

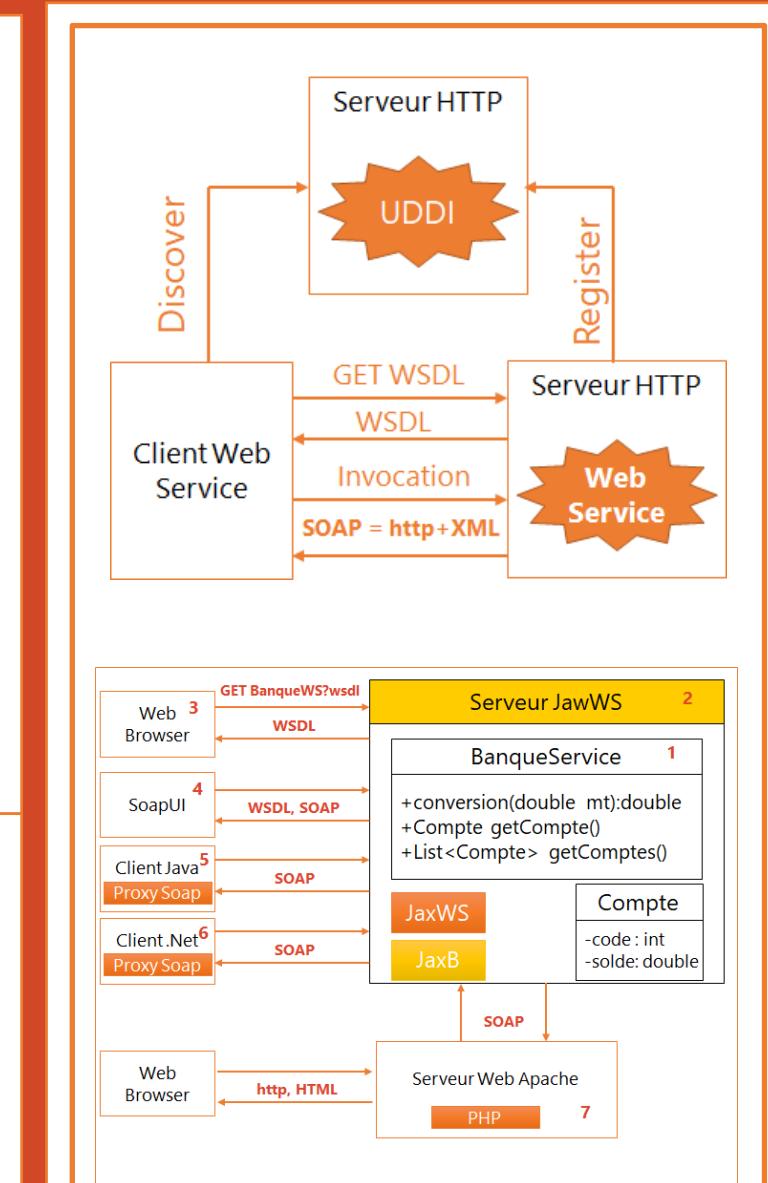
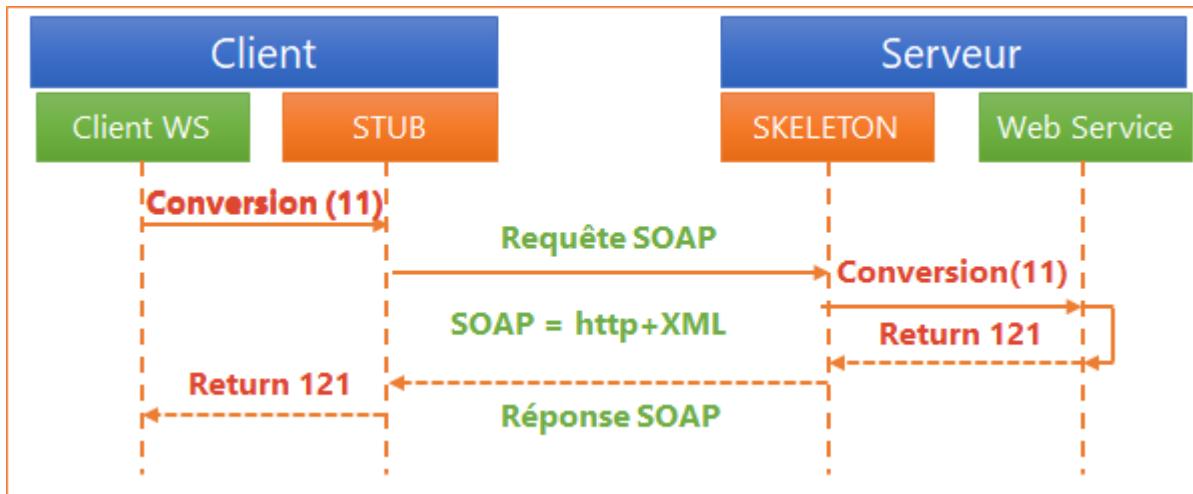
# Web Services : SOAP, WSDL, UDDI



ENSET

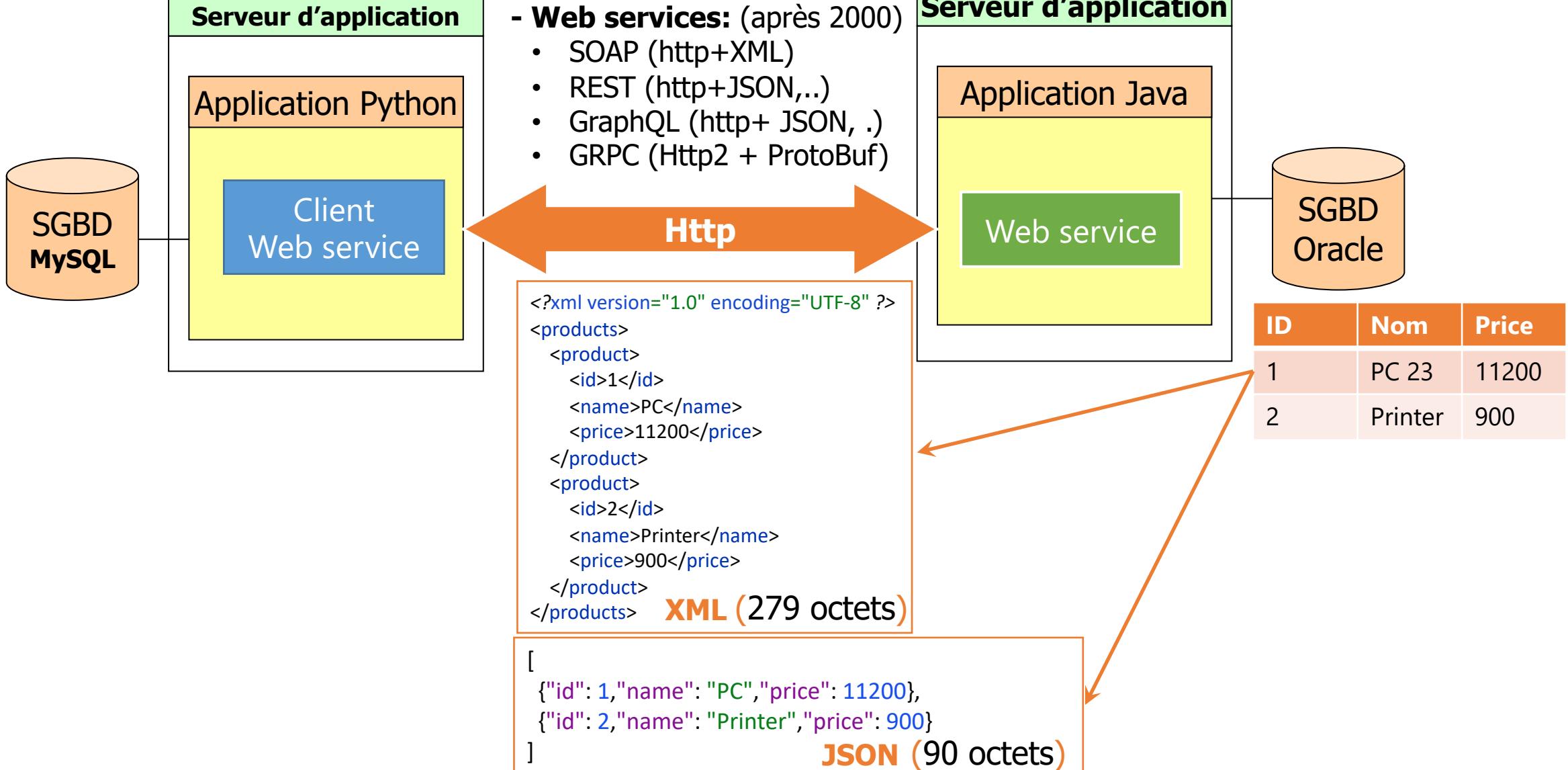


Master : Ingénierie Informatique, Big Data et Cloud Computing  
ENSET Mohammedia



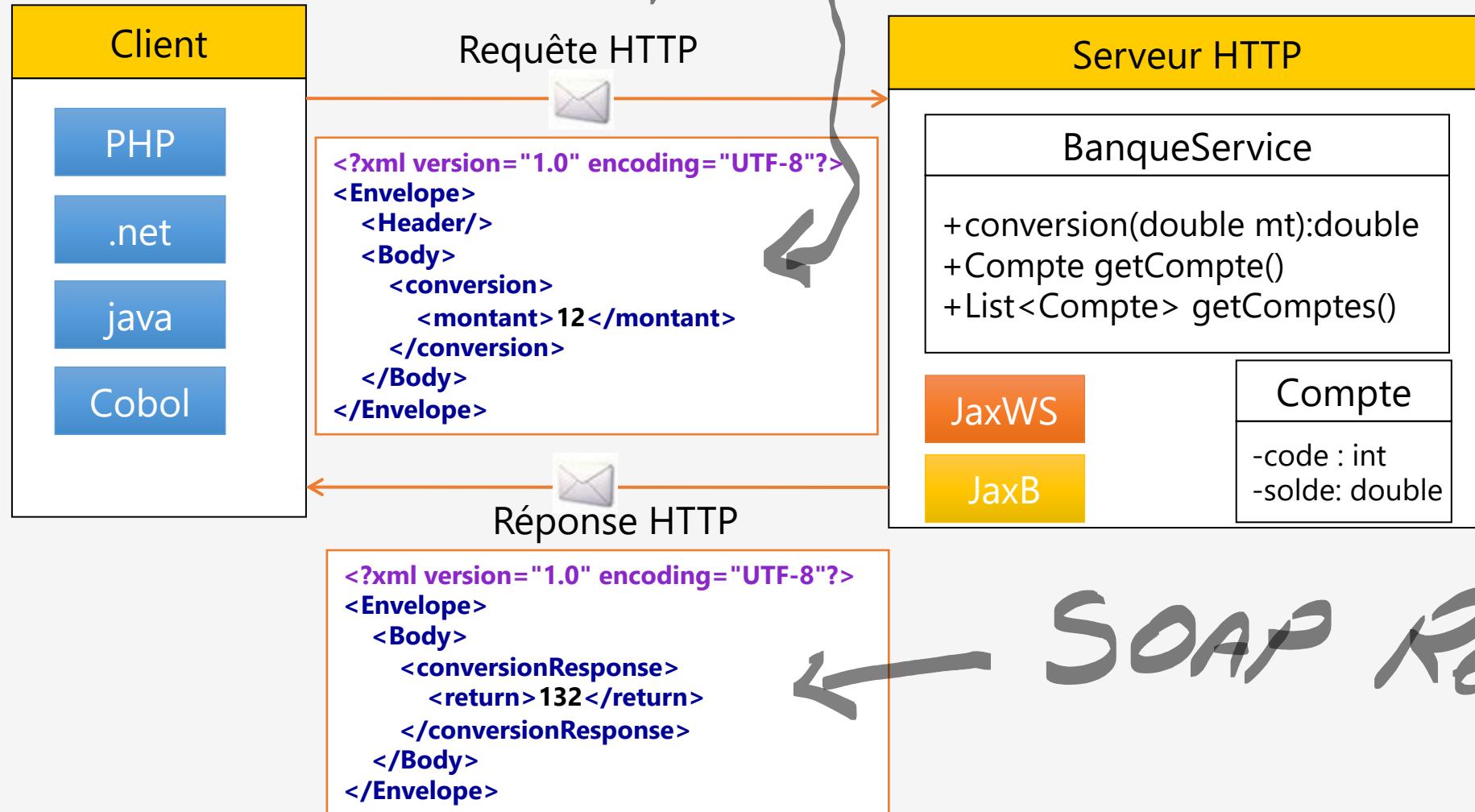
Mohamed Youssfi, Enseignant Chercheur, ENSET Mohammedia, Université Hassan II  
de Casablanca, Consultant R&D Ingénierie Logicielle,  
Directeur du Laboratoire Informatique, Intelligence Artificielle et Cyber Sécurité

# Systèmes Distribués



# Architecture de base des Web Services

*SOAP Request*



*SOAP Response*

## Requête SOAP avec POST

### Entête de la requête

Post /WebServicePath HTTP/1.1

accept: application/xml

Content-Type : application/xml

\*\*\* saut de ligne \*\*\*

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <conversion xmlns="http://bk/test">
      <montant xmlns="">12</montant>
    </conversion>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### corps de la requête HTTP

## Réponse SOAP :

### Entête de la réponse

**HTTP/1.0 200 OK**

**Date : Wed, 05Feb02 15:02:01 GMT**

**Server : Apache/1.3.24**

**Mime-Version 1.0**

**Last-Modified : Wed 02Oct01 24:05:01GMT**

**Content-Type : application/xml**

**Content-length : 4205**

**\*\*\* saut de ligne \*\*\***

```
<?xml version="1.0" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns1="http://bk/test">
  <soapenv:Body>
    <ns1:conversionResponse>
      <return>132.0</return>
    </ns1:conversionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

### corps de la réponse

# Concepts fondamentaux des web services

Le concept des Web Services s'articule actuellement autour des trois concepts suivants :

- **SOAP (*Simple Object Access Protocol*)**

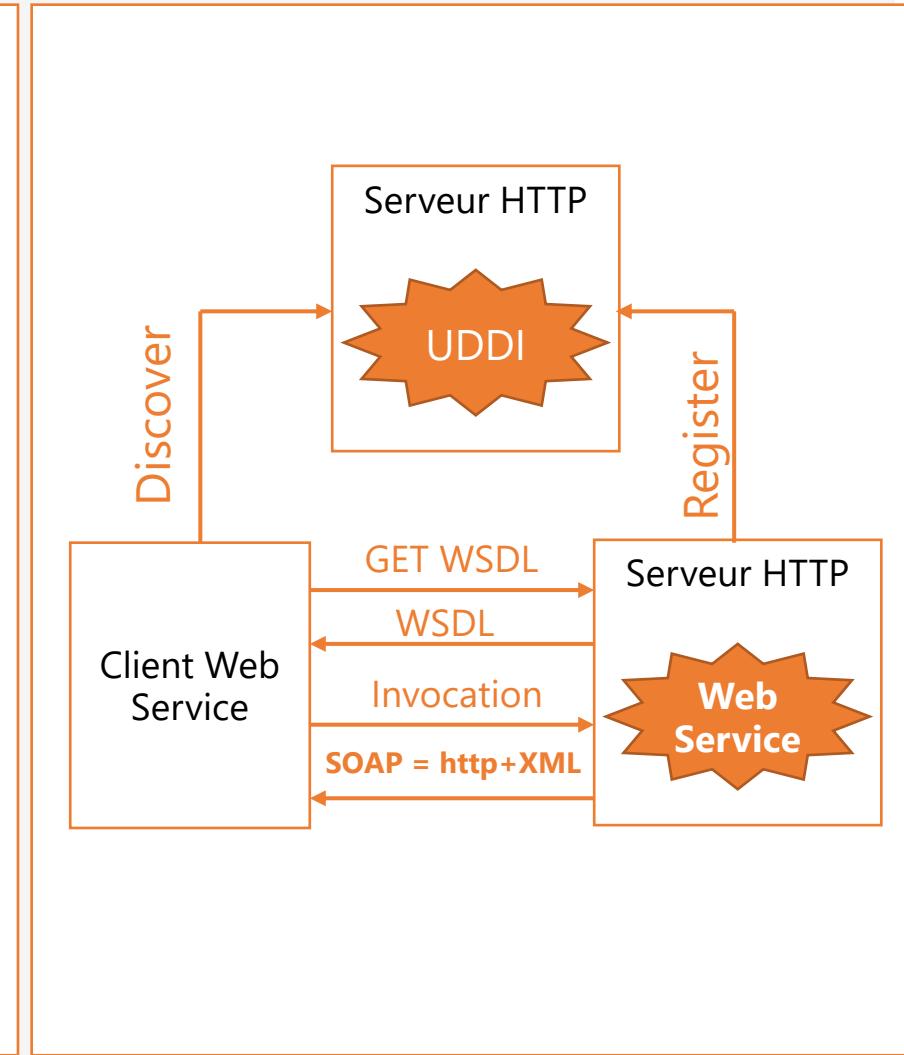
- est un protocole d'échange inter-applications indépendant de toute plate-forme, basé sur le langage XML.
- Un appel de service SOAP est un flux ASCII encadré dans des balises XML et transporté dans le protocole HTTP.

- **WSDL (*Web Services Description Language*)**

- donne la description au format XML des Web Services en précisant les méthodes pouvant être invoquées, leurs signatures et le point d'accès (URL, port, etc..).
- C'est, en quelque sorte, l'équivalent du langage IDL pour la programmation distribuée CORBA.

- **UDDI (*Universal Description, Discovery and Integration*)**

- normalise une solution d'annuaire distribué de Web Services, permettant à la fois la publication et l'exploration (recherche) de Web Services.
- UDDI se comporte lui-même comme un Web service dont les méthodes sont appelées via le protocole SOAP.

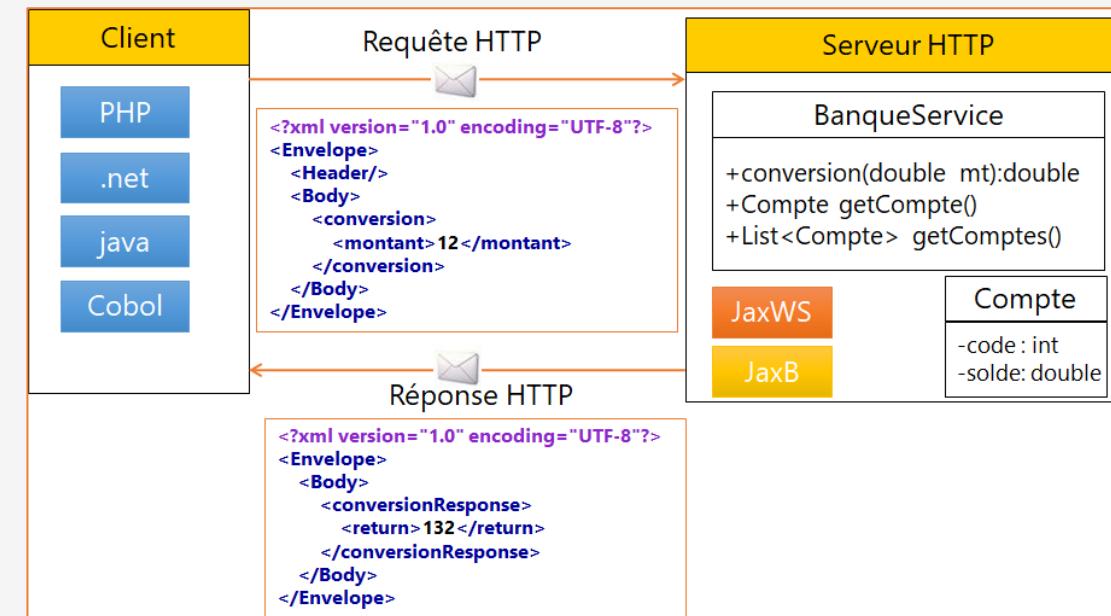


# Mise en œuvre des web services avec JAX-WS

# JAX-WS

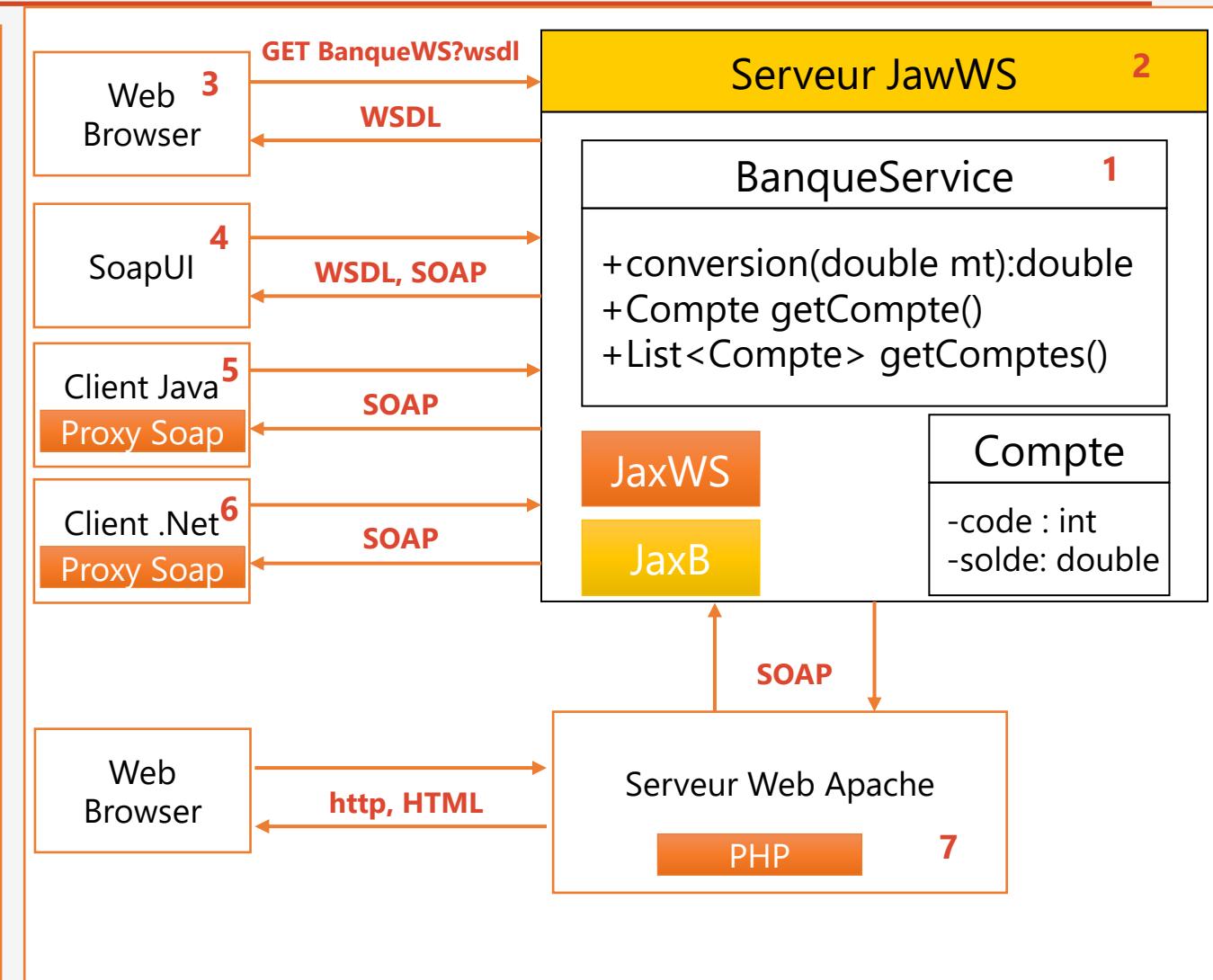
- JAX-WS est la nouvelle appellation de JAX-RPC (Java API for XML Based RPC) qui permet de développer très simplement des services web en Java.
- JAX-WS fournit un ensemble d'annotations pour mapper la correspondance Java-WSDL. Il suffit pour cela d'annoter directement les classes Java qui vont représenter le service web.
- Dans l'exemple ci-dessous, une classe Java utilise des annotations JAX-WS qui vont permettre par la suite de générer le document WSDL. Le document WSDL est auto-généré par le serveur d'application au moment du déploiement :

```
@WebService(serviceName="BanqueWS")
public class BanqueService {
    @WebMethod(operationName="ConversionEuroToDh")
    public double conversion(@WebParam(name="montant")double mt){
        return mt*11;
    }
    @WebMethod
    public String test(){ return "Test"; }
    @WebMethod
    public Compte getCompte(){ return new Compte (1,7000); }
    @WebMethod
    public List<Compte> getComptes(){
        List<Compte> cptes=new ArrayList<Compte>();
        cptes.add (new Compte (1,7000)); return cptes;
    }
}
```

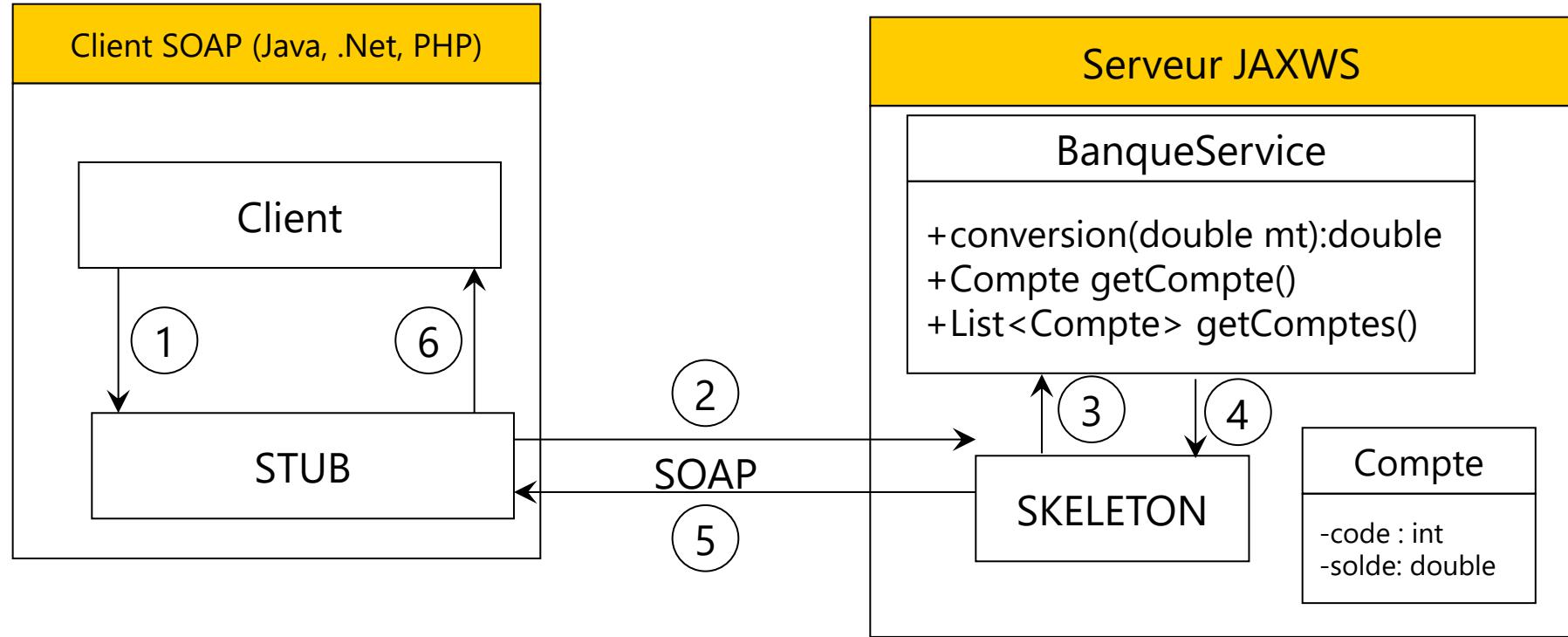


# Application

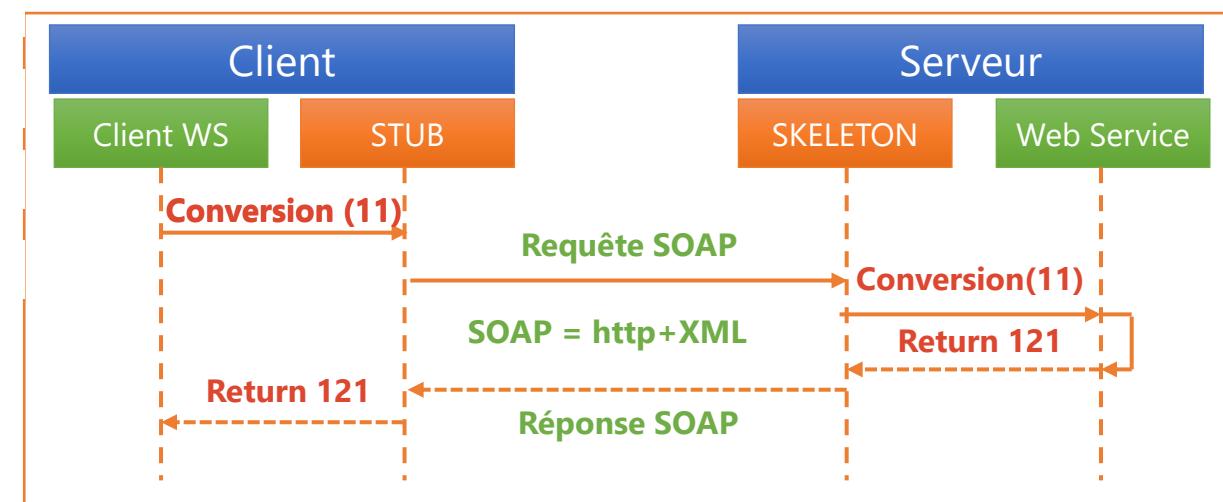
1. Créer un Web service qui permet de :
  - Convertir un montant de l'auro en DH
  - Consulter un Compte
  - Consulter une Liste de comptes
2. Déployer le Web service avec un simple Serveur JaxWS
3. Consulter et analyser le WSDL avec un Browser HTTP
4. Tester les opérations du web service avec un outil comme SoapUI ou Oxygen
5. Créer un Client SOAP Java
6. Créer un Client SOAP Dot Net
7. Créer un Client SOAP PHP
8. Déployer le Web Service dans un Projet Spring Boot



# Architecture 1

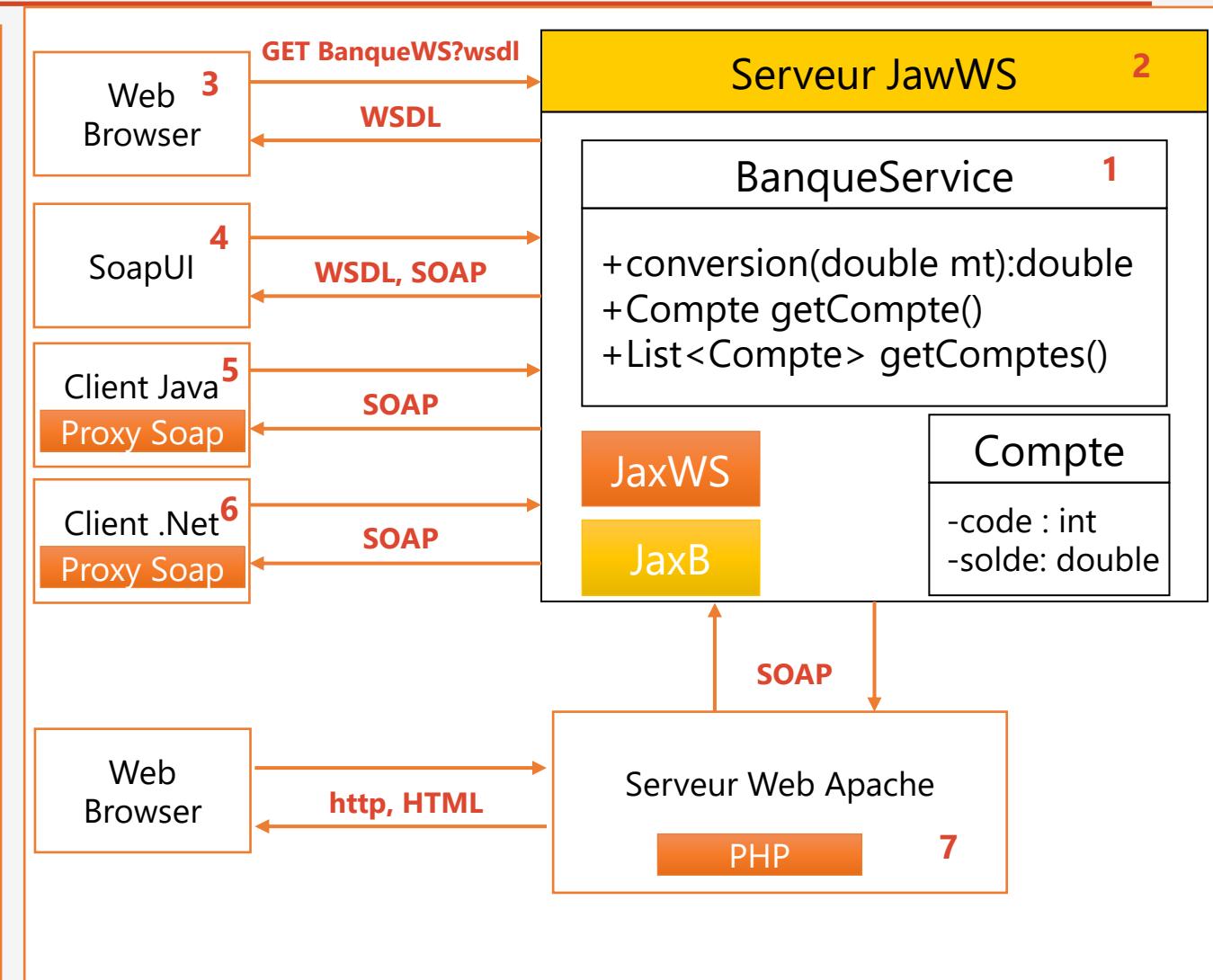


- 1 Le client demande au stub de faire appel à la méthode conversion(12)
- 2 Le Stub se connecte au Skeleton et lui envoie une requête SOAP
- 3 Le Skeleton fait appel à la méthode du web service
- 4 Le web service retourne le résultat au Skeleton
- 5 Le Skeleton envoie le résultat dans une la réponse SOAP au Stub
- 6 Le Stub fournie le résultat au client



# Application

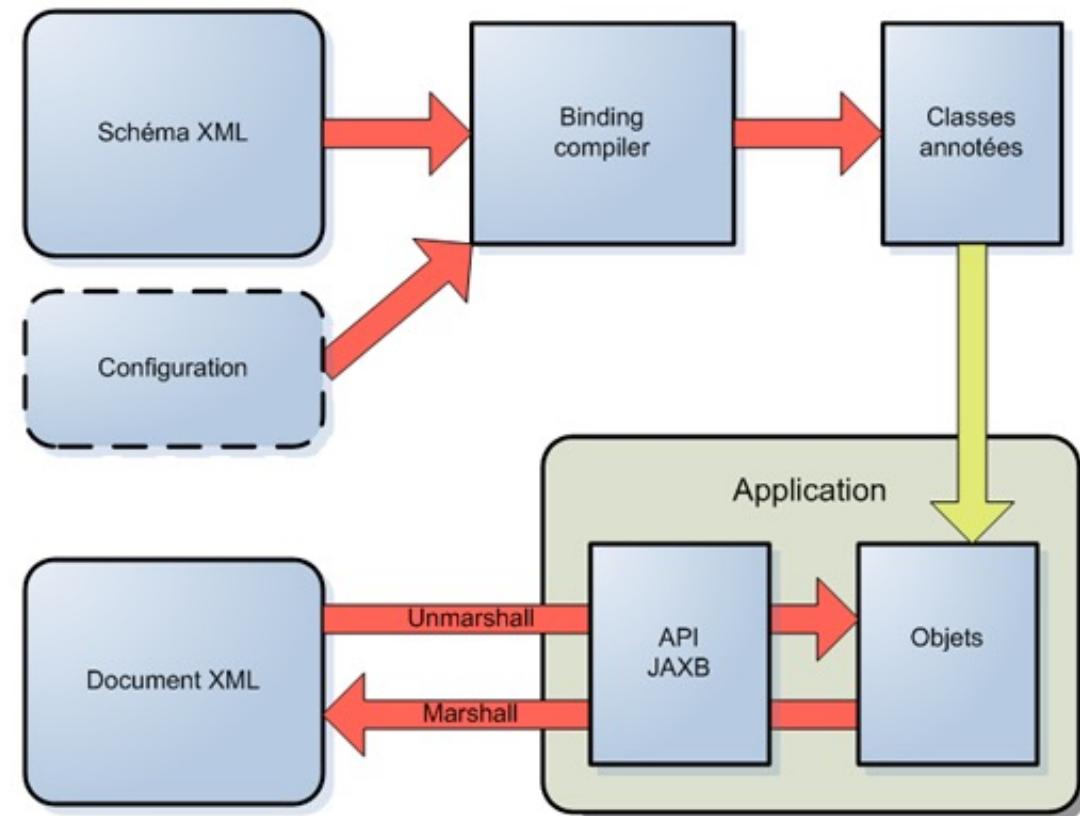
1. Créer un Web service qui permet de :
  - Convertir un montant de l'auro en DH
  - Consulter un Compte
  - Consulter une Liste de comptes
2. Déployer le Web service avec un simple Serveur JaxWS
3. Consulter et analyser le WSDL avec un Browser HTTP
4. Tester les opérations du web service avec un outil comme SoapUI ou Oxygen
5. Créer un Client SOAP Java
6. Créer un Client SOAP Dot Net
7. Créer un Client SOAP PHP
8. Déployer le Web Service dans un Projet Spring Boot



# **Mapping Objet XML (OXM) Avec Jax Binding (JAXB2)**

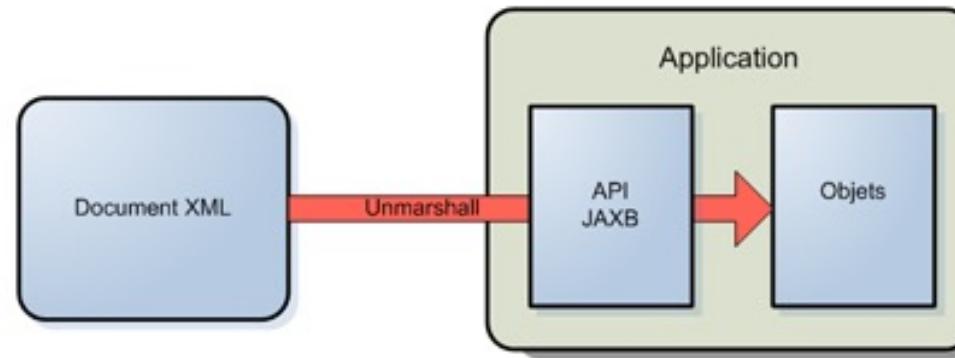
## JAX-WS / JAXB

- ▶ JAX-WS s'appuie sur l'API JAXB 2.0 pour tout ce qui concerne la correspondance entre document XML et objets Java.
- ▶ JAXB 2.0 permet de mapper des objets Java dans un document XML et vice versa.
- ▶ Il permet aussi de générer des classes Java à partir un schéma XML et vice et versa.

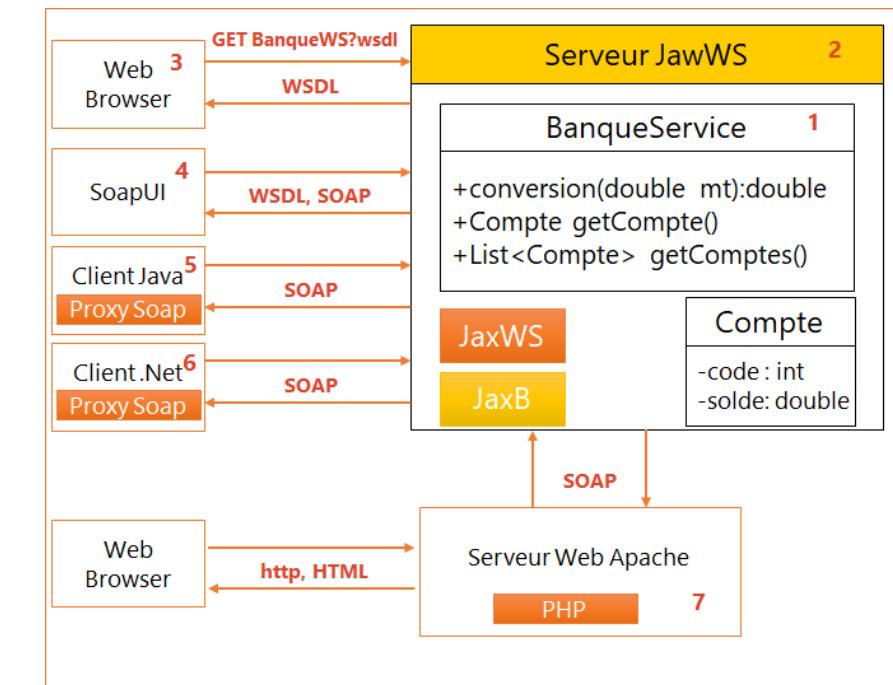
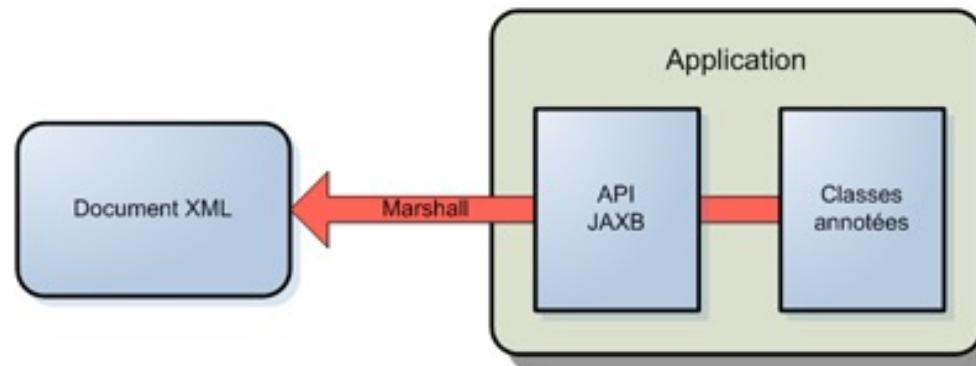


## Principe de JAXB

- Le mapping d'un document XML à des objets (unmarshal)

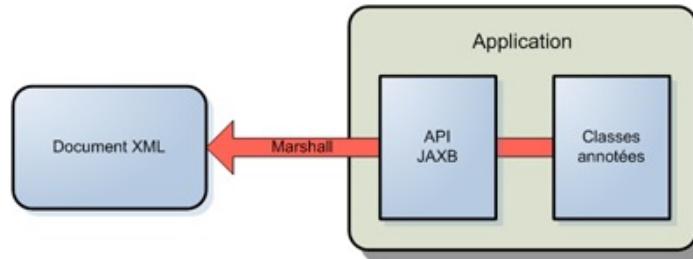


- La création d'un document XML à partir d'objets (marshal)



## Générer XML à partir des objet java avec JAXB

```
package ws;
import java.util.Date;
import javax.xml.bind.annotation.*;
@XmlRootElement
public class Compte {
    private int code;
    private float solde;
    private Date dateCreation;
    // Constructeur sans paramètre
    // Constructeur avec paramètres
    // Getters et Setters
}
```



```
package ws;
import java.io.File; import java.util.Date;
import javax.xml.bind.*;
public class Banque {
    public static void main(String[] args) throws Exception {
        JAXBContext context=JAXBContext.newInstance(Compte.class);
        Marshaller marshaller=context.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,true);
        Compte cp=new Compte(1,8000,new Date());
        marshaller.marshal(cp,new File("comptes.xml"));
    }
}
```

Fichier XML Généré : comptes.xml ➔

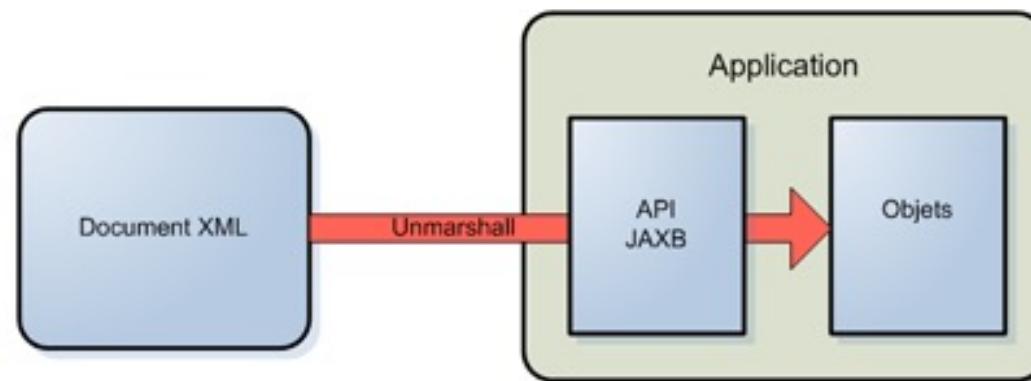
```
<?xml version="1.0" encoding="UTF-8"?>
<compte>
    <code>1</code>
    <dateCreation>
        2014-01-16T12:33:16.960Z
    </dateCreation>
    <solde>8000.0</solde>
</compte>
```

## Générer des objets java à partir des données XML

```
package ws;
import java.io.*;
import javax.xml.bind.*;
public class Banque2 {
public static void main(String[] args) throws Exception {
JAXBContext jc=JAXBContext.newInstance(Compte.class);
Unmarshaller unmarshaller=jc.createUnmarshaller();
Compte cp=(Compte) unmarshaller.unmarshal(new File("comptes.xml"));
System.out.println(cp.getCode()+" - "+cp.getSolde()+" - "+cp.getDateCreation());
}
}
```

Fichier XML Source : comptes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<compte>
  <code>1</code>
  <dateCreation>
    2014-01-16T12:33:16.960Z
  </dateCreation>
  <solde>8000.0</solde>
</compte>
```



# Générer un schéma XML à partir d'une classe avec JAXB

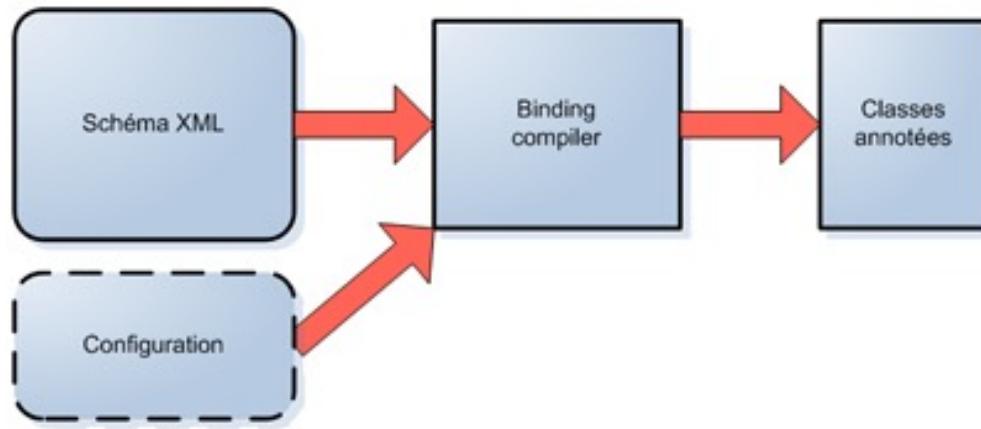
```
package ws;
import java.io.*;
import javax.xml.bind.*;import javax.xml.transform.Result;
import javax.xml.transform.stream.StreamResult;
public class Banque {
    public static void main(String[] args) throws Exception {
        JAXBContext context=JAXBContext.newInstance(Compte.class);
        context.generateSchema(new SchemaOutputResolver() {
            @Override
            public Result createOutput(String namespaceUri, String suggestedFileName)
                File f=new File("compte.xsd");
                StreamResult result=new StreamResult(f);
                result.setSystemId(f.getName());
                return result;
        });
    }
});}}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xss:schema version="1.0" xmlns:xss="http://www.w3.org/2001/XMLSchema">
    <xss:element name="compte" type="compte"/>
    <xss:complexType name="compte">
        <xss:sequence>
            <xss:element name="code" type="xs:int"/>
            <xss:element name="dateCreation" type="xs:dateTime" minOccurs="0"/>
            <xss:element name="solde" type="xs:float"/>
        </xss:sequence>
    </xss:complexType>
</xss:schema>
```

```
package ws;
import java.util.Date;
import javax.xml.bind.annotation.*;
@XmlRootElement
public class Compte {
    private int code;
    private float solde;
    private Date dateCreation;
    // Constructeur sans paramètre
    // Constructeur avec paramètres
    // Getters et Setters
}
```

## Génération des classes à partir d'un schéma XML avec XJC

- ▶ Pour permettre l'utilisation et la manipulation d'un document XML, JAXB propose de générer un ensemble de classes à partir du schéma XML du document.



C:\Windows\system32\cmd.exe

```
C:\Users\youssf i\w\JB2\src>xjc compte.xsd
parsing a schema...
compiling a schema...
generated\Compte.java
generated\ObjectFactory.java

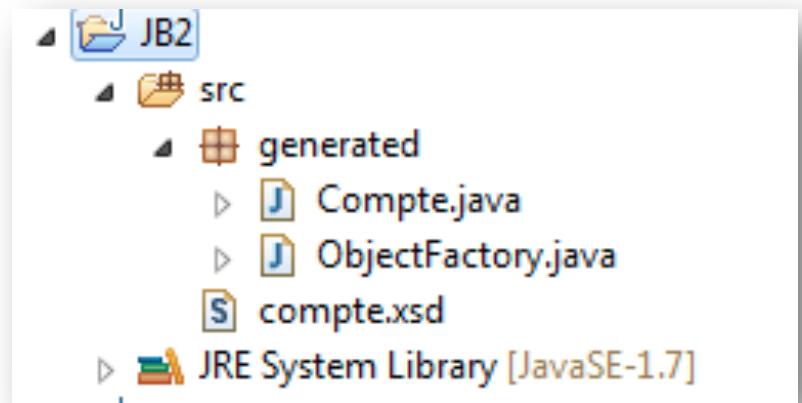
C:\Users\youssf i\w\JB2\src>
```

A screenshot of a Windows command prompt window titled 'cmd.exe'. The command entered is 'xjc compte.xsd'. The output shows the tool parsing the schema, compiling it, and generating two Java files: 'Compte.java' and 'ObjectFactory.java'. The command prompt then returns to the directory 'C:\Users\youssf i\w\JB2\src'.

- ▶ L'implémentation de référence fournit l'outil xjc pour générer les classes à partir d'un schéma XML.
- ▶ L'utilisation la plus simple de l'outil xjc est de lui fournir simplement le fichier qui contient le schéma XML du document à utiliser.

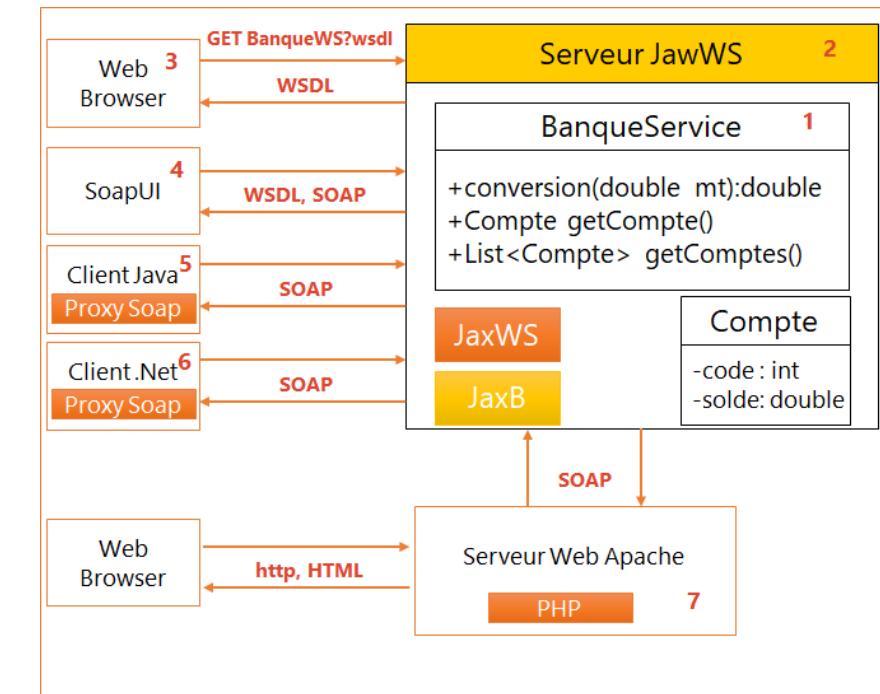
## Classes générées par XJC par XJC

- ▶ L'outil XJC génère deux classes :
  - Compte.java qui correspond au type complexe Compte dans le schéma xml.
  - ObjectFactory.java : une fabrique qui permet de créer des objets de type Compte.



# Quelques annotations JAXB

Annotation	Description
@XmlRootElement	Associer une classe ou une énumération à un élément XML
@XmlSchema	Associer un espace de nommage à un package
@XmlTransient	Marquer une entité pour ne pas être mappée dans le document XML
@XmlAttribute	Convertir une propriété en un attribut dans le document XML
@XmlElement	Convertir une propriété en un élément dans le document XML
@XmlAccessorType	Préciser comment un champ ou une propriété est sérialisé
@XmlNs	Associer un prefixe d'un espace de nommage à un URI



# Implémentation du Web Service JaxWS

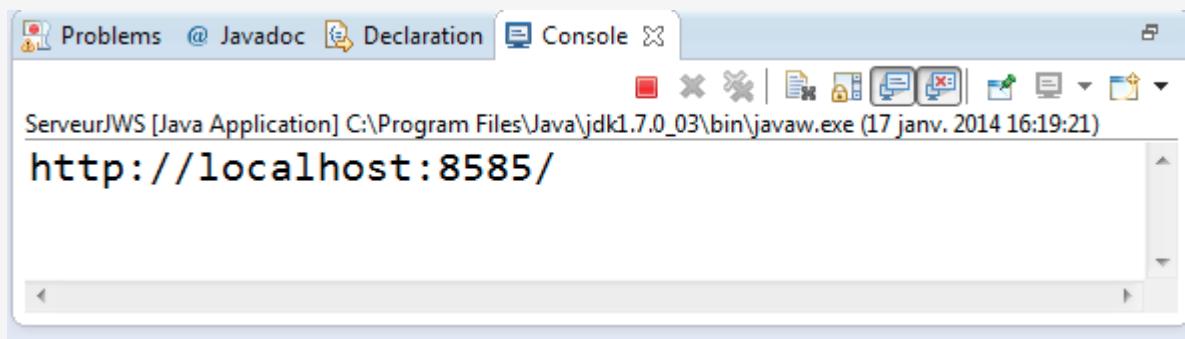
```
package ws;
import java.util.*;
import javax.jws.*;
import metier.Compte;
@WebService(serviceName="BanqueWS")
public class BanqueService {
    @WebMethod(operationName="ConversionEuroToDh")
    public double conversion(@WebParam(name="montant")double mt){
        return mt*11;
    }
    @WebMethod
    public Compte getCompte(@WebParam(name="code")Long code){
        return new Compte (code,7000,new Date());
    }
    @WebMethod
    public List<Compte> getComptes(){
        List<Compte> cptes=new ArrayList<Compte>();
        cptes.add (new Compte (1L,7000,new Date()));
        cptes.add (new Compte (2L,7000,new Date()));
        return cptes;
    }
}
```

```
package metier;
import java.util.Date;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
public class Compte {
    private Long code;
    private double solde;
    @XmlTransient
    private Date dateCreation;
    // Constructeurs
    // Getters et setters
}
```

# Simple Serveur JAX WS

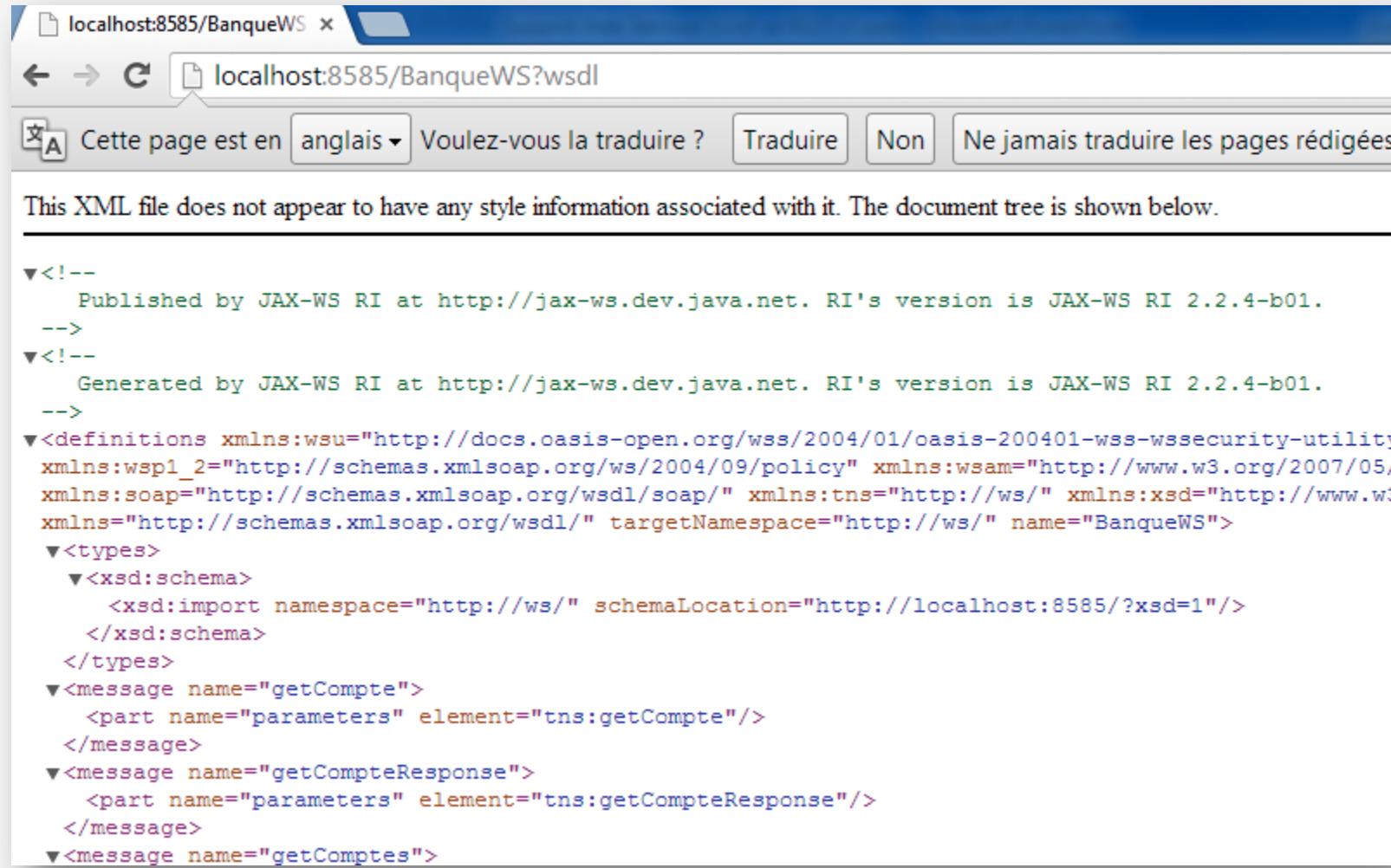
```
import javax.xml.ws.Endpoint;
import ws.BanqueService;
public class ServeurJWS {
public static void main(String[] args) {
    String url="http://localhost:8585/";
    Endpoint.publish(url, new BanqueService());
    System.out.println(url);
}
}
```

## Exécution du serveur Java



# Analyser le WSDL

Pour Visualiser le WSDL, vous pouvez utiliser un navigateur web



The screenshot shows a web browser window with the URL `localhost:8585/BanqueWS?wsdl`. The page content is an XML document representing the Web Services Description Language (WSDL) for the `BanqueWS` service.

```
<!--
  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
-->
<!--
  Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility"
  xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://ws/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://ws/" name="BanqueWS">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://ws/" schemaLocation="http://localhost:8585/?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="getCompte">
    <part name="parameters" element="tns:getCompte"/>
  </message>
  <message name="getCompteResponse">
    <part name="parameters" element="tns:getCompteResponse"/>
  </message>
  <message name="getComptes">
```

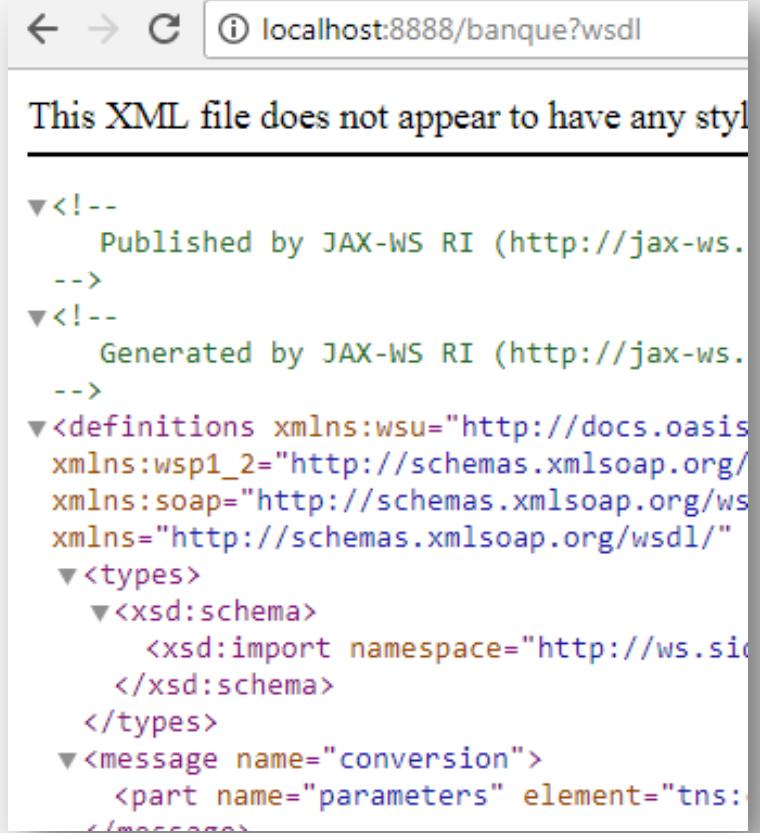
# Déployer un Web service dans un projet Spring Boot

## Web Service

```
@Component  
@WebService  
public class BanqueService {  
    // Opérations du web service  
}
```

## Classe de configuration

```
@Configuration  
public class MyConfig {  
    @Bean  
    public SimpleJaxWsServiceExporter getJWS() {  
        SimpleJaxWsServiceExporter exporter=new SimpleJaxWsServiceExporter();  
        exporter.setBaseAddress("http://0.0.0.0:8888/banque");  
        return exporter;  
    }  
}
```



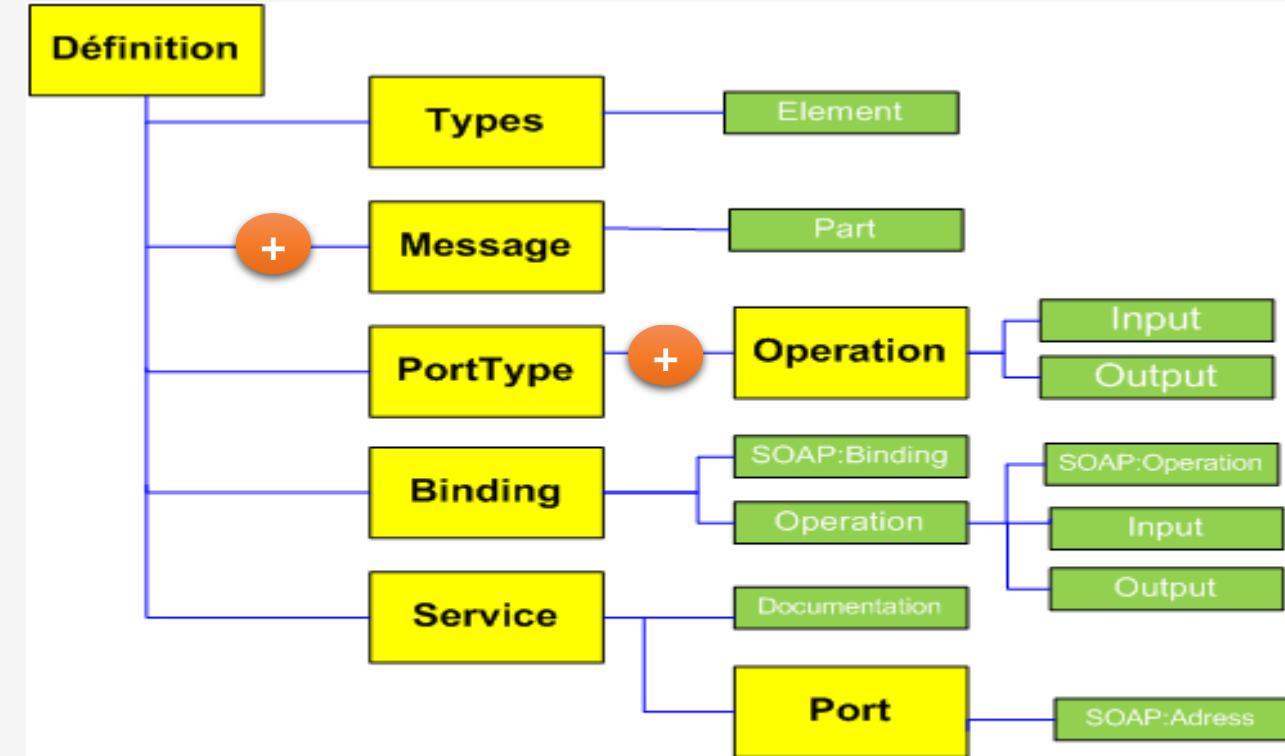
The screenshot shows a browser window with the URL `localhost:8888/banque?wsdl`. The page content is an XML document representing the Web Services Description Language (WSDL). The XML starts with a header indicating it was published by JAX-WS RI and generated by JAX-WS RI. It defines a service with a single message named "conversion" that has a part named "parameters". The XML uses various namespaces including wsu, wsp1\_2, soap, and wsdl.

```
<!-- Published by JAX-WS RI (http://jax-ws.java.net) -->  
<!-- Generated by JAX-WS RI (http://jax-ws.java.net) -->  
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/ws-securityutils" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/02/policy/1_2" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns="http://schemas.xmlsoap.org/wsdl/">  
    <types>  
        <xsd:schema>  
            <xsd:import namespace="http://ws.svc.idm.oclc.org/Banque"/>  
        </xsd:schema>  
    </types>  
    <message name="conversion">  
        <part name="parameters" element="tns:conversion"/>  
    </message>
```

# Structure du WSDL

Un document WSDL se compose d'un ensemble d'éléments décrivant les types de données utilisés par le service, les messages que le service peut recevoir, ainsi que les liaisons SOAP associées à chaque message.

Le schéma suivant illustre la structure du langage WSDL qui est un document XML, en décrivant les relations entre les sections constituant un document WSDL.



Structure d'un document WSDL

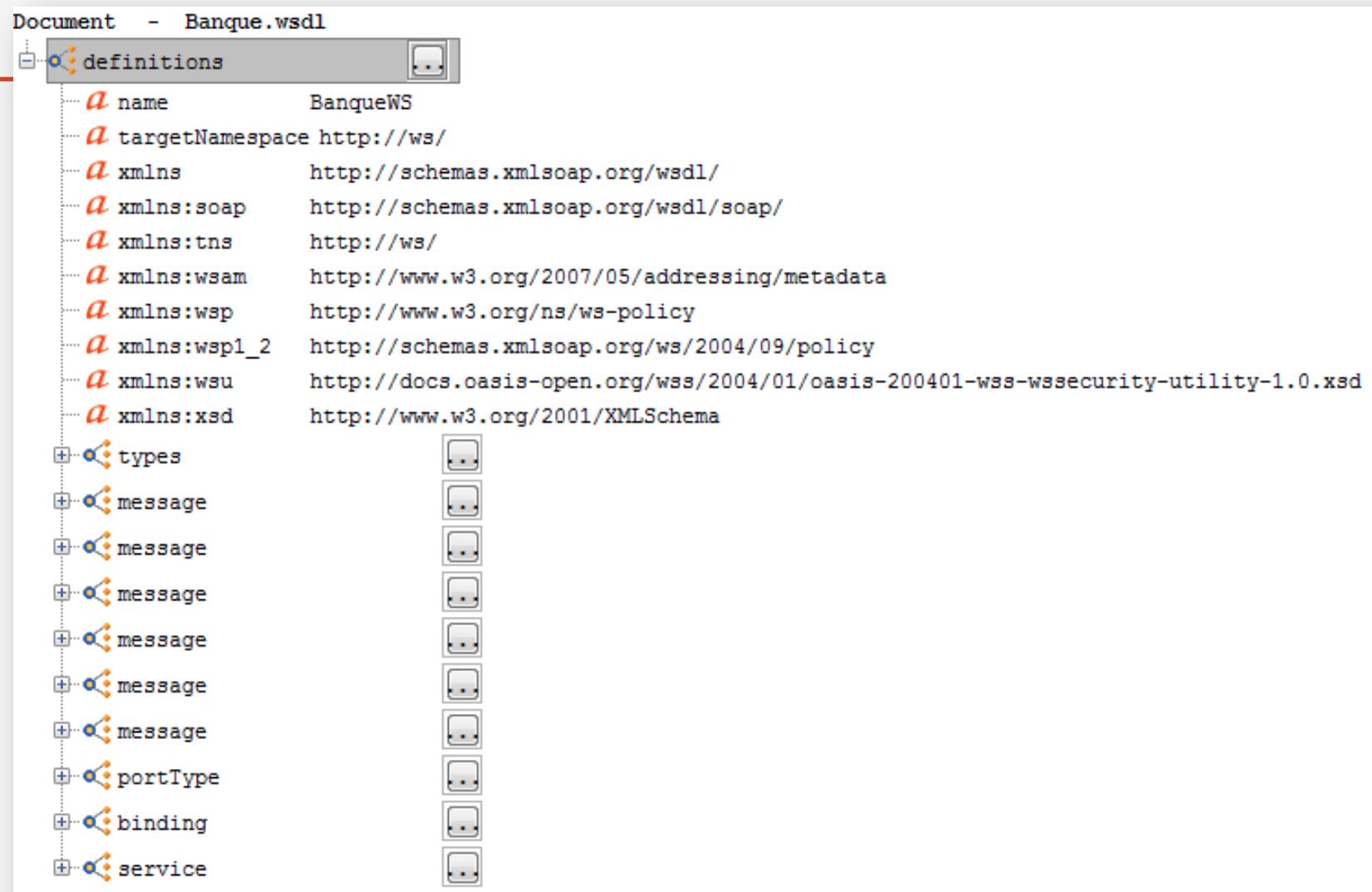
# Structure du WSDL

---

Un fichier WSDL contient donc sept éléments.

- **Types** : fournit la définition de types de données utilisés pour décrire les messages échangés.
- **Messages** : représente une définition abstraite (noms et types) des données en cours de transmission.
- **PortTypes** : décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou d'erreurs.
- **Binding** : spécifie une liaison entre un `<portType>` et un protocole concret (SOAP, HTTP...).
- **Service** : indique les adresses de port de chaque liaison.
- **Port** : représente un point d'accès de services défini par une adresse réseau et une liaison.
- **Opération** : c'est la description d'une action exposée dans le port.

# Structure du WSDL



# Elément Types

---



# XML Schema

localhost:8585/?xsd=1

Cette page est en [anglais](#) ▾ Voulez-vous la traduire ? [Traduire](#) [Non](#) [Ne jamais traduire les pages rédigées en anglais](#)

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!--
  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
-->
<xss:schema xmlns:tns="http://ws/" xmlns:xss="http://www.w3.org/2001/XMLSchema" version="1.0" targetNamespace="http://ws/">
  <xss:element name="ConversionEuroToDh" type="tns:ConversionEuroToDh"/>
  <xss:element name="ConversionEuroToDhResponse" type="tns:ConversionEuroToDhResponse"/>
  <xss:element name="compte" type="tns:compte"/>
  <xss:element name="getCompte" type="tns:getCompte"/>
  <xss:element name="getCompteResponse" type="tns:getCompteResponse"/>
  <xss:element name="getComptes" type="tns:getComptes"/>
  <xss:element name="getComptesResponse" type="tns:getComptesResponse"/>
  <xss:complexType name="ConversionEuroToDh">
    <xss:sequence>
      <xss:element name="montant" type="xs:double"/>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="ConversionEuroToDhResponse">
    <xss:sequence>
      <xss:element name="return" type="xs:double"/>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="getCompte">
    <xss:sequence>
      <xss:element name="code" type="xs:long" minOccurs="0"/>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="getCompteResponse">
    <xss:sequence>
      <xss:element name="return" type="tns:compte" minOccurs="0"/>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="compte">
    <xss:sequence>
      <xss:element name="code" type="xs:long" minOccurs="0"/>
      <xss:element name="solde" type="xs:double"/>
    </xss:sequence>
  </xss:complexType>
  <xss:complexType name="getComptes">
    <xss:sequence/>
  </xss:complexType>
  <xss:complexType name="getComptesResponse">
    <xss:sequence>
      <xss:element name="return" type="tns:compte" minOccurs="0" maxOccurs="unbounded"/>
    </xss:sequence>
  </xss:complexType>
</xss:schema>
```

# XML Schema

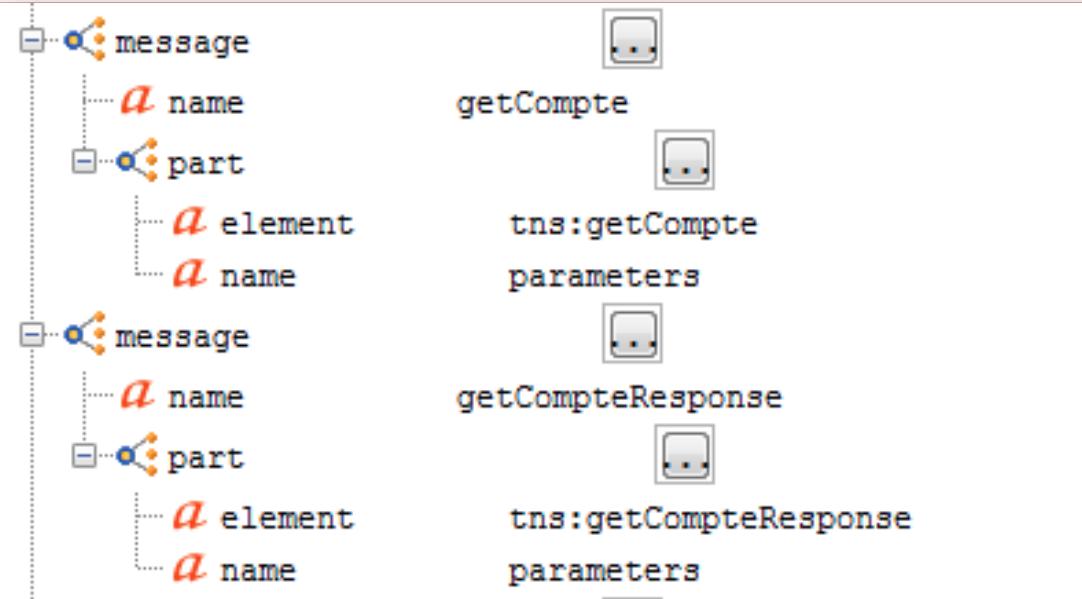
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:tns="http://ws/" xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
targetNamespace="http://ws/">
<xs:element name="ConversionEuroToDh" type="tns:ConversionEuroToDh"></xs:element>
<xs:element name="ConversionEuroToDhResponse" type="tns:ConversionEuroToDhResponse"/>
<xs:element name="compte" type="tns:compte"></xs:element>
<xs:element name="getCompte" type="tns:getCompte"></xs:element>
<xs:element name="getCompteResponse" type="tns:getCompteResponse"></xs:element>
<xs:element name="getComptes" type="tns:getComptes"></xs:element>
<xs:element name="getComptesResponse" type="tns:getComptesResponse"></xs:element>
<xs:complexType name="ConversionEuroToDh">
<xs:sequence>
    <xs:element name="montant" type="xs:double"></xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="ConversionEuroToDhResponse">
<xs:sequence>
    <xs:element name="return" type="xs:double"></xs:element>
</xs:sequence>
</xs:complexType>
```

## XML Schema

---

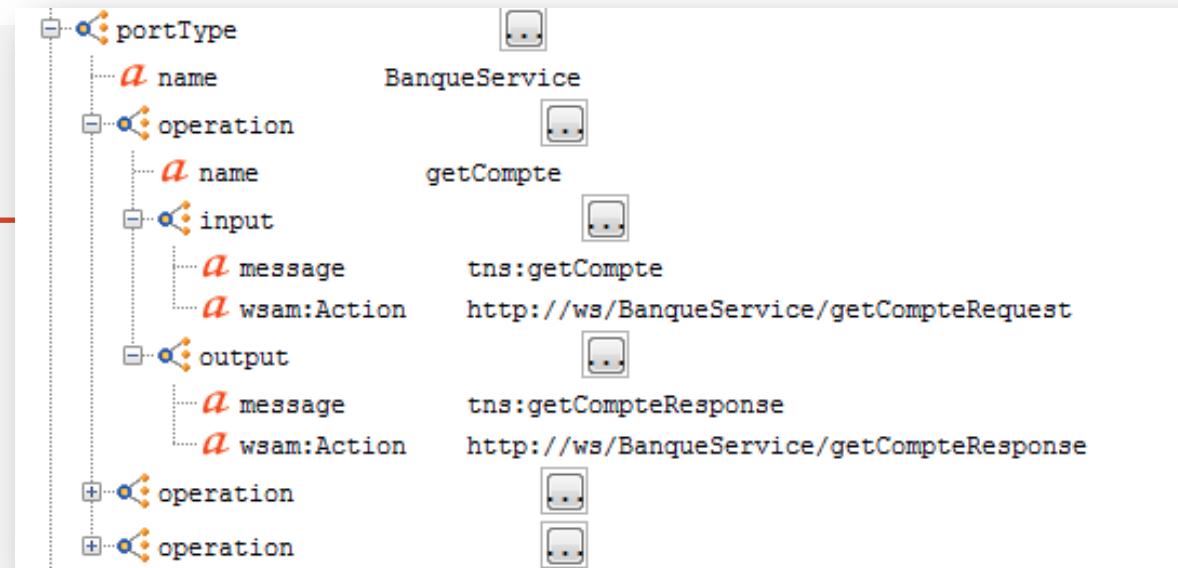
```
<xs:complexType name="getCompte">
<xs:sequence>
    <xs:element name="code" type="xs:long" minOccurs="0"></xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="getCompteResponse">
<xs:sequence>
    <xs:element name="return" type="tns:compte" minOccurs="0"></xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="compte">
<xs:sequence>
    <xs:element name="code" type="xs:long" minOccurs="0"></xs:element>
    <xs:element name="solde" type="xs:double"></xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="getComptes">
<xs:sequence></xs:sequence>
</xs:complexType>
<xs:complexType name="getComptesResponse">
<xs:sequence>
    <xs:element name="return" type="tns:compte" minOccurs="0" maxOccurs="unbounded"></xs:element>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

# Elément message



```
<message name="getCompte">
    <part name="parameters" element="tns:getCompte"></part>
</message>
<message name="getCompteResponse">
    <part name="parameters" element="tns:getCompteResponse"></part>
</message>
```

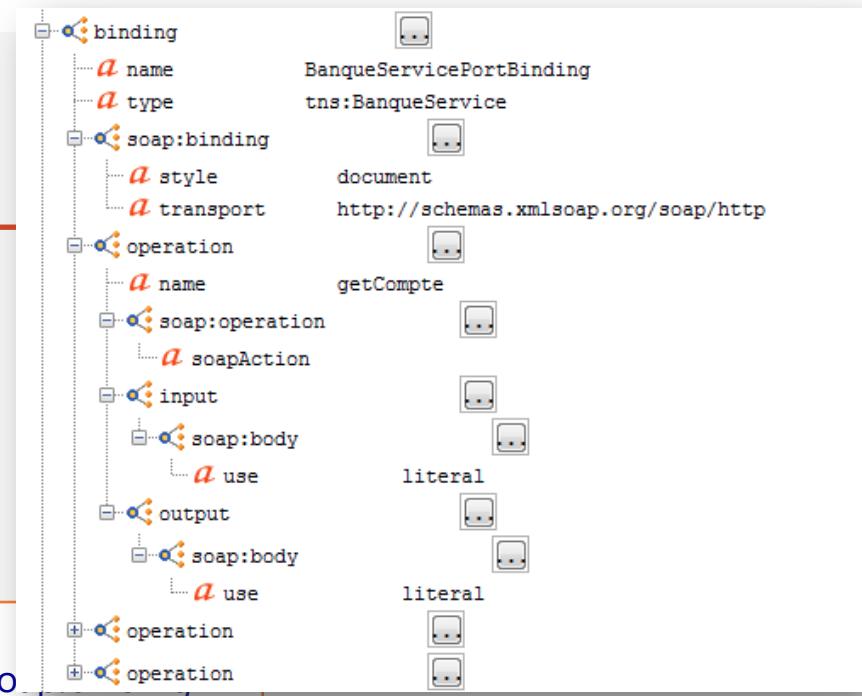
# Elément portType



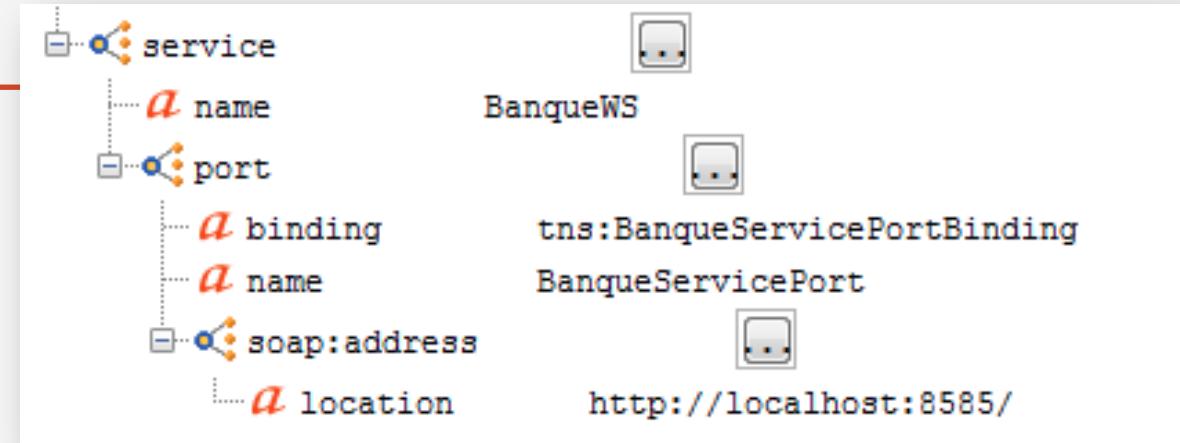
```
<portType name="BanqueService">
  <operation name="getCompte">
    <input wsam:Action="http://ws/BanqueService/getCompteRequest" message="tns:getCompte"></input>
    <output wsam:Action="http://ws/BanqueService/getCompteResponse"
           message="tns:getCompteResponse"></output>
  </operation>
  <operation name="getComptes">
    <input wsam:Action="http://ws/BanqueService/getComptesRequest" message="tns:getComptes"></input>
    <output wsam:Action="http://ws/BanqueService/getComptesResponse"
           message="tns:getComptesResponse"></output>
  </operation>
  <operation name="ConversionEuroToDh">
    ....
  </operation>
</portType>
```

# Elément binding

```
<binding name="BanqueServicePortBinding" type="tns:BanqueService">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"></so
<operation name="getCompte">
<soap:operation soapAction=""></soap:operation>
<input>
    <soap:body use="literal"></soap:body>
</input>
<output>
    <soap:body use="literal"></soap:body>
</output>
</operation>
<operation name="getComptes">
.....
</operation>
</binding>
```



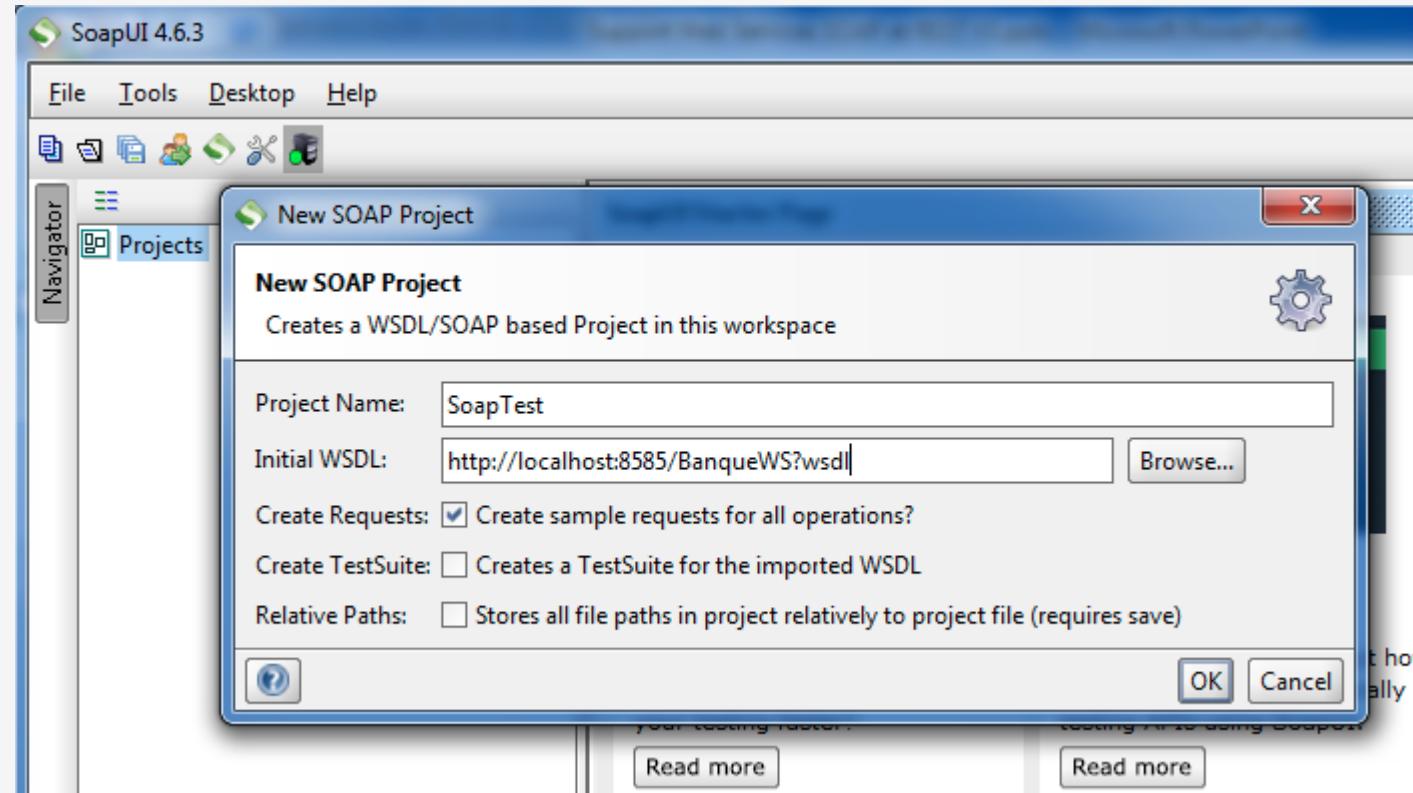
## Elément service



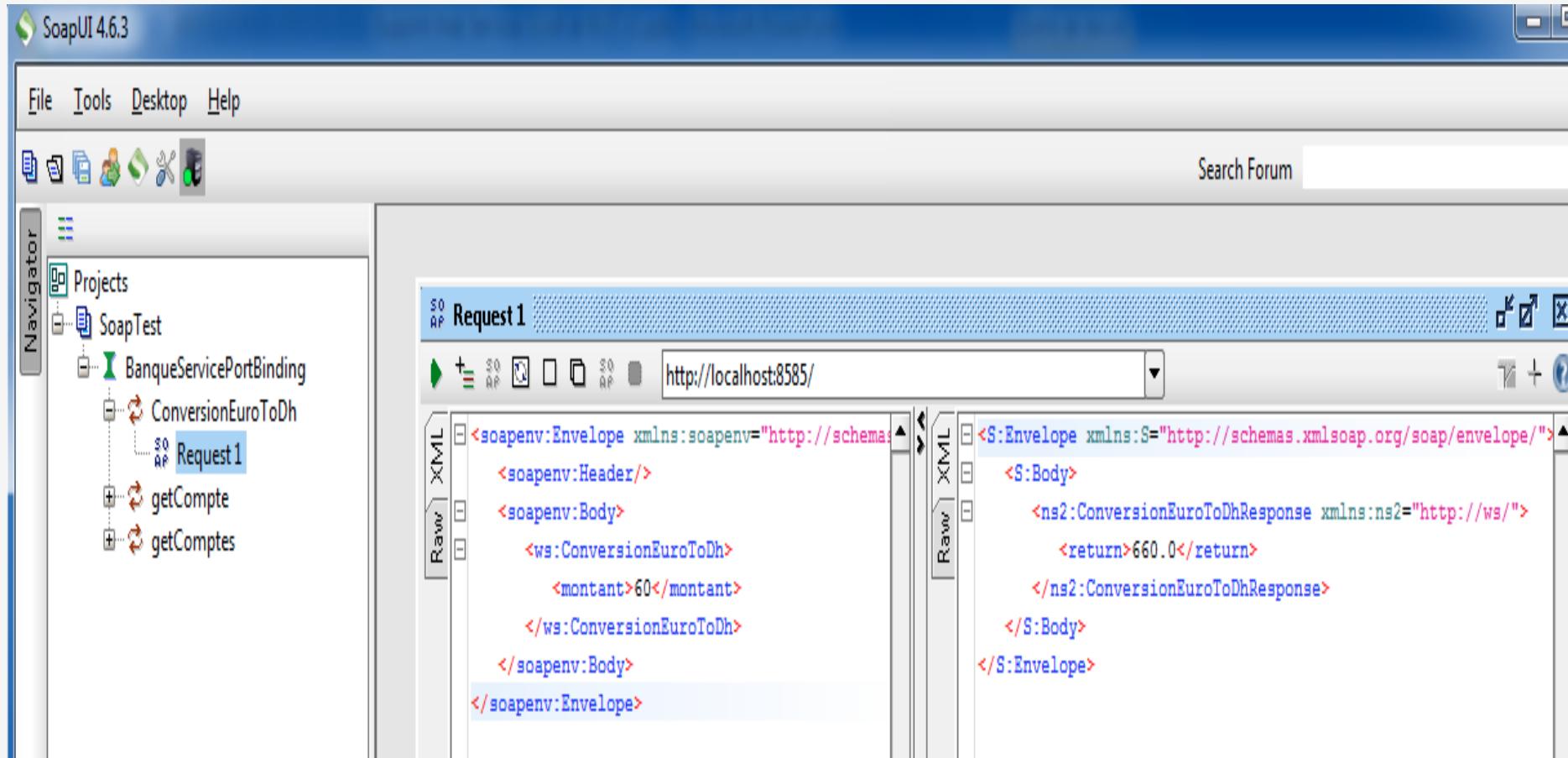
```
<service name="BanqueWS">
  <port name="BanqueServicePort"
    binding="tns:BanqueServicePortBinding">
    <soap:address location="http://localhost:8585/"></soap:address>
  </port>
</service>
```

# Tester les méthodes du web service avec un analyseur SOAP : SoapUI

---



# Tester les méthodes du web service avec un analyseur SOAP : SoapUI



# Tester la méthode conversion

## Requête SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws/">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:ConversionEuroToDh>
      <montant>60</montant>
    </ws:ConversionEuroToDh>
  </soapenv:Body>
</soapenv:Envelope>
```

## Réponse SOAP

- <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  
 <S:Body>  
 <ns2:ConversionEuroToDhResponse xmlns:ns2="http://ws/">  
 <return>660.0</return>  
 </ns2:ConversionEuroToDhResponse>  
 </S:Body>  
</S:Envelope>

# Tester la méthode getCompte

## Requête SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws/">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:getCompte>
      <code>2</code>
    </ws:getCompte>
  </soapenv:Body>
</soapenv:Envelope>
```

## Réponse SOAP

- <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  
 <S:Body>
 <ns2:getCompteResponse xmlns:ns2="http://ws/">
 <return>
 <code>2</code>
 <solde>7000.0</solde>
 </return>
 </ns2:getCompteResponse>
 </S:Body>
</S:Envelope>

# Tester la méthode getComptes

## Requête SOAP

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ws="http://ws/">
  <soapenv:Header/>
  <soapenv:Body>
    <ws:getComptes/>
  </soapenv:Body>
</soapenv:Envelope>
```

## Réponse SOAP

- <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">  
  <S:Body>  
    <ns2:getComptesResponse xmlns:ns2="http://ws/">  
      <return>  
        <code>1</code>  
        <solde>7000.0</solde>  
      </return>  
      <return>  
        <code>2</code>  
        <solde>7000.0</solde>  
      </return>  
    </ns2:getComptesResponse>  
  </S:Body>  
</S:Envelope>

# Client Java

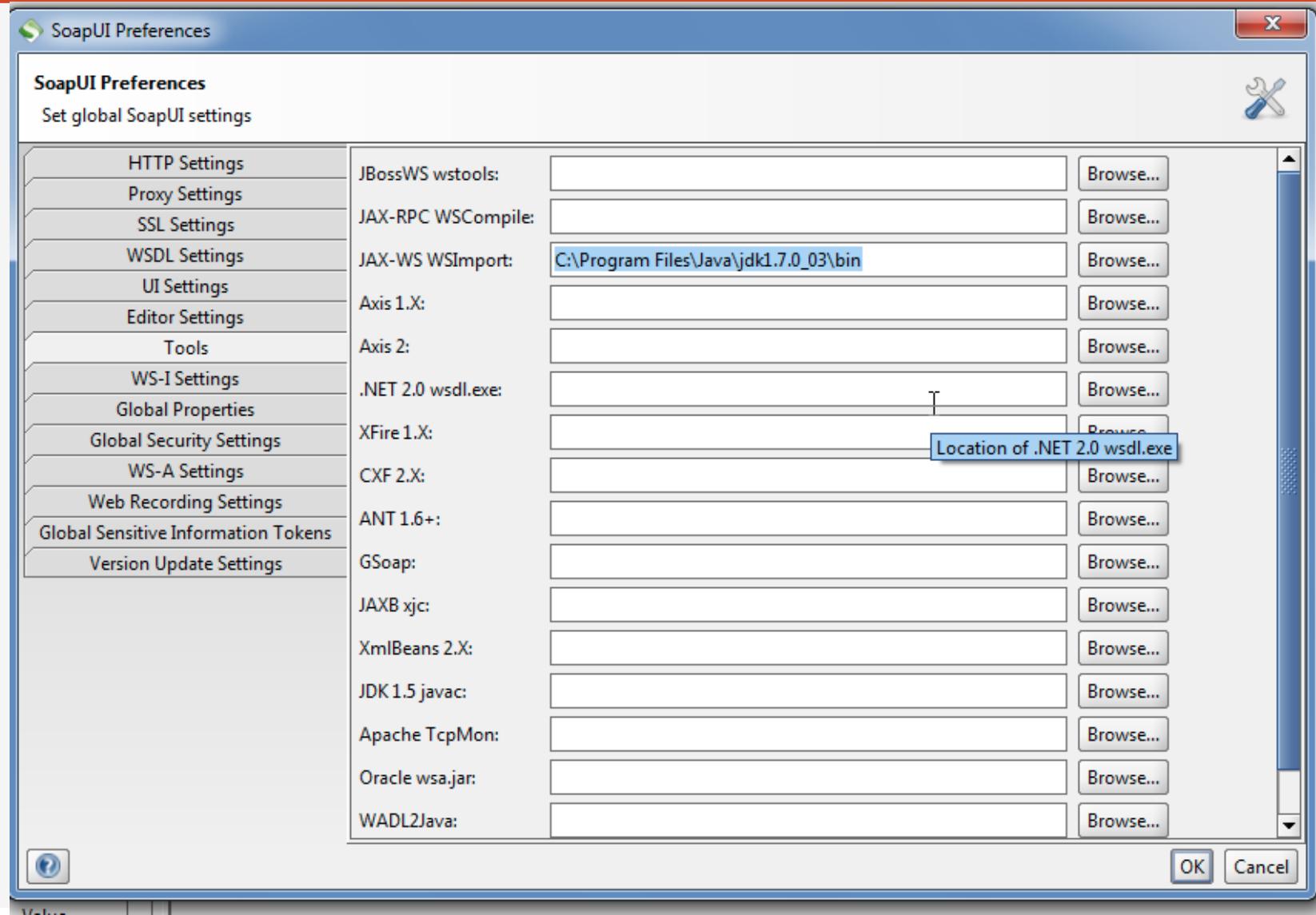
---

Créer un nouveau projet Java

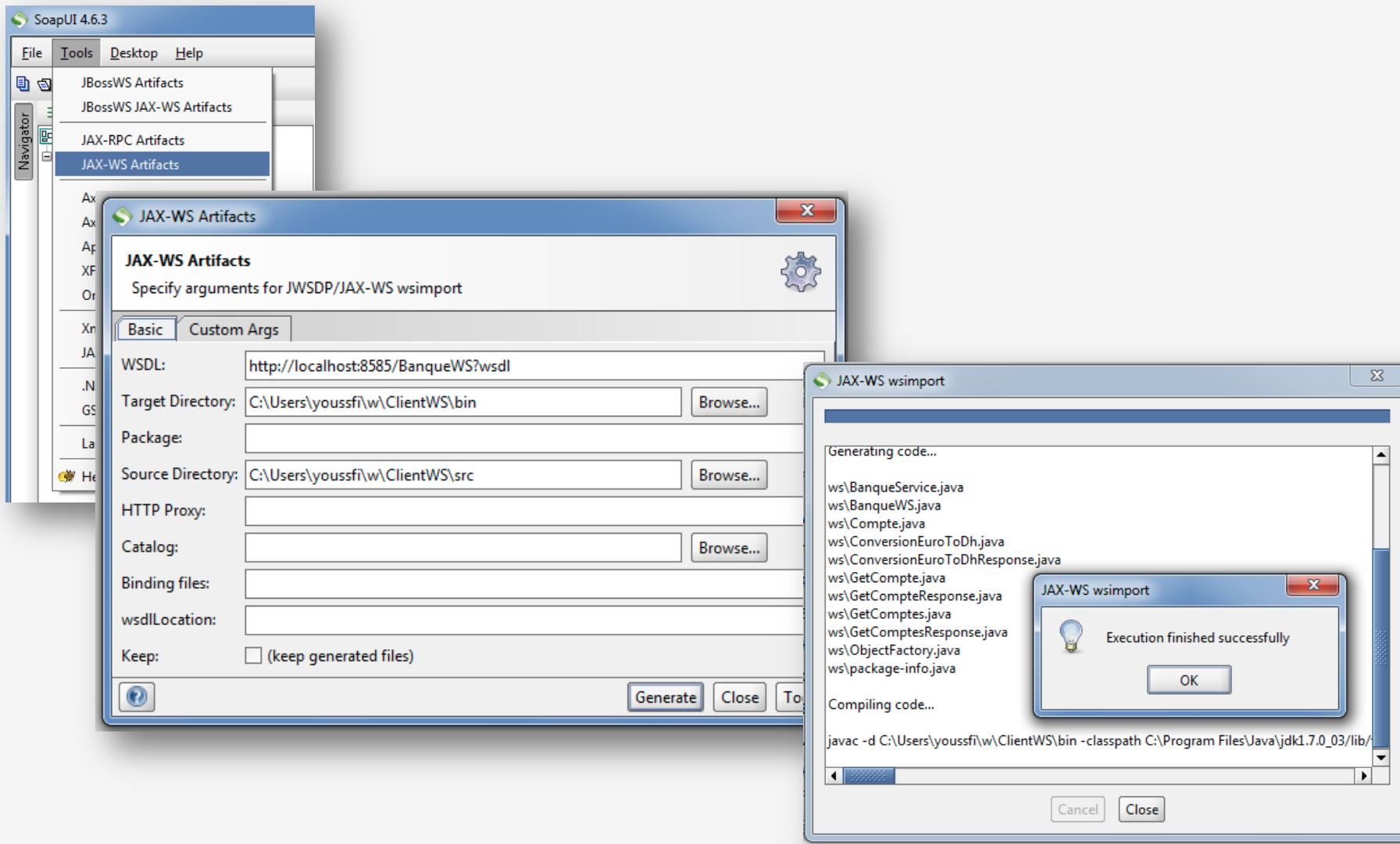
Générer un proxy

- SoapUI est l'un des outils qui peuvent être utilisés pour générer les artefacts client en utilisant différents Framework (Axis, CXF, JaxWS, etc...)
- Le JDK fournit une commande simple qui permet de générer un STUB JaxWS pour l'accès à un web service. Cette commande s'appelle **wsimport**.
- SoapUI a besoin de savoir le chemin de cette commande
- Avec la commande `File > Preferences > Tools`, vous pouvez configurer ce chemin comme le montre la figure suivante :

# Préférence générale de SoapUI

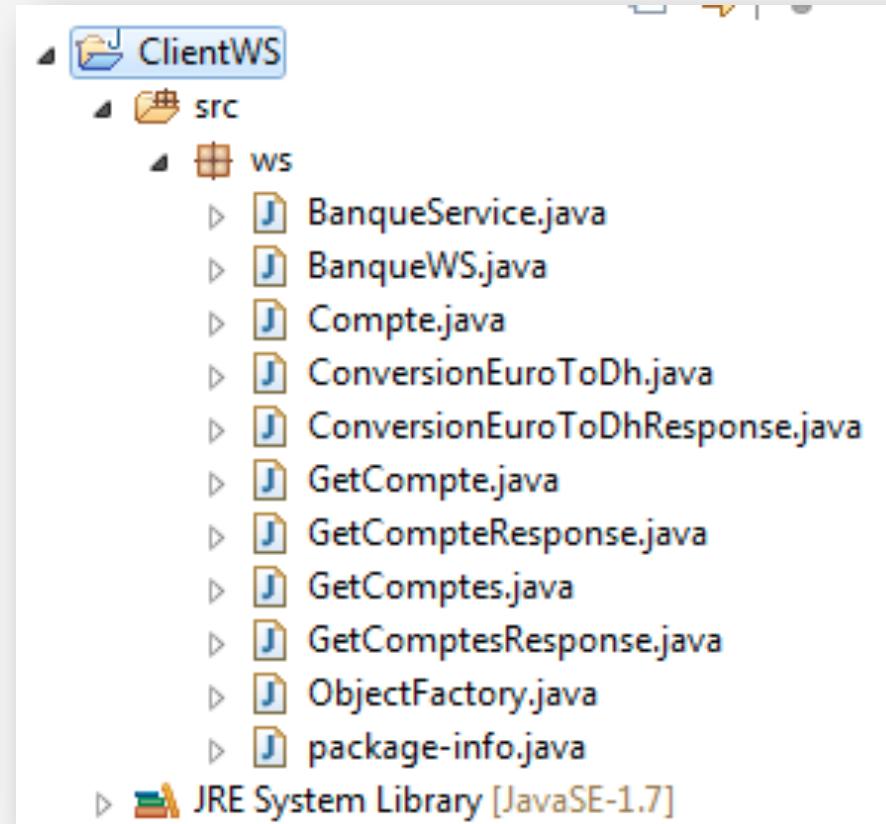


# Générer le STUB JaxWS



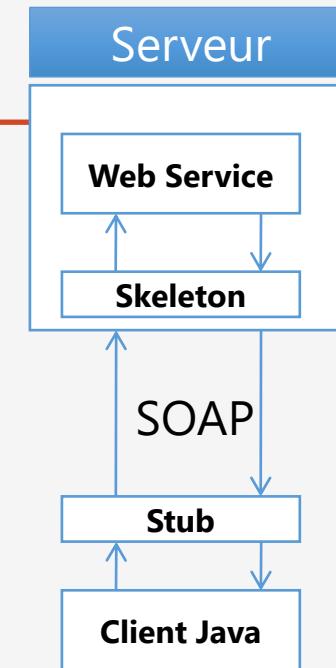
## Fichiers Générés

---



## Client Java

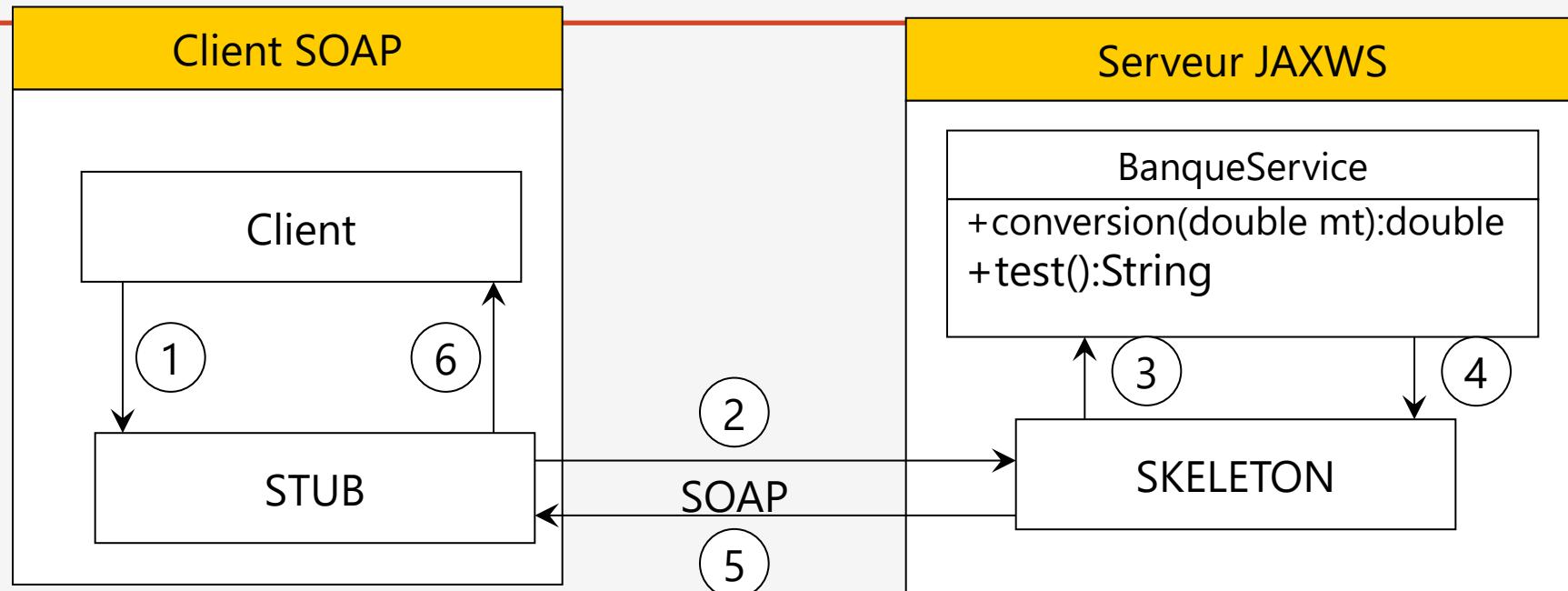
```
import java.util.List;
import ws.BanqueService;
import ws.BanqueWS;
import ws.Compte;
public class ClientWS {
public static void main(String[] args) {
BanqueService stub=new BanqueWS().getBanqueServicePort();
System.out.println("Conversion");
System.out.println(stub.conversionEuroToDh(9000));
System.out.println("Consulter un compte");
Compte cp=stub.getCompte(2L);
System.out.println("Solde="+cp.getSolde());
System.out.println("Liste des comptes");
List<Compte> cptes=stub.getComptes();
for(Compte c:cptes){
System.out.println(c.getCode()+"----"+c.getSolde());
}
}
```



```
Problems @ Javadoc Declaration Console
<terminated> ClientWS (1) [Java Application] C:\Program Files\Java\
Conversion
99000.0
Consulter un compte
Solde=7000.0
Liste des comptes
1----7000.0
2----7000.0
```

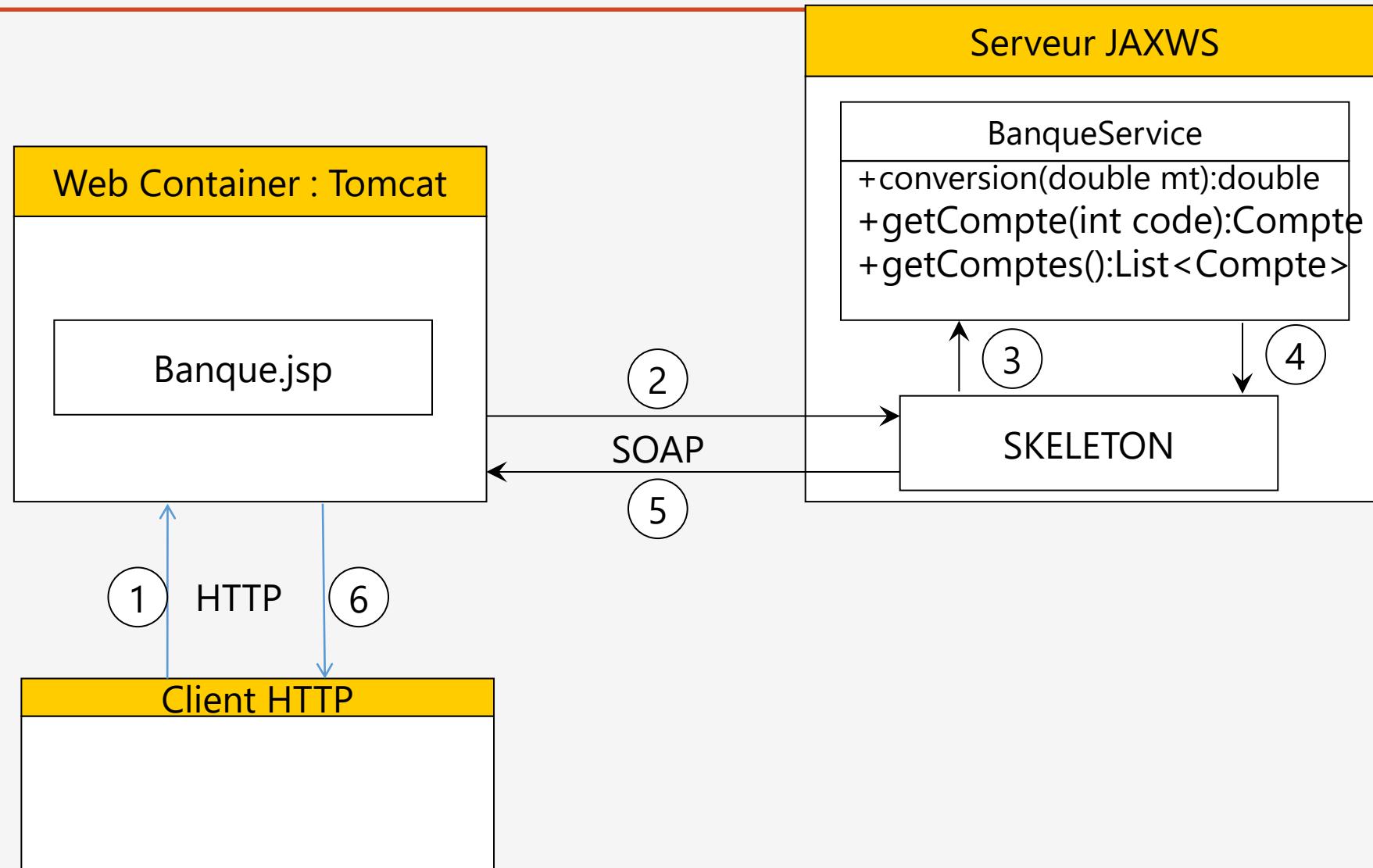
The screenshot shows the Eclipse IDE's Console view displaying the output of the Client Java application. The output includes the results of the conversion method and the list of accounts, each with its code and balance.

# Architecture



- 1 Le client demande au stub de faire appel à la méthode conversion(12)
- 2 Le Stub se connecte au Skeleton et lui envoie une requête SOAP
- 3 Le Skeleton fait appel à la méthode du web service
- 4 Le web service retourne le résultat au Skeleton
- 5 Le Skeleton envoie le résultat dans une la réponse SOAP au Stub
- 6 Le Stub fournie le résultat au client

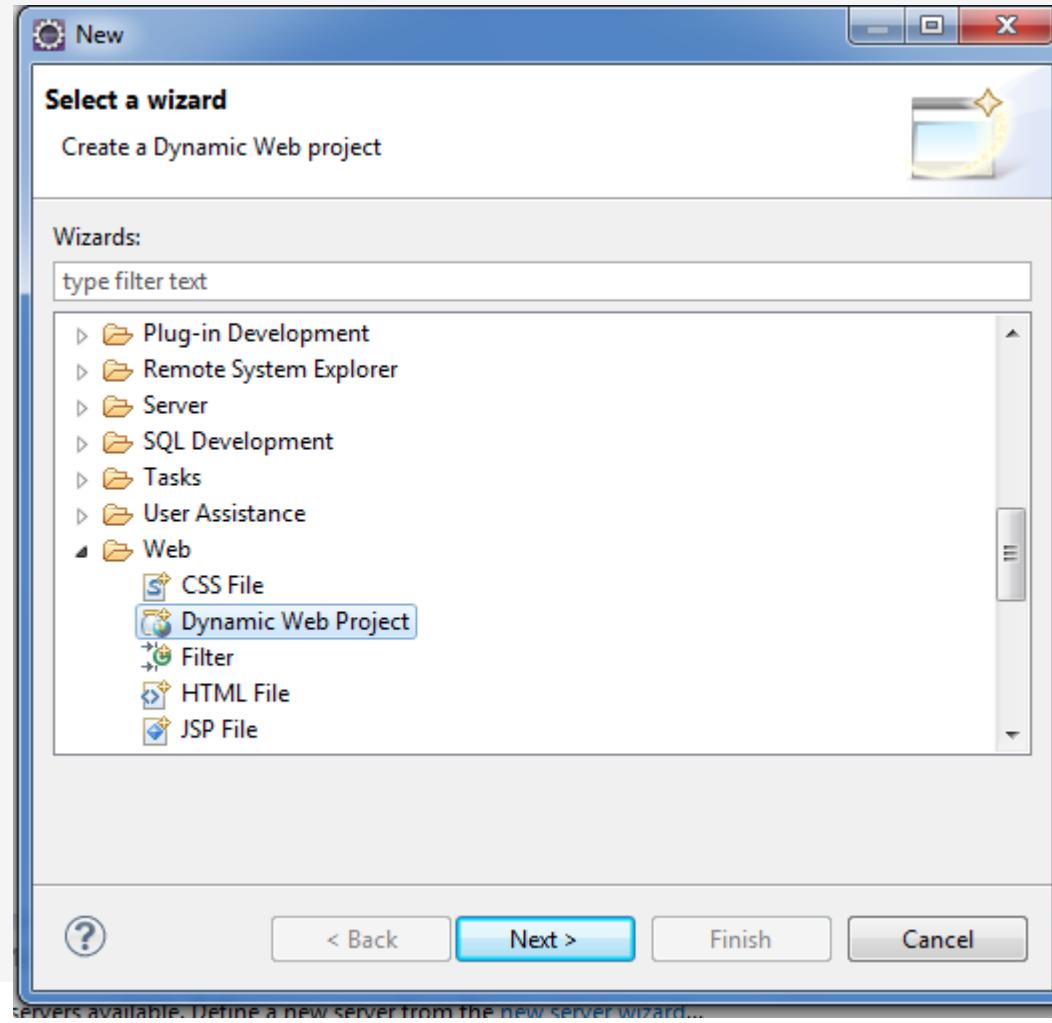
# Client JSP



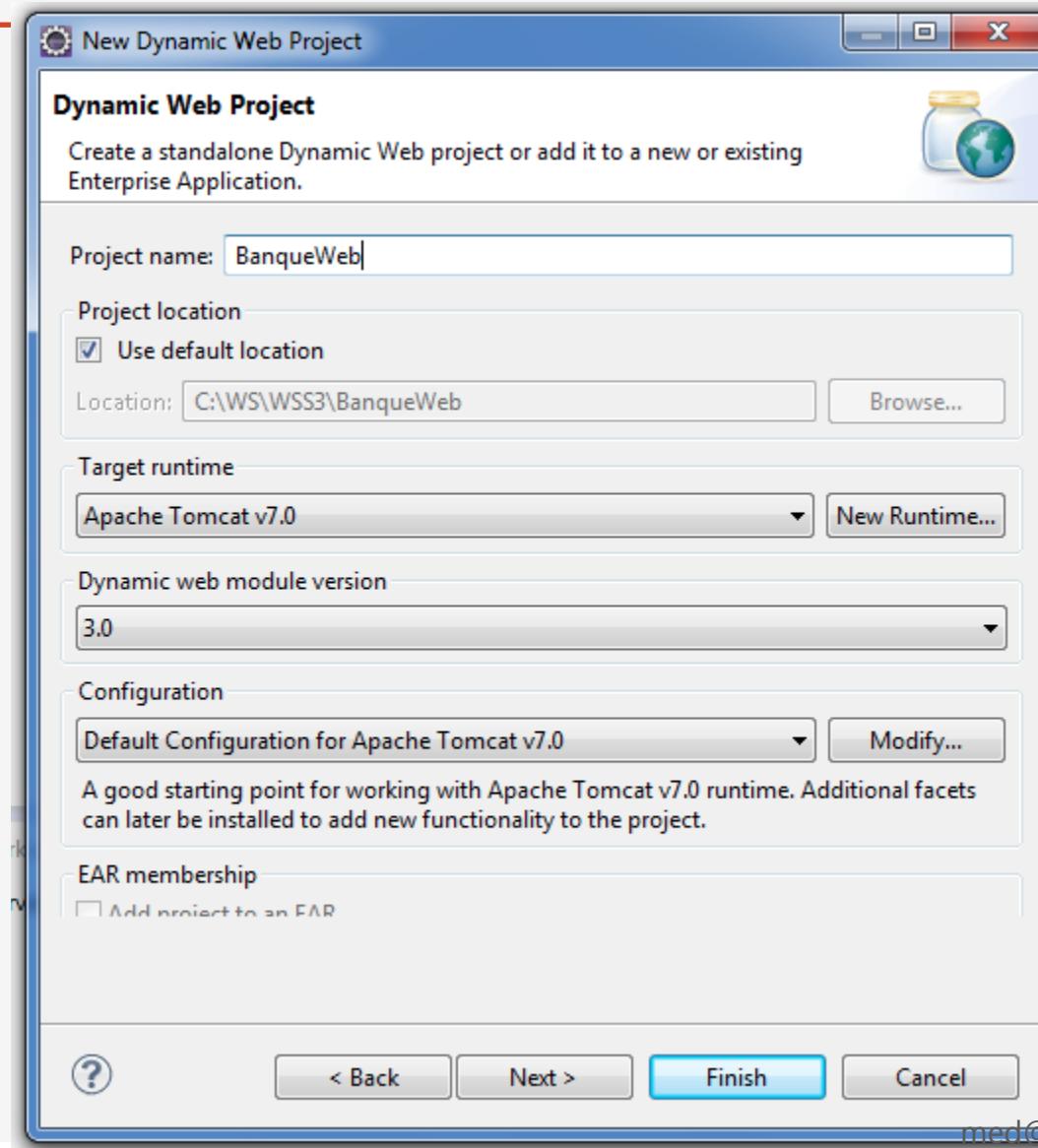
# Création d'un projet Web Dynamique basé sur Tomcat 7

---

## Créer un projet Web Dynamique basé sur Tomcat 7



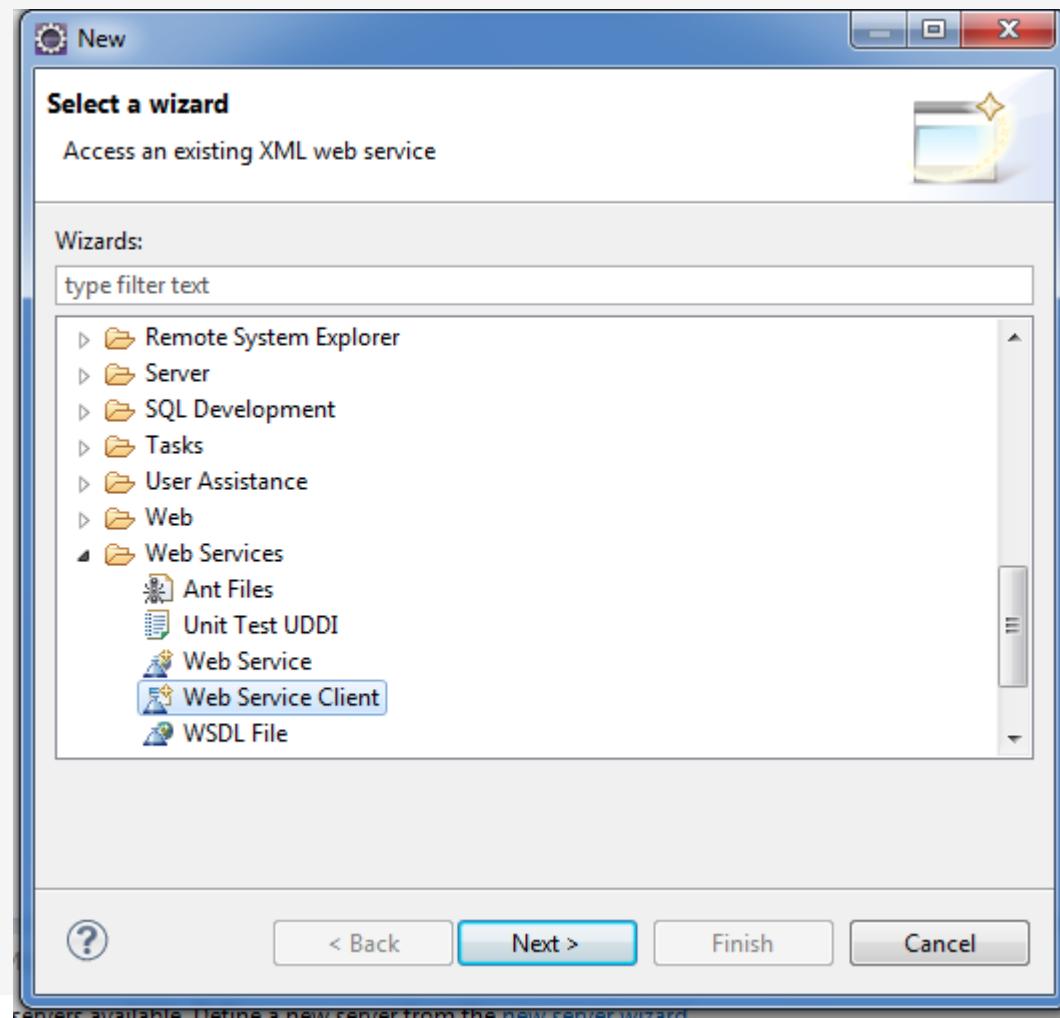
# Projet Web Dynamique



# Générer un proxy (Stub) pour le web Service

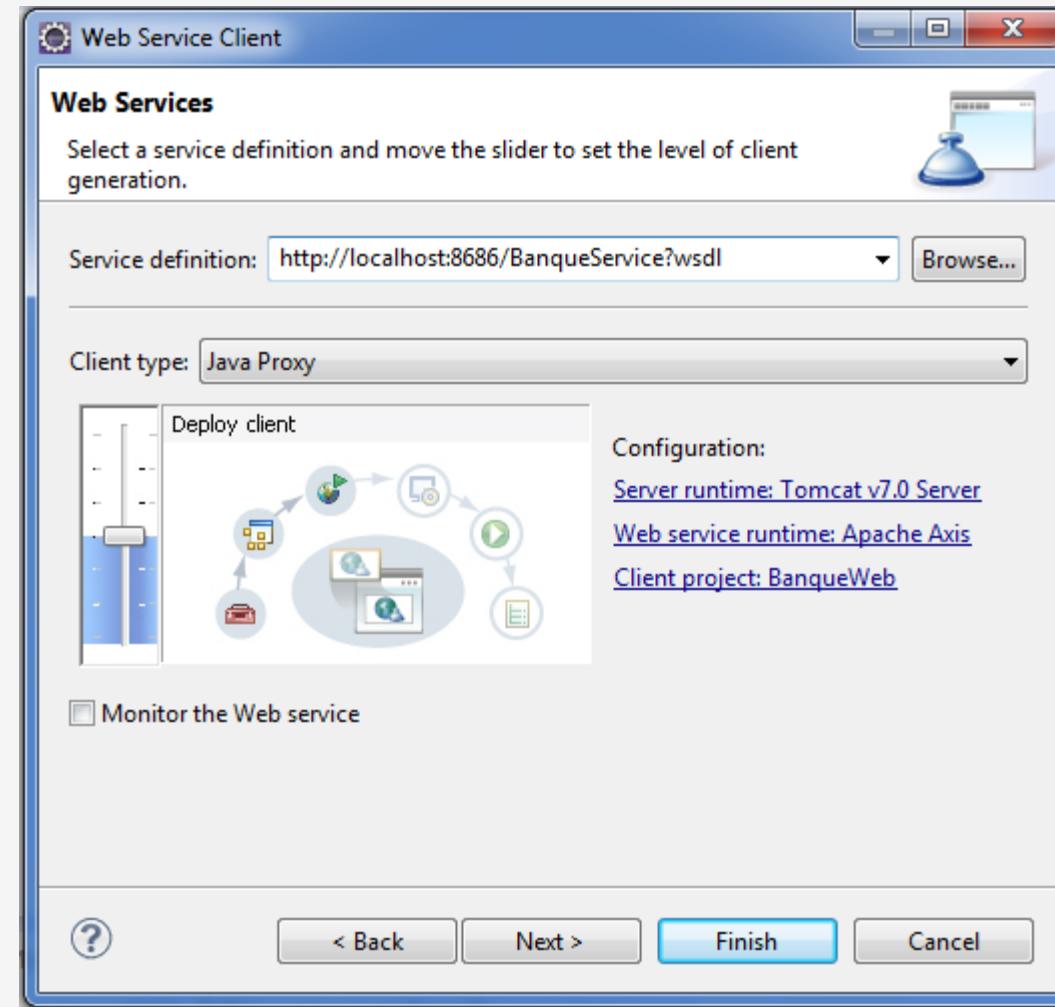
---

Fichier > Nouveau > Web Service Client



# Génération du proxy

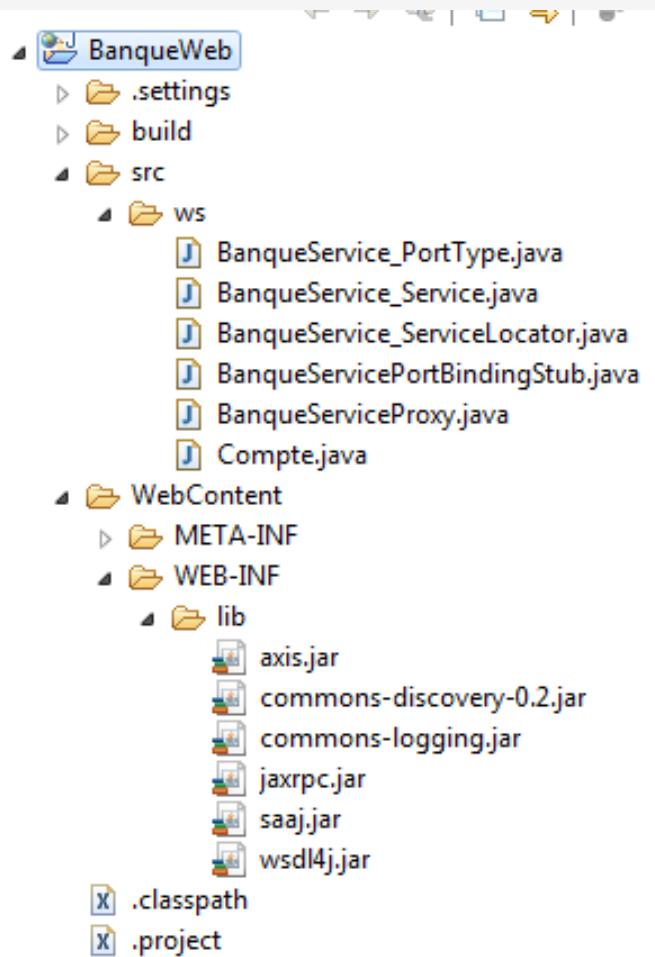
## Générer le proxy à partir du WSDL



## Fichier Générés

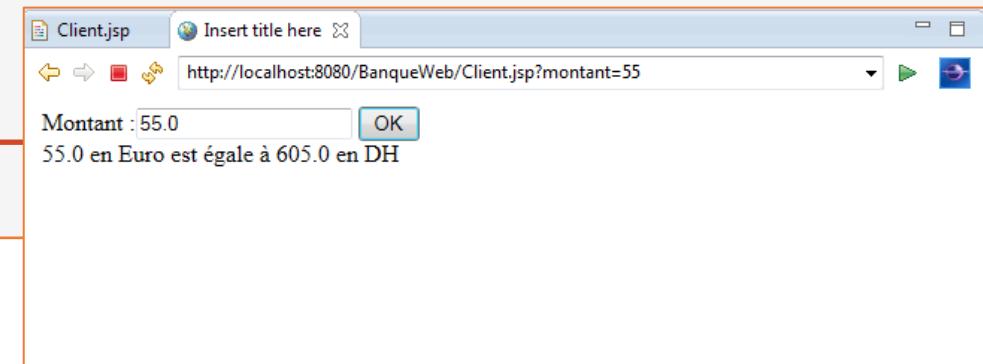
---

Le proxy généré est basé sur AXIS



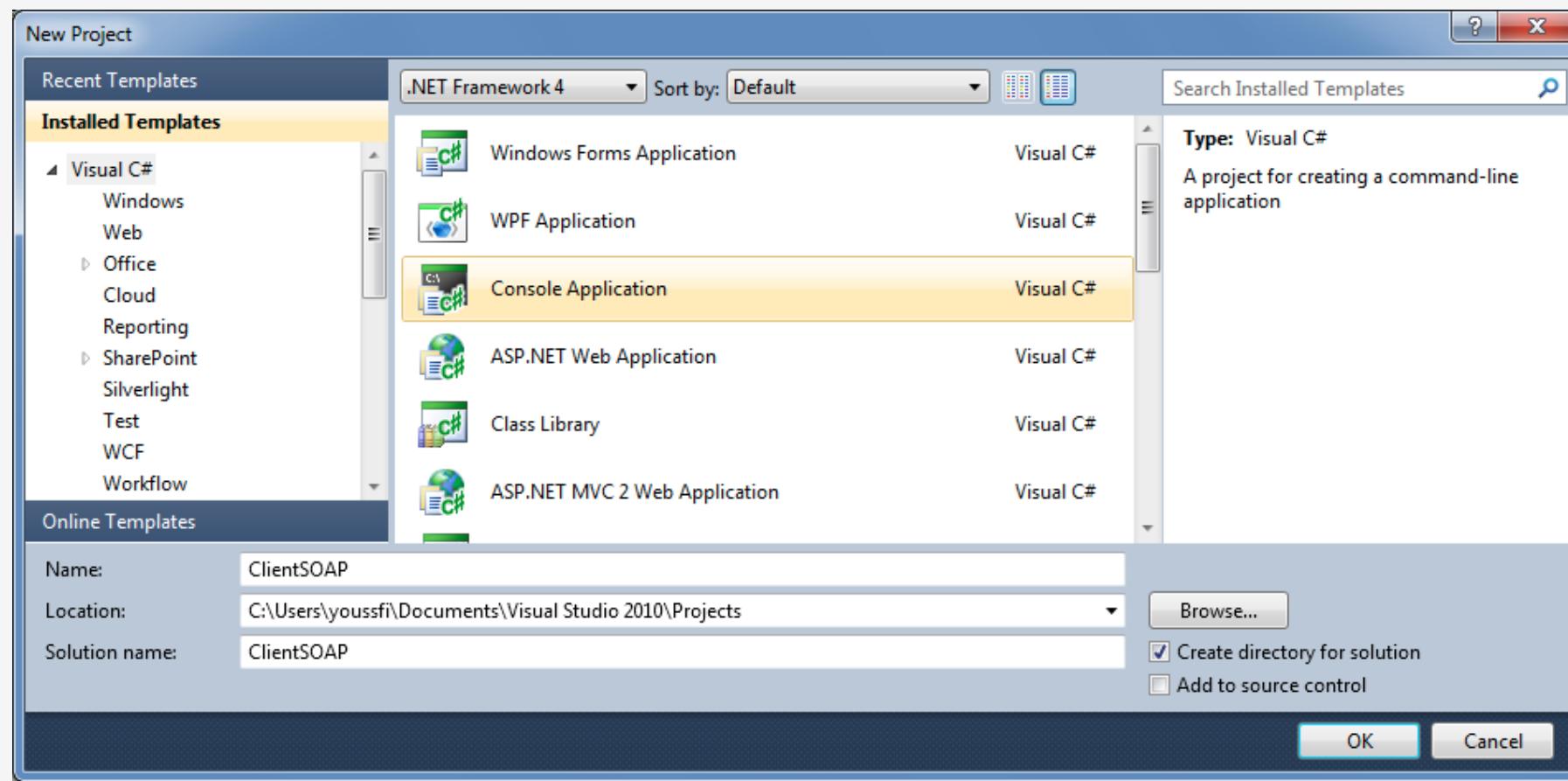
## Client JSP

```
<%@page import="ws.BanqueServiceProxy"%>
<%
    double montant=0;  double resultat=0;
    if(request.getParameter("montant")!=null){
        montant=Double.parseDouble(request.getParameter("montant"));
        BanqueServiceProxy service=new BanqueServiceProxy();
        resultat=service.conversionEuroDH(montant);
    }
%>
<html><body>
<form action="Client.jsp">
    Montant :<input type="text" name="montant" value="<%=montant%>">
    <input type="submit" value="OK">
</form>
<%=montant %> en Euro est égale à <%=resultat %> en DH
</body></html>
```

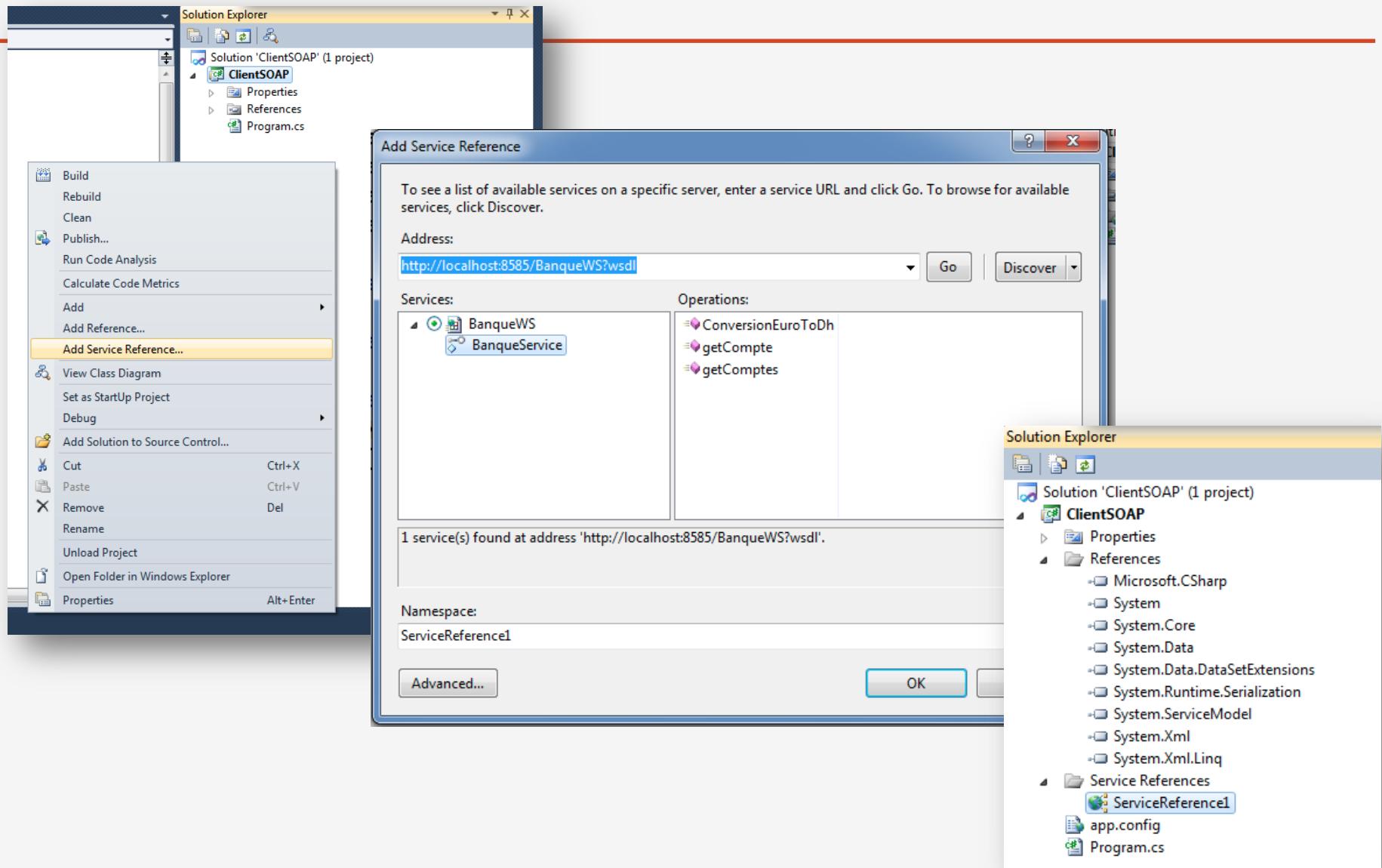


# Client SOAP avec .Net

En utilisant Visual Studio (2010), Créez un projet C# de type Console

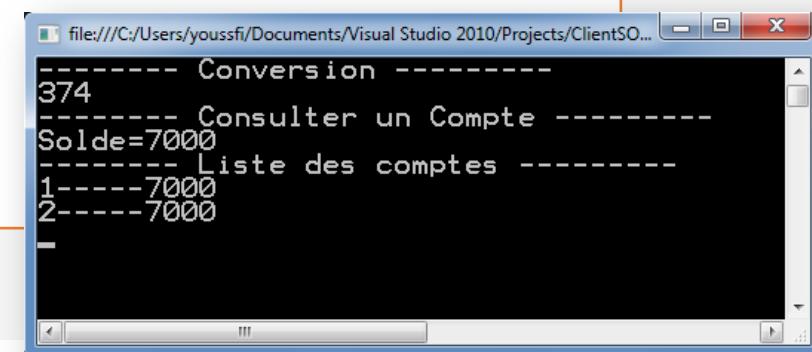
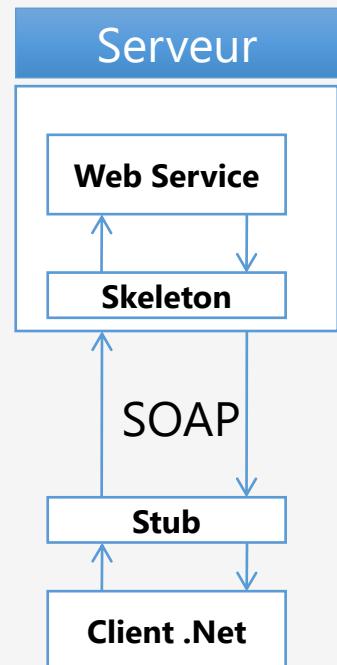


# Générer Le Proxy client Soap



# Code C# du client

```
using System;
namespace ClientSOAP
{
    class Program
    {
        static void Main(string[] args)
        {
            ServiceReference1.BanqueServiceClient stub =
                new ServiceReference1.BanqueServiceClient();
            Console.WriteLine("----- Conversion -----");
            Console.WriteLine(stub.ConversionEuroToDh(34));
            Console.WriteLine("----- Consulter un Compte -----");
            ServiceReference1.compte cp = stub.getCompte(2L);
            Console.WriteLine("Solde=" + cp.solde);
            Console.WriteLine("----- Liste des comptes -----");
            ServiceReference1.compte[] cptes = stub.getComptes();
            for (int i = 0; i < cptes.Length; i++)
            {
                Console.WriteLine(cptes[i].code + "----" + cptes[i].solde);
            }
            Console.ReadLine();
        }
    }
}
```



# Dessiner les composants graphique de l'interface

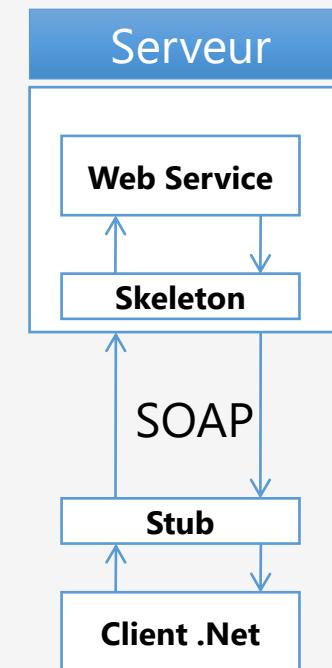
---

Form1

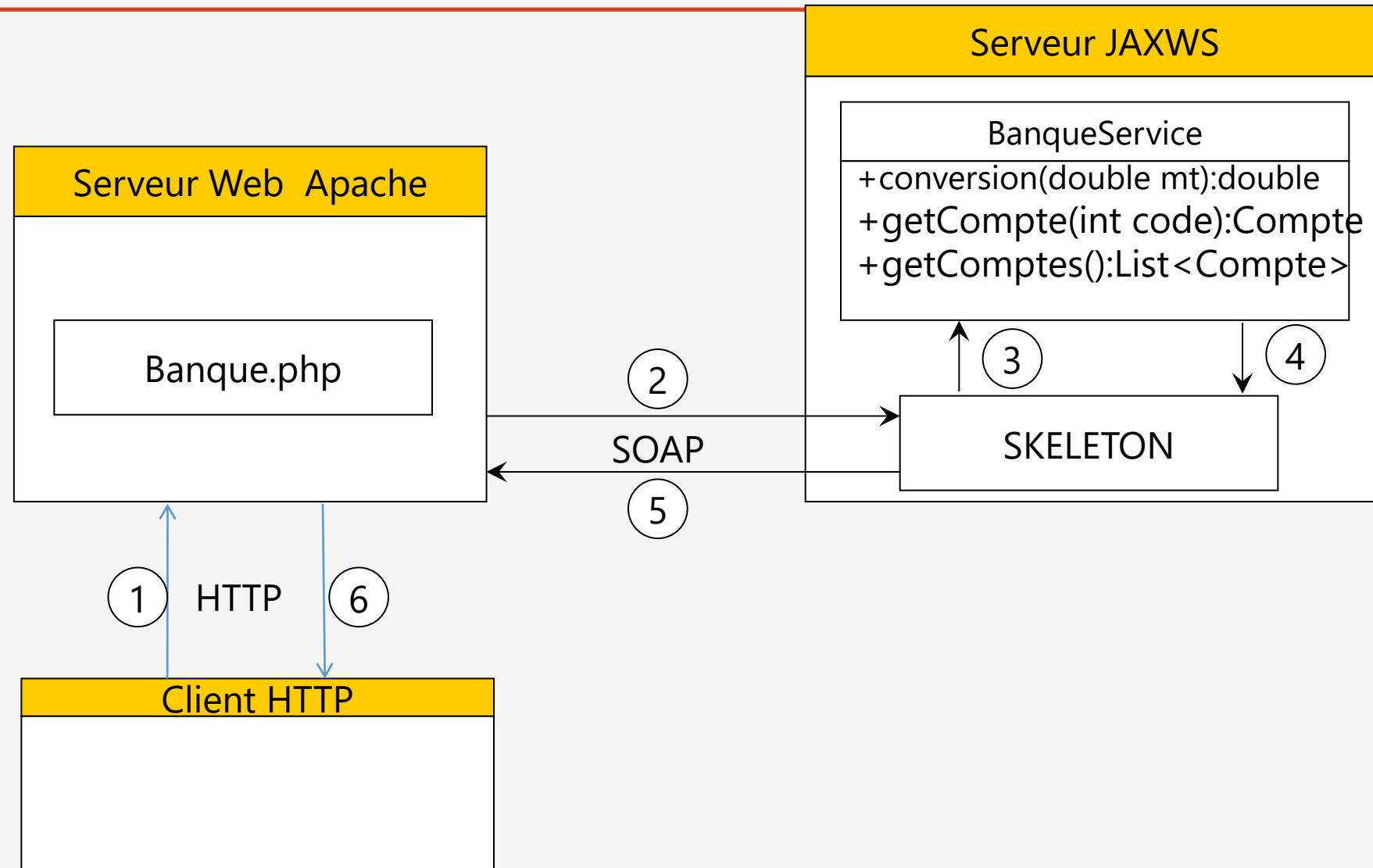
Montant:  Conversion Comptes

Résultat:

	CODE	SOLDE
▶	1	7000
	2	7000
*		



# Web Service Java et Client PHP



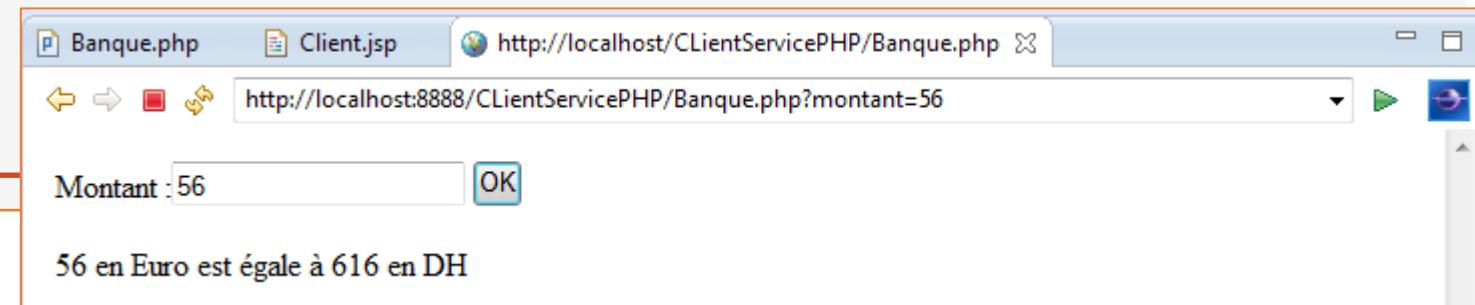
# Exemple de Client SOAP PHP

---

```
<?php
$client = new SoapClient('http://localhost:8585/BanqueWS?wsdl');
$param=new stdClass();
$param->montant=23;
$res=$client->__soapCall("conversionEuroDH",array($param));
//var_dump($res);
echo($res->return);
$param2=new stdClass();
$param2->arg0=2;
$res2=$client->__soapCall("getCompte",array($param2));
//var_dump($res2);
echo("Code=".$res2->return->code);
echo("<br/>Solde=".$res2->return->solde);
$res3=$client->__soapCall("getComptes",array());
//var_dump($res3);
echo ("<hr/>");
foreach($res3->return as $cpte){
    echo("Code=".$cpte->code);
    echo("<br/>Solde=".$cpte->solde);
    echo("<br/>");
}
?>
```

# Code PHP

```
<?php
$montant=0;$resultat=0;
if (isset($_GET['montant'])){
$montant=$_GET['montant'];
$client = new SoapClient('http://localhost:8686/BanqueService?wsdl');
$param=new stdClass();
$param->montant=$montant;
$rep=$client->__soapCall("conversionEuroDH",array($param));
$resultat=$rep->return;
}
?>
<html>
<body>
<form action="Banque.php">
Montant :<input type="text" name="montant" value="<?php echo($montant)?>">
<input type="submit" value="OK">
</form>
<?php echo($montant)?> en Euro est égale à <?php echo($resultat)?> en DH
</body>
</html>
```



## Un autre exemple Client SOAP PHP

```
<?php
$mt=0;
if(isset($_POST['action'])){
    $action=$_POST['action'];
    if($action=="OK"){
        $mt=$_POST['montant'];
        $client=new SoapClient("http://localhost:8585/BanqueWS?wsdl");
        $param=new stdClass();
        $param->montant=$mt;
        $rep=$client->__soapCall("conversionEuroToDh",array($param));
        $res=$rep->return;
    }
    elseif($action=="listComptes"){
        $client=new SoapClient("http://localhost:8585/BanqueWS?wsdl");
        $res2=$client->__soapCall("getComptes",array());
    }
}
?>
```

localhost/ClientPHP/clientSOAP.php

Montant: 0 OK listComptes

Rsltat:

CODE	SOLDE
1	7000
2	7000

# Suite de l'exemple du Client SOAP PHP

```
<html>
<body>
    <form method="post" action="clientSOAP.php">
        Montant:<input type="text" name="montant" value="<?php echo($mt)?>">
        <input type="submit" value="OK" name="action">
        <input type="submit" value="ListComptes" name="action">
    </form>

    Rsltat:
    <?php if (isset($res)){
        echo($res);
    }
    ?>
    <?php if(isset($res2)){?>
        <table border="1" width="80%">
            <tr>
                <th>CODE</th><th>SOLDE</th>
            </tr>
            <?php foreach($res2->return as $cp) {?>
                <tr>
                    <td><?php echo($cp->code)?></td>
                    <td><?php echo($cp->solde)?></td>
                </tr>
            <?php }?>
        </table>
    <?php }?>
    </body>
</html>
```

CODE	SOLDE
1	7000
2	7000

# UDDI

---

- L'annuaire des services UDDI est un standard pour la publication et la découverte des informations sur les services Web.
- La spécification UDDI est une initiative lancée par *ARIBA*, *Microsoft* et *IBM*.
- Cette spécification n'est pas gérée par le W3C mais par le groupe OASIS.
- La spécification UDDI vise à créer une plate-forme indépendante, un espace de travail (framework) ouvert pour la description, la découverte et l'intégration des services des entreprises.

## Consultation de l'annuaire

---

L'annuaire UDDI se concentre sur le processus de découverte de l'architecture orientée services (SOA), et utilise des technologies standards telles que XML, SOAP et WSDL qui permettent de simplifier la collaboration entre partenaires dans le cadre des échanges commerciaux.

L'accès au référentiel s'effectue de différentes manières.

- **Les pages blanches** : comprennent la liste des entreprises ainsi que des informations associées à ces dernières (coordonnées, description de l'entreprise, identifiants...).
- **Les pages jaunes** : recensent les services Web de chacune des entreprises sous le standard WSDL.
- **Les pages vertes** : fournissent des informations techniques précises sur les services fournis.

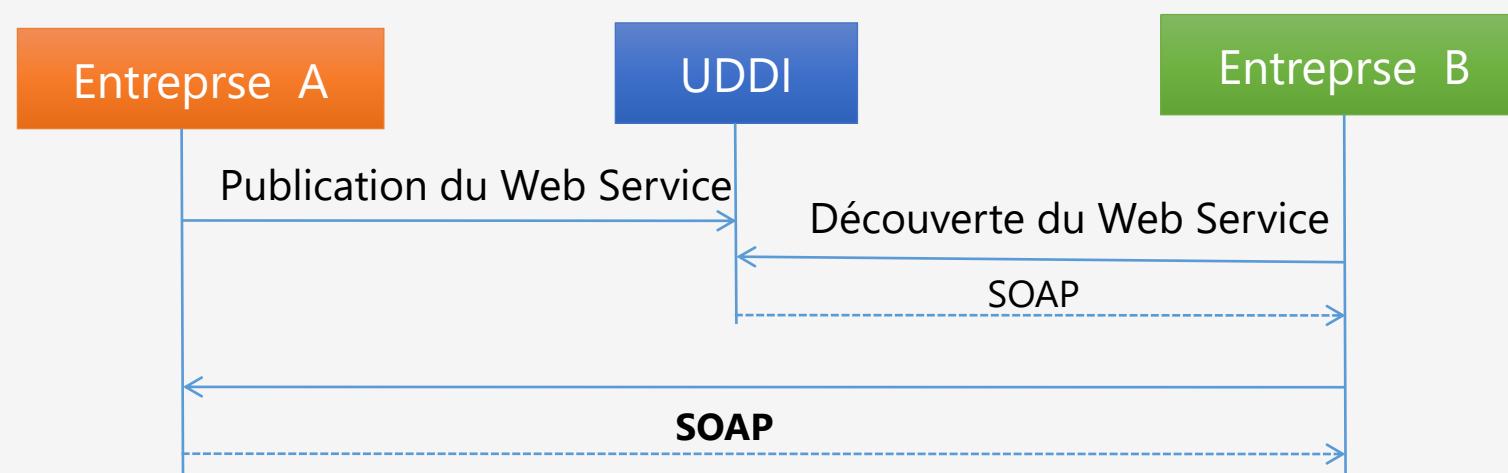
# Architecture

Les entreprises publient les descriptions de leurs services Web en UDDI, sous la forme de fichiers WSDL.

Ainsi, les clients peuvent plus facilement rechercher les services Web dont ils ont besoin en interrogeant le registre UDDI.

Lorsqu'un client trouve une description de service Web qui lui convient, il télécharge son fichier WSDL depuis le registre UDDI. Ensuite, à partir des informations inscrites dans le fichier WSDL, notamment la référence vers le service Web, le client peut invoquer le service Web et lui demander d'exécuter certaines de ses fonctionnalités.

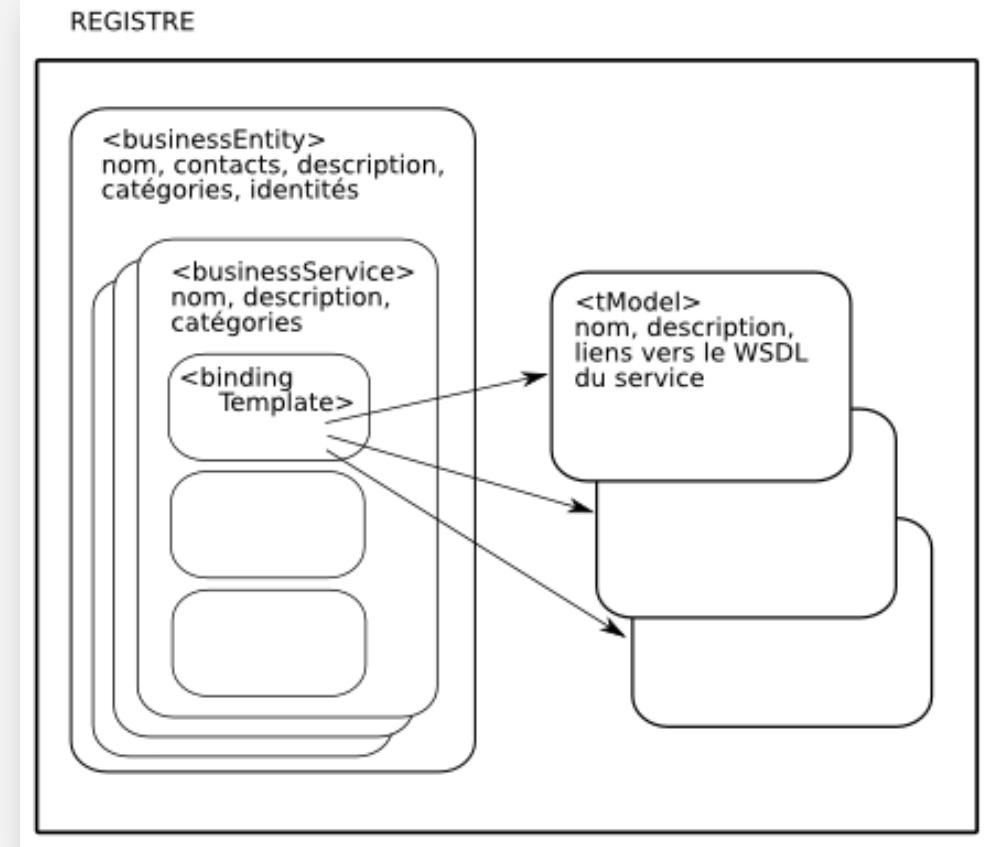
Le scénario classique d'utilisation de UDDI est illustré ci-dessous. L'entreprise B a publié le service Web S, et l'entreprise A est client de ce service :



# Structures de données UDDI

Un registre UDDI se compose de quatre types de structures de données,

- le **businessEntity**,
- Le **businessService**,
- le **bindingTemplate**
- et la **tModel**.



## BusinessEntity (entité d'affaires)

---

Les « businessEntities » sont en quelque sorte les pages blanches d'un annuaire UDDI.

Elles décrivent les organisations ayant publié des services dans le répertoire.

On y trouve notamment

- le nom de l'organisation,
- ses adresses (physiques et Web),
- des éléments de classification,
- une liste de contacts
- ainsi que d'autres informations.

## BusinessService (service d'affaires)

---

Les « businessServices » sont en quelque sorte les pages jaunes d'un annuaire UDDI.

Elles décrivent de manière non technique les services proposés par les différentes organisations.

On y trouve essentiellement

- le nom et la description textuelle des services
- ainsi qu'une référence à l'organisation proposant le service
- et un ou plusieurs « bindingTemplate ».

# BindingTemplate (modèle de rattachement)

---

- UDDI permet de décrire des services Web utilisant HTTP, mais également des services invoqués par d'autres moyens (SMTP, FTP...).
- Les « bindingTemplates » donnent les coordonnées des services.
- Ce sont les pages vertes de l'annuaire UDDI.
- Ils contiennent notamment une description, la définition du **point d'accès** (une URL) et les éventuels « tModels » associés.

## tModel (index)

---

- Les « tModels » sont les descriptions techniques des services.
- UDDI n'impose aucun format pour ces descriptions qui peuvent être publiées sous n'importe quelle forme et notamment sous forme de documents textuels (XHTML, par exemple).
- C'est à ce niveau que WSDL intervient comme le vocabulaire de choix pour publier des descriptions techniques de services.

# L'interface UDDI

---

L'interface UDDI est définie sous forme de documents UDDI et implémentée sous forme de service Web SOAP.

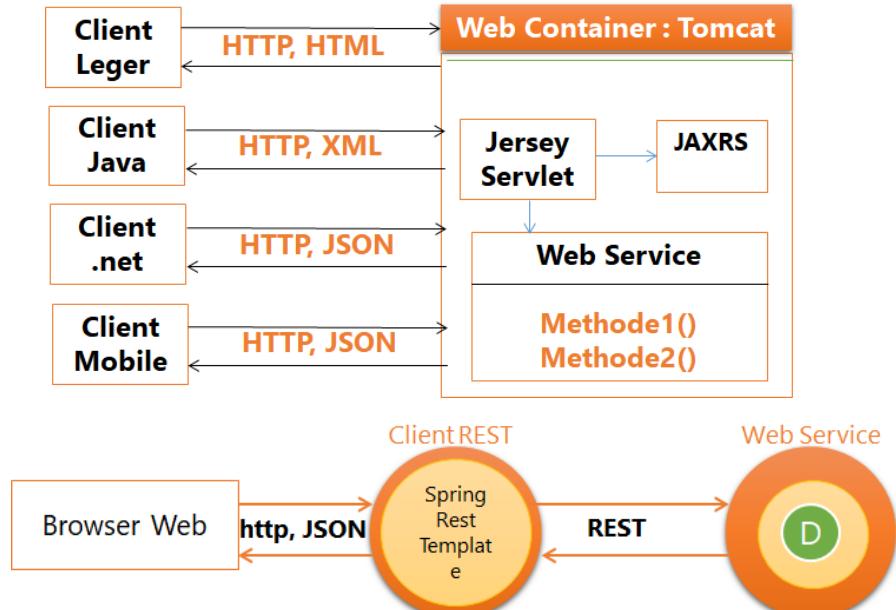
Elle est composée des modules suivants :

- **Interrogation inquiry** : Cette interface permet de rechercher des informations dans un répertoire UDDI.
- **Publication** : Cette interface permet de publier des informations dans un répertoire UDDI.
- **Sécurité** : cette interface est utilisée pour obtenir et révoquer les jetons d'authentification nécessaires pour accéder aux enregistrements protégés dans un annuaire UDDI.
- **Contrôle d'accès et propriété custody and ownership transfer**: Cette interface permet de transférer la propriété d'informations (qui est à l'origine attribuée à l'utilisateur ayant publié ces informations) et de gérer les droits d'accès associés.
- **Abonnement Subscription** : Cette interface permet à un client de s'abonner à un ensemble d'informations et d'être averti lors des modifications de ces informations.

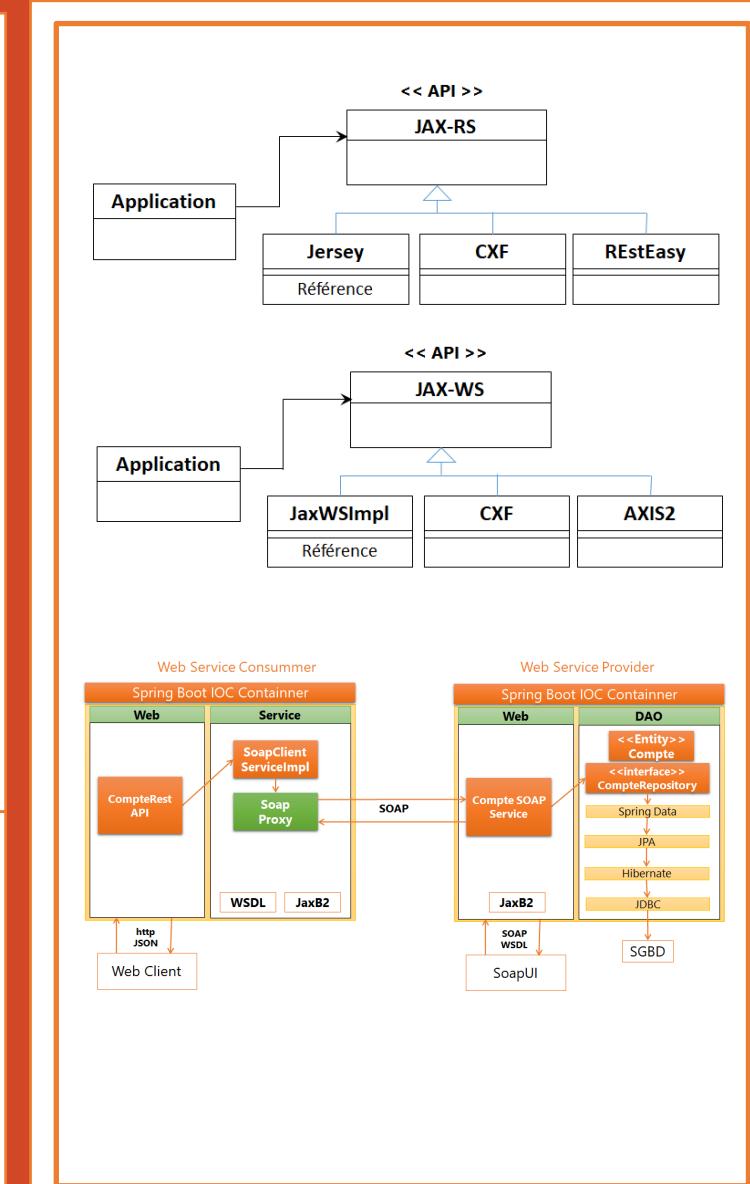
# Systèmes Distribués : Web Services : RESTful



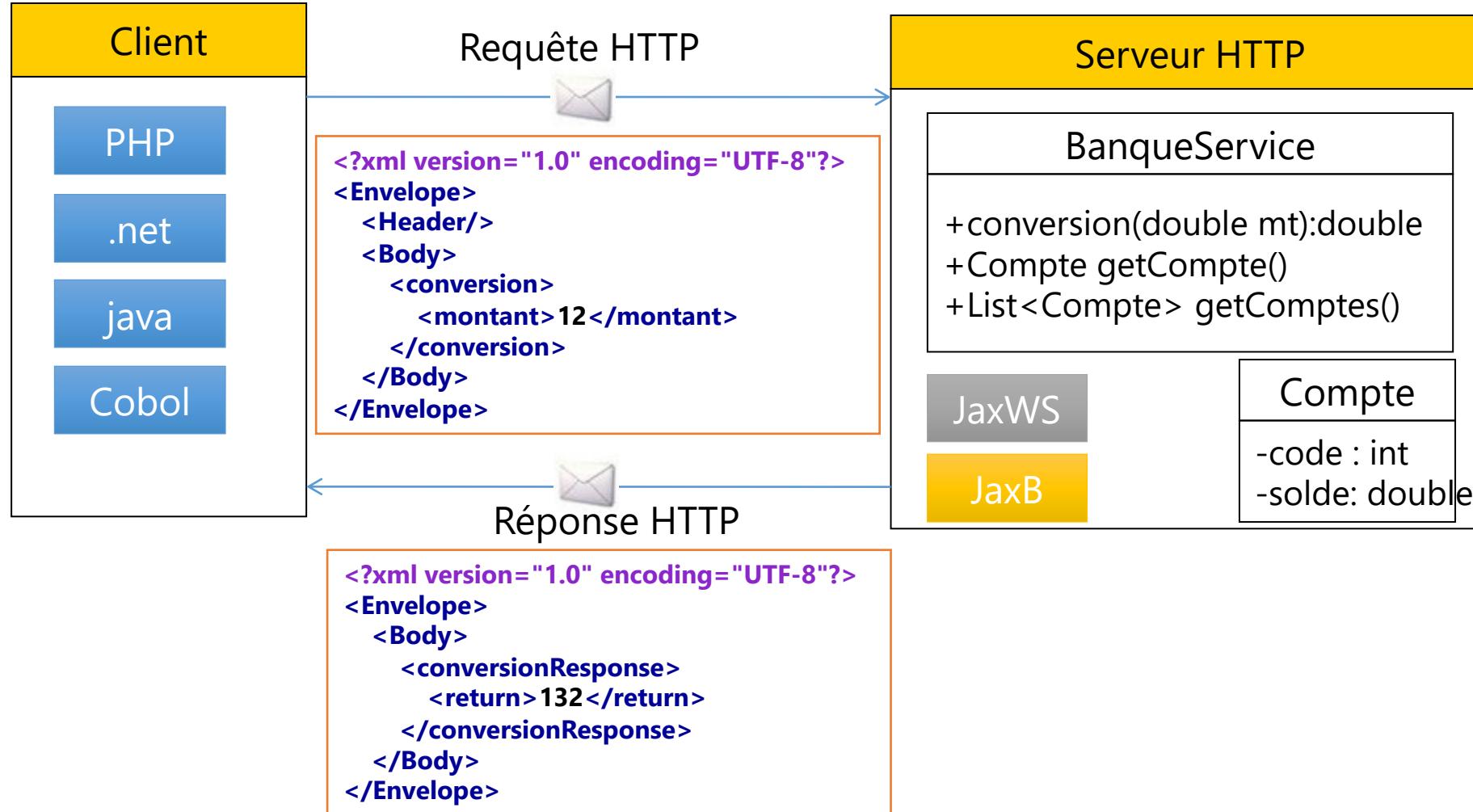
## RESTful avec JAXRS, RestController, Spring Data Rest



Mohamed Youssfi  
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)  
ENSET, Université Hassan II Casablanca, Maroc  
Email : [med@youssfi.net](mailto:med@youssfi.net)  
Supports de cours : <http://fr.slideshare.net/mohamedyoussfi9>  
Chaîne vidéo : <http://youtube.com/mohamedYoussfi>  
Recherche : [http://www.researchgate.net/profile/Youssfi\\_Mohamed/publications](http://www.researchgate.net/profile/Youssfi_Mohamed/publications)



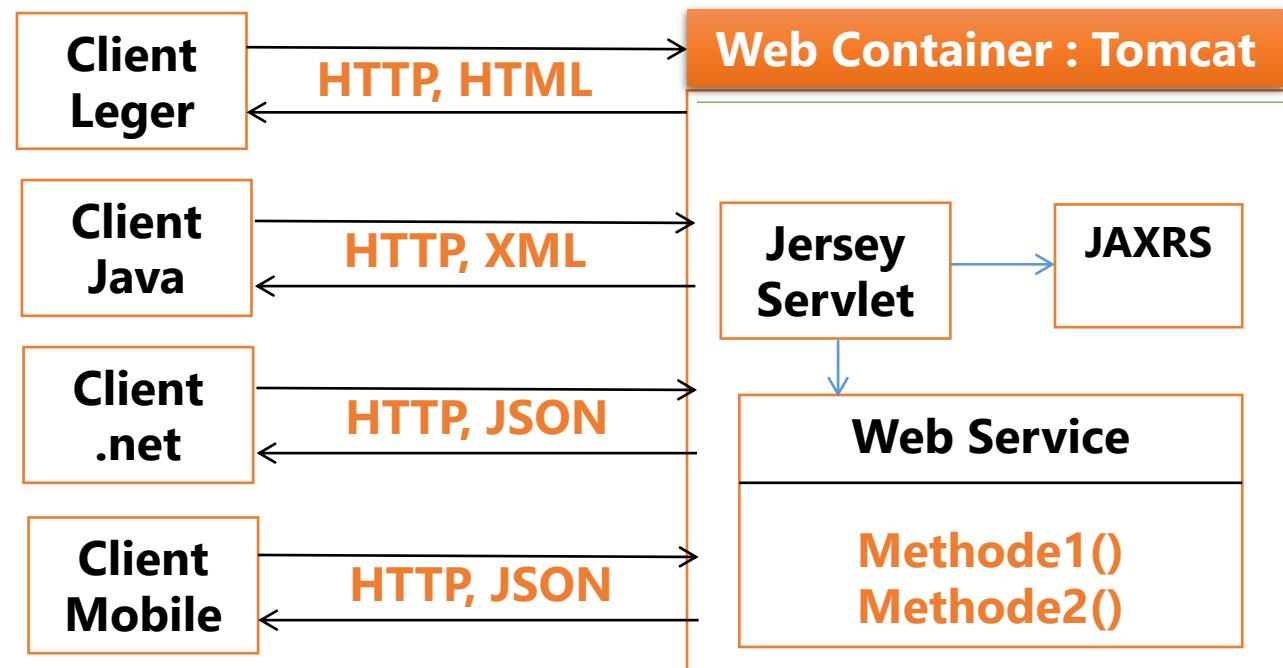
## Problème des Web services SOAP



# RESTful

- **REST (REpresentational State Transfer)** ou **RESTful** est un style d'architecture pour les systèmes hypermédia distribués,
- Crée par Roy Fielding en 2000 dans le chapitre 5 de sa thèse de doctorat.
- **REST** est un style d'architecture permettant de construire des applications (Web, Intranet, Web Service).
- Il s'agit d'un ensemble de conventions et de bonnes pratiques à respecter et non d'une technologie à part entière.
- L'architecture REST utilise les spécifications originelles du **protocole HTTP**, plutôt que de réinventer une surcouche (comme le font SOAP ou XML-RPC par exemple).

**WADL (Web Application Description Language)**  
Equivalent à WSDL pour SOAP



# Les 5 Règles de RESTful

- **Règle n°1 :**

- l'URI comme identifiant des ressources :
  - Respecter un standard pour construire les URLs
- **Règle n°2 :**
- les méthodes HTTP comme identifiant des opérations:
  - POST : Pour Ajouter (Create)
  - GET : pour consulter (Read)
  - PUT : pour mise à jour (Update) ou PATCH pour update partiel
  - DELETE : pour supprimer (DELETE)

- **Règle n°3 :**

- les réponses HTTP comme représentation des ressources
  - Réponses HTTP en différents formats (XML, JSON, ...) en fonction de la demande du client

- **Règle n°4 :**

- les liens comme relation entre ressources
  - <a href="..." rel="payment" >Effectuer un paiement</a>

- **Règle n°5 :**

- Un paramètre comme jeton d'authentification :
  - Envoyer un jeton d'authentification différent pour chaque requête

- REST se base sur les **URI (Uniform Resource Identifier)** afin d'identifier une ressource.
- Ainsi une application se doit de construire ses URI (et donc ses URLs) de manière précise, en tenant compte des contraintes REST.
- Il est nécessaire de prendre en compte la hiérarchie des ressources et la sémantique des URL pour les éditer :
- **Liste des livres**
  - `GET http://mywebsite.com/books`
- **Filtre et tri sur les livres**
  - `GET http://mywebsite.com/books?filter=policier&tri=asc`
- **Consulter un livre**
  - `GET http://mywebsite.com/books/87`
- **Tous les commentaires sur un livre**
  - `GET http://mywebsite.com/books/87/comments`
- **Ajoute un livre**
  - `POST http://mywebsite.com/books`
- **Mettre à jour un livre**
  - `PUT http://mywebsite.com/books/5`
- **Supprimer un livre**
  - `DELETE http://mywebsite.com/books/5`

# Les 5 Règles de RESTful

- Règle n°1 :
  - l'URI comme identifiant des ressources :
    - Respecter un standard pour construire les URLs
- **Règle n°2 :** 
  - les méthodes HTTP comme identifiant des opérations:
    - **POST** : Pour Ajouter (Create)
    - **GET** : pour consulter (Read)
    - **PUT** : pour mise à jour (Update) ou PATCH pour update partiel
    - **DELETE** : pour supprimer (DELETE)
- Règle n°3 :
  - les réponses HTTP comme représentation des ressources
    - Réponses HTTP en différents formats (XML, JSON, ...) en fonction de la demande du client
- Règle n°4 :
  - les liens comme relation entre ressources
    - <a href="..." rel="payment" >Effectuer un paiement</a>
- Règle n°5 :
  - Un paramètre comme jeton d'authentification :
    - Envoyer un jeton d'authentification différent pour chaque requête

- La seconde règle d'une architecture REST est d'utiliser les verbes HTTP existants plutôt que d'inclure l'opération dans l'URI de la ressource.
- Ainsi, généralement pour une ressource, il y a 4 opérations possibles (CRUD) :
  - Créer (create)
  - Afficher (read)
  - Mettre à jour (update)
  - Supprimer (delete)
- HTTP propose les verbes correspondant :
  - Créer (create) => **POST**
  - Afficher (read) => **GET**
  - Mettre à jour (update) => **PUT ou PATCH**
  - Supprimer (delete) => **DELETE**

# Les 5 Règles de RESTful

- Règle n°1 :
  - l'URI comme identifiant des ressources:
    - Respecter un standard pour construire les URLs
- Règle n°2 :
  - les méthodes HTTP comme identifiant des opérations:
    - POST : Pour Ajouter (Create)
    - GET : pour consulter (Read)
    - PUT : pour mise à jour (Update) ou PATCH pour update partiel
    - DELETE : pour supprimer (DELETE)
- **Règle n°3 :** 
- les réponses HTTP comme représentation des ressources
  - Réponses HTTP en différents formats (XML, JSON, ...) en fonction de la demande du client
- Règle n°4 :
  - les liens comme relation entre ressources
    - <a href="..." rel="payment" >Effectuer un paiement</a>
- Règle n°5 :
  - Un paramètre comme jeton d'authentification :
    - Envoyer un jeton d'authentification différent pour chaque requête

- Il est important d'avoir à l'esprit que la réponse envoyée n'est pas une ressource, mais plutôt la représentation d'une ressource.
- Ainsi, une ressource peut avoir plusieurs représentations dans des formats divers : **HTML, XML, CSV, JSON, etc.**
- C'est au client de définir quel format de réponse il souhaite recevoir via l'entête http **Accept**.
- Il est possible de définir plusieurs formats.
  - **Demander une réponse en HTML**
    - GET /books  
Host: mywebsite.com  
**Accept: application/xml**
  - **Demande une réponse en XML**
    - GET /books  
Host: mywebsite.com  
**Accept: application/json**

# Les 5 Règles de RESTful

- Règle n°1 :
  - l'URI comme identifiant des ressources:
    - Respecter un standard pour construire les URLs
- Règle n°2 :
  - les méthodes HTTP comme identifiant des opérations:
    - POST : Pour Ajouter (Create)
    - GET : pour consulter (Read)
    - PUT : pour mise à jour (Update) ou PATCH pour update partiel
    - DELETE : pour supprimer (DELETE)
- Règle n°3 :
  - les réponses HTTP comme représentation des ressources
    - Réponses HTTP en différents formats (XML, JSON, ...) en fonction de la demande du client
- **Règle n°4 :**
  - les liens comme relation entre ressources
    - <a href="..." rel="payment" >Effectuer un paiement</a>
- Règle n°5 :
  - Un paramètre comme jeton d'authentification :
    - Envoyer un jeton d'authentification différent pour chaque requête

- Les liens d'une ressource vers une autre indiquent la présence d'une relation.
- Il est cependant possible de la décrire afin d'améliorer la compréhension du système.
- Exemple quand on consulte une ressource qui représente un produit d'une catégorie, on trouve dans la description du produit un lien significatif qui permet de consulter la catégorie de ce produit
- Ainsi l'IANA donne une liste de relations parmi lesquelles :
  - Contents, edit, next, last, payment, etc.
- La liste complète sur le site de l'IANA :
  - <https://www.iana.org/assignments/link-relations/link-relations.xhtml>

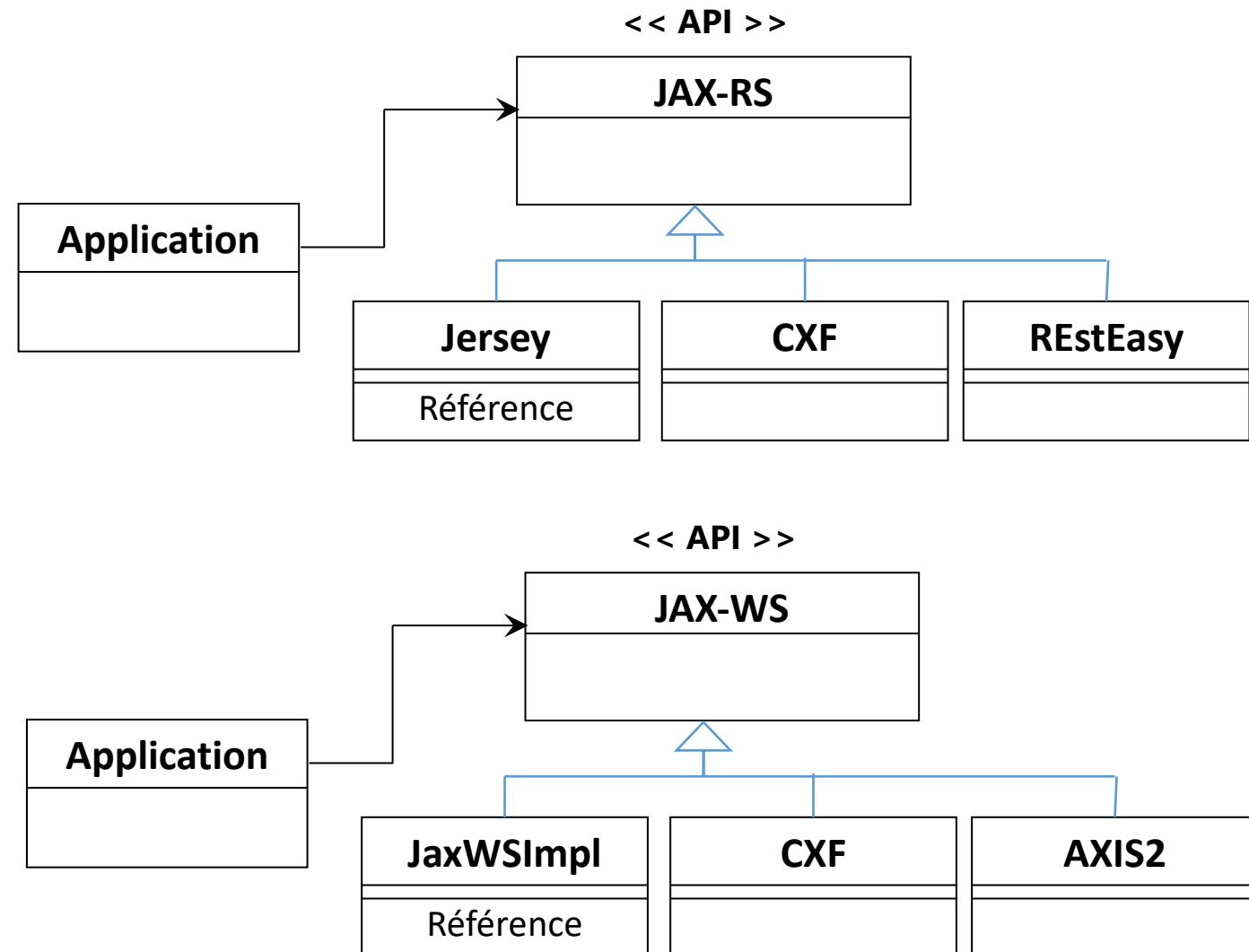
# Les 5 Règles de RESTful

- Règle n°1 :
  - l'URI comme identifiant des ressources:
    - Respecter un standard pour construire les URLs
- Règle n°2 :
  - les méthodes HTTP comme identifiant des opérations:
    - POST : Pour Ajouter (Create)
    - GET : pour consulter (Read)
    - PUT : pour mise à jour (Update) ou PATCH pour update partiel
    - DELETE : pour supprimer (DELETE)
- Règle n°3 :
  - les réponses HTTP comme représentation des ressources
    - Réponses HTTP en différents formats (XML, JSON, ...) en fonction de la demande du client
- Règle n°4 :
  - les liens comme relation entre ressources
    - <a href="..." rel="payment" >Effectuer un paiement</a>
- **Règle n°5 :**
- Un paramètre comme jeton d'authentification :
  - Envoyer un jeton d'authentification différent pour chaque requête

- C'est un des sujets les plus souvent abordés quand on parle de REST : comment authentifier une requête ?
- Pour authentifier les requêtes HTTP, il faut envoyer dans la requête un jeton d'authentification de la requête
- Le premier Jeton X est obtenu au moment de l'authentification.
- Pour chaque requête HTTP, ce jeton X peut être mélangé avec l'URL de la requête en utilisant une opération de hachage.
- Cette opération permet d'avoir un jeton différent pour chaque requête
- Exemple de Token d'authentification : JWT (Json Web Token)

## Spécification JEE : JAX-RS pour RESTFUL

- JAX-RS est l'acronyme Java API for RESTful Web Services
- Décrise par la JSR 311 ([jcp.org/en/jsr/summary?id=311](http://jcp.org/en/jsr/summary?id=311))
- Version courante de la spécification est la 1.1
- Depuis la version 1.1, JAX-RS fait partie intégrante de la spécification Java EE 6 au niveau de la pile Service Web
- Cette spécification décrit uniquement la mise en œuvre de services Web REST coté serveur
- Le développements des Services Web REST repose sur l'utilisation de classes Java et d'annotations
- Différentes implémentations de la spécification JAX-RS sont disponibles
  - **JERSEY** : implémentation de référence fournie par Oracle  
Site projet : <http://jersey.java.net>
  - **CXF** : fournie par Apache, la fusion entre XFire et Celtix  
Site projet : [cxf.apache.org](http://cxf.apache.org)
  - **RESTEasy** : fournie par JBoss Site projet :  
[www.jboss.org/resteasy](http://www.jboss.org/resteasy)
  - **RESTlet** : un des premiers framework implémentant REST pour Java : Site projet : [www.restlet.org](http://www.restlet.org)



# Format JSON Vs XML

- **JSON** (JavaScript Object Notation – Notation Objet issue de JavaScript) est un format léger d'échange de données.
- Il est facile à lire ou à écrire pour des humains. Il est aisément analysable ou générable par des machines.
- JSON est un format texte complètement indépendant de tout langage, mais les conventions qu'il utilise seront familières à tout programmeur habitué aux langages descendant du C, comme par exemple : C lui-même, C++, C#, Java, JavaScript, Perl, Python et bien d'autres.
- Ces propriétés font de JSON un langage d'échange de données idéal.

JSON

```
[  
    {"type": "courant", "code": 1, "solde": 4300, "dateCreation": 1596711188451},  
    {"type": "epargne", "code": 2, "solde": 9600, "dateCreation": 1596711179308}  
]
```

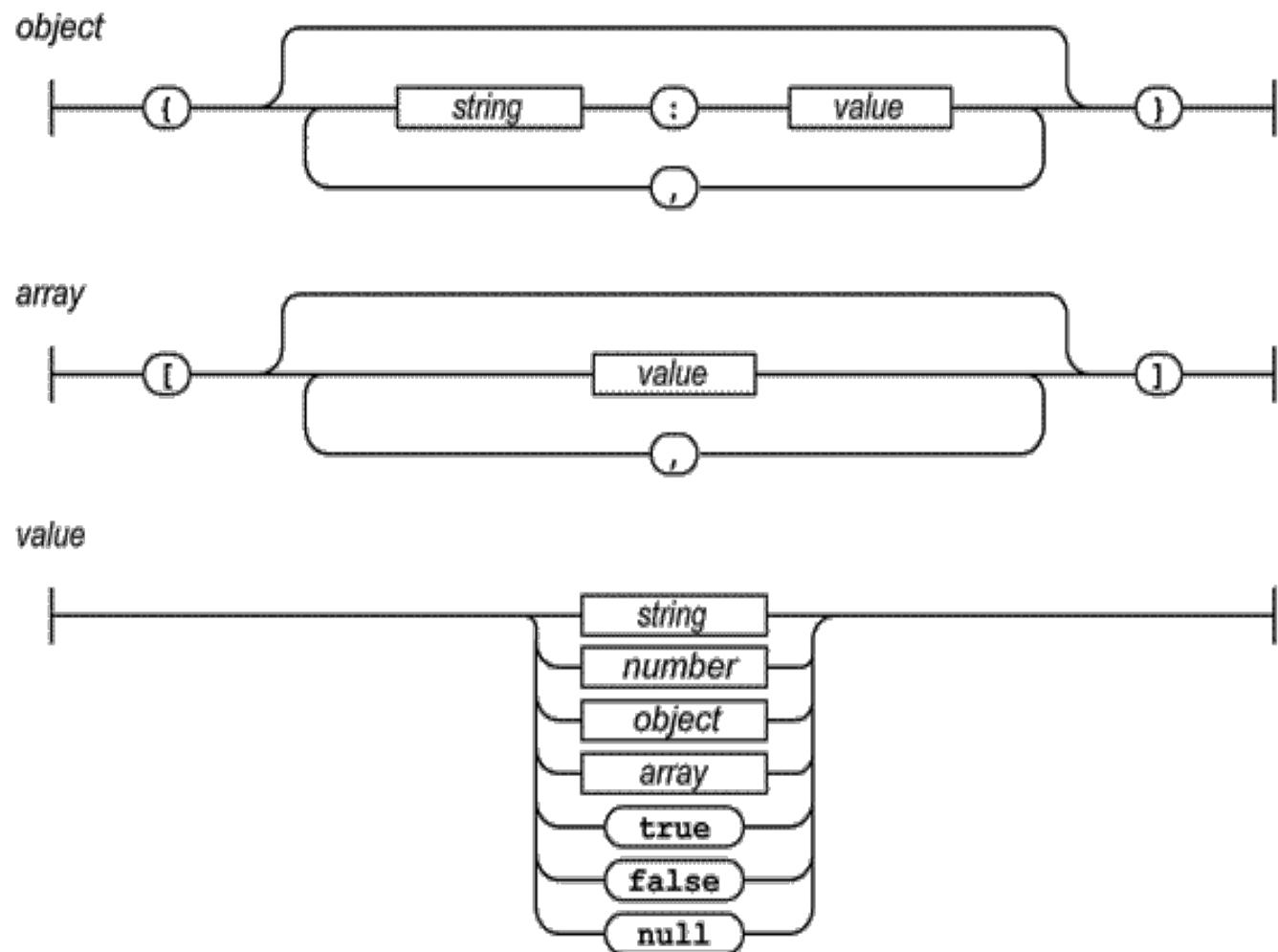
```
// x est un objet java script  
var x={a:4,b:9};  
// T1 est un tableau java script  
var T1=[8,6,8];  
// T2 est un tableau d'objets  
var T2=[{a:2,b:6},{a:1,b:8},{a:2,b:4}];
```

XML

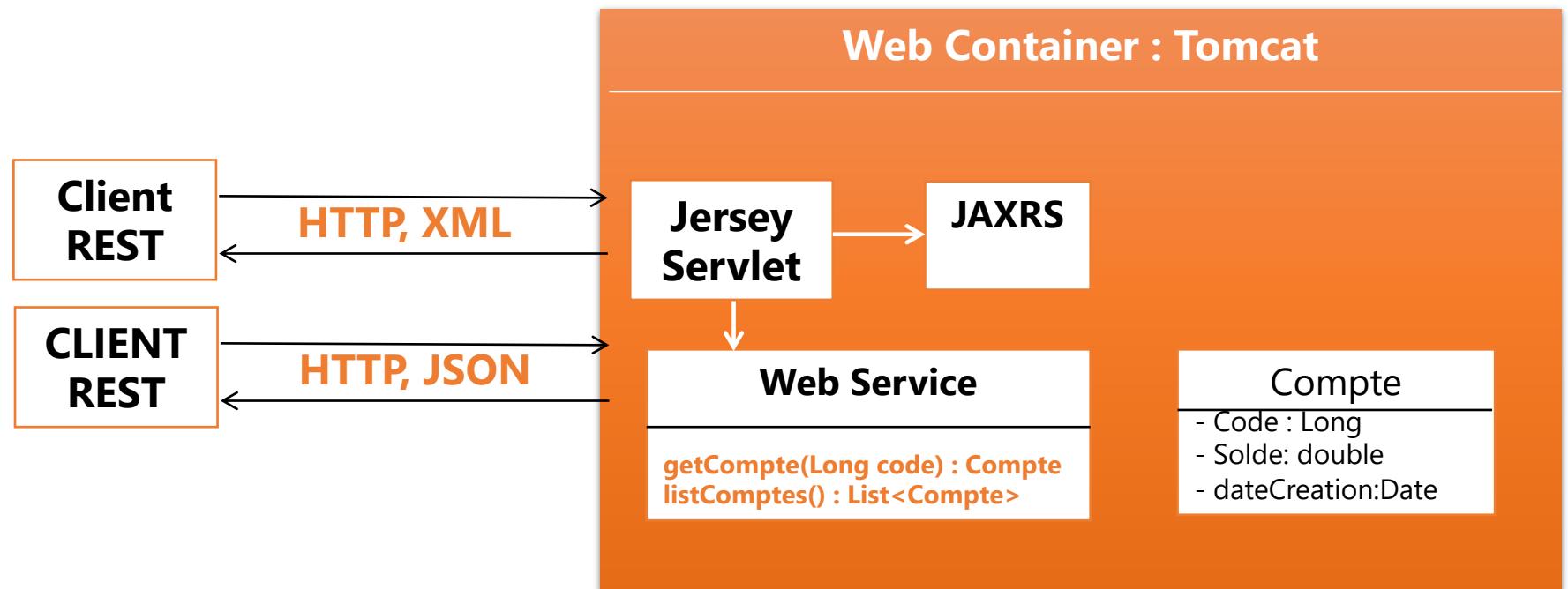
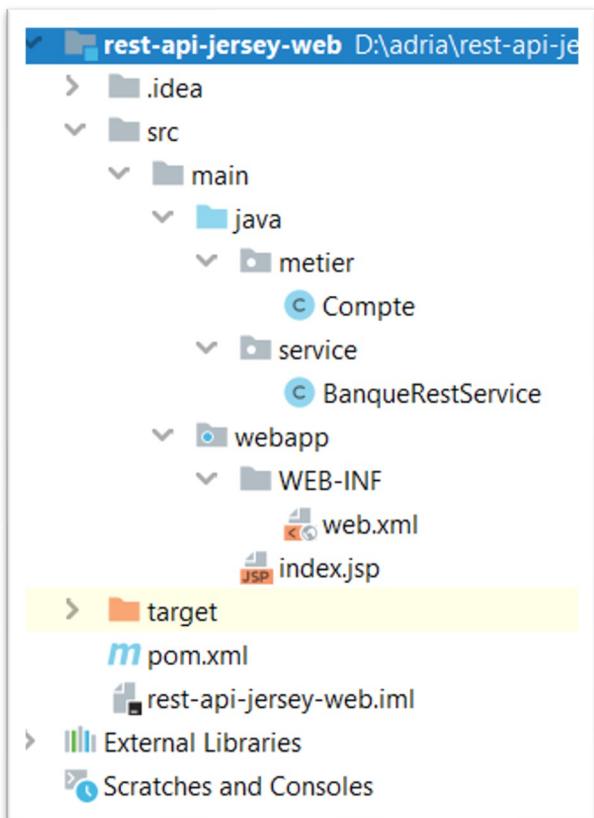
```
<?xml version="1.0" encoding="UTF-8"?>  
  <comptes>  
    <compte type="courant">  
      <code>1</code>  
      <solde>4300</solde>  
      <dateCreation>2012-11-11</dateCreation>  
    </compte>  
    <compte type="epargne">  
      <code>2</code>  
      <solde>96000</solde>  
      <dateCreation>2012-12-11</dateCreation>  
    </compte>  
  </comptes>
```

## Les structures de données JSON

- En JSON, les structures de données prennent les formes suivantes :
- Un **objet**, qui est un ensemble de couples nom/valeur non ordonnés. Un objet commence par { (accolade gauche) et se termine par } (accolade droite). Chaque nom est suivi de : (deux-points) et les couples nom/valeur sont séparés par , (virgule).
- Un **tableau** est une collection de valeurs ordonnées. Un tableau commence par [ (crochet gauche) et se termine par ] (crochet droit). Les valeurs sont séparées par , (virgule).
- Une **valeur** peut être soit une *chaîne de caractères* entre guillemets, soit un *nombre*, soit true ou false ou null, soit un *objet* soit un *tableau*. Ces structures peuvent être imbriquées.
- Une **chaîne de caractères** est une suite de zéro ou plus caractères Unicode, entre guillemets, et utilisant les échappements avec antislash. Un caractère est représenté par une chaîne d'un seul caractère.



# JAX RS: Application



## Implémentation d'un Web service RESTful avec JAX RS

- Le développement de Services Web avec JAX-RS est basé sur des POJO (Plain Old Java Object) en utilisant des annotations spécifiques à JAX-RS
- Pas description requise dans des fichiers de configuration
- Seule la configuration de la Servlet « JAX-RS » est requise pour réaliser le pont entre les requêtes HTTP et les classes Java annotées
- Un Service Web REST est déployé dans une application Web
- Contrairement aux Services Web entendus il n'y a pas de possibilité de développer un service REST à partir du fichier de description WADL
- Seule l'approche Bottom / Up est disponible
- Créer et annoter un POJO
- Compiler, Déployer et Tester
- Possibilité d'accéder au document WADL
- Le fichier de description WADL est généré automatiquement par JAX-RS (exemple : <http://host/context/application.wadl>)

```
@Path("/banque")
public class BanqueRestService {

    @Path("/comptes")
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Compte> listComptes() {
        List<Compte> cptes=new ArrayList<>();
        ...
        return cptes;
    }

    @Path("/comptes/{code}")
    @GET
    @Produces({MediaType.APPLICATION_JSON,MediaType.APPLICATION_XML})
    public Compte getCompte(@PathParam(value="code")Long code) {
        return new Compte(1L, Math.random()*9999, new Date());
    }

    @Path("/comptes")
    @POST
    @Produces({MediaType.APPLICATION_JSON,MediaType.APPLICATION_XML})
    public Compte save(Compte cp) {
        ...
        return cp;
    }
}
```

## Implémentation d'un Web service RESTful avec JAX RS

- Le développement de Services Web avec JAX-RS est basé sur des POJO (Plain Old Java Object) en utilisant des annotations spécifiques à JAX-RS
- Pas description requise dans des fichiers de configuration
- Seule la configuration de la Servlet « JAX-RS » est requise pour réaliser le pont entre les requêtes HTTP et les classes Java annotées
- Un Service Web REST est déployé dans une application Web
- Contrairement aux Services Web entendus il n'y a pas de possibilité de développer un service REST à partir du fichier de description WADL
- Seule l'approche Bottom / Up est disponible
- Créer et annoter un POJO
- Compiler, Déployer et Tester
- Possibilité d'accéder au document WADL
- Le fichier de description WADL est généré automatiquement par JAX-RS (exemple : <http://host/context/application.wadl>)

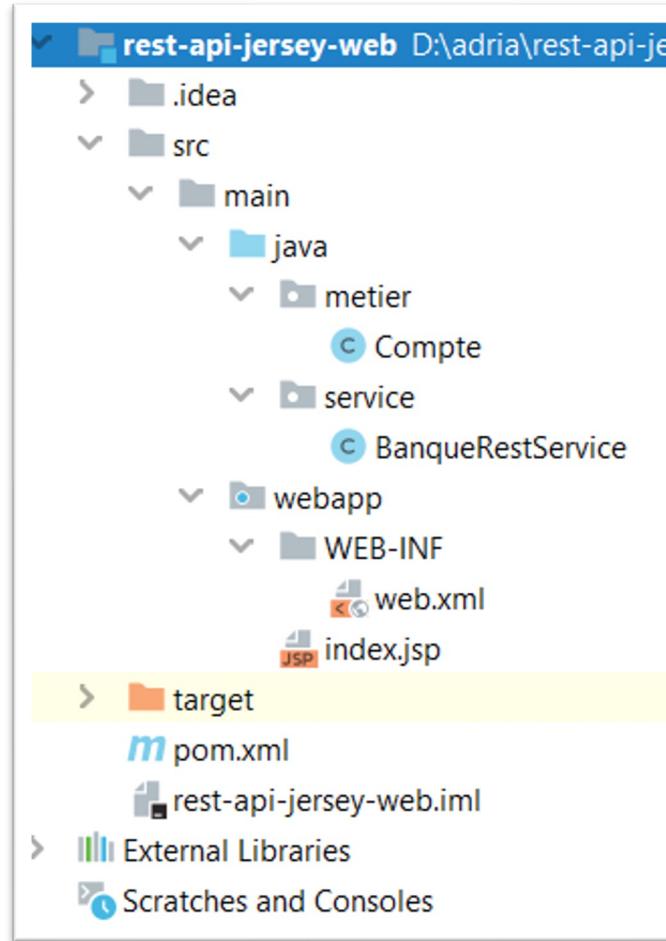
```
@Path("/comptes/{code}")
@PUT
@Produces(MediaType.APPLICATION_JSON)
public Compte update(@PathParam("code")Long code,Compte cp) {
    cp.setCode(code);
    // .. Mise à jour du compte
    System.out.println("Mise à jour du compte :" +cp.getCode());
    return cp;
}
```

```
@Path("/comptes/{code}")
@DELETE
@Produces(MediaType.APPLICATION_JSON)
public void delete(@PathParam("code")Long code) {
    // .. Suppression du compte
    System.out.println("Suppression du compte :" +code);
}
```

```
@XmlRootElement
public class Compte {
    private Long code; private double solde;
    private Date dateCreation;
    // Getters et Setters
    // Constructeurs
}
```

# Structure du projet Maven

```
<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-web-api</artifactId>
  <version>7.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-server</artifactId>
  <version>1.19.4</version>
</dependency>
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-servlet</artifactId>
  <version>1.19.4</version>
</dependency>
<dependency>
  <groupId>com.sun.jersey</groupId>
  <artifactId>jersey-json</artifactId>
  <version>1.19.4</version>
</dependency>
```



```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <configuration>
    <port>8080</port>
    <path>/web-api</path>
  </configuration>
</plugin>
```

## Déploiement de Jersey : Web.xml

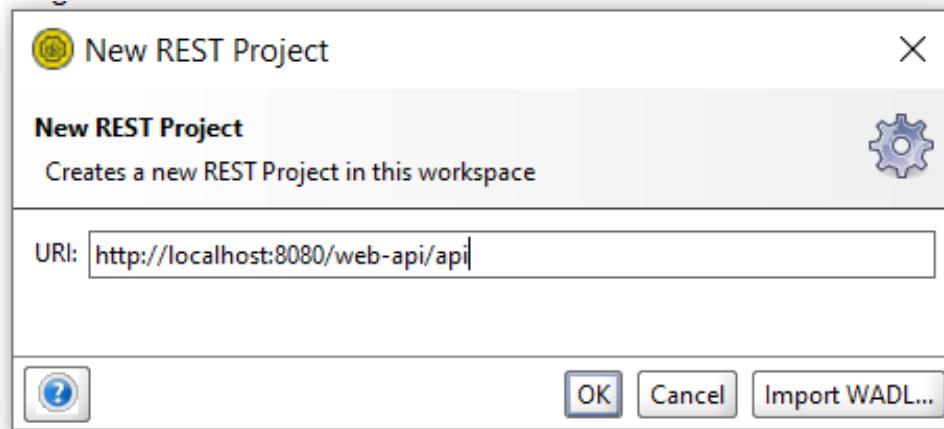
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
  <servlet>
    <servlet-name>rs</servlet-name>
    <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>service</param-value>
    </init-param>
    <init-param>
      <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
      <param-value>true</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>rs</servlet-name>
    <url-pattern>/api/*</url-pattern>
  </servlet-mapping>
</web-app>
```

## Déploiement de Jersey : Web.xml

```
@Path("/")
public class BanqueRestService {
    static Map<Long,Compte> banque=new HashMap<>();
    @GET @Path("/test")
    public String test(){ return "Test Jersey"; }
    @GET @Path("/comptes/{code}")
    @Produces({MediaType.APPLICATION_XML,MediaType.APPLICATION_JSON})
    public Compte getCompte(@PathParam(value="code") Long code){
        return banque.get(code);
    }
    @GET @Path("/comptes") @Produces(MediaType.APPLICATION_JSON)
    public Collection<Compte> listComptes(){
        return banque.values();
    }
    @Path("/comptes") @POST @Produces(MediaType.APPLICATION_JSON)
    public Compte save(Compte cp){
        if(banque.get(cp.getId())!=null) throw new RuntimeException("Ce compte existe déjà");
        cp.setDateCreation(new Date());
        banque.putIfAbsent(cp.getId(),cp);
        return cp;
    }
    @Path("/comptes/{code}") @PUT @Produces(MediaType.APPLICATION_JSON)
    public Compte update(@PathParam("code") Long code,Compte cp){
        cp.setId(code); banque.put(cp.getId(),cp); return cp;
    }
    @Path("/comptes/{code}") @DELETE @Produces(MediaType.APPLICATION_JSON)
    public Compte delete(@PathParam("code") Long code){
        return banque.remove(code);
    }
}
```

```
@XmlRootElement
@Data @AllArgsConstructor
@NoArgsConstructor
public class Compte {
    private Long id;
    private double solde;
    private Date dateCreation;
}
```

# Test du Web service RESTful



REST Request 1

Method: POST    Endpoint: http://localhost:8080    Resource: /web-api/api/comptes

Request:

Name	Value	Style	Level
Raw			

Required:  Sets if parameter is required

Type:

Media Type: application/json

JSON Content:

```
{"id": 1, "solde": 9000, "dateCreation": 1596712883147}
```

REST Request 1

Method: GET    Endpoint: http://localhost:8080    Resource: /web-api/api/comptes

Request:

Name	Value	Style	Level
Raw			

Required:  Sets if parameter is required

Type:

JSON Content:

```
[{"id": 1, "solde": 9000, "dateCreation": 1596712883147}, {"id": 2, "solde": 8000, "dateCreation": 1596712975807}]
```

# Test du Web service RESTful

REST Request 1

Method: GET    Endpoint: http://localhost:8080    Resource: /web-api/api/comptes/1    Parameters:

Request

Raw    HTML    JSON    XML

```
1 {
  "id": 1,
  "solde": 9000,
  "dateCreation": 1596712883147
}
```

Header    Value  
accept    application/json

REST Request 1

Method: PUT    Endpoint: http://localhost:8080    Resource: /web-api/api/comptes/1    Parameters:

Request

Raw    Name    Value    Style    Level

Media Type: application/json    Post Qu

```
{"id":2,"solde":30000}
```

Raw    HTML    JSON    XML

```
1 {
  "id": 1,
  "solde": 30000,
  "dateCreation": null
}
```

REST Request 1

Method: GET    Endpoint: http://localhost:8080    Resource: /web-api/api/comptes/1    Parameters:

Request

Raw    HTML    JSON    XML

```
<compte>
  <dateCreation>2020-08-06T12:21:23.147+01:00</dateCreation>
  <id>1</id>
  <solde>9000.0</solde>
</compte>
```

Header    Value  
accept    application/xml

REST Request 1

Method: DELETE    Endpoint: http://localhost:8080    Resource: /web-api/api/comptes/2    Parameters:

Request

Raw    Name    Value    Style    Level

Media Type: application/json    Post Qu

Header    Value  
accept    application/json

```
1 {
  "id": 2,
  "solde": 8000,
  "dateCreation": 1596713516857
}
```

# WADL : Web Application Description Language

- WADL est un fichier XML qui permet de faire la description des services web d'une application basée sur REST.
- Le WADL est généré automatiquement par le conteneur REST.  
<http://localhost:8080/web-api/api/application.wadl>
- Les types de données structurés échangés via ce web service sont décrites par un schéma XML lié au WSDL.
- Le schéma xml de l'application REST peut être consulté par l'adresse de suivante : <http://localhost:8080/web-api/api/application.wadl/xsd0.xsd>

This XML file does not appear to have any style information associated with it. The document tree is shown below.

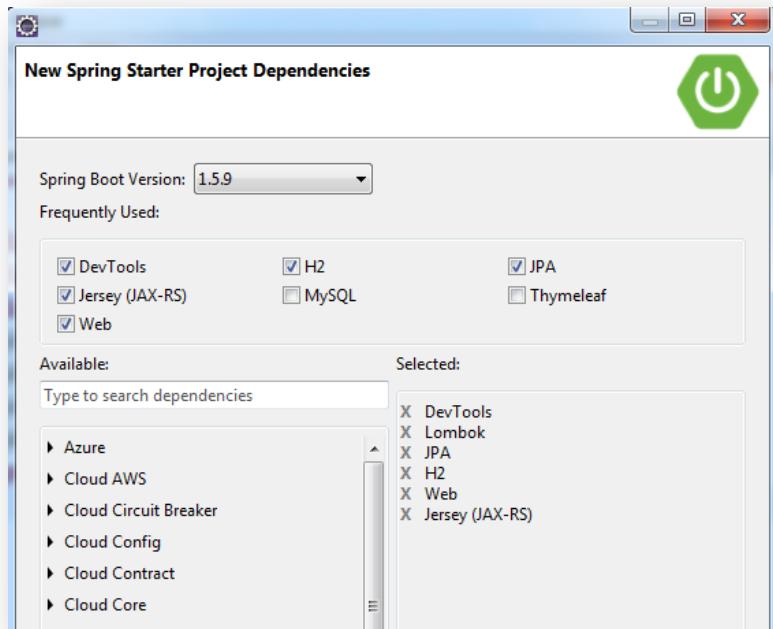
```
<application xmlns="http://wadl.dev.java.net/2009/02">
  <doc xmlns:jersey="http://jersey.java.net/" jersey:generatedBy="Jersey: 1.19.4 05/24/2017 03:20 PM"/>
  <grammars>
    <include href="application.wadl/xsd0.xsd">
      <doc title="Generated" xml:lang="en"/>
    </include>
  </grammars>
  <resources base="http://localhost:8080/web-api/api/">
    <resource path="/">
      <resource path="/comptes/{code}">
        <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="code" style="template" type="xs:long"/>
        <method id="update" name="PUT">
          <request>
            <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="compte" mediaType="*/*"/>
          </request>
          <response>
            <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="compte" mediaType="application/json"/>
          </response>
        </method>
        <method id="delete" name="DELETE">
          <response>
            <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="compte" mediaType="application/json"/>
          </response>
        </method>
        <method id="getCompte" name="GET">
          <response>
            <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="compte" mediaType="application/xml"/>
            <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="compte" mediaType="application/json"/>
          </response>
        </method>
      </resource>
    </resource path="/comptes">
```

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<xss:schema xmlns:xss="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xss:element name="compte" type="compte"/>
  <xss:complexType name="compte">
    <xss:sequence>
      <xss:element name="dateCreation" type="xs:dateTime" minOccurs="0"/>
      <xss:element name="id" type="xs:long" minOccurs="0"/>
      <xss:element name="solde" type="xs:double"/>
    </xss:sequence>
  </xss:complexType>
</xss:schema>
```

# JaxRS dans un projet Spring Boot

```
public interface CompteRepository extends JpaRepository<Compte, Long> {}
```



```
@Configuration  
public class MyConfig {  
    @Bean  
    public ResourceConfig getJaxRSExporter() {  
        ResourceConfig exporter=new ResourceConfig();  
        exporter.register(BanqueRestService.class);  
        return exporter;  
    }  
}
```

```
@Component  
@Path("/banque")  
public class BanqueRestService {  
    @Autowired  
    private CompteRepository compteRepository;  
  
    @Path("/comptes") @GET  
    @Produces(MediaType.APPLICATION_JSON)  
    public List<Compte> listComptes() {  
        return compteRepository.findAll();  
    }  
    @Path("/comptes/{code}")  
    @GET @Produces({MediaType.APPLICATION_JSON,MediaType.APPLICATION_XML})  
    public Compte getCompte(@PathParam("code")Long code) {  
        return compteRepository.findOne(code);  
    }  
    @Path("/comptes")  
    @POST @Produces({MediaType.APPLICATION_JSON,MediaType.APPLICATION_XML})  
    public Compte save(Compte cp) {  
        compteRepository.save(cp);  
        return cp;  
    }  
    @Path("/comptes/{code}") @PUT @Produces(MediaType.APPLICATION_JSON)  
    public Compte update(@PathParam("code")Long code,Compte cp) {  
        cp.setCode(code);  compteRepository.save(cp);  
        return cp;  
    }  
    @Path("/comptes/{code}") @DELETE @Produces(MediaType.APPLICATION_JSON)  
    public void delete(@PathParam("code")Long code) {  
        compteRepository.delete(code);  
    }  
}
```

```
@Entity @Data  
@AllArgsConstructor  
@NoArgsConstructor  
public class Compte {  
    @Id @GeneratedValue  
    private Long code;  
    private double solde;  
    private Date dateCreation;  
}
```

## Web Service Restful avec Spring MVC

- Spring MVC offre des annotations qui permettent de créer facilement des web services Restful, sans avoir besoin d'utiliser JAXRS.
- Les annotations principales suivantes peuvent être utilisées pour ce fait:
  - **@RestController** : Pour annoter le web service
  - **@RequestMapping** : Pour associer une requête HTTP quelconque
  - **@GetMapping** : Pour associer une requête HTTP de type GET
  - **@PostMapping** : Pour associer une requête HTTP de type POST
  - **@PutMapping** : Pour associer une requête HTTP de type PUT
  - **@DeleteMapping** : Pour associer une requête HTTP de type DELETE
  - **@RequestBody** : Pour associer le corps de la requête HTTP à un paramètre d'une méthode du web service
  - **@PathVariable** : Pour associer un paramètre de path URL un paramètre d'une méthode du web service
  - **@RequestParam** : Pour associer un Query param à un paramètre d'une méthode du web service

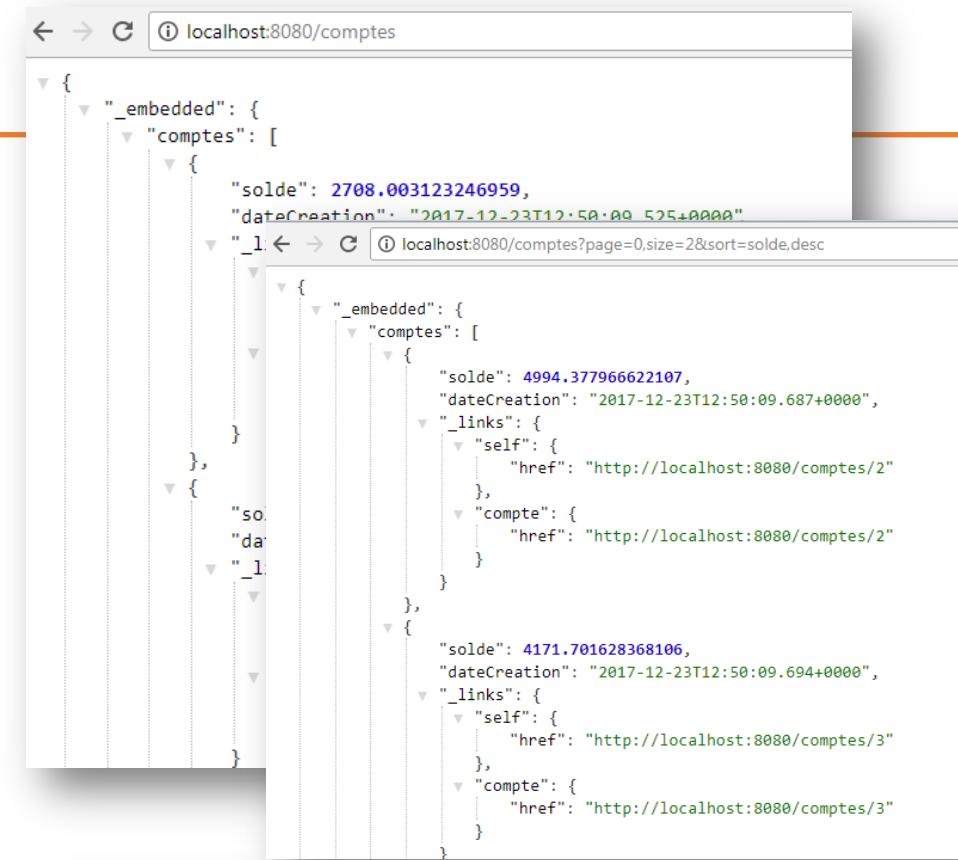
```
@RestController
@RequestMapping(value="/banque")
public class BanqueRestServiceRC {
@Autowired
private CompteRepository compteRepository;
@GetMapping("/comptes")
public List<Compte> listComptes() {
    return compteRepository.findAll();
}
@GetMapping("/comptes/{code}")
public Compte getCompte(@PathVariable(value="code")Long code) {
    return compteRepository.findOne(code);
}
@PostMapping("/comptes")
public Compte save(@RequestBody Compte cp) {
    compteRepository.save(cp);
    return cp;
}
@PutMapping("/comptes/{code}")
public Compte update(@PathVariable(value="code")Long code,Compte cp) {
    cp.setCode(code);
    compteRepository.save(cp);
    return cp;
}
@DeleteMapping("/comptes/{code}")
public void delete(@PathVariable(value="code")Long code) {
    compteRepository.delete(code);
}
}
```

# Web Service Restful avec Spring Data Rest

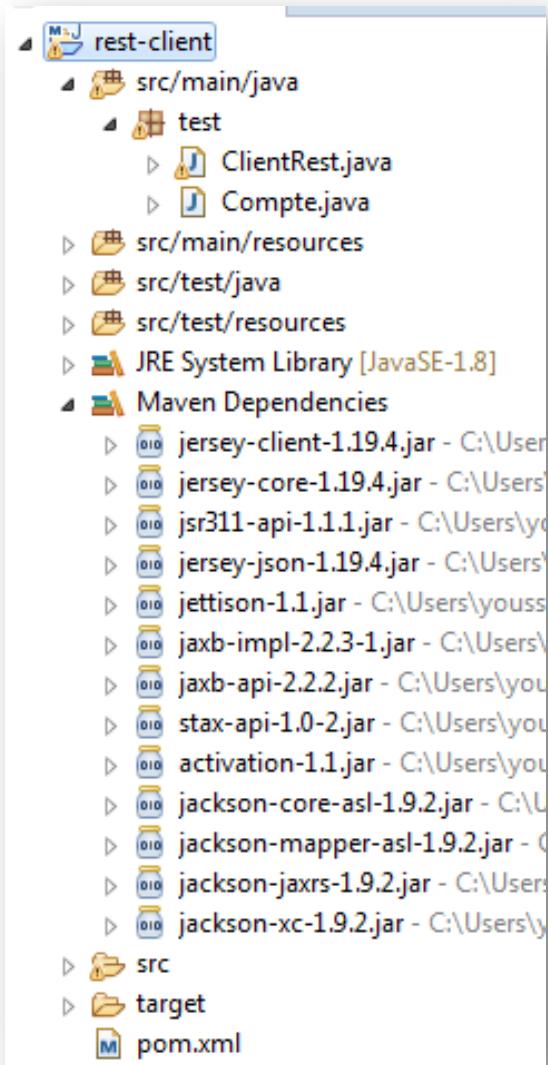
- Spring Data rest offre un moyen très rapide qui permet d'exposer une API Rest.
- Il suffit d'utiliser l'annotation **@RepositoryRestResource** dans l'interface Spring Data pour que toutes les méthodes de cette interface soient accessibles via une API Rest générée automatiquement.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
```

```
@RepositoryRestResource
public interface CompteRepository extends JpaRepository<Compte, Long> { }
```



## Client REST Java avec Jersey Client



```
<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-client</artifactId>
    <version>1.19.4</version>
</dependency>
<dependency>
    <groupId>com.sun.jersey</groupId>
    <artifactId>jersey-json</artifactId>
    <version>1.19.4</version>
</dependency>
```

Client Java  
Jersey Client

HTTP  
JSON

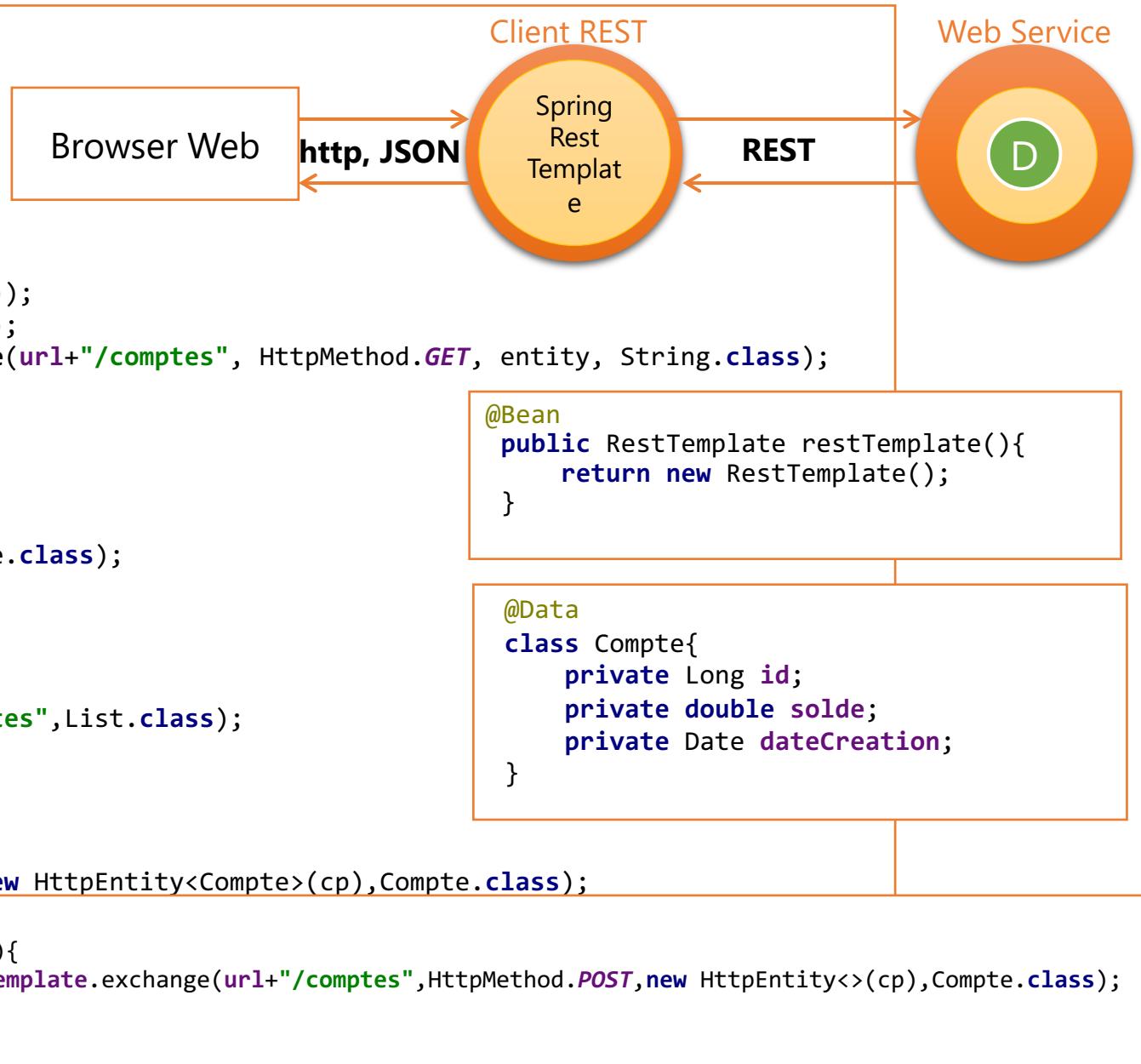
Server  
API REST

```
ClientConfig config=new DefaultClientConfig();
Client client=Client.create(config);
URI uri=UriBuilder.fromUri("http://localhost:8080/web-api").build();
WebResource service=client.resource(uri);
WebResource resource=service.path("banque").path("comptes");
/*
 * Ajouter un compte : POST http://localhost:8080/banque/comptes
 * {"solde":9999,"dateCreation":1514110620417}
 */
Compte cp=new Compte(); cp.solde=9999;cp.dateCreation=new Date();
ObjectMapper mapper=new ObjectMapper();
ClientResponse resp1=resource
    .accept("application/json").type("application/json")
    .post(ClientResponse.class,mapper.writeValueAsString(cp));
System.out.println("Status Code:"+resp1.getStatus());
String corpsReponse=resp1.getEntity(String.class);
System.out.println("Corps Réponse:"+corpsReponse);
Compte cp1=mapper.readValue(corpsReponse, Compte.class);
System.out.println("Code="+cp1.code); System.out.println("Solde="+cp1.solde);
System.out.println("Date Crédation="+cp1.dateCreation);
/* GET http://localhost:8080/banque/comptes */
String resp2=resource.get(String.class);
System.out.println(resp2);
Compte[] cptes=mapper.readValue(resp2, Compte[].class);
```

```
package test;
import java.util.Date;
public class Compte {
    public Long code;
    public double solde;
    public Date dateCreation;
}
```

# Client REST avec Spring RestTemplate

```
@RestController
class ClientRestController{
    @Autowired
    private RestTemplate restTemplate;
    private String url="http://localhost:8080/web-api/api";
    @GetMapping("/listComptes")
    public String listCompte(){
        HttpHeaders headers = new HttpHeaders();
        headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
        HttpEntity<String> entity = new HttpEntity<String>(headers);
        ResponseEntity<String> responseEntity=restTemplate.exchange(url+"/comptes", HttpMethod.GET, entity, String.class);
        String response=responseEntity.getBody();
        return response;
    }
    @GetMapping("/listComptes2")
    public Compte listCompte2(){
        Compte cp=restTemplate.getForObject(url+"/comptes/1",Compte.class);
        return cp;
    }
    @GetMapping("/listComptes3")
    public List<Compte> listCompte3(){
        List<Compte> comptes=restTemplate.getForObject(url+"/comptes",List.class);
        return comptes;
    }
    @PostMapping("/comptes")
    public Compte save(@RequestBody Compte cp){
        Compte compte= restTemplate.postForObject(url+"/comptes",new HttpEntity<Compte>(cp),Compte.class);
        return compte;
    }
}
```



# Application Spring Web Services : Use Case

