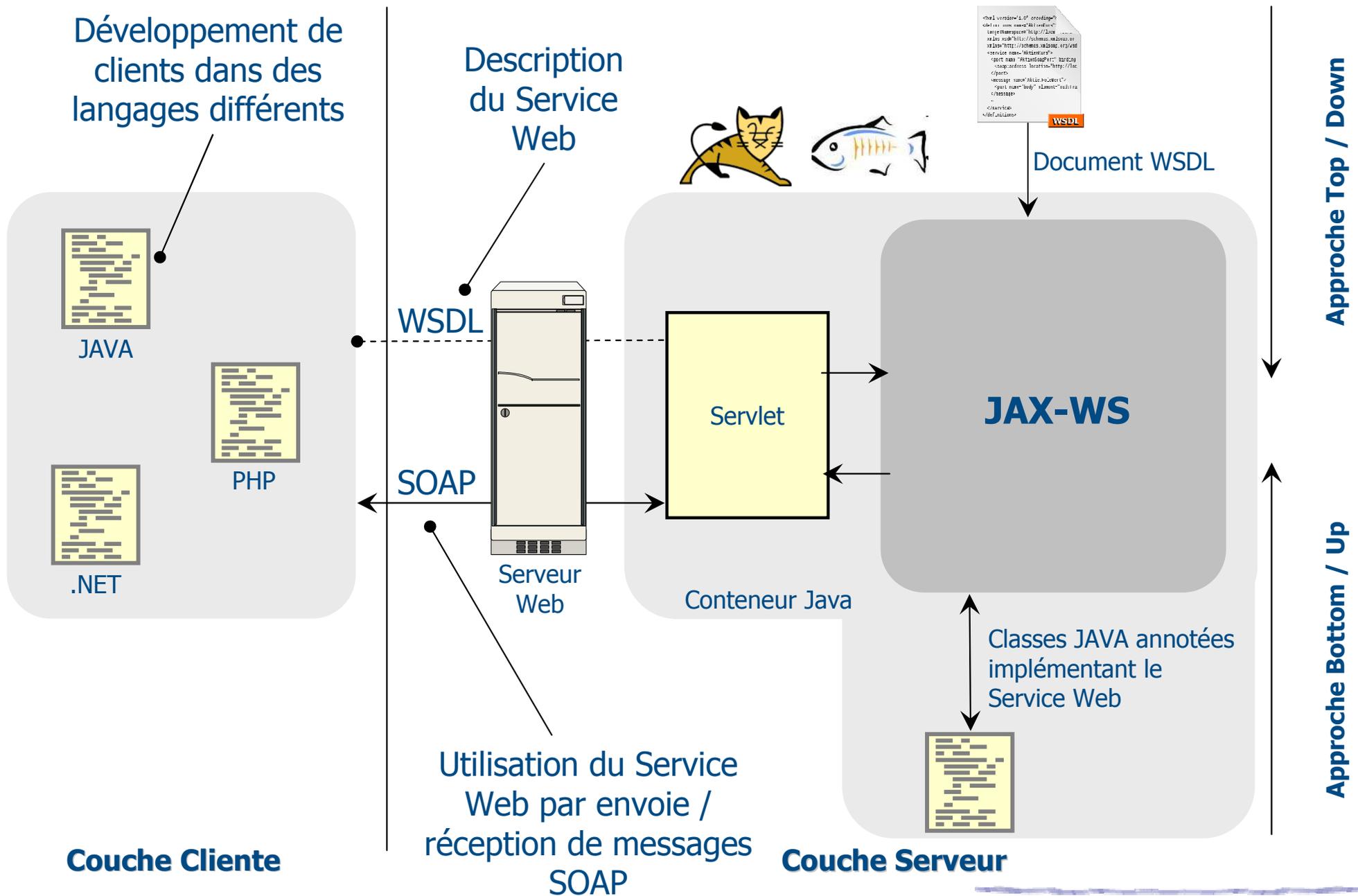
- **JAX-WS** est l'acronyme **J**ava **A**PI for **X**ML **W**eb **S**ervices
- JAX-WS est à la fois un standard et une implémentation
- La version courante est JAX-WS 2.0, précédemment JAX-WS s'appelait JAX-RPC
- JAX-WS s'appuie sur un ensemble de JSR
  - JSR 224 : JAX-WS
  - JSR 181 : Web Services Metadata
  - JSR 222 : JAXB
  - JSR 250 : Common Annotations

# Généralités JAX-WS

---

- L'implémentation de référence est fournie par **METRO** appelée également JAX-WS RI (Reference Implementation)
  - Site projet Metro : <https://metro.dev.java.net/>
- **WSIT** (Web Services Interoperability Technologies) est un complément pour gérer les Web Services avancés (WS-\*)
  - Site projet WSIT : <http://wsit.dev.java.net/>
- L'implémentation JAX-WS est intégrée nativement à la JRE depuis la version 6
- Il est possible de développer des Services Web en dehors d'un serveur d'application en mode autonome (étudié à la fin de ce support de cours)

# Généralités JAX-WS



# Généralités JAX-WS

---

- Le développement de Services Web avec JAX-WS est basé sur des POJO (**P**lain **O**ld **J**ava **O**bject)
- Les fonctionnalités de base pour le développement de Web Services avec JAX-WS requiert simplement l'utilisation d'annotations Java
- Par conséquent aucun fichier de déploiement n'est requis
- Toutefois, les fonctionnalités avancées (appels asynchrones) nécessitent d'utiliser une API
- JAX-WS permet d'assurer l'indépendance du protocole (SOAP) et du transport (HTTP)

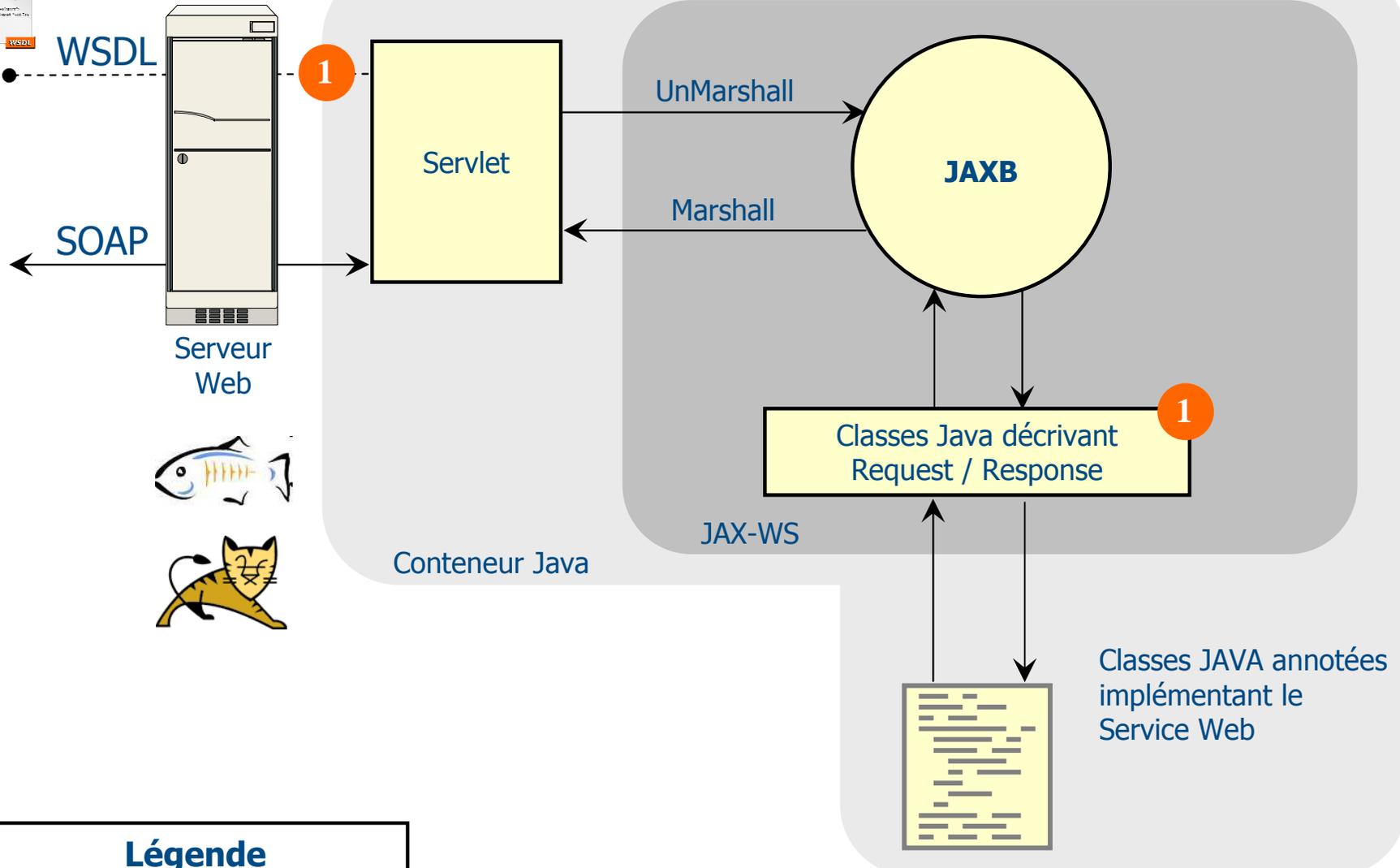
# Développement Serveur : généralités

---

- Deux façons pour développer un Service Web avec JAX-WS
- **Approche Top / Down** (à partir d'un document WSDL)
  - Génération des différentes classes Java (JAXB et squelette du Web Service) en utilisant l'outil *wsimport*
  - Compléter le squelette de classe de l'implémentation
  - Compiler, déployer et tester
- **Approche Bottom / Up** (à partir d'un POJO)
  - Créer et annoter un POJO
  - Compiler, déployer et tester
  - Le document WSDL est automatiquement généré

# Développement Serveur : Bottom / Up

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions name="Service" targetNamespace="http://schemas.xmlsoap.org/wsdl/" >
  <wsdl:import namespace="http://schemas.xmlsoap.org/soap/envelope/" wsdl:base="http://schemas.xmlsoap.org/soap/envelope/" />
  <wsdl:types base="http://schemas.xmlsoap.org/wsdl/" />
  <wsdl:message name="Request" />
  <wsdl:message name="Response" />
  <wsdl:portType name="ServicePortType" />
  <wsdl:binding name="ServiceBinding" type="ServicePortType" />
  <wsdl:service name="Service" />
</wsdl:definitions>
```



Légende	
1	Générer implicitement par l'outil <i>WSGEN</i>

## Développement Serveur : Bottom / Up

---

- L'approche Bottom / Up consiste à démarrer le développement à partir d'une classe Java (POJO)
- Ajouter l'annotation *@WebService*
- Déployer l'application sur un serveur d'application (ou via directement Java SE 6)
- Le document WSDL est généré automatiquement en respectant les valeurs par défaut
  - URL du WSDL : *http://monserveur/app/Service?WSDL*
- Toutes les méthodes du POJO sont des opérations du Web Service
- La surcharge de méthodes n'est pas supportée

# Développement Serveur : Bottom / Up

## ➤ Exemple : Implémentation du Service Web *HelloWorld*

Le package fait parti intégrante de la JRE 6

```
package fr.ensma.lisi.helloworldquietwebservice;

import javax.jws.WebService;

@WebService
public class HelloWorldService {

    public String makeHelloWorld(String value) {
        return "Hello World to " + value;
    }

    public String simpleHelloWorld() {
        return "Hello World to everybody";
    }
}
```

Deux opérations sont définies dans la classe *HelloWorldService*

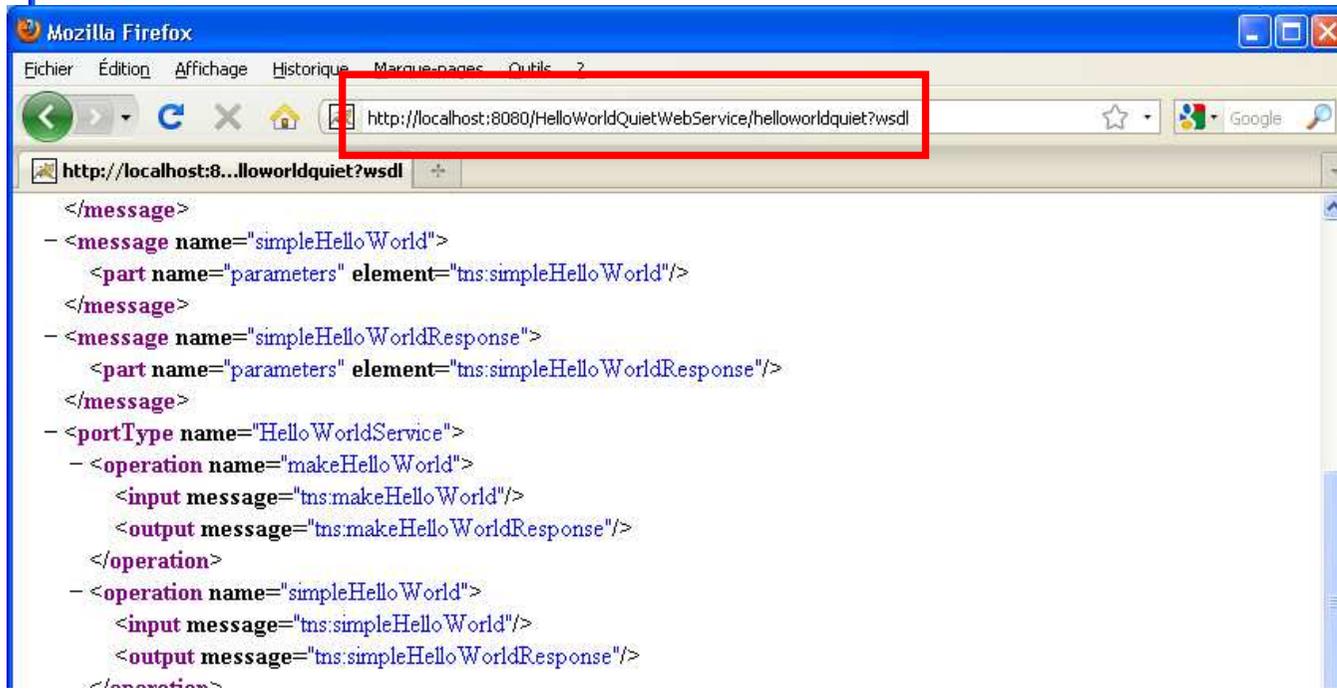
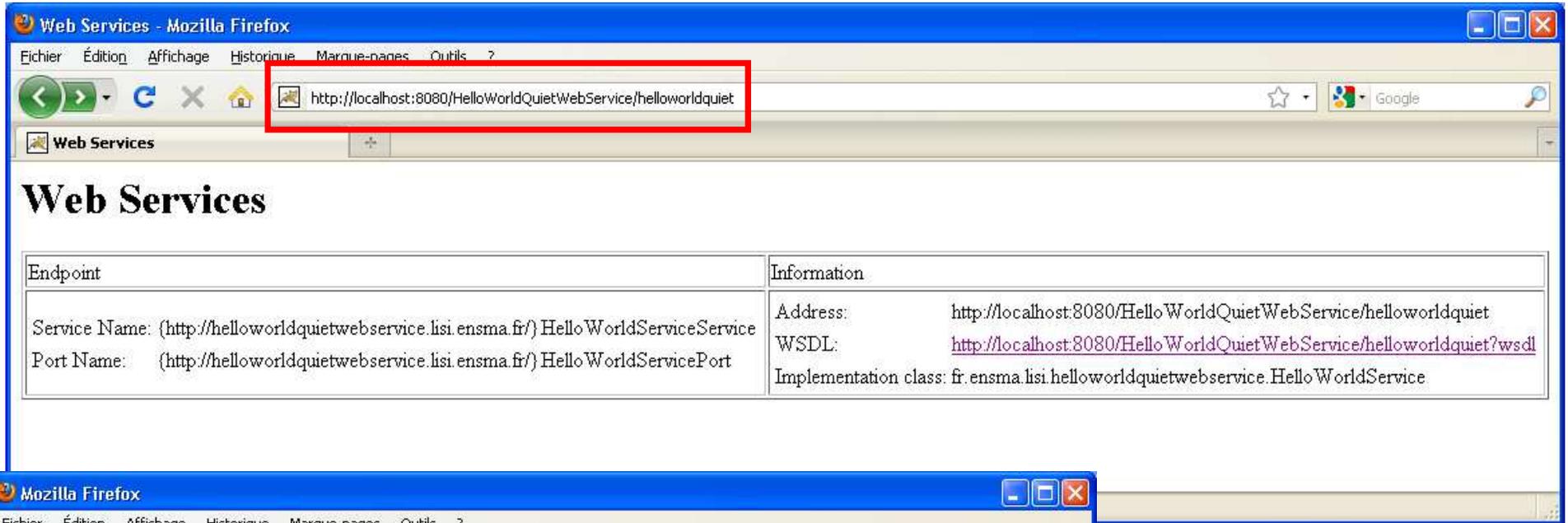
L'opération *makeHelloWorld* contenant un message input et un message output

L'opération *simpleHelloWorld* contenant un message output uniquement

*HelloWorldService.java* du projet **HelloWorldQuietWebService**

# Développement Serveur : Bottom / Up

## ➤ Exemple (suite) : Implémentation du Service Web *HelloWorld*



Document WSDL du Service Web développé

## ➤ Exemple : Paramétrer le Service Web *HelloWorld*

```
package fr.ensma.lisi.helloworldwebservice;

@WebService(name="HelloWorld",targetNamespace="http://helloworldwebservice.lisi.ensma.fr/")
@SOAPBinding(style=Style.RPC, use=Use.LITERAL)
public interface HelloWorldService {

    @WebMethod(operationName="makeHelloWorld")
    @WebResult(name="helloWorldResult")
    public String makeHelloWorld(
        @WebParam(name = "value")String value);

    @WebMethod(operationName="simpleHelloWorld")
    @WebResult(name="helloWorldResult")
    public String simpleHelloWorld();
}
```

Utilisation d'une interface pour  
définir les paramètres du  
Service Web

*HelloWorldService.java* du projet  
**HelloWorldWebService**

```
package fr.ensma.lisi.helloworldwebservice;

@WebService(endpointInterface="fr.ensma.lisi.helloworldwebservice.HelloWorldService",
serviceName="HelloWorld", portName="HelloWorldPort")
public class HelloWorldServiceImpl implements HelloWorldService {

    public String makeHelloWorld(String value) {
        return "Hello World to " + value;
    }

    public String simpleHelloWorld() {
        return "Hello World to everybody";
    }
}
```

Classe qui fournit  
l'implémentation du Service Web

*HelloWorldServiceImpl.java* du projet  
**HelloWorldWebService**

## ➤ Exemple (bis) : Paramétrer le Service Web *HelloWorld*

```
package fr.ensma.lisi.helloworldwebservice;

@WebService(name="HelloWorld",targetNamespace="http://helloworldwebservice.lisi.ensma.fr/")
@SOAPBinding(style=Style.RPC, use=Use.LITERAL)
public interface HelloWorldService {

    @WebMethod(operationName="makeHelloWorld")
    @WebResult(name="helloWorldResult")
    public String makeHelloWorld(
        @WebParam(name = "value")String value);

    @WebMethod(operationName="simpleHelloWorld")
    @WebResult(name="helloWorldResult")
    public String simpleHelloWorld();
}
```

Utilisation d'une interface pour définir les paramètres du Service Web

*HelloWorldService.java* du projet **HelloWorldWebService**

```
package fr.ensma.lisi.helloworldwebservice;

@WebService(endpointInterface="fr.ensma.lisi.helloworldwebservice.HelloWorldService",
serviceName="HelloWorld", portName="HelloWorldPort")
public class HelloWorldServiceImpl {

    public String makeHelloWorld(String value) {
        return "Hello World to " + value;
    }

    public String simpleHelloWorld() {
        return "Hello World to everybody";
    }
}
```

Pas nécessaire d'indiquer l'implémentation puisqu'elle est précisée dans l'annotation (vérification à l'exécution)

Classe qui fournit l'implémentation du Service Web

*HelloWorldServiceImpl.java* du projet **HelloWorldWebService**

# Développement Serveur : Bottom / Up

- Comme indiqué précédemment, la JRE 6 fournit des API et des outils pour manipuler des Services Web
- L'outil *wsgen* génère des *artifacts* (JAXB, WSDL) à partir de classes Java annotées via JAX-WS
- L'utilisation de cet outil n'est pas obligatoire puisque cette génération est implicite lors de l'exécution
- Exemples d'utilisation

```
wsgen -cp . fr.ensma.lisi.helloworldquietwebservice.HelloWorldService -keep
```

Génère les classes Java annotées JAXB  
(marshall et unmarshall)

```
wsgen -cp . fr.ensma.lisi.helloworldquietwebservice.HelloWorldService -keep -wsdl
```

Génère le document WSDL

**Nécessite la dépendance vers la bibliothèque *webservices-rt.jar***



# Développement Serveur : Bottom / Up

## ➤ Exemple : *wsgen* avec Maven 2

```
<project ...>
  ...
  <dependencies>
    <dependency>
      <groupId>com.sun.xml.ws</groupId>
      <artifactId>jaxws-rt</artifactId>
      <version>2.1.7</version>
    </dependency>
  </dependencies>
  <build><plugins><plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>jaxws-maven-plugin</artifactId>
    <executions><execution>
      <goals><goal>wsgen</goal></goals>
      <configuration>
        <sei>fr.ensma.lisi.helloworldquietwebservice.HelloWorldService</sei>
        <genWsdL>true</genWsdL>
        <keep>true</keep>
      </configuration>
    </execution></executions>
  </plugin></plugins></build>
  <repositories>
    <repository>
      <id>maven-repository.dev.java.net</id>
      <name>Java.net Repository for Maven 1</name>
      <url>http://download.java.net/maven/1/</url>
      <layout>legacy</layout>
    </repository>
    <repository>
      <id>maven2-repository.dev.java.net</id>
      <name>Java.net Repository for Maven 2</name>
      <url>http://download.java.net/maven/2/</url>
    </repository>
  </repositories>
</project>
```

Implémentation JAX-WS

Plug-in à utiliser pour la génération des artifacts

Demande génération des ressources par rapport à la classe *HelloWorldService*

Nécessaire pour le téléchargement des dépendances

*pom.xml* du projet

**HelloWorldQuietWebService**

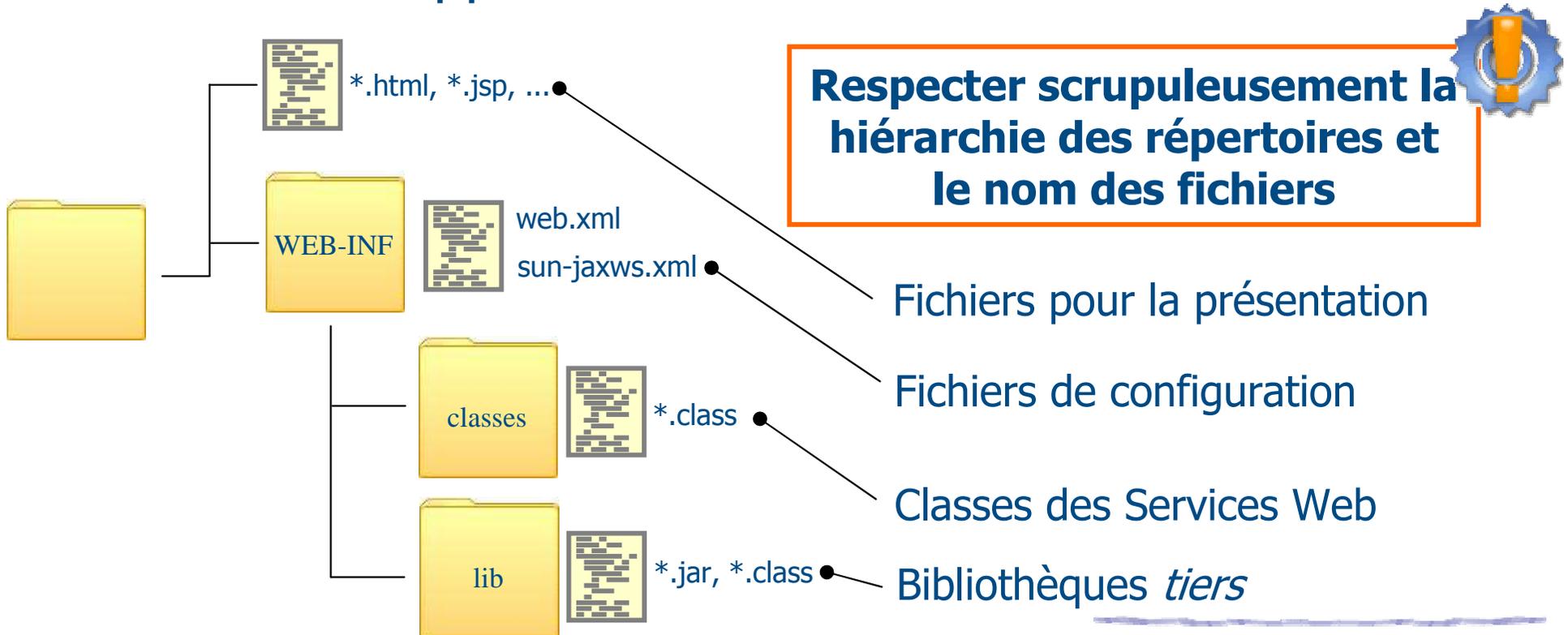
# Développement Serveur : Bottom / Up

---

- Un Service Web est déployé dans une application Web (un Service Web par application Web)
- Différentes catégories de serveur d'application pour gérer les Services Web avec JAX-WS
  - Conteneur respectant JSR 109 (Implementing Enterprise Web Services)
    - La gestion du Service Web est transparente et maintenue par le serveur d'application
    - Exemple : **Glassfish** 
    - *<http://jcp.org/en/jsr/summary?id=109>*
  - Conteneur nécessitant une gestion par Servlet
    - Nécessite une configuration explicite du Service Web
    - Exemple : **Tomcat** 
    - *Note* : un composant additionnel se propose de fournir le support JSR 109 (*<http://tomcat.apache.org/tomcat-6.0-doc/extras.html>*)

# Développement Serveur : Bottom / Up

- Dans le cas d'un conteneur dont la gestion du Service Web est gérée par une Servlet
  - *web.xml* : précise la Servlet assurant la gestion
  - *sun-jaxws.xml* : utilisé pour effectuer une relation entre le contexte de l'application et la classe du Service Web
- Structure d'une application Web fournissant un Service Web



# Développement Serveur : Bottom / Up

## ➤ Exemple : Tomcat et le fichier de configuration *web.xml*

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

<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <description>HelloWorld</description>
  <display-name>HelloWorld</display-name>
  <listener>
    <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
  </listener>
  <servlet>
    <description>JAX-WS endpoint - helloworld</description>
    <display-name>helloworld</display-name>
    <servlet-name>helloworld</servlet-name>
    <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>helloworld</servlet-name>
    <url-pattern>/helloworld</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

Servlet assurant la gestion du Service Web

La Servlet est accessible via cette URL

*web.xml* du projet  
**HelloWorldWebService**

# Développement Serveur : Bottom / Up

## ➤ Exemple : Tomcat et le fichier de configuration *sun-jaxws.xml*

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

<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime' version='2.0'>
  <endpoint
    name='helloworld'
    implementation='fr.ensma.lisi.helloworldwebservice.HelloWorldServiceImpl'
    url-pattern='/helloworld' />
</endpoints>
```

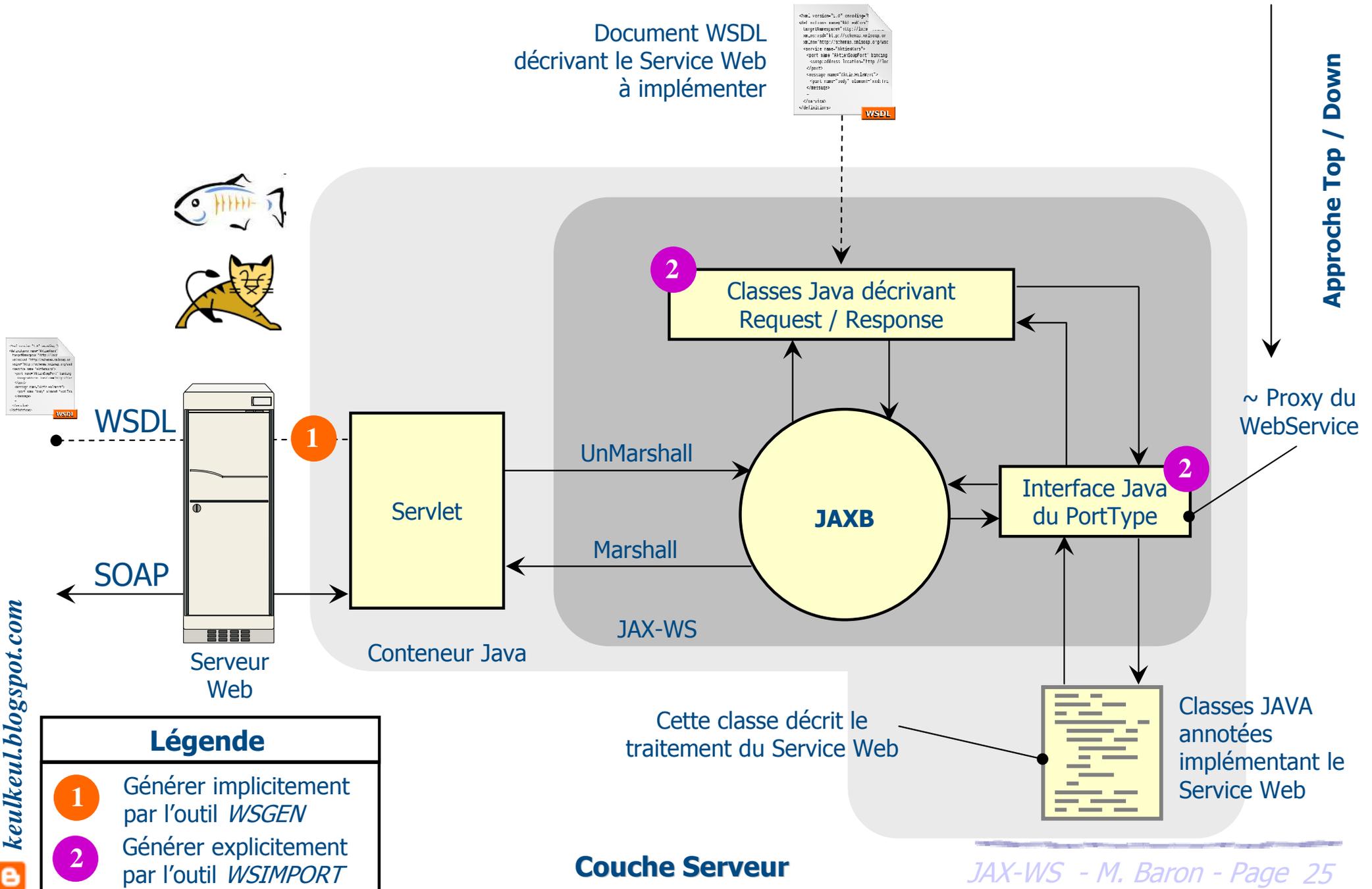
*sun-jaxws.xml* du projet  
**HelloWorldWebService**

Relation entre la classe  
implémentant le Service Web et  
la Servlet assurant sa gestion

Le fichier *sun-jaxws.xml* est également  
utilisé pour ajouter des informations  
supplémentaires (wsdl, binding, ...)



# Développement Serveur : Top / Down



# Développement Serveur : Top / Down

---

- L'approche Top / Down consiste à démarrer le développement à partir d'un document WSDL
- Le document WSDL est accessible via une URL ou via un fichier physique
- Utilisation explicite de l'outil *wsimport* pour la génération du squelette du Service Web
  - Génération des classes liées à JAXB
  - Génération des interfaces WS (interface décrivant le *PortType*)
- Création d'un POJO annotée *@WebService* en précisant l'emplacement de l'interface du *portType* (Proxy)
- Déployer l'application sur un serveur d'application
- Le reste du processus de développement est identique à celui de l'approche Bottom / Up

# Développement Serveur : Top / Down

- Exemple : Développer le Service Web *Notebook* à partir du document WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions ...>
  <types>...</types>
  <portType name="Notebook">
    <operation name="addPersonWithComplexType">
      <input message="tns:addPersonWithComplexType"/>
      <output message="tns:addPersonWithComplexTypeResponse"/>
    </operation>
    <operation name="getPersonByName">
      <input message="tns:getPersonByName"/>
      <output message="tns:getPersonByNameResponse"/>
    </operation>
    <operation name="getPersons">
      <input message="tns:getPersons"/>
      <output message="tns:getPersonsResponse"/>
    </operation>
    <operation name="addPersonWithSimpleType">
      <input message="tns:addPersonWithSimpleType"/>
    </operation>
  </portType>
  <binding name="NoteBookPortBinding" type="tns:Notebook">...</binding>
  <service name="Notebook">...</service>
</definitions>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="AktienKurs"
  targetNamespace="http://localhost:8080/NotebookWebService"
  xmlns:xs="http://schemas.xmlsoap.org/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
  >
  <service name="AktienKurs">
    <port name="AktienKursPort" binding="AktienKursPortBinding"
      soap:address location="http://localhost:8080/NotebookWebService" />
    <message name="Aktie.HoleWert">
      <part name="body" element="xsd:Text" />
    </message>
  </service>
</definitions>
```

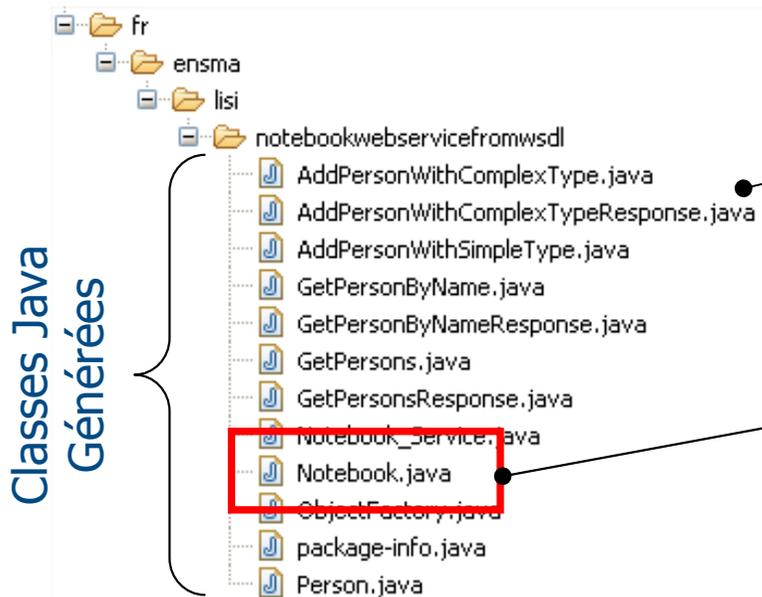
WSDL

WSDL utilisé pour générer le squelette du Service Web

Document WSDL issu du Service Web *Notebook* (Projet **NotebookWebService**)

# Développement Serveur : Top / Down

- Exemple (suite) : Développer le Service Web *Notebook* à partir du document WSDL



Pour chaque type défini dans le WSDL une classe Java (JAXB) est générée

L'interface *Notebook* définit la classe Java annotée avec JAX-WS

```
@WebService(name = "Notebook", targetNamespace = "http://notebookwebservice.lisi.ensma.fr/")
@XmlSeeAlso({ObjectFactory.class})
public interface Notebook {

    @WebMethod
    @WebResult(name = "addPersonWithComplexTypeResult", targetNamespace = "")
    @RequestWrapper(localName = "addPersonWithComplexType", targetNamespace = "http://notebookwebservice.lisi.ensma.fr/",
        className = "fr.ensma.lisi.notebookwebservicefromwsdl.AddPersonWithComplexType")
    @ResponseWrapper(localName = "addPersonWithComplexTypeResponse", targetNamespace = "http://notebookwebservice.lisi.ensma.fr/",
        className = "fr.ensma.lisi.notebookwebservicefromwsdl.AddPersonWithComplexTypeResponse")
    public boolean addPersonWithComplexType(
        @WebParam(name = "newPerson", targetNamespace = "")Person newPerson);

    ...
}
```

Interface *Notebook.java* du projet  
**NotebookWebServiceFromWSDL**

# Développement Serveur : Top / Down

## ➤ Exemple (suite) : Développer le Service Web *Notebook* à partir du document WSDL

```
@WebService(endpointInterface = "fr.ensma.lisi.notebookwebservicefromwsdl.Notebook")
public class NotebookServiceImpl {
    public boolean addPersonWithComplexType(Person newPerson) {
        ...
        return true;
    }

    public Person getPersonByName(String name) {
        Person current = new Person();
        current.setName(name);
        current.setBirthyear("1976");
        current.setAddress("17 Square Mickael BARON");
        return current;
    }

    public List<Person> getPersons() {
        Person first = new Person();
        first.setName("Ken BLOCK");
        first.setBirthyear("1967");
        first.setAddress("United States");
        Person second = new Person();
        second.setName("Colin MCRAE");
        second.setBirthyear("1968");
        second.setAddress("Scotland");

        List<Person> tabPerson = new ArrayList<Person>();
        tabPerson.add(first);
        tabPerson.add(second);
        return tabPerson;
    }

    public void addPersonWithSimpleType(String name, String address, String birthyear) {
        System.out.println("Name : " + name + " Address : " + address + " birthyear : " + birthyear);
    }
}
```

L'utilisation du mot clé implémentation n'est pas obligatoire, l'annotation se charge d'effectuer cette relation

Implémentation du traitement du Service Web (Non généré à développer)

*NotebookServiceImpl.java* du projet **NotebookWebServiceFromWSDL**

# Développement Serveur : Top / Down

## ➤ Exemple : *wsimport* avec Maven 2

```
<project ...>
  ...
  <dependencies>
    <dependency>
      <groupId>com.sun.xml.ws</groupId>
      <artifactId>jaxws-rt</artifactId>
      <version>2.1.7</version>
    </dependency>
  </dependencies>
  <build><plugins><plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>jaxws-maven-plugin</artifactId>
    <executions> <execution>
      <goals><goal>wsimport</goal></goals>
      <configuration>
        <wsdlDirectory>${basedir}/src/wsdl</wsdlDirectory>
        <genWsdL>true</genWsdL>
        <keep>true</keep>
        <packageName>fr.ensma.lisi.notebookwebservicefromwsdl</packageName>
      </configuration>
    </execution> </executions>
  </plugin></plugins></build>
  <repositories>
    <repository>
      <id>maven-repository.dev.java.net</id>
      <name>Java.net Repository for Maven 1</name>
      <url>http://download.java.net/maven/1/</url>
      <layout>legacy</layout>
    </repository>
    <repository>
      <id>maven2-repository.dev.java.net</id>
      <name>Java.net Repository for Maven 2</name>
      <url>http://download.java.net/maven/2/</url>
    </repository>
  </repositories>
</project>
```

Plug-in à utiliser pour la  
génération des classes Java

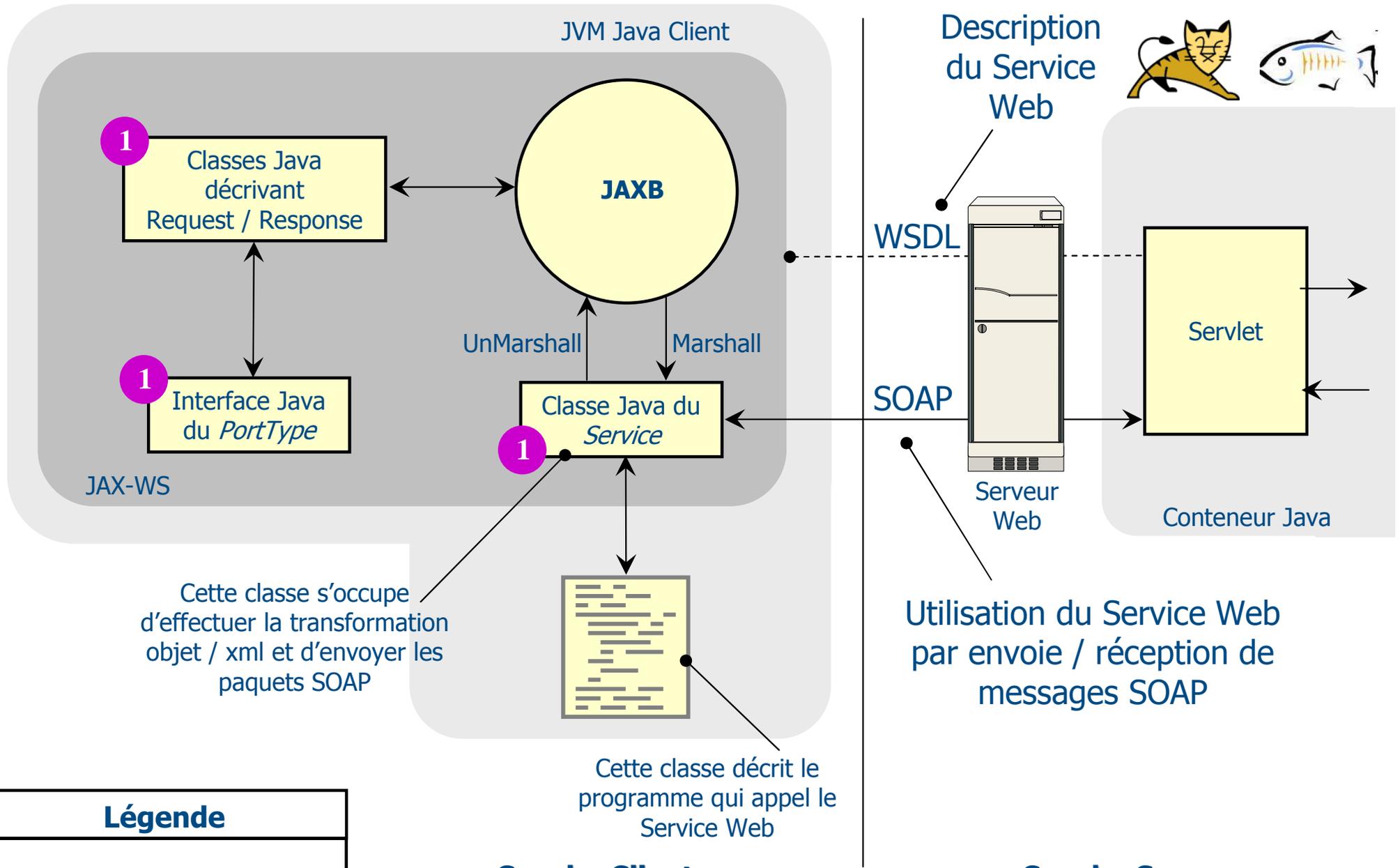
Le fichier WSDL se trouve  
physiquement dans un  
répertoire WSDL (peut se  
trouver sur le réseau  
également)

Package par défaut des  
classes générées

*pom.xml* du projet

**NotebookWebServiceFromWSDL**

# Développement Client Java



**Légende**

1 Générer explicitement par l'outil *WSIMPORT*

## Couche Cliente

## Couche Serveur

# Développement Client Java

---

- Le développement du client consiste à appeler des opérations du Service Web à partir d'un programme Java
- Le client peut être une application développée
  - Java SE (Swing, Eclipse RCP)
  - Java EE avec les EJB (JSP, Servlet, ...) (voir section EJB)
- Possibilité de générer des appels aux Services Web de manière synchrone et asynchrone
- Le développeur ne manipule que du code Java, le code XML est caché (JAXB)

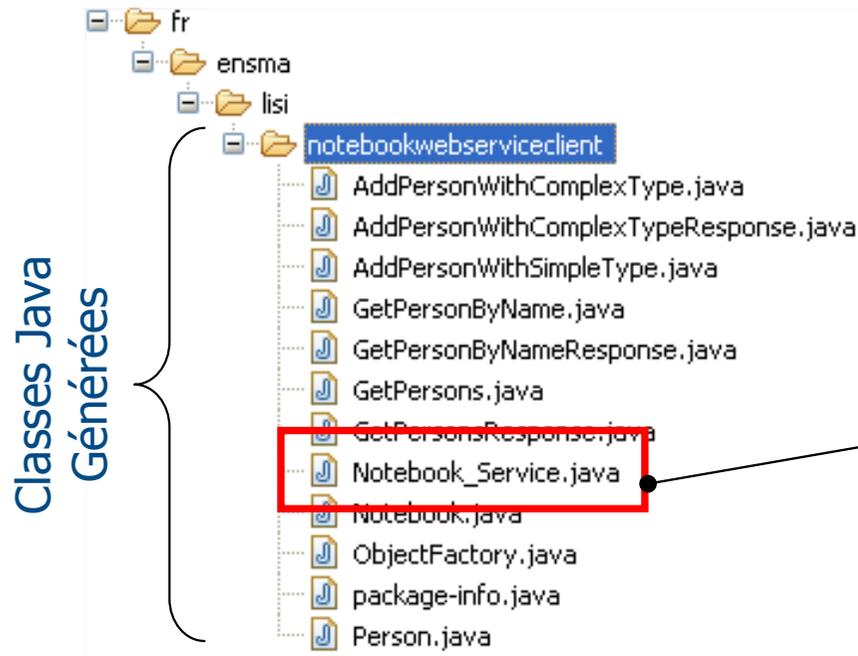
# Développement Client Java

---

- Le développement du client suit une procédure similaire à l'approche Top / Down où le point de départ est le document WSDL (via une URL ou via un fichier physique)
- Utilisation explicite de l'outil *wsimport* pour la génération du squelette du Service Web
  - Génération des classes liées à JAXB
  - Génération de classes Service Web (*PortType* et *Service*)
- Création d'une instance de la classe *Service*
- Récupération d'un port via *get<ServiceName>Port()*
- Invocation des opérations

# Développement Client Java

- Exemple : Développer un client Java (Synchrone) pour le Service Web *Notebook*



Pour chaque type défini dans le WSDL une classe Java (JAXB) est générée

La classe *Service* s'occupe de gérer les appels distants au Service Web

Le résultat de la génération est identique à la génération effectuée pour l'approche Top / Down



# Développement Client Java

- Exemple (suite) : Développer un client Java (Synchrone) pour le Service Web *Notebook*

Création d'une instance de la classe *Service*

Récupération d'un port via *getNoteBookPort()*

```
public class NotebookClient {
    public static void main(String[] args) {
        Notebook_Service notebook_Service = new Notebook_Service();
        Notebook noteBookPort = notebook_Service.getNoteBookPort();

        Person refPerson = new Person();
        refPerson.setName("Baron Mickael");
        refPerson.setAddress("Poitiers");
        refPerson.setBirthyear("1976");

        boolean addPersonWithComplexType = noteBookPort.addPersonWithComplexType(refPerson);

        System.out.println(addPersonWithComplexType);
    }
}
```

Appel de l'opération du Service Web

*NotebookClient.java* du projet **NotebookWebServiceClient**

# Développement Client Java

## ➤ Exemple (suite) : Développer un client Java (Synchrone) pour le Service Web *Notebook*

```
@WebService(endpointInterface = "fr.ensma.lisi.notebookwebservicefromwsdl.Notebook")
public class NotebookServiceImpl {
    public boolean addPersonWithComplexType(Person newPerson) {
        System.out.println("Starting process ...");

        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Name : " + newPerson.getName() + " Address : " + newPerson.getAddress() +
            " birthyear : " + newPerson.getBirthyear());

        return true;
    }
    ...
}
```

Le traitement du Service Web peut prendre un certain temps



**Dans le cas où le temps de réponse d'un Service Web est particulièrement long, prévoir un appel asynchrone**

*NotebookServiceImpl.java* du projet **NotebookWebService**

# Développement Client Java

---

- Si le temps de réponse de l'appel à une opération d'un Web Service est long, prévoir un appel asynchrone
- JAX-WS permet d'appeler des Services Web en mode asynchrone si l'information est précisée dans le binding
- Lors de la génération des fichiers classes (via *wsimport*) fournir un fichier binding suivant

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bindings wsdlLocation="http://localhost:8080/NotebookWebService/notebook?wsdl"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <enableAsyncMapping>true</enableAsyncMapping>
</bindings>
```

# Développement Client Java

## ➤ Exemple : Développer un client Java (Asynchrone) pour le Service Web *Notebook*

```
public class NotebookAsyncClient {
    public static void main(String[] args) {
        Notebook_Service notebook_Service = new Notebook_Service();
        Notebook noteBookPort = notebook_Service.getNoteBookPort();

        Person refPerson = new Person();
        refPerson.setName("Baron Mickael");
        refPerson.setAddress("Poitiers");
        refPerson.setBirthyear("1976");

        noteBookPort.addPersonWithComplexTypeAsync(refPerson, new AsyncHandler<AddPersonWithComplexTypeResponse>() {
            public void handleResponse(Response<AddPersonWithComplexTypeResponse> res) {
                if (!res.isCancelled() && res.isDone()) {
                    try {
                        AddPersonWithComplexTypeResponse value = res.get();
                        System.out.println(value.isAddPersonWithComplexTypeResult());
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        try {
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Terminé");
    }
}
```

Des méthodes spécifiques au mode asynchrone sont générées

Déclenché quand le traitement de l'opération sera terminé

Pour éviter de terminer le programme

## Annotations : généralités

---

- JAX-WS repose sur l'utilisation massive d'annotations pour la configuration d'un Service Web
- JSR utilisées (JSR 224, 222, 181 et 250)
- Les principales annotations sont les suivantes
  - *@WebService* : POJO implémentant un Service Web
  - *@WebMethod* : Paramétrer une opération
  - *@WebParam* : Paramétrer un message
  - *@WebResult* : Paramétrer un message de sortie
  - *@WebFault* : Paramétrer un message fault
- A noter que seule l'utilisation de l'annotation *@WebService* est nécessaire (utilisation de valeurs par défaut)
- Nous détaillerons chacune des annotations de manière à découvrir les paramètres disponibles

## Annotations : @WebService

---

- Annote une classe Java pour définir l'implémentation du Service Web
- Annote une interface Java pour définir la description du Service Web
- Attributs de l'annotation *@WebService*
  - *String name* : nom du Service Web
  - *String endpointInterface* : nom de l'interface décrivant le Service Web
  - *String portName* : nom du port
  - *String serviceName* : nom du service du Service Web
  - *String targetNamespace* : le namespace du Service Web
  - *String wsdlLocation* : l'emplacement du WSDL décrivant le Service Web

## Annotations : @WebMethod

---

- Annote une méthode d'une classe Java exposée comme une opération du Service Web
- Attributs de l'annotation : *@WebMethod*
  - *String action* : l'action de l'opération. Dans le cas d'un binding SOAP, cela détermine la valeur de l'action SOAP
  - *boolean exclude* : précise que la méthode ne doit pas être exposée comme une opération. Ne pas utiliser dans une interface Java
  - *String operationName* : précise le nom de l'attribut *name* défini dans l'élément operation du document WSDL

## Annotations : @WebParam

---

- Décrit la relation entre un paramètre d'entrée d'une méthode et un message part d'une opération
- Attributs de l'annotation
  - *boolean header* : précise si le paramètre doit être transmis dans l'en-tête du message (*true*) ou dans le corps (*false*)
  - *WebParam.Mode mode* : précise le type d'accès au paramètre (*IN*, *OUT* ou *INOUT*)
  - *String name* : nom du paramètre
  - *String partName* : le nom du *wsdl:part* représentant ce paramètre
  - *String targetNamespace* : l'espace de nommage de ce paramètre

## Annotations : @WebResult

---

- Décrit la relation entre le paramètre de sortie d'une méthode et un message part d'une opération
- Attributs de l'annotation
  - *boolean header* : précise si le paramètre de sortie doit être transmis dans l'en-tête du message (*true*) ou dans le corps (*false*)
  - *String name* : nom du paramètre de sortie
  - *String partName* : le nom du *wsdl:part* représentant ce paramètre de sortie
  - *String targetNamespace* : l'espace de nommage de ce paramètre de sortie

# Handler : généralités

---

- Les « handlers » sont des intercepteurs permettant de réaliser des traitements lors de la réception et l'émission de messages
  - Lors de la réception ils sont déclenchés avant l'appel à une opération
  - Lors de l'émission ils sont déclenchés après l'appel à une opération
- Un « handler » est disponible dans la couche JAX-WS et par conséquent autant sur la partie cliente que celle du serveur
- Quand l'utiliser ?
  - Pour filtrer les appels aux opérations d'un Service Web
  - Pour l'écriture des logs
- Deux types de « handlers »
  - Handlers liés au protocole de transport (ex : SOAP)
  - Handlers liés au contenu transféré appelé *logical handlers* qui est indépendant du protocole

# Handler : généralités

---

- JAX-WS définit l'interface *Handler* contenant les principales méthodes
  - *boolean handleMessage(C context)* : invoquée lors des messages entrants et sortants. Si false est retourné le processus est arrêté
  - *boolean handleFault(C context)* : invoquée lors des messages en erreur (fault)
- Le type générique *C* hérite de *MessageContext* qui est une *Map* contenant un ensemble de propriétés
  - *MESSAGE\_OUTBOUND\_PROPERTY* : pour savoir s'il s'agit de messages entrants ou sortants
  - *HTTP\_REQUEST\_HEADERS* : pour récupérer l'en-tête HTTP de la requête
  - *WSDL\_OPERATION* : nom de l'opération WSDL
  - *WSDL\_SERVICE* : nom du service WSDL



# Handler : généralités

- Deux sous interfaces à *handler* sont proposées pour décrire chaque type de « handler »
- ***SOAPHandler*** a un accès sur l'intégralité du message SOAP incluant les en-têtes
- ***LogicalHandler*** pour une indépendance du protocole de transport et un accès sur le contenu du message



# Handler : côté Serveur

---

- Développement d'une classe Java qui implémente soit l'interface *SOAPHandler* soit *LogicalHandler*
- Définir un fichier de correspondance (*handlers.xml*) qui précise les classes implémentant les handlers

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <handler>
      <handler-name>NOM DU HANDLER</handler-name>
      <handler-class>CLASSE DU HANDLER</handler-class>
    </handler>
  </handler-chain>
</handler-chains>
```

- Ajouter l'annotation *@HandlerChain* sur le POJO du Web Service

# Handler : côté Serveur

- Exemple : Ajouter un *handler* de type *SOAPHandler* sur le Service Web *Notebook*

```
public class SOAPLoggingHandler implements SOAPHandler<SOAPMessageContext> {  
  
    private static PrintStream out = System.out;  
  
    public Set<QName> getHeaders() { return null; }  
    public void close(MessageContext context) { }  
  
    public boolean handleMessage(SOAPMessageContext smc) {  
        logToSystemOut(smc);  
        return true;  
    }  
    public boolean handleFault(SOAPMessageContext smc) {  
        logToSystemOut(smc);  
        return true;  
    }  
  
    private void logToSystemOut(SOAPMessageContext smc) {  
        Boolean outboundProperty = (Boolean) smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);  
        if (outboundProperty.booleanValue()) {  
            out.println("\nOutgoing message from web service provider:");  
        } else {  
            out.println("\nIncoming message to web service provider:");  
        }  
        SOAPMessage message = smc.getMessage();  
        try {  
            message.writeTo(out);  
            out.println("");  
        } catch (Exception e) {  
            out.println("Exception in handler: " + e);  
        }  
    }  
}
```

*SOAPLoggingHandler.java* du projet  
**NotebookWebServiceWithSOAPHandler**

# Handler : côté Serveur

- Exemple (suite) : Ajouter un *handler* de type *SOAPHandler* sur le Service Web *Notebook*

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <handler>
      <handler-name>fr.ensma.lisi.notebookwebservicewithsoaphandler.SOAPLoggingHandler</handler-name>
      <handler-class>fr.ensma.lisi.notebookwebservicewithsoaphandler.SOAPLoggingHandler</handler-class>
    </handler>
  </handler-chain>
</handler-chains>
```

*handlers.xml* du projet  
**NotebookWebServiceWithSOAPHandler**

```
@WebService(endpointInterface = "fr.ensma.lisi.notebookwebservicefromwsdl.Notebook")
@HandlerChain(file = "handlers.xml")
public class NotebookServiceImpl {
    public boolean addPersonWithComplexType(Person newPerson) {
        ...
    }

    public Person getPersonByName(String name) {
        ...
    }

    public List<Person> getPersons() {
        ...
    }

    public void addPersonWithSimpleType(String name, String address, String birthyear) {
        ...
    }
}
```

*NotebookServiceImpl.java* du projet  
**NotebookWebServiceWithSOAPHandler**

# Handler : côté Serveur

- Exemple (suite) : Ajouter un *handler* de type *SOAPHandler* sur le Service Web *Notebook*

Les messages entrants et sortants sont affichés avec la structure de l'enveloppe SOAP

```
Tomcat
Incoming message to web service provider:
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:notebookwebservice="http://notebookwebservice.lisi.ensma.fr/"><soapenv:Header/><soapenv:Body><not:addPersonWithComplexType>
  <newPerson>
    <address>Poitiers</address>
    <birthyear>1976</birthyear>
    <name>BARON Mickael</name>
  </newPerson>
</not:addPersonWithComplexType></soapenv:Body></soapenv:Envelope>
Starting process ...
Name : BARON Mickael Address : Poitiers birthyear : 1976
Outgoing message from web service provider:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:addPersonWithComplexTypeResponse xmlns:ns2="http://notebookwebservice.lisi.ensma.fr/"><addPersonWithComplexTypeResult>true</addPersonWithComplexTypeResult></ns2:addPersonWithComplexTypeResponse></S:Body></S:Envelope>
```

# Handler : côté Client

- Exemple : Ajouter un *handler* de type *SOAPHandler* sur le client du Service Web *Notebook*

```
public class SOAPLoggingHandler implements SOAPHandler<SOAPMessageContext> {  
  
    private static PrintStream out = System.out;  
  
    public Set<QName> getHeaders() { return null; }  
    public void close(MessageContext context) { }  
  
    public boolean handleMessage(SOAPMessageContext smc) {  
        logToSystemOut(smc);  
        return true;  
    }  
  
    public boolean handleFault(SOAPMessageContext smc) {  
        logToSystemOut(smc);  
        return true;  
    }  
  
    private void logToSystemOut(SOAPMessageContext smc) {  
        Boolean outboundProperty = (Boolean) smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);  
        if (outboundProperty.booleanValue()) {  
            out.println("\nOutgoing message from web service client:");  
        } else {  
            out.println("\nIncoming message to web service client:");  
        }  
        SOAPMessage message = smc.getMessage();  
        try {  
            message.writeTo(out);  
            out.println("");  
        } catch (Exception e) {  
            out.println("Exception in handler: " + e);  
        }  
    }  
}
```

*SOAPLoggingHandler.java* du projet  
**NotebookWebServiceClientWithSOAPHandler**

# Handler : côté Client

- Exemple (suite) : Ajouter un *handler* de type *SOAPHandler* sur le client du Service Web *Notebook*

```

public class NotebookClient {
    public static void main(String[] args) {
        Notebook_Service notebook_Service = new Notebook_Service();
        Notebook noteBookPort = notebook_Service.getNoteBookPort();

        List<Handler> myHandler = new ArrayList<Handler>();
        myHandler.add(new SOAPLoggingHandler());

        ((BindingProvider)noteBookPort).getBinding().setHandlerChain(myHandler);

        Person refPerson = new Person();
        refPerson.setName("Baron Mickael");
        refPerson.setAddress("Poitiers");
        refPerson.setBirthyear("1976");
        boolean addPersonWithComplexType = noteBookPort.addPersonWithComplexType(refPerson);
        System.out.println(addPersonWithComplexType);
    }
}

```

*NotebookClient.java* du projet

## NotebookWebServiceClientWithSOAPHandler

```

<terminated> NotebookClient (1) [Java Application] C:\Program Files\Java\jdk1.6.0_12\bin\javaw.exe (1 janv. 2010 20:50:51)

Outgoing message from web service client:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:addPersonWithComplexType >

Incoming message to web service client:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Header/><S:Body><ns2:addPersonWithCo
true

```

# Service Web avec Java 6 : généralités

---

- Précédemment nous avons vu comment développer un client pour appeler un Service Web
- Le développement serveur est possible directement à partir de Java SE 6 puisqu'il intègre un serveur Web embarqué
  - Inutile de gérer un serveur Web (Tomcat, Glassfish)
  - Le déploiement est immédiat
- Les deux approches (*Top / Down* et *Bottom / Up*) sont applicables à ce mode de développement
- Usages
  - Fournir des Services Web à une application type client lourd
  - Pour les tests unitaires, fournir des Mock de Services Web
- A noter que le développement du client reste similaire à ce qui a été présenté précédemment

## Service Web avec Java 6 : serveur

---

- Développer une classe Java implémentant le Service Web (ou à partir d'un WSDL et en utilisant wsimport)
- Ajouter l'annotation *@WebService*
- Créer explicitement une instance du POJO
- Publier le Service Web par l'intermédiaire de la méthode

```
Endpoint Endpoint.publish(String adresse, Object implementor)
```

- Le document WSDL est généré automatiquement à l'exécution de l'application
- URL du WSDL : *http://monserveur/service?WSDL*
- Toutes les méthodes du POJO sont des opérations du Web Service

## Service Web avec Java 6 : serveur

---

- La méthode *publish* est utilisée pour démarrer la publication du Service Web
  - *String adresse* : adresse de déploiement du Service Web depuis le serveur Web embarqué (*http://localhost:8080/ws*)
  - *Object implementor* : instance de l'implémentation du Service Web
- Différentes méthodes proposées par la classe *Endpoint* (type de retour de la méthode *publish*)
  - *boolean isPublished()* : vérifie si le service est en publication
  - *void stop()* : arrête la publication du Service Web

# Service Web avec Java 6 : serveur

## ➤ Exemple : Publier le Service Web Notebook à partir de Java SE 6

```
@WebService(name="Notebook",targetNamespace="http://notebookwebservice.lisi.ensma.fr/")
@SOAPBinding(style=Style.DOCUMENT,use=Use.LITERAL)
public interface NotebookService {
    @WebMethod(operationName="addPersonWithComplexType")
    @WebResult(name="addPersonWithComplexTypeResult")
    public boolean addPersonWithComplexType(
        @WebParam(name = "newPerson")Person newPerson);
    ...
}
```

Interface *NotebookService.java* du projet  
**NotebookWebServiceFromJavaSE**

```
@WebService(endpointInterface="fr.ensma.lisi.notebookwebservicefromjavase.NotebookService",
    serviceName="Notebook",
    portName="NoteBookPort")
public class NotebookServiceImpl implements NotebookService {
    @Override
    public boolean addPersonWithComplexType(Person newPerson) { ... }
    @Override
    public Person getPersonByName(String name) { ... }
    @Override
    public Person[] getPersons() { ... }
    @Override
    public void addPersonWithSimpleType(String name, String address, String birthyear) { ... }
}
```

Classe *NotebookServiceImpl.java* du projet  
**NotebookWebServiceFromJavaSE**

# Service Web avec Java 6 : serveur

## ► Exemple (suite) : Publier le Service Web Notebook à partir de Java SE 6

```
public class NotebookServicePublish extends JFrame {
    private Endpoint publish;
    public NotebookServicePublish() {
        final JButton startPublish = new JButton();
        startPublish.setText("Start Publish");
        final JButton stopPublish = new JButton();
        stopPublish.setText("Stop Publish");

        startPublish.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                NotebookService current = new NotebookServiceImpl();
                publish = Endpoint.publish("http://localhost:8080/notebook", current);
                startPublish.setEnabled(false);
                stopPublish.setEnabled(true);
            }
        });

        stopPublish.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                publish.stop();
                stopPublish.setEnabled(false);
                startPublish.setEnabled(true);
            }
        });

        ...
    }
    public static void main(String[] args) {
        new NotebookServicePublish();
    }
}
```



Utilisée pour publier le Service Web

Utilisée arrêter la publication du Service Web

Classe *NotebookServicePublish.java* du projet

**NotebookWebServiceFromJavaSE**

# Service Web avec les EJB : généralités

---

- A partir d'un EJB Session, il est possible de définir le POJO associé comme étant un Service Web
- Pour rappel, l'appel à un EJB Session passe obligatoirement par un client écrit en Java
- Les avantages à transformer en EJB Session en Service Web
  - Caractéristiques des EJBs (transactions, sécurité, scalabilité, ...)
  - Plus grande hétérogénéité, le client n'est pas forcément écrit en Java
  - Réutilisabilité du code
  - Modularité (avec les annotations JAX-WS possibilité de masquer les méthodes qui ne doivent pas être découvertes)
- Nécessite d'avoir un conteneur EJB pour le déploiement
  - Glassfish, OpenEJB, ...

## Service Web avec les EJB : serveur

---

- Partir d'une classe Java définissant un EJB Session (*statefull* ou *stateless*)
- Ajouter l'annotation *@WebService*
- Déployer l'application sur un serveur d'application ayant le support EJB (Glassfish par exemple)
- Le conteneur EJB s'occupe de la gestion du Service Web, aucune Servlet n'est nécessaire
- Le document WSDL est généré automatiquement en respectant les valeurs par défaut
  - URL du WSDL : *http://monserveur/app/Service?WSDL*
- Toutes les méthodes de l'EJB sont par défaut des opérations du Service Web

# Service Web avec les EJB : serveur

## ➤ Exemple : Définir un EJB Stateless comme étant un Web Service

Seule l'annotation *@Stateless* a été ajoutée

```
@Stateless
@WebService(endpointInterface = "fr.ensma.lisi.notebookwebservicefromejb.NotebookService")
public class NotebookServiceImpl {
    public boolean addPersonWithComplexType(Person newPerson) {
        ...
        return true;
    }

    public Person getPersonByName(String name) {
        Person current = new Person();
        current.setName(name);
        current.setBirthyear("1976");
        current.setAddress("17 Square Mickael BARON");
        return current;
    }

    public List<Person> getPersons() {
        Person first = new Person();
        ...
        Person second = new Person();
        ...

        List<Person> tabPerson = new ArrayList<Person>();
        tabPerson.add(first);
        tabPerson.add(second);
        return tabPerson;
    }

    public void addPersonWithSimpleType(String name, String address, String birthyear) {
        System.out.println("Name : " + name + " Address : " + address + " birthyear : " + birthyear);
    }
}
```

Le reste du code est le même que dans les exemples précédents

Classe *NotebookServiceImpl.java* du projet  
**NotebookWebServiceFromEJB**

## Service Web avec les EJB : client

---

- Le développement du client est similaire au client développé précédemment où le point de départ est le document WSDL (via une URL ou via un fichier physique)
- Utilisation explicite de l'outil *wsimport* pour la génération du squelette du Service Web
  - Génération des classes liées à JAXB
  - Génération de classes Service Web (*PortType* et *Service*)
- Injecter un *@WebServiceReference* pour récupérer une instance du Service à partir du conteneur EJB
- Récupération d'un port via *get<ServiceName>Port()*
- Invocation des opérations

# Service Web avec les EJB : client

## ➤ Exemple : appel d'un Service Web à partir d'une Servlet

```

public class NotebookWebServiceFromEJBClientServlet extends HttpServlet {

    @WebServiceRef(wsdlLocation = "http://localhost:8080/NotebookWebServiceFromEJB/Notebook?wsdl")
    private Notebook_Service service;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet NotebookWebServiceFromEJBClientServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet NotebookWebServiceFromEJBClientServlet at " + request.getContextPath() + "</h1>");

            try {
                Notebook port = service.getNoteBookPort();
                Person newPerson = new Person();
                newPerson.setName("BARON Mickael");
                newPerson.setAddress("Poitiers");
                newPerson.setBirthyear("1976");
                boolean result = port.addPersonWithComplexType(newPerson);
                out.println("<div>Result = " + result + "</div>");
            } catch (Exception ex) {
                ex.printStackTrace();
            }
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
}

```

La référence à  
*Notebook\_Service* est donnée  
par le conteneur EJB

Classe *NotebookWebServiceFromEJBClientServlet*  
du projet **NotebookWebServiceFromEJBClient**

## JAX-WS « in actions » : Services Web eBay

---

- eBay fournit via son programme *eBay developers program* un accès à ses services via des Services Web
  - <http://developer.ebay.com/>
  - <http://developer.ebay.com/products/finding/>
  - <http://developer.ebay.com/products/shopping/>
  - <http://developer.ebay.com/products/best-match-item-details/>
- L'utilisation de ces Services Web est soumis à une inscription pour obtenir une clé d'autorisation
  - <https://developer.ebay.com/DevZone/account/>
- Nous montrons à partir de JAX-WS comment générer un client d'accès au service de recherche d'eBay
- Le code du client est fortement inspiré de
  - [http://ebay.custhelp.com/cgi-bin/ebay.cfg/php/enduser/std\\_adp.php?p\\_faqid=1484](http://ebay.custhelp.com/cgi-bin/ebay.cfg/php/enduser/std_adp.php?p_faqid=1484)

# JAX-WS « in actions » : Services Web eBay

---

- Utilisation de l'outil *wsimport*
  - <http://developer.ebay.com/webservices/finding/latest/FindingService.wsdl>
  - Utilisation du paramètre *extension* de l'outil *wsimport*
- Génération des artifacts du Service Web
  - Génération des classes liées à JAXB (51 classes)
  - Génération de classes Service Web (2 classes)
- Création d'une instance de la classe *FindingService*
- Récupération d'un port *getFindingServiceSOAP12PortHttp()*
- Renseigner dans l'en-tête HTTP les informations de protocole et d'autorisation
- Préparer le paramètre de l'opération *findItemsAdvanced*
- Invocation de l'opération

# JAX-WS « in actions » : Services Web eBay

## ➤ Exemple : création d'un client pour les Services Web eBay

```

public class EBAYFindingserviceClient {
    public static void main (String[] args) {
        String strBaseURL = "http://svcs.ebay.com/services/search/FindingService/v1";
        FindingService service = new FindingService();
        FindingServicePortType port = service.getFindingServiceSOAP12PortHttp();

        BindingProvider bp = (BindingProvider) port;
        List<Handler> myHandler = new ArrayList<Handler>();
        myHandler.add(new SOAPLoggingHandler());

        bp.getBinding().setHandlerChain(myHandler);

        Map<String, Object> requestProperties = bp.getRequestContext();
        Map<String, List<String>> httpHeaders = new HashMap<String, List<String>>();
        // Set the headers
        httpHeaders.put("X-EBAY-SOA-MESSAGE-PROTOCOL", Collections.singletonList("SOAP12"));
        httpHeaders.put("X-EBAY-SOA-OPERATION-NAME", Collections.singletonList("findItemsAdvanced"));

        // Edit the following line to insert your AppID to set the X-EBAY-SOA-SECURITY-APPNAME correctly
        httpHeaders.put("X-EBAY-SOA-SECURITY-APPNAME", Collections.singletonList(KEY_PERSONAL));

        requestProperties.put(MessageContext.HTTP_REQUEST_HEADERS, httpHeaders);
        requestProperties.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, strBaseURL);

        ... Suite dans le prochain transparent
    }
}

```

Création et paramétrage du Service Web

Fournit à partir du compte personnel

Classe *EBAYFindingserviceClient* du projet  
**EBAYFindingServiceClient**

# JAX-WS « in actions » : Services Web eBay

## ➤ Exemple (suite) : création d'un client pour Service Web eBay

```
public class EBAYFindingserviceClient {
    public static void main (String[] args) {
        ...
        FindItemsAdvancedRequest req = new FindItemsAdvancedRequest();
        List<OutputSelectorType> opSelector = req.getOutputSelector();
        opSelector.add(OutputSelectorType.SELLER_INFO);

        ItemFilter objFilter1 = new ItemFilter();
        objFilter1.setName(ItemFilterType.AVAILABLE_TO);objFilter1.getValue().add("US");
        ItemFilter objFilter2 = new ItemFilter();
        objFilter2.setName(ItemFilterType.LISTING_TYPE);objFilter2.getValue().add("All");
        ItemFilter objFilter3 = new ItemFilter();
        objFilter3.setName(ItemFilterType.HIDE_DUPLICATE_ITEMS);objFilter3.getValue().add("true");

        List<ItemFilter> itemFilter = req.getItemFilter();
        itemFilter.add(objFilter1);
        itemFilter.add(objFilter2);
        itemFilter.add(objFilter3);

        List<String> catID = req.getCategoryId();
        catID.add("279");
        req.setSortOrder(SortOrderType.END_TIME_SOONEST);
        req.setKeywords("Harry Potter");

        // Suite dans le prochain transparent
    }
}
```

Création du paramètre à passer à l'opération *findItemsAdvanced(...)*

Recherche des produits en relation avec *Harry Potter*

Classe *EBAYFindingserviceClient* du projet **EBAYFindingServiceClient**

# JAX-WS « in actions » : Services Web eBay

## ➤ Exemple (suite) : création d'un client pour Service Web eBay

```
public class EBAYFindingserviceClient {
    public static void main (String[] args) {
        ...

        FindItemsAdvancedResponse res = port.findItemsAdvanced(req);
        SearchResult searchResult = res.getSearchResult();
        List<SearchItem> item = searchResult.getItem();
        for (SearchItem searchItem : item) {
            System.out.println(searchItem.getGalleryURL());
        }
        System.out.println("Search Count: " + searchResult.getCount());
    }
}
```

Un appel synchrone à l'opération *findItemsAdvanced(...)*

Classe *EBAYFindingserviceClient* du projet **EBAYFindingServiceClient**

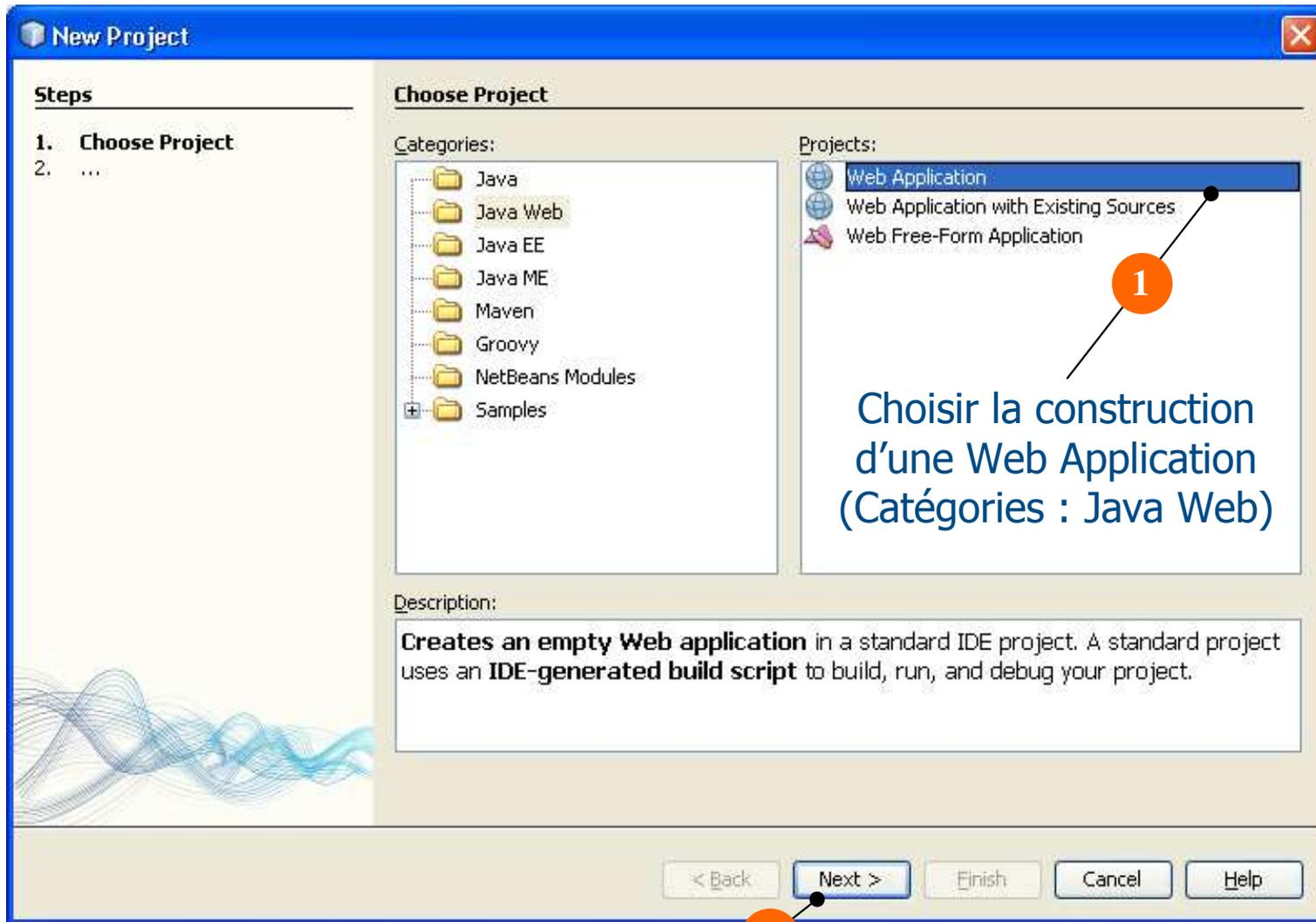
```
<terminated> EBAYFindingserviceClient [Java Application] C:\Program Files\Java\jdk1.6.0_12\bin\javaw.exe (4
http://thumbs2.ebaystatic.com/pict/1303499688938080_1.jpg
http://thumbs2.ebaystatic.com/pict/1804509681898080_1.jpg
http://thumbs1.ebaystatic.com/pict/3500793015888080_18.jpg
http://thumbs4.ebaystatic.com/pict/2705084981278080_1.jpg
http://thumbs3.ebaystatic.com/pict/1303552690948080_1.jpg
http://thumbs1.ebaystatic.com/pict/2205222870088080_1.jpg
http://thumbs3.ebaystatic.com/pict/1603847116988080_1.jpg
http://thumbs1.ebaystatic.com/pict/3204683036928080_1.jpg
http://thumbs2.ebaystatic.com/pict/1403705628818080_1.jpg
http://thumbs1.ebaystatic.com/pict/3303661137808080_3.jpg
http://thumbs4.ebaystatic.com/pict/3702712268118080_3.jpg
http://thumbs2.ebaystatic.com/pict/2505570634178080_1.jpg
http://thumbs3.ebaystatic.com/pict/3101868596668080_1.jpg
http://thumbs4.ebaystatic.com/pict/2602864311878080_16.jpg
```

Une liste de liens d'aperçu d'image est générée par rapport à la recherche effectuée



# Service Web avec les EJB : client

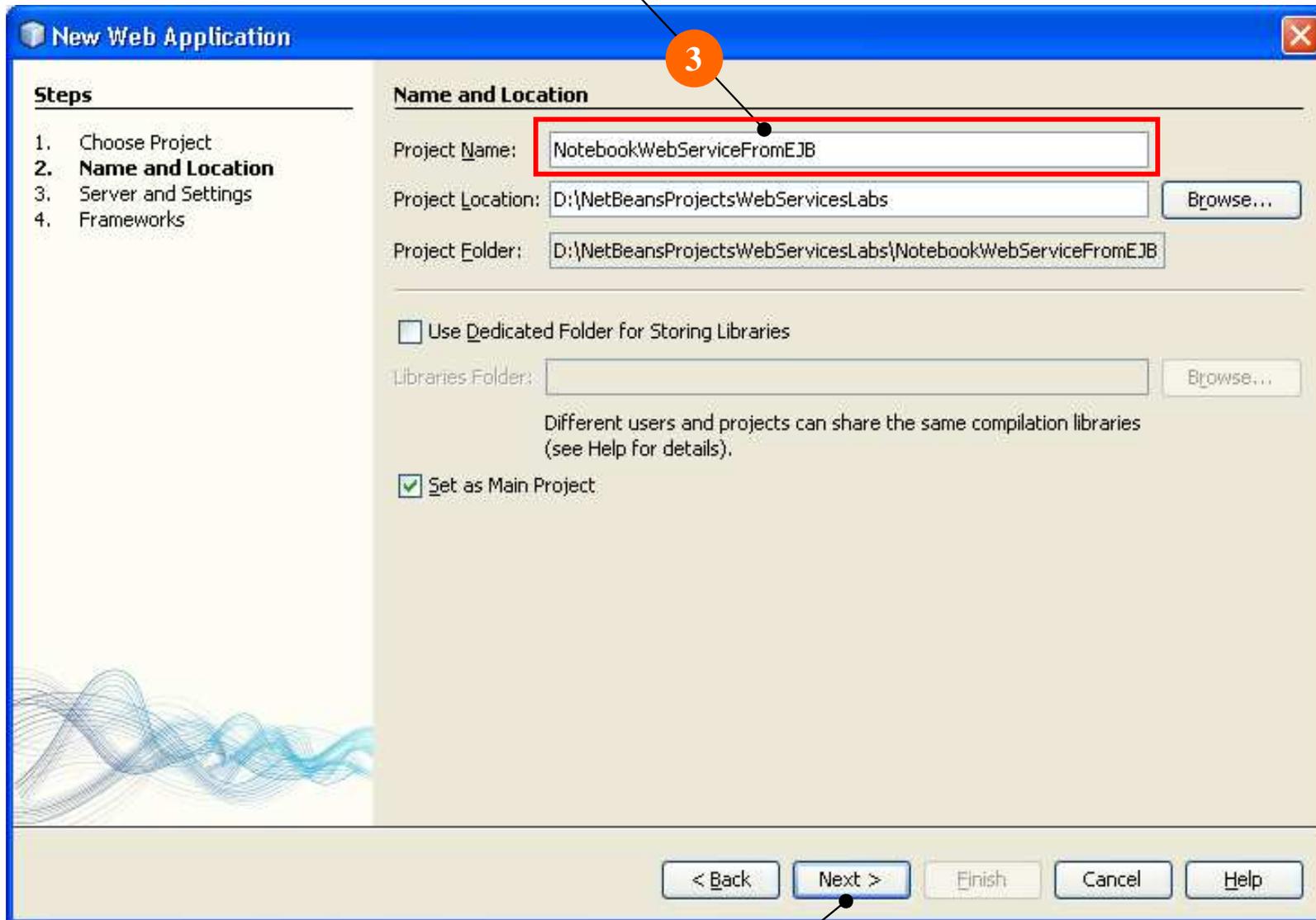
- Démarrer la création d'un nouveau Projet (File -> New Project ...)



Faire *Next*

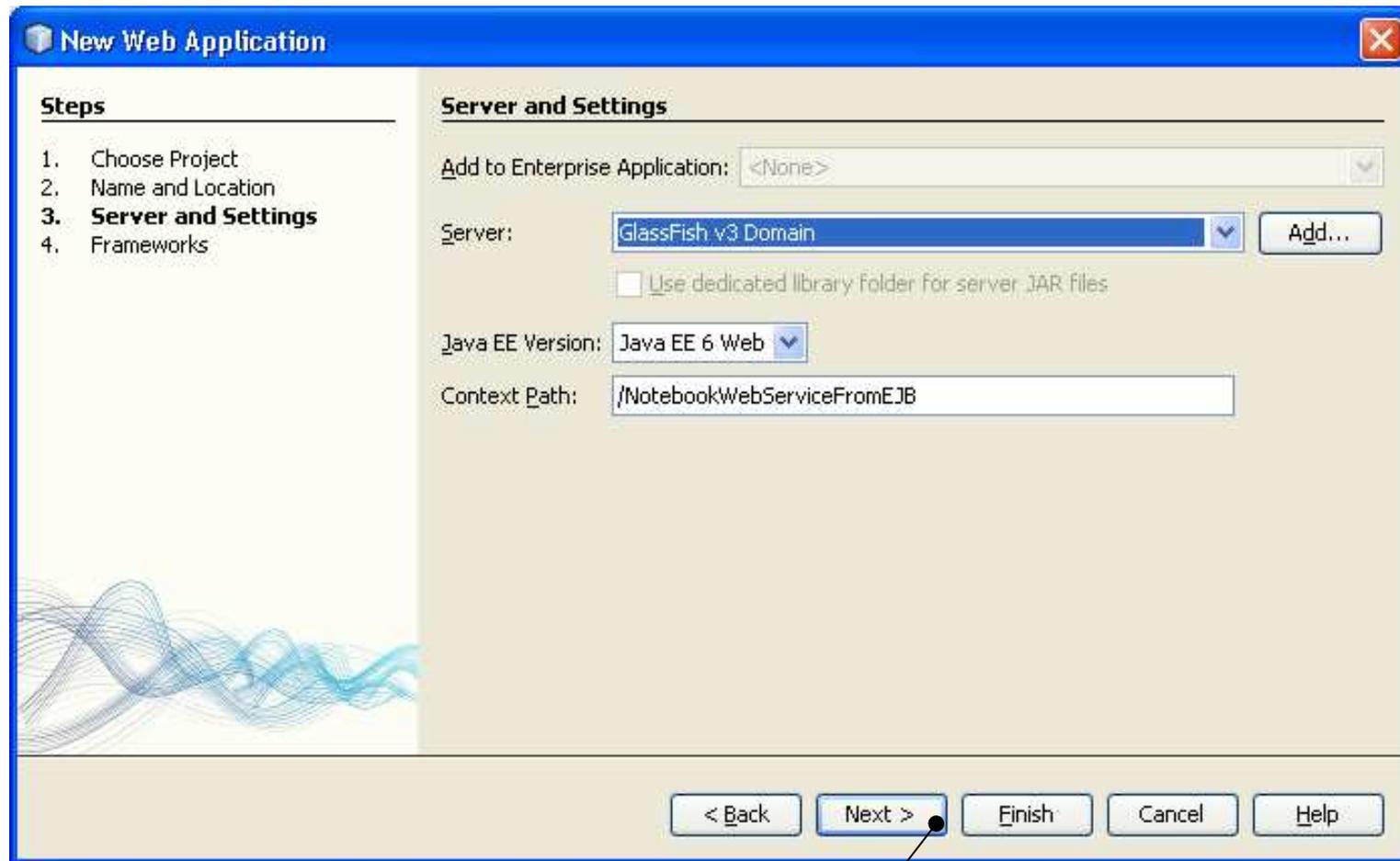
# Service Web avec les EJB : serveur

Définir le nom du projet  
*NotebookWebServiceFromEJB*



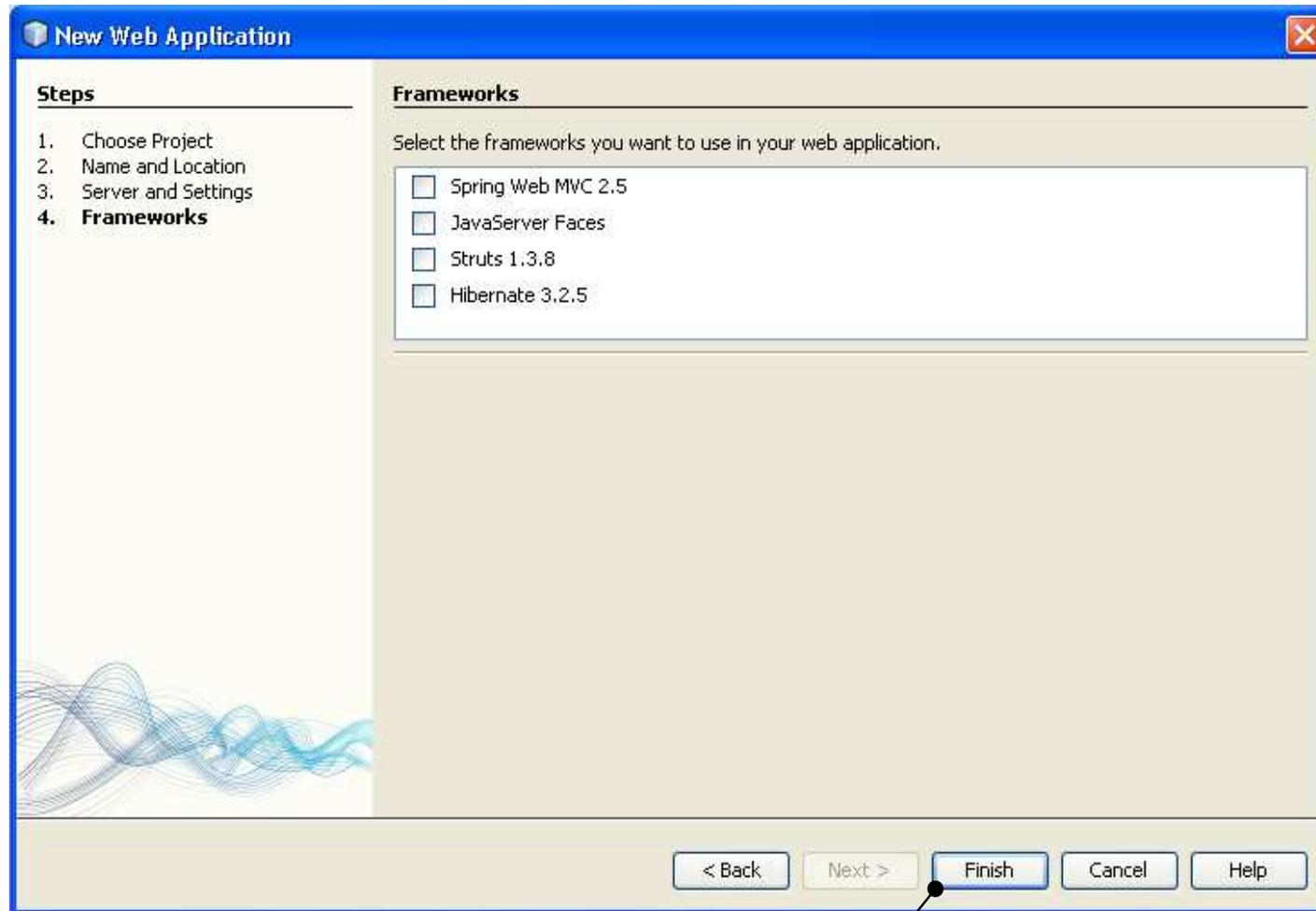
Faire *Next*

# Service Web avec les EJB : serveur



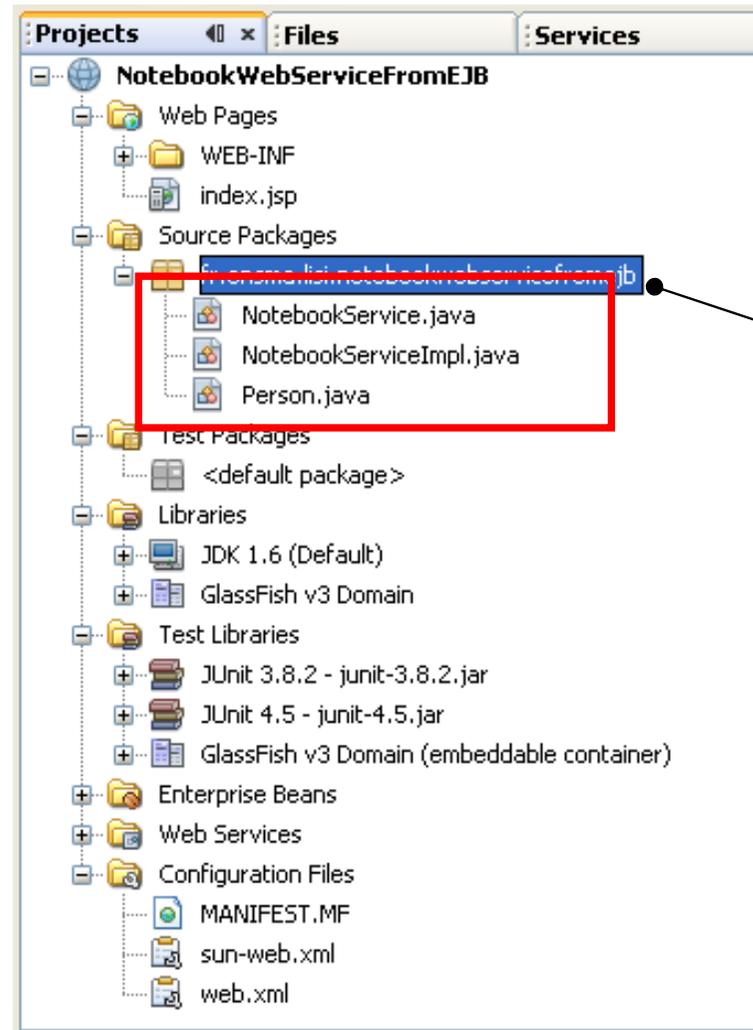
Faire *Next*

# Service Web avec les EJB : serveur



# Service Web avec les EJB : serveur

- Génération de la structure des fichiers du projet



Recopier les fichiers Java dans ce nouveau package

# Service Web avec les EJB : serveur

Ajouter l'annotation `@Stateless` pour définir ce POJO comme un EJB Session sans état

8

```
@Stateless
@WebService(endpointInterface = "fr.ensma.lisi.notebookwebservicefromejb.NotebookService")
public class NotebookServiceImpl {
    public boolean addPersonWithComplexType(Person newPerson) {
        ...
    }

    public Person getPersonByName(String name) {
        ...
    }

    public List<Person> getPersons() {
        ...
    }

    public void addPersonWithSimpleType(String name, String address, String birthyear) {
        ...
    }
}
```

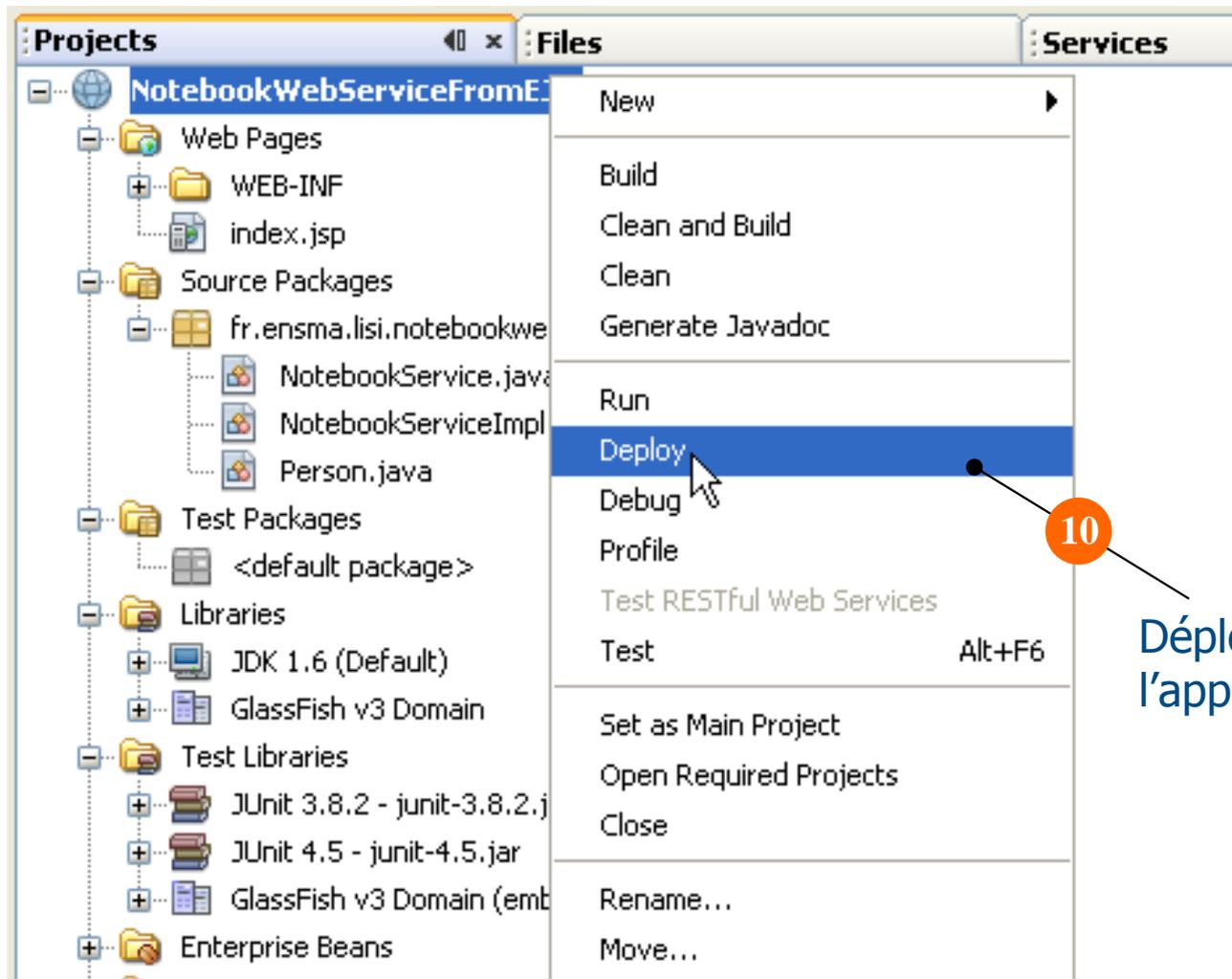
9

S'assurer que le chemin de l'interface du Service Web est correcte

Classe `NotebookServiceImpl.java` du projet **NotebookWebServiceFromEJB**

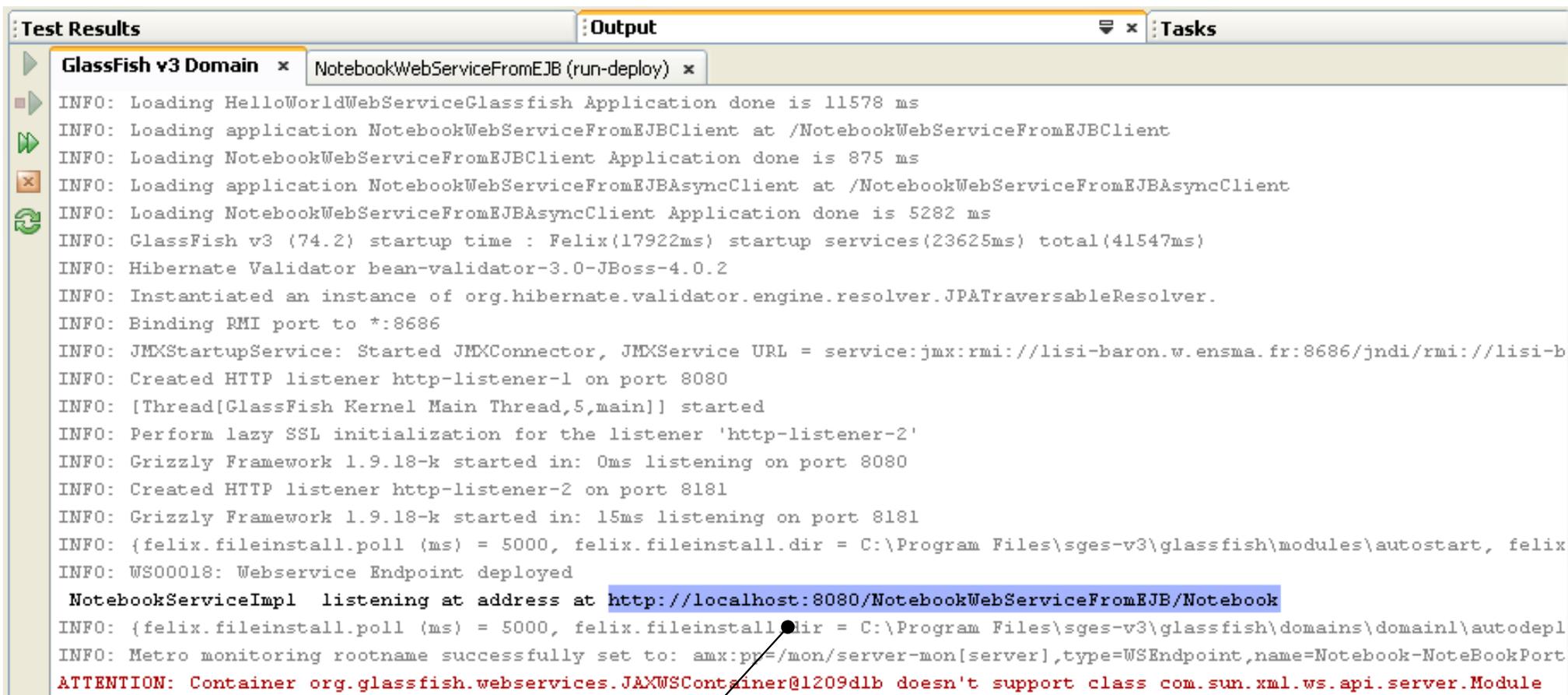
# Service Web avec les EJB : serveur

- Déployer l'application sur Glassfish 3



Déploiement de l'application sur Glassfish

# Service Web avec les EJB : serveur



```
Test Results | Output | Tasks
GlassFish v3 Domain x NotebookWebServiceFromEJB (run-deploy) x
INFO: Loading HelloWorldWebServiceGlassfish Application done is 11578 ms
INFO: Loading application NotebookWebServiceFromEJBClient at /NotebookWebServiceFromEJBClient
INFO: Loading NotebookWebServiceFromEJBClient Application done is 875 ms
INFO: Loading application NotebookWebServiceFromEJBAsyncClient at /NotebookWebServiceFromEJBAsyncClient
INFO: Loading NotebookWebServiceFromEJBAsyncClient Application done is 5282 ms
INFO: GlassFish v3 (74.2) startup time : Felix(17922ms) startup services(23625ms) total(41547ms)
INFO: Hibernate Validator bean-validator-3.0-JBoss-4.0.2
INFO: Instantiated an instance of org.hibernate.validator.engine.resolver.JPATraversableResolver.
INFO: Binding RMI port to *:8686
INFO: JMXStartupService: Started JMXConnector, JMXService URL = service:jmx:rmi://lisi-baron.w.ensma.fr:8686/jndi/rmi://lisi-b
INFO: Created HTTP listener http-listener-1 on port 8080
INFO: [Thread[GlassFish Kernel Main Thread,5,main]] started
INFO: Perform lazy SSL initialization for the listener 'http-listener-2'
INFO: Grizzly Framework 1.9.18-k started in: 0ms listening on port 8080
INFO: Created HTTP listener http-listener-2 on port 8181
INFO: Grizzly Framework 1.9.18-k started in: 15ms listening on port 8181
INFO: {felix.fileinstall.poll (ms) = 5000, felix.fileinstall.dir = C:\Program Files\sges-v3\glassfish\modules\autostart, felix
INFO: WS00018: Webservice Endpoint deployed
NotebookServiceImpl listening at address at http://localhost:8080/NotebookWebServiceFromEJB/Notebook
INFO: {felix.fileinstall.poll (ms) = 5000, felix.fileinstall.dir = C:\Program Files\sges-v3\glassfish\domains\domain1\autodepl
INFO: Metro monitoring rootname successfully set to: amx:pr=/mon/server-mon[server],type=WEEndpoint,name=Notebook-NoteBookPort
ATTENTION: Container org.glassfish.webservices.JAXWSContainer@1209d1b doesn't support class com.sun.xml.ws.api.server.Module
```

L'application est déployée et le Service Web est découvert

# Service Web avec les EJB : serveur

## ► Visualiser le document WSDL

Endpoint	Information
Service Name: {http://notebookwebservicefromejb.lisi.ensma.fr/} Notebook Port Name: {http://notebookwebservicefromejb.lisi.ensma.fr/} NoteBookPort	Address: http://localhost:8080/NotebookWebServiceFromEJB/Notebook WSDL: <a href="http://localhost:8080/NotebookWebServiceFromEJB/Notebook?wsdl">http://localhost:8080/NotebookWebServiceFromEJB/Notebook?wsdl</a> Implementation class: fr.ensma.lisi.notebookwebservicefromejb.NotebookServiceImpl

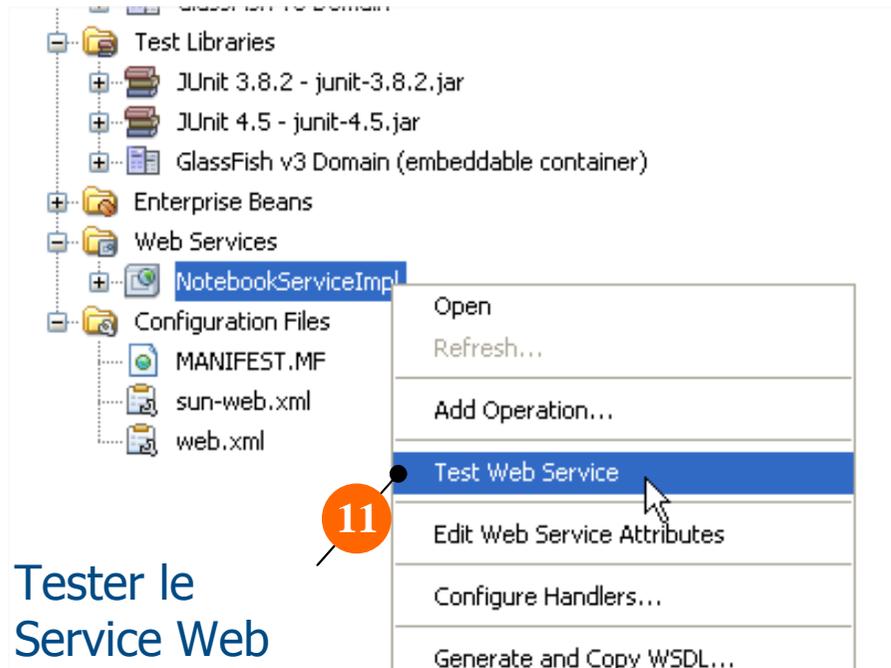
```

- <!--
  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2-hudson-752-.
-->
- <!--
  Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2-hudson-752-.
-->
- <definitions targetNamespace="http://notebookwebservicefromejb.lisi.ensma.fr" name="Notebook">
  <import namespace="http://notebookwebservice.lisi.ensma.fr" location="http://localhost:8080/NotebookWebServiceFromEJB/Notebook?wsdl=1"/>
  <binding name="NoteBookPortBinding" type="ns1:Notebook">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="addPersonWithComplexType">
      <soap:operation soapAction="">
        <input>
          <soap:body use="literal"/>
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
      </operation>
    <operation name="getPersonByName">

```

# Service Web avec les EJB : serveur

## ➤ Tester le Service Web



Génération d'un formulaire pour tester le Service Web

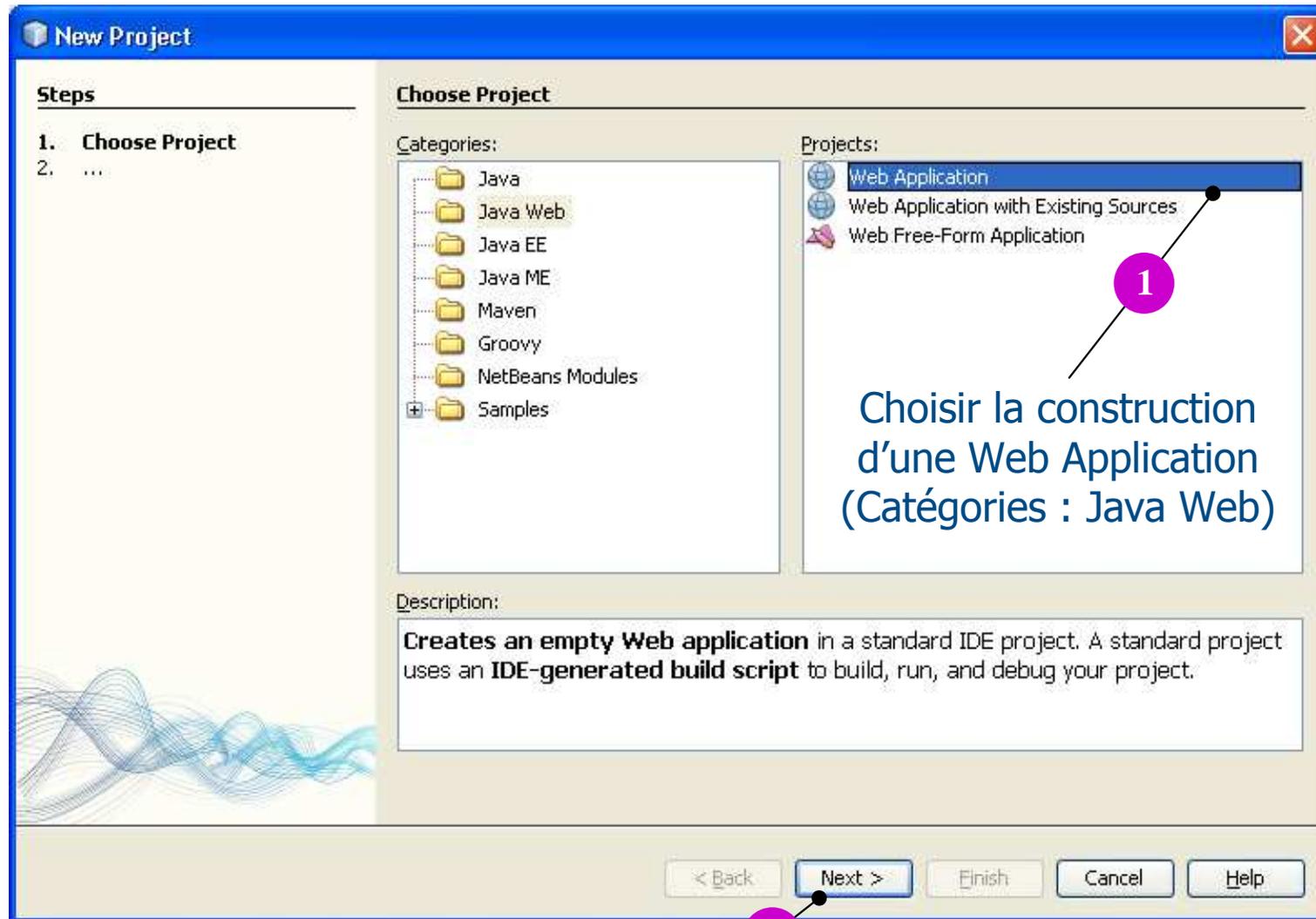


Possibilité de saisir les paramètres et de tester le résultat



# Service Web avec les EJB : client

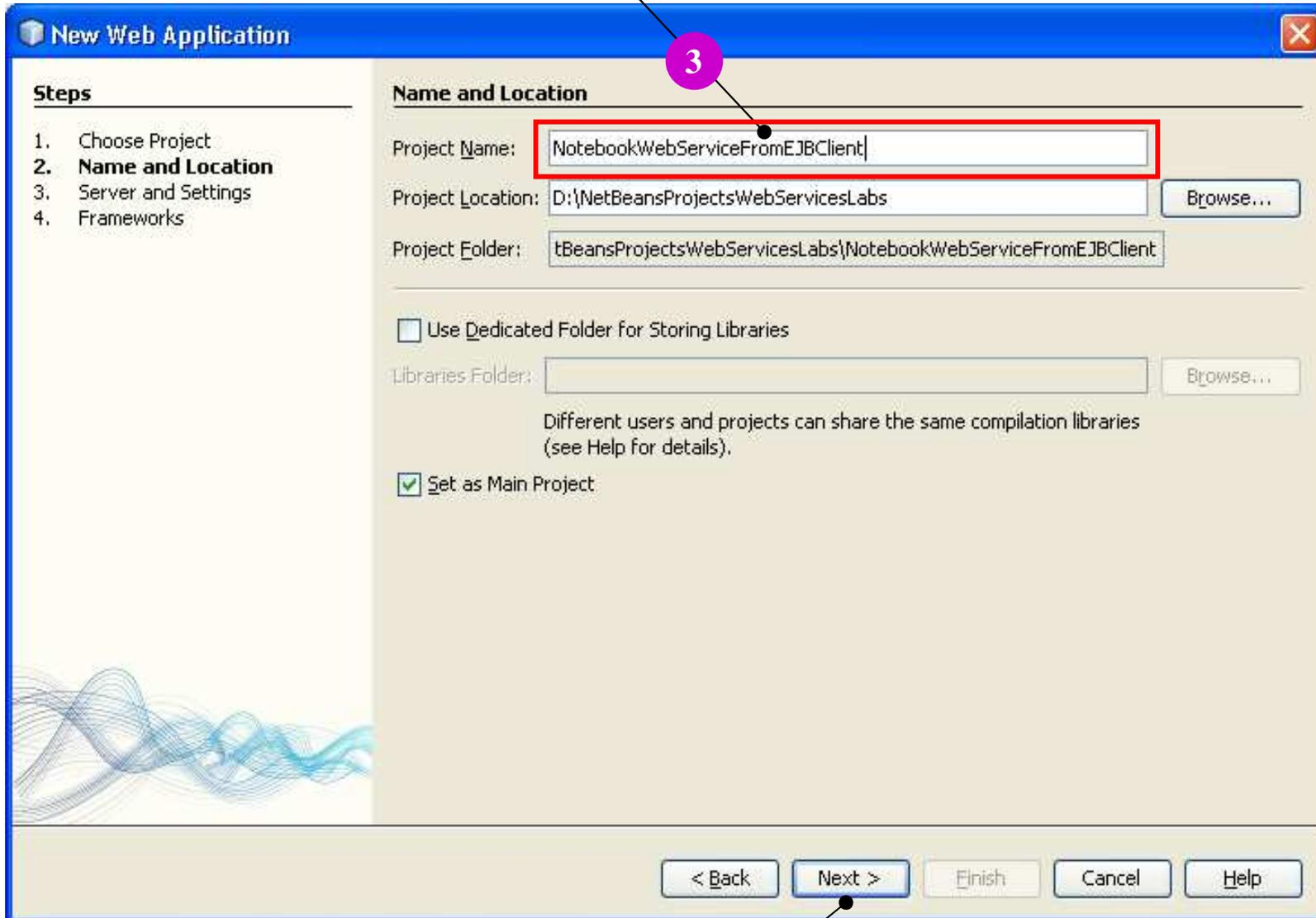
- Démarrer la création d'un nouveau Projet (File -> New Project ...)



Faire *Next*

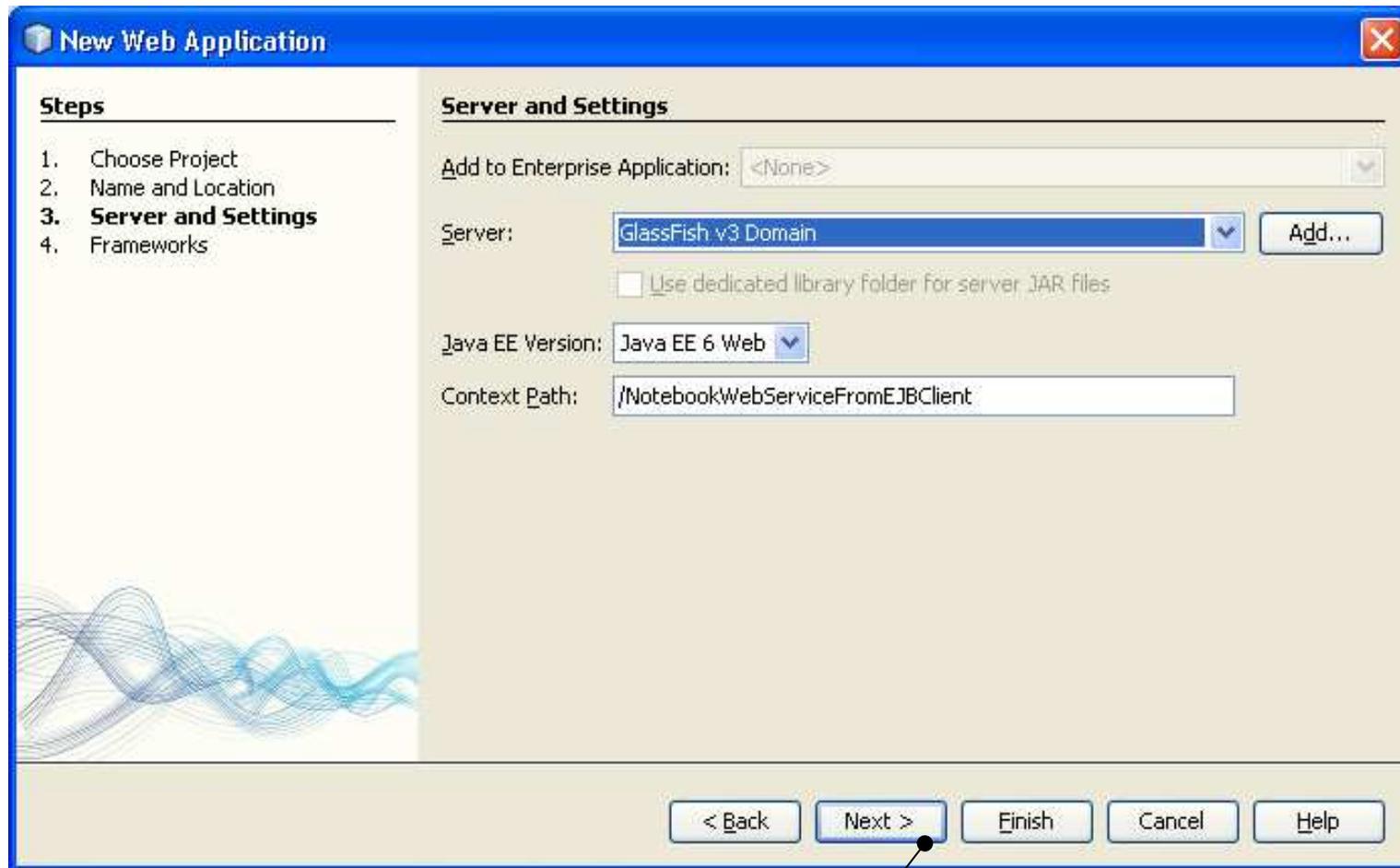
# Service Web avec les EJB : client

Définir le nom du projet  
*NotebookWebServiceFromEJBClient*



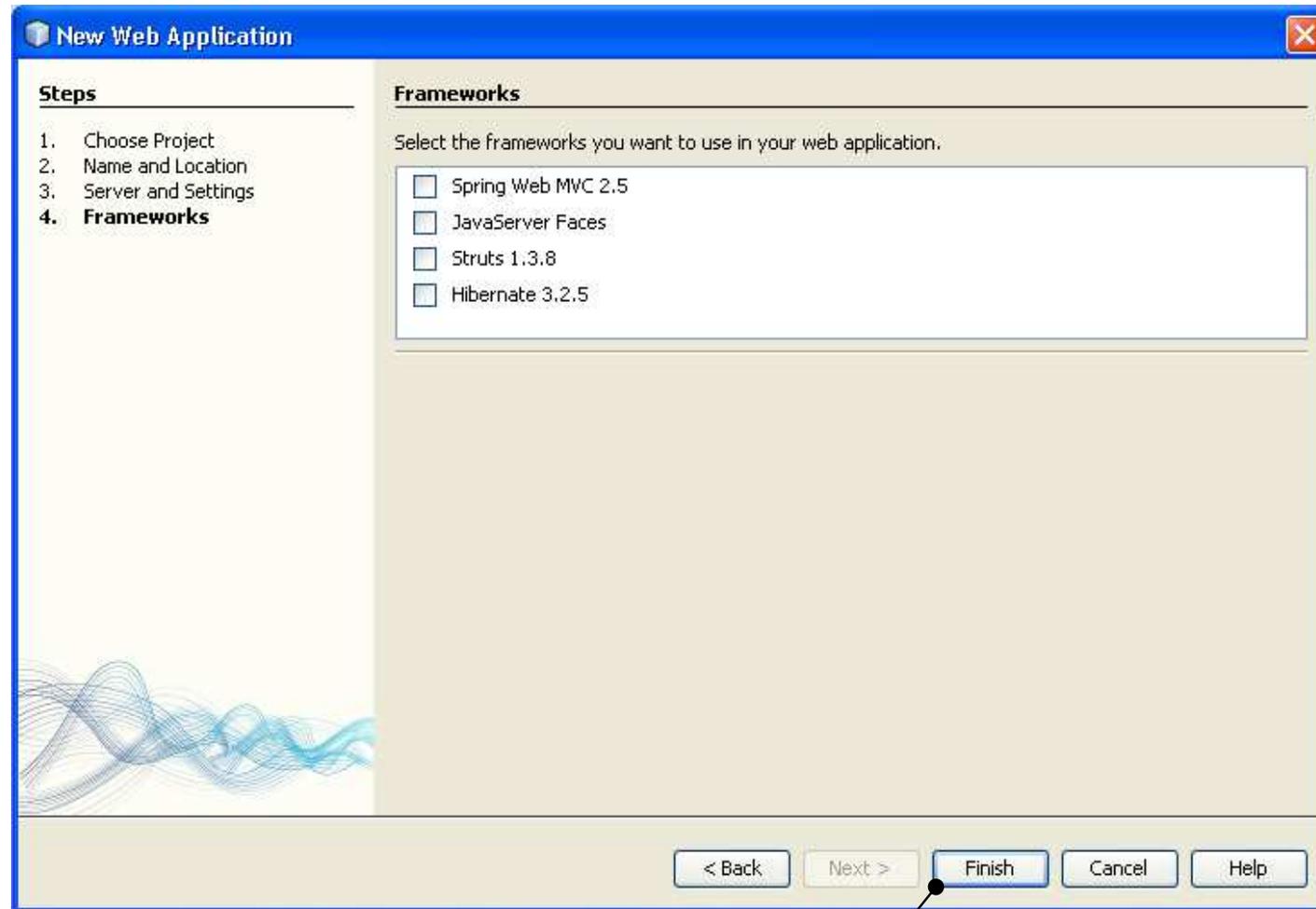
Faire *Next*

# Service Web avec les EJB : client



Faire *Next*

# Service Web avec les EJB : client



Faire *Finish*

# Service Web avec les EJB : client

- Ajouter la bibliothèque JAX-WS pour la génération des *artifacts* (JAXB, ...) à partir de l'outil *wsimport*

Bouton droit sur le nœud *Libraries*

7

8

9

Choisir la bibliothèque JAX-WS 2.2

Faire *Add Library*

Libraries

- JAX-WS 2.2 - activation.jar
- JAX-WS 2.2 - jaxb-api.jar
- JAX-WS 2.2 - jaxb-impl.jar
- JAX-WS 2.2 - jaxb-xjc.jar
- JAX-WS 2.2 - jsr173\_api.jar
- JAX-WS 2.2 - FastInfoset.jar
- JAX-WS 2.2 - gmbal-api-only.jar
- JAX-WS 2.2 - http.jar
- JAX-WS 2.2 - jaxws-api.jar
- JAX-WS 2.2 - jaxws-rt.jar
- JAX-WS 2.2 - jaxws-tools.jar
- JAX-WS 2.2 - jsr181-api.jar
- JAX-WS 2.2 - jsr250-api.jar
- JAX-WS 2.2 - management-api.jar
- JAX-WS 2.2 - mimepull.jar
- JAX-WS 2.2 - policy.jar
- JAX-WS 2.2 - saaj-api.jar
- JAX-WS 2.2 - saaj-impl.jar
- JAX-WS 2.2 - stax-ex.jar
- JAX-WS 2.2 - streambuffer.jar
- JAX-WS 2.2 - woodstox.jar
- JDK 1.6 (Default)
- GlassFish v3 Domain

Available Libraries:

- Java ME CDC AGUI Tree Layout
- Java ME CDC NSIcom Plugin Impl
- Java ME CDC Personal Profile Free
- Java Tree API
- JAXB 2.2
- JAX-RS 1.1
- JAX-WS 2.2
- Jersey 1.1 (JAX-RS RI)
- JMUnit for CLDC10
- JMUnit for CLDC11
- JSF 1.2
- JSF 2.0
- JSTL 1.1
- JUnit 3.8.2
- JUnit 4.5

Add Library

Cancel

Create...

# Service Web avec les EJB : client

- Créer un client à partir d'un document WSDL (New -> Web Services -> Web Service Client)

**New Web Service Client**

**Steps**

1. Choose File Type
2. **WSDL and Client Location**

**WSDL and Client Location**

Specify the WSDL file of the Web Service.

Project:  Browse...

Local File:  Browse...

**10** WSDL URL:  Set Proxy...

Specify a package name where the client java artifacts will be generated:

Project: NotebookWebServiceFromEJBClient

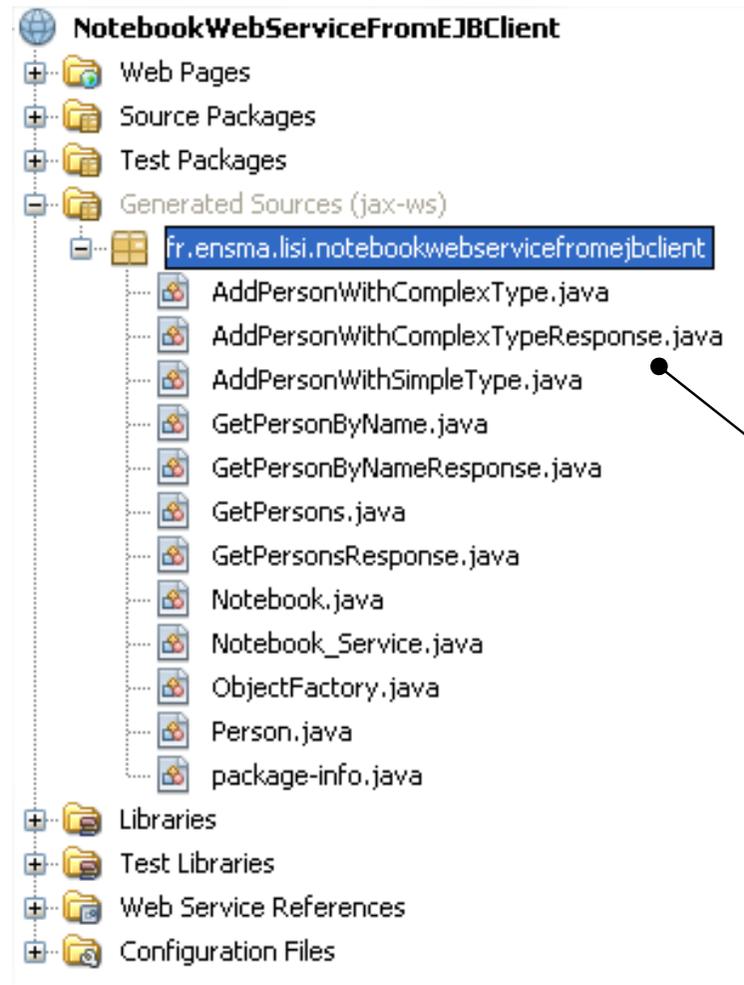
**11** Package:

Client Style:

Generate Dispatch code

< Back Next > **12** Finish Cancel Help

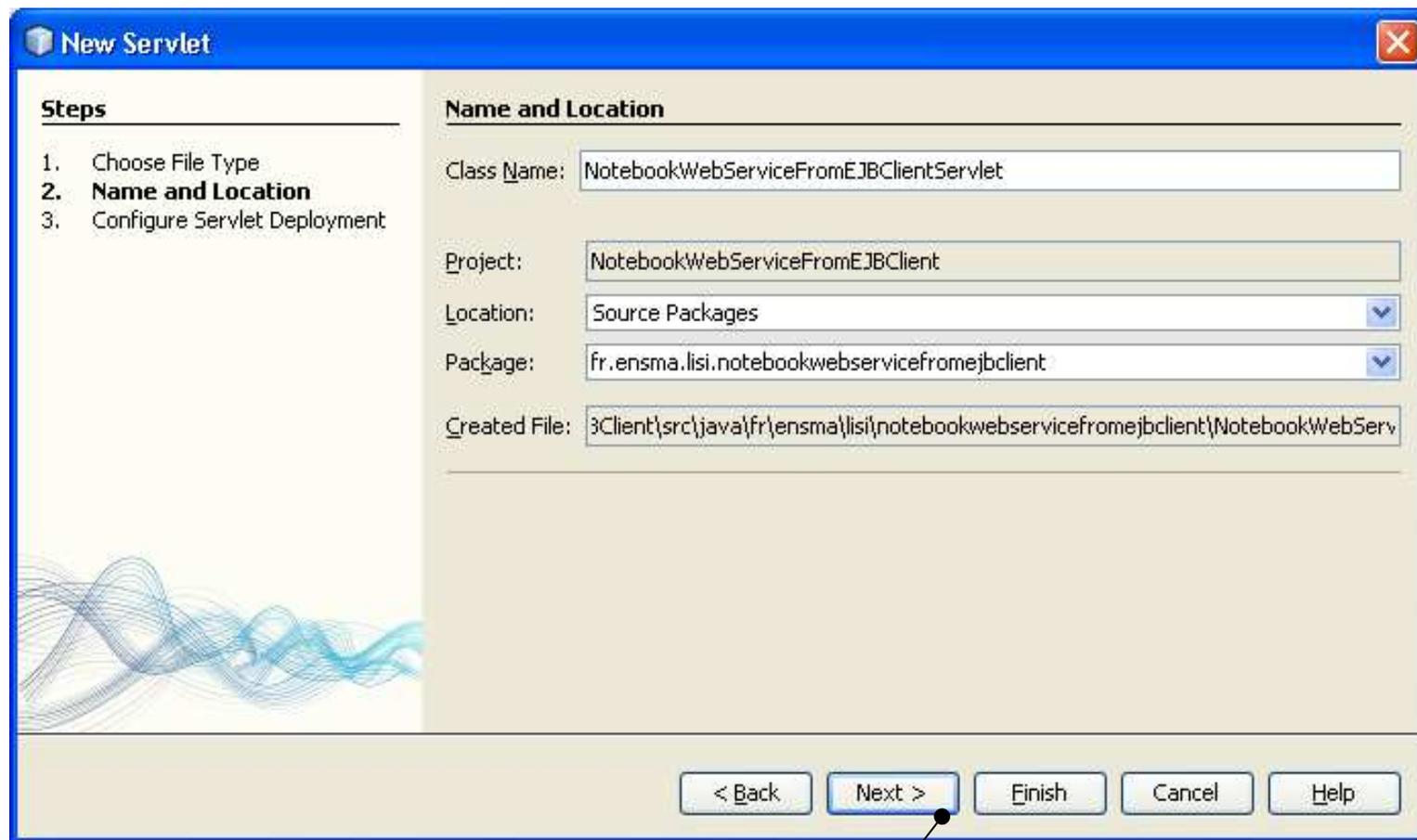
# Service Web avec les EJB : client



Les artefacts du Service Web Notebook sont générés

# Service Web avec les EJB : client

- Création d'une Servlet pour appeler un Service Web (File -> New Files ... -> Servlet)

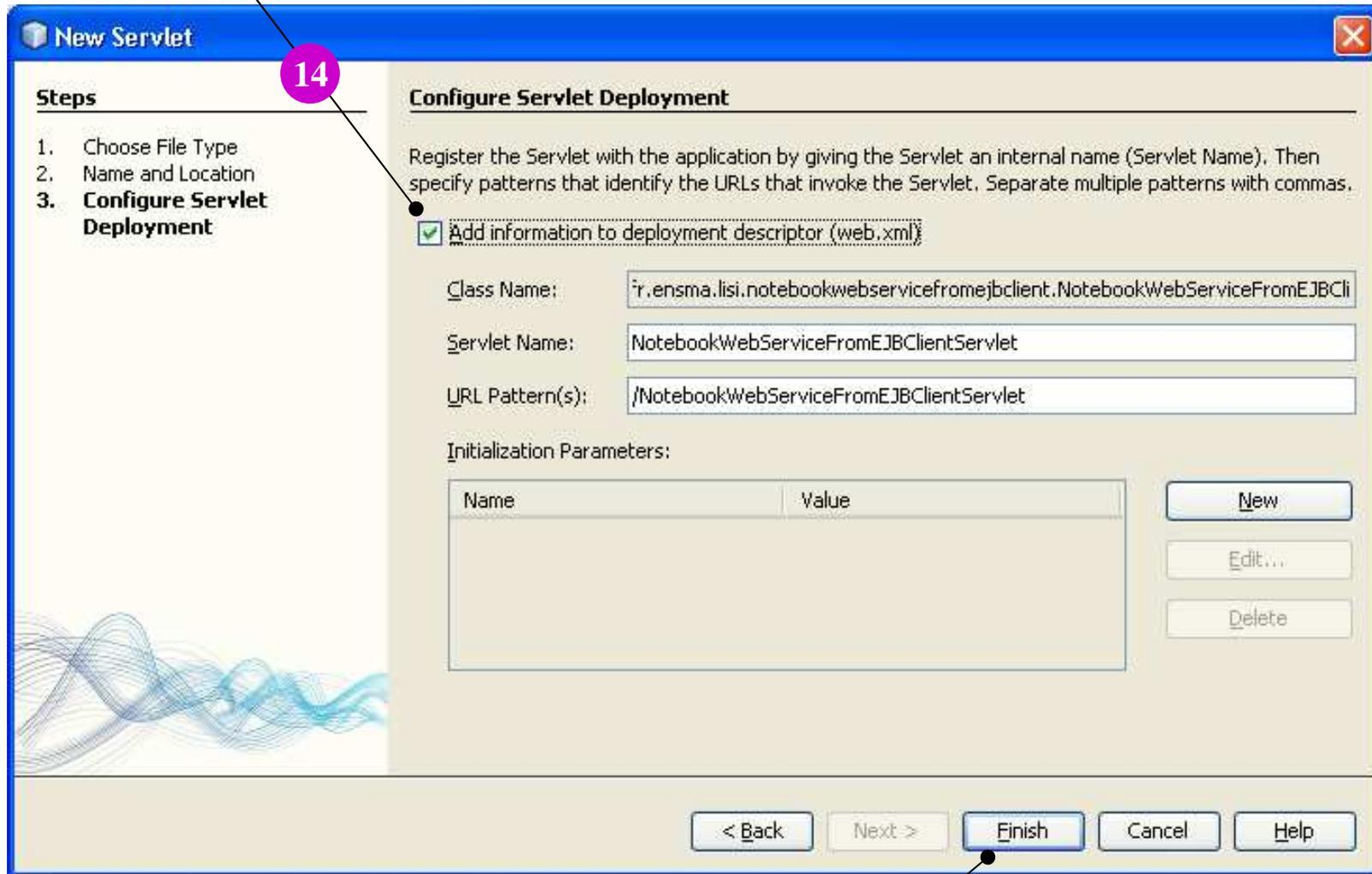


13

Faire *Next*

# Service Web avec les EJB : client

Ajouter l'information dans le fichier *web.xml*



Faire *Finish*

## ► Compléter le code de la Servlet

```
public class NotebookWebServiceFromEJBClientServlet extends HttpServlet {

    @WebServiceRef(wsdlLocation = "http://localhost:8080/NotebookWebServiceFromEJB/Notebook?wsdl")
    private Notebook_Service service;

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet NotebookWebServiceFromEJBClientServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet NotebookWebServiceFromEJBClientServlet at " + request.getContextPath() + "</h1>");

            try {
                Notebook port = service.getNoteBookPort();
                Person newPerson = new Person();
                newPerson.setName("BARON Mickael");
                newPerson.setAddress("Poitiers");
                newPerson.setBirthyear("1976");
                boolean result = port.addPersonWithComplexType(newPerson);
                out.println("<div>Result = " + result + "</div>");
            } catch (Exception ex) {
                ex.printStackTrace();
            }
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }
}
```

Classe *NotebookWebServiceFromEJBClientServlet*  
du projet **NotebookWebServiceFromEJBClient**