

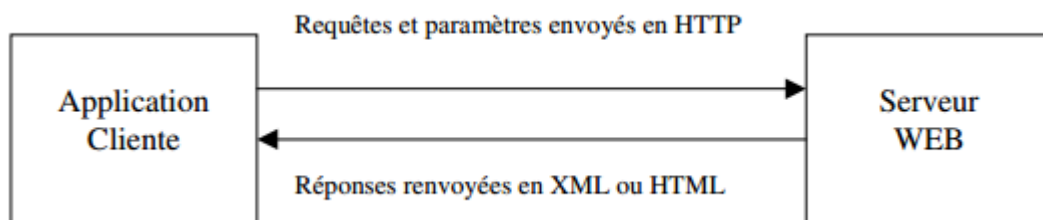
1 Introduction

Un "Service Web" est une application logicielle à laquelle on peut accéder à distance à partir de différents langages basés sur **XML (SOAP)**, **JSON (REST)**.

Un "Service Web" est identifié par une URL, comme n'importe quel site Web. Il s'exécute sur un "Serveur d'Applications".

De nombreuses normes sous-tendent cette architecture : "**SOAP**" pour l'échange de messages, "**XML**" langage de base pour décrire tous les documents sur lesquels les messages sont construits, "**HTTP**" pour transporter les messages, "**WSDL**" pour décrire les services et enfin "**UDDI**" pour les publier.

L'approche "Services Web" constitue un changement fondamental dans la manière de concevoir et réaliser les applications informatiques et de programmer les ordinateurs. A court terme, il est probable que cette approche se substitue aux architectures et systèmes basés sur des LAN ou sur Internet.



2 TP : Réaliser un service web qui offre deux méthodes :

Pour développer les web service on va utiliser **JAXWS**

JAXWS fournis un certain nombre d'annotation :

- **@WebService**(serviceName="BanqueWS")
- **@WebMethod**(operationName="conversionEuroDinar")
- **@WebParam**(name="montant")double mnt

Etape 1 :

Pour commencer on va créer un nouveau projet java classique (new java project) : TPws
Choisir le jdk(1.8.0_77)

Ensuite on va commencer par créer les objets métier

➔ Dans le package metier on crée la classe compte

```

*Compte.java BanqueService.java
1 package metier;
2
3 import java.io.Serializable;
4 import java.util.Date;
5
6 import javax.xml.bind.annotation.XmlRootElement;
7
8 //pour dire que le xml qu'on va générer représente un compte
9
10 @XmlRootElement(name="Compte")
11 public class Compte implements Serializable{
12
13     private Long code;
14     private double solde;
15     private Date dateCreation;
16
17     // générer les getters et les setters
18     //générer les constructeurs (un vide, avec parametre)
19

```

Par la suite on va créer la classe BanqueService dans le package service « c'est une classe java classique »

```

Compte.java BanqueService.java
1 package service;
2
3
4
5 import java.util.ArrayList;
6 import java.util.Date;
7 import java.util.List;
8
9 import metier.Compte;
10
11 public class BanqueService {
12     //methode pour faire la conversion
13     public double conversion(double mt){
14         return mt*6.55;
15     }
16     //methode pour retourner le solde du compte
17     public Compte getCompte(long code){
18         return new Compte(code, Math.random()*8000, new Date());
19     }
20     //methode retourne la liste des comptes
21     public List<Compte> getComptes(){
22         List<Compte> cptes = new ArrayList<>();
23         cptes.add( new Compte(1L, Math.random()*8000, new Date()));
24         cptes.add( new Compte(2L, Math.random()*8000, new Date()));
25         return cptes;
26     }
27
28 }

```

On rajoute les annotations JAX-WS pour que ça soit un web service

```
Compte.java BanqueService.java
1 package service;
2
3
4
5 import java.util.ArrayList;
6 import java.util.Date;
7 import java.util.List;
8
9 import javax.jws.WebMethod;
10 import javax.jws.WebParam;
11 import javax.jws.WebService;
12 import metier.Compte;
13
14 @WebService(name="BanqueWS")
15 public class BanqueService {
16     @WebMethod(operationName="ConversionEuroToFranc")
17     public double conversion(@WebParam(name="montant") double mt) {
18         return mt*6.55;
19     }
20     @WebMethod
21     public Compte getCompte(@WebParam(name="code") long code) {
22         return new Compte(code, Math.random()*8000, new Date());
23     }
24
25     @WebMethod
26     public List<Compte> getComptes() {
27         List<Compte> cptes = new ArrayList<>();
28         cptes.add( new Compte(1L, Math.random()*8000, new Date()));
29         cptes.add( new Compte(2L, Math.random()*8000, new Date()));
30         return cptes;
31     }
}
```

Voilà on a développé un web service maintenant on doit le déployer, pour cela il nous faut un serveur (on va créer notre propre serveur et dans notre cas c'est une classe nommée ServeurJWS avec la méthode main....)

Note : Si on veut déployer dans un serveur d'application pas besoin d'y ajouter quoi que ce soit le serveur d'application va comprendre que c'est un **Web Service**

Mais dans notre exemple on va ajouter une application java; ça veut dire je vais créer mon propre serveur dans lequel je vais déployer le web service

```

Compte.java  BanqueService.java  ServeurJWS.java
1 package service;
2
3 import javax.xml.ws.Endpoint;
4
5 public class ServeurJWS {
6
7     public static void main(String[] args) {
8         //specifier le port ou l'adresse d'accès
9         //localhost peut être remplacé par l'adresse IP c
10        String url="http://localhost:8585/";
11        Endpoint.publish(url, new BanqueService());
12        System.out.println(url);
13
14    }
15
16 }
17 }

```

Executer l'application avec run java Application

Maintenant on peut tester sur un navigateur tapez : <http://localhost:8585/BanqueWS?wsdl>
On aura un wsdl généré automatiquement par le server

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

```

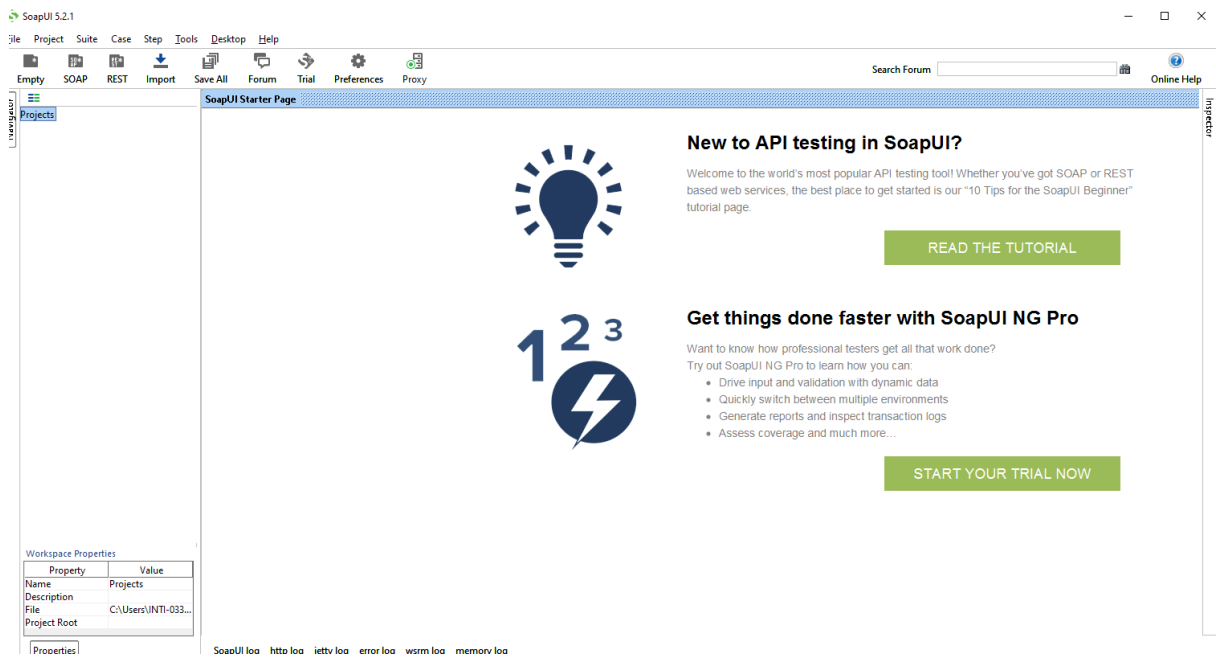
<!--
  Published by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e.
-->
<!--
  Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e.
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/soap/" xmlns:tns="http://service/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="BanqueServiceService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://service/" 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="ConversionEuroToFranc">
    <part name="parameters" element="tns:ConversionEuroToFranc"/>
  </message>
  <message name="ConversionEuroToFrancResponse">
    <part name="parameters" element="tns:ConversionEuroToFrancResponse"/>
  </message>
  <message name="getComptes">
    <part name="parameters" element="tns:getComptes"/>
  </message>
  <message name="getComptesResponse">
    <part name="parameters" element="tns:getComptesResponse"/>
  </message>
  <portType name="BanqueWS">
    <operation name="getCompte">
      <input wsam:Action="http://service/BanqueWS/getCompteRequest" message="tns:getCompte"/>
      <output wsam:Action="http://service/BanqueWS/getCompteResponse" message="tns:getCompteResponse"/>
    </operation>
    <operation name="ConversionEuroToFranc">
      <input wsam:Action="http://service/BanqueWS/ConversionEuroToFrancRequest" message="tns:ConversionEuroToFranc"/>
      <output wsam:Action="http://service/BanqueWS/ConversionEuroToFrancResponse" message="tns:ConversionEuroToFrancResponse"/>
    </operation>
    <operation name="getComptes">
      <input wsam:Action="http://service/BanqueWS/getComptesRequest" message="tns:getComptes"/>
      <output wsam:Action="http://service/BanqueWS/getComptesResponse" message="tns:getComptesResponse"/>
    </operation>
  </portType>
  <binding name="BanqueWSPortBinding" type="tns:BanqueWS">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="getCompte">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
    <operation name="ConversionEuroToFranc">
      <soap:operation soapAction="" />

```

Etape 2 : Un client pour accéder au WS SOAP il a besoin de WSDL

Pour tester les méthodes du web service on aura besoin d'un client, on peut utiliser un éditeur XML comme oxygène ou une interface SoapUi (interface graphique qui permet de tester les web service), que vous pouvez télécharger

Test notre service WEB avec le client : SoapUi

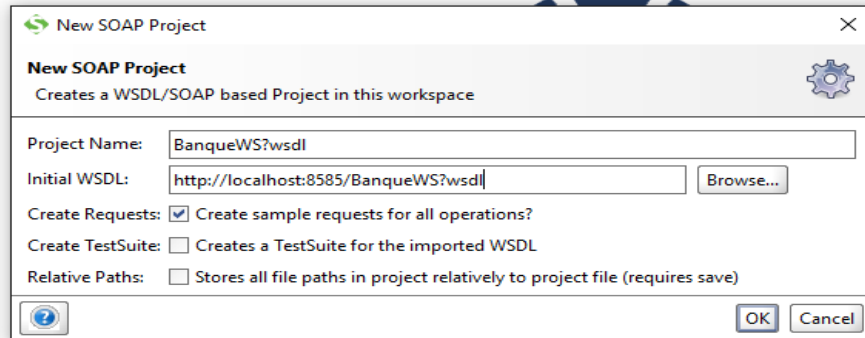


Il existe d'autres outils pour tester les WS comme oXygen:



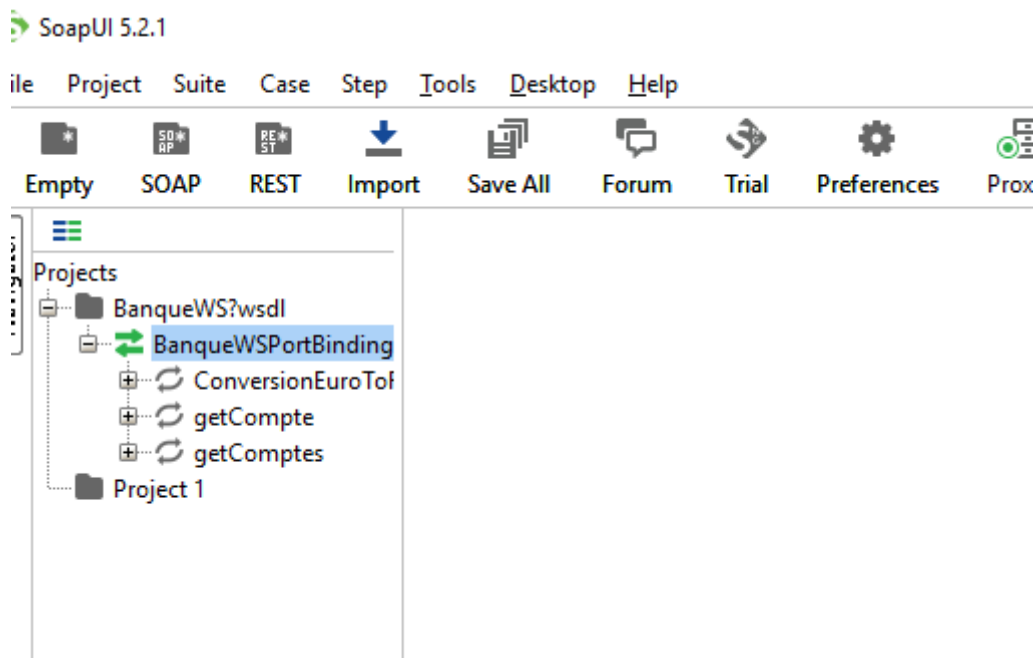
Avec le client SoapUI :

on crée un nouveau projet SOAP on copie notre URL

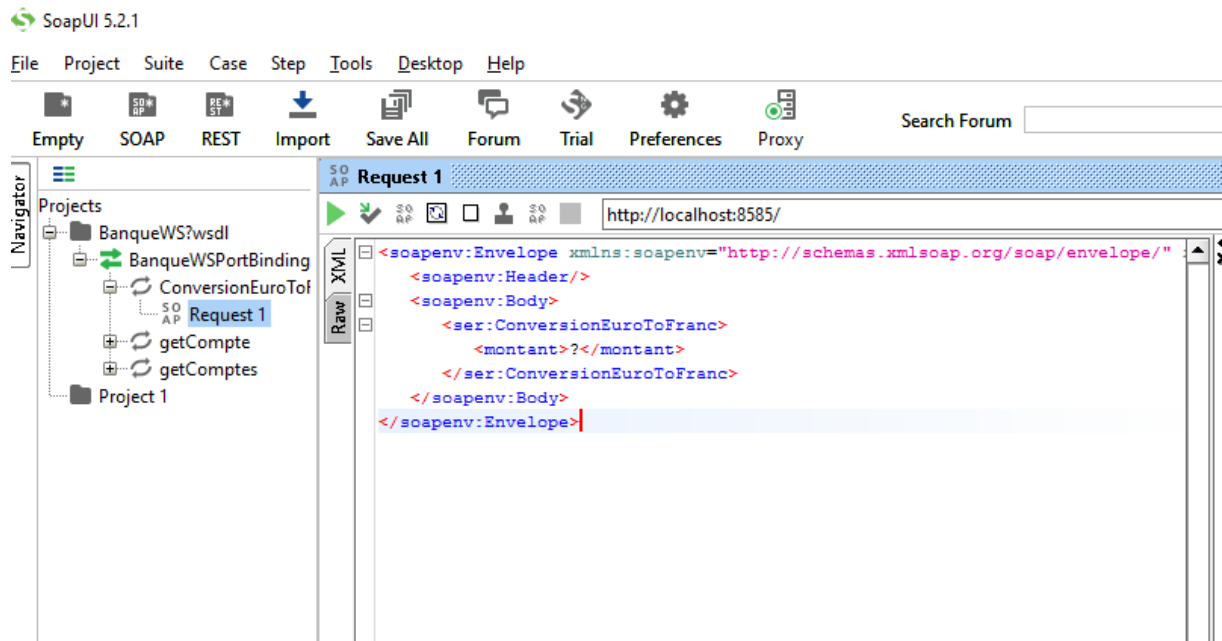


SOAPUI va se connecter au web service, il va récupérer le WSDL

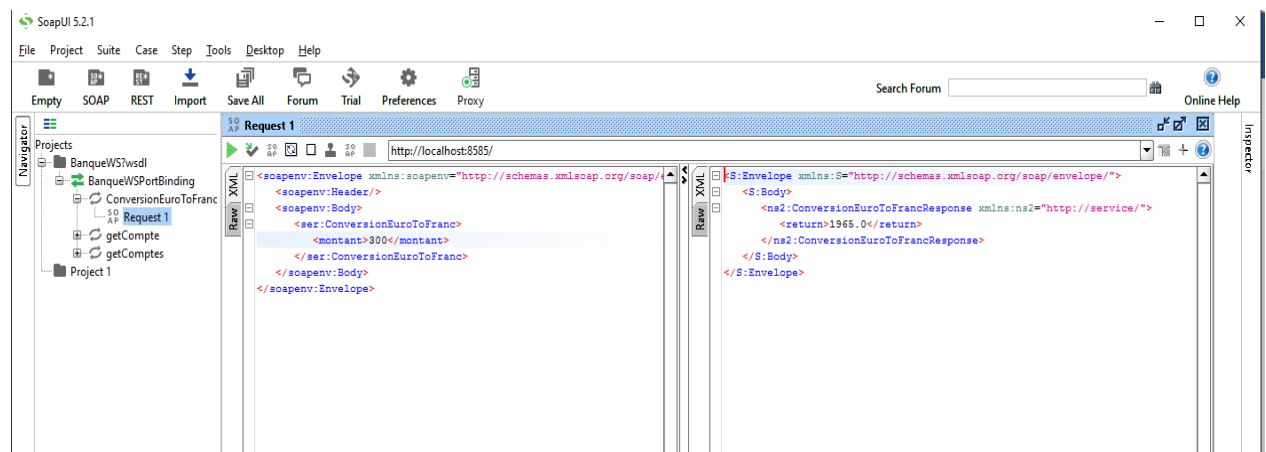
Par la suite il nous affiche la structure du web service comme on le voit sur l'image ci-dessous



Si je veux tester la méthode conversion je fais un double clic sur la méthode, il nous affiche la requête SOAP à envoyer pour tester cette méthode



Par la suite entrez le montant à convertir



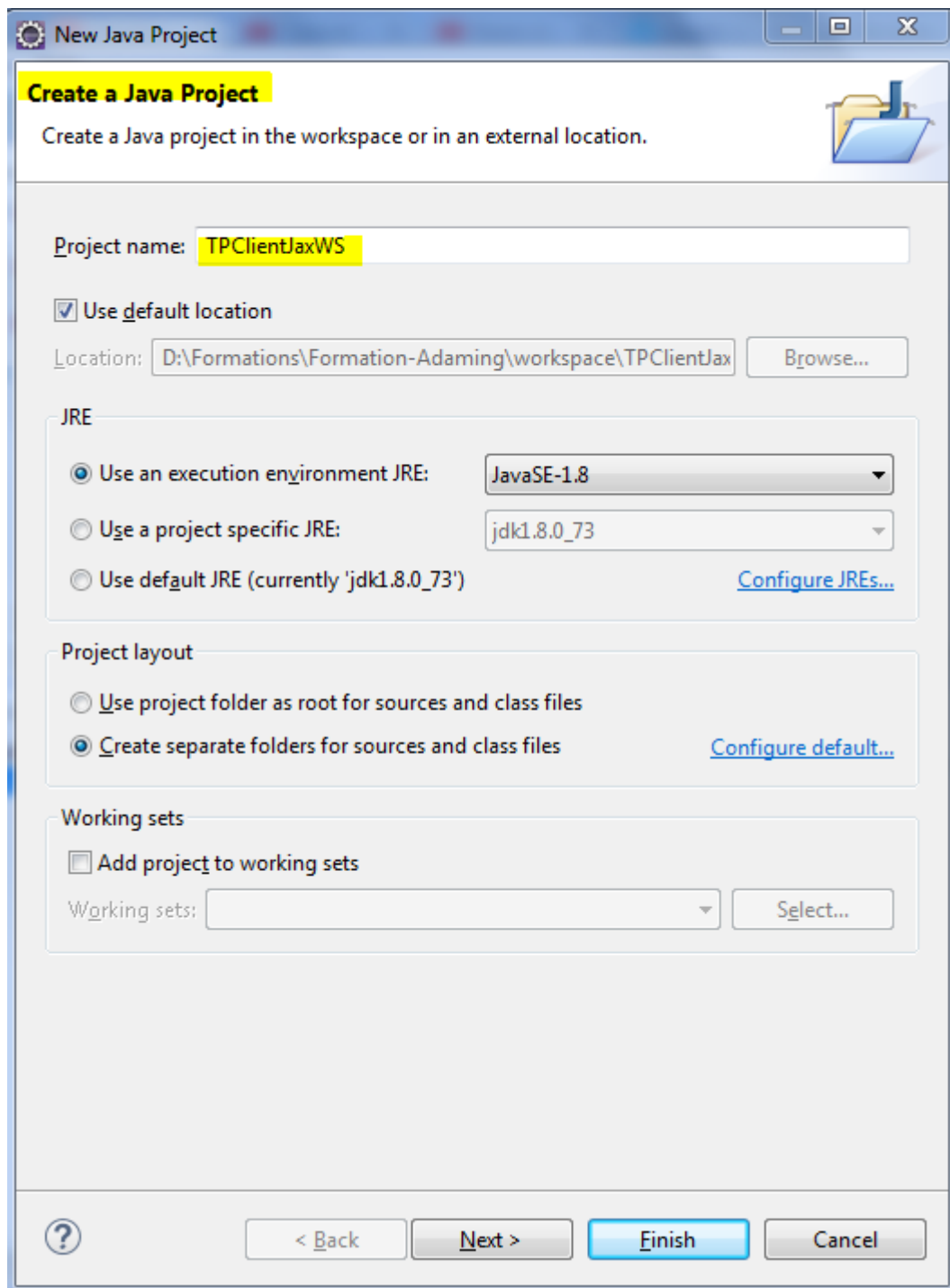
Voilà la réalité !!!! c'est ces messages qui circule quand on envoie une requête
(vous pouvez tester les deux autres méthodes)

Bravoooooooooo...;)

On a réalisé notre Web service **BanqueWS** qui est déployer dans notre serveur maintenant on peut créer notre client afin de consommer notre Web Service

On va créer notre application java **client consommateur** ?

Pour information le client a besoin juste de l'adresse url (ou se trouve le WS) pour récupérer le WSDL



The screenshot shows the 'New Java Project' dialog box in the Eclipse IDE. The title bar reads 'New Java Project'. The main heading is 'Create a Java Project' with a sub-instruction: 'Create a Java project in the workspace or in an external location.' The 'Project name' field is filled with 'TPClientJaxWS'. The 'Use default location' checkbox is checked, and the 'Location' field shows the path 'D:\Formations\Formation-Adaming\workspace\TPClientJax'. Under the 'JRE' section, 'Use an execution environment JRE:' is selected with 'JavaSE-1.8' chosen from the dropdown. Other options include 'Use a project specific JRE:' (set to 'jdk1.8.0_73') and 'Use default JRE (currently 'jdk1.8.0_73')'. The 'Project layout' section has 'Create separate folders for sources and class files' selected. The 'Working sets' section has 'Add project to working sets' unchecked. At the bottom, there are buttons for '< Back', 'Next >', 'Finish' (highlighted in blue), and 'Cancel'.

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: **TPClientJaxWS**

☒ Use default location

Location: D:\Formations\Formation-Adaming\workspace\TPClientJax [Browse...](#)

JRE

☒ Use an execution environment JRE: JavaSE-1.8

☐ Use a project specific JRE: jdk1.8.0_73

☐ Use default JRE (currently 'jdk1.8.0_73') [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

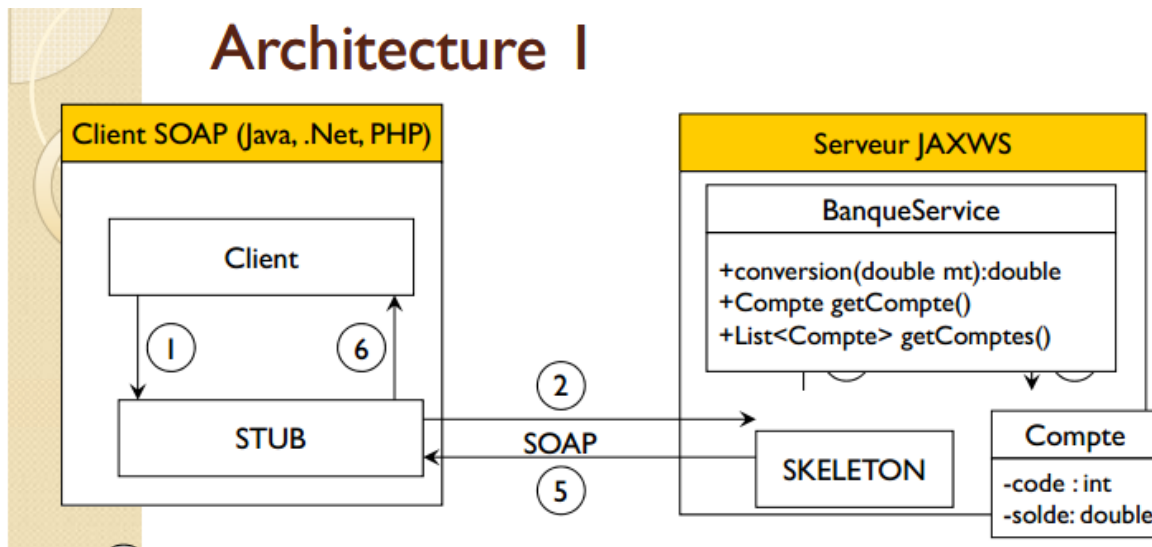
Working sets

☐ Add project to working sets

Working sets: [Select...](#)

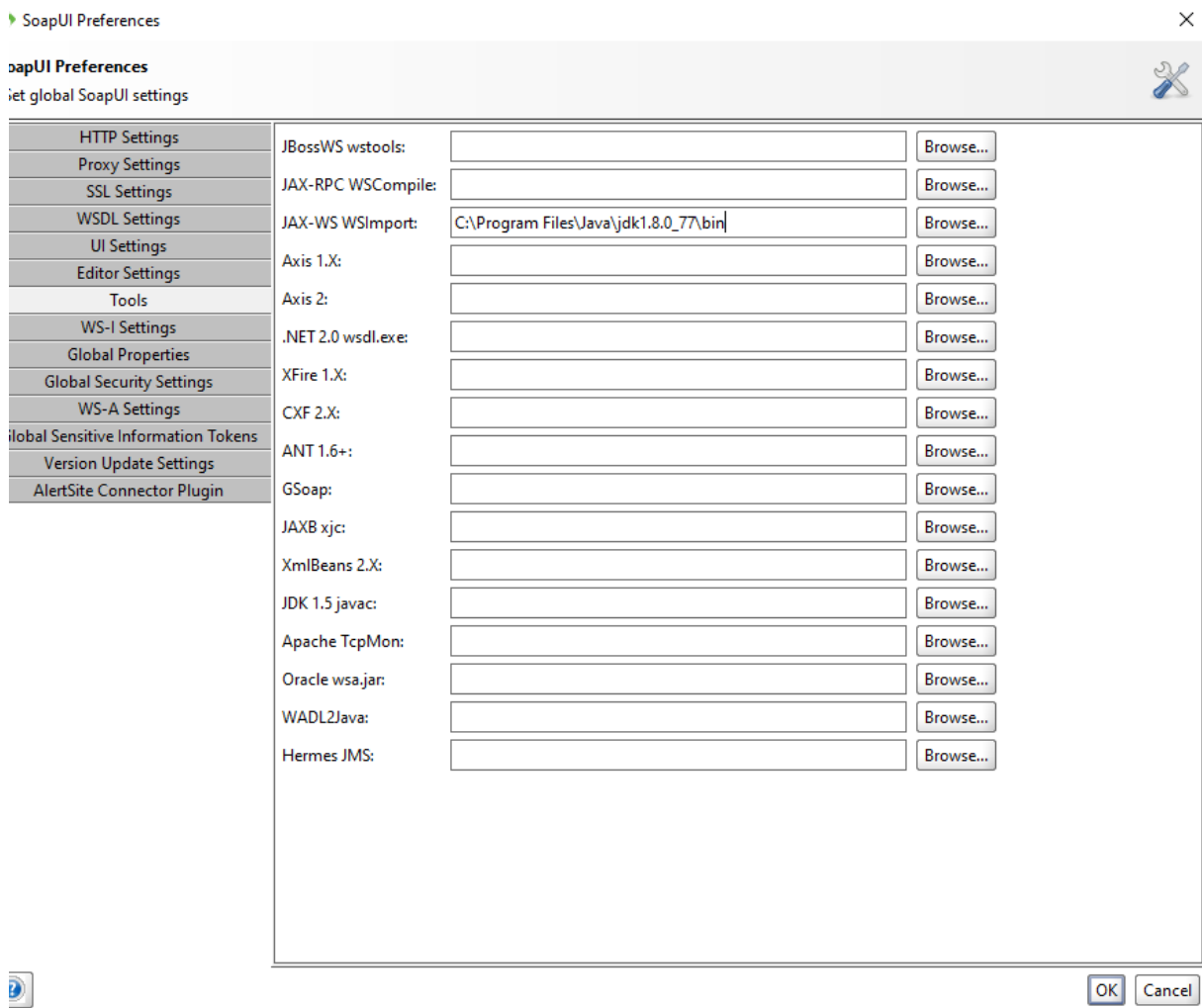
[?](#) [< Back](#) [Next >](#) **Finish** [Cancel](#)

Ensuite je dois générer un proxy ou un STUB qui permet au client d'envoyer la requête SOAP

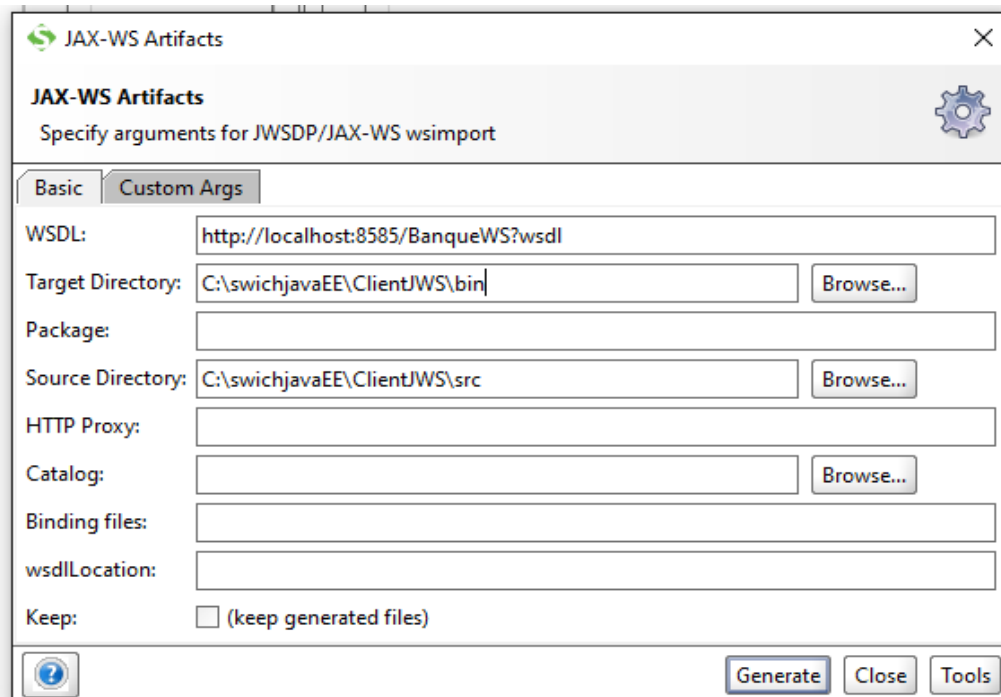


On va faire les configurations nécessaires

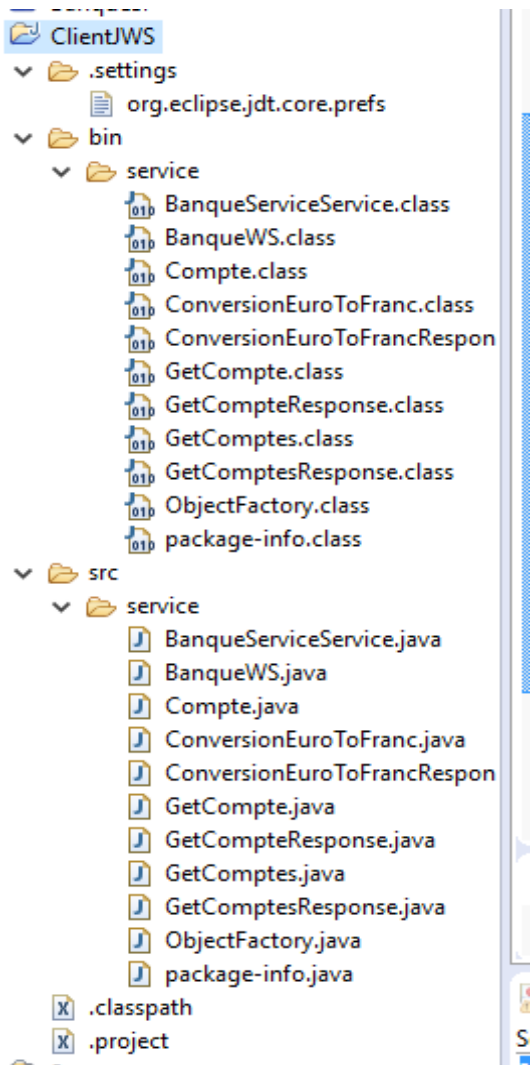
La première spécifier le chemin du Jdk (File → preferences)



La deuxième : pour générer les classes (tools → JAX-WS Artifacts) on donne l'adresse du web service & le chemin ou il va générer les classe (pour nous c'est dans src du projet ClientJWS)



Voilà les classes générer



Par la suite on va créer la classe client pour utiliser le proxy ou le stub

```
SeueurJWS.java  Compte.java  ClientJWS.java  X
1 package service;
2
3 public class ClientWS {
4
5     public static void main(String[] args) {
6         //créer un Stub
7         BanqueWS stub=new BanqueServiceService().getBanqueWSPort();
8         //pour appeler les methodes
9         double res=stub.conversionEuroToFranc(600);
10        System.out.println("la conversion de 600 euro en Franc = " +res);
11    }
12
13 }
14
```

Problems @ Javadoc Declaration Console X

terminated> ClientWS [Java Application] C:\Program Files\Java\jdk1.8.0_77\bin\javaw.exe (6 juin 2016 00:29:08)

la conversion de 600 euro en Franc = 3930.0

Ici le client appelle la méthode conversion et c'est stub qui envoi la requête SOAP vers le serveur il récupère le résultat et il l'envoi
On fait de même pour les autres méthodes

```
ServerWS.java  Compte.java  ClientWS.java  ✖
3 import java.util.List;
4
5 public class ClientWS {
6
7     public static void main(String[] args) {
8         //créer un Stub
9         BanqueWS stub=new BanqueServiceService().getBanqueWSPort();
10        //pour appeler les methodes
11        double res=stub.conversionEuroToFranc(600);
12        System.out.println("la conversion de 600 euro en Franc = " +res);
13
14        //on va tester les autres méthodes
15        Compte cp=stub.getCompte(1L);
16        System.out.println(cp.getSolde());
17        System.out.println("-----");
18        List<Compte> cptes=stub.getComptes();
19        for(Compte c:cptes){
20            System.out.println(c.getCode());
21            System.out.println(c.getSolde());|
22        }
23
24
25    }
```

Problems Javadoc Declaration Console ✖

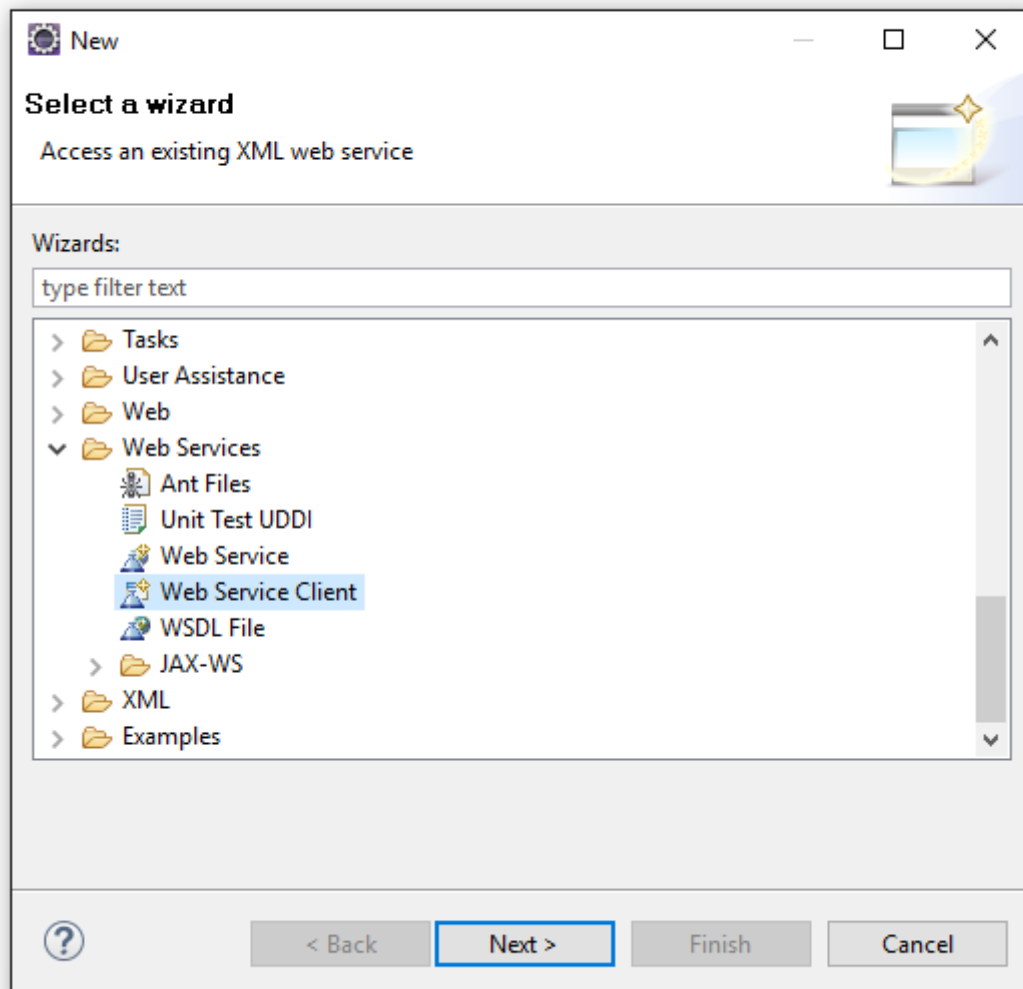
<terminated> ClientWS [Java Application] C:\Program Files\Java\jdk1.8.0_77\bin\javaw.exe (6 juin 2016 00:40:17)

la conversion de 600 euro en Franc = 3930.0
4635.362928133085


1
5701.766146933637
2
3214.842136852079

Jusqu'au là nos applications sont des applications desktop on va créer un nouveau projet web dynamique « Dynamic Web Project » ClientBanque

➔ J'ai besoin d'un proxy ou stub que je vais générer à partir d'éclipse cette fois-ci



On co
lientB



 Web Service Client

Web Services

Select a service definition and move the slider to set the level of client generation.

Service definition:

Client type:



Configuration:
[Server runtime: Tomcat v8.0 Server](#)
[Web service runtime: Apache Axis](#)
[Client project: ClientBanque](#)

☐ Monitor the Web service

