# Travaux pratiques Services Web SOAP
# Environnement de travail

Les services seront déployés sous Glassfish, sous la forme d'une archive war (WEB-INF mais sans fichier web.xml, et répertoire classes qui contient les .class des services déployés).

Vous placerez le(s) classes représentant les services dans un package.

L'utilitaire wsgen (à utiliser après avoir compilé le service) permet de générer les fichiers qui permettront la communication avec le service Web déployé

```
wsgen -cp WEB-INF/classes:$CLASSPATH -d WEB-INF/classes -wsdl ServiceSOAP
```

où *ServiceSOAP* est le nom de la classe implémentant le service.

L'archive web (war) que vous déployez contient les fichiers précédemment générés

```
jar cvf WSSOAP.war WEB-INF/* (avec éventuellement index.html)
```

Pour consulter le WSDL après déploiement

```
localhost:8080/WSSOAP/ServiceSOAPService?WSDL
```

Notez l'ajout de **Service** après le nom de votre classe !

Pour tester le service

```
localhost:8080/WSSOAP/ServiceSOAPService?Tester
```

Rappel pour déployer en ligne de commandes :

```
asadmin deploy --force NomArchive.war
```

Pour communiquer avec le service via un client C#, du côté du client on génère les classes avec l'outil wsdl du framework .NET:

```
wsdl http://NomMachineServeur:NumPortHTTP/WSSOAP/ServiceSOAPService?wsdl
```

Pour compiler le client

```
mcs ClientWS.cs ServiceSOAPService.cs -r:System.Web.Services
```

Pour communiquer avec le service via un client Java, du côté du client on génère les classes avec l'outil `wsimport` :

```
wsimport http://localhost:8080/WSSOAP/ServiceSOAPService?WSDL
```

Vous pouvez utiliser l'option –keep si vous souhaitez conserver les .java générés. Sinon ils sont automatiquement supprimés pour nettoyer le répertoire.

# TP1 – Production et consommation de services SOAP

**Question 1 – Premier service SOAP : le traditionnel "Salut" !**

Créez (avec un éditeur de texte basique) un service auquel on transmet une chaîne de caractères et qui retourne un message de salutation. Consultez le fichier WSDL associé au service via le navigateur, utilisez le service dans le navigateur.

**Question 2 – Consommation du service en Java – Première version**

Ecrivez une première version du client en utilisant un objet Service, construit en transmettant l'URL du wsdl, ainsi qu'un qualified name QName (construit avec 2 arguments, le namespace et le nom de ce service). Sur ce service, obtenez le port pour utiliser la méthode proposée. Vous aurez copié chez le client le fichier compilé qui correspond à l'interface uniquement.

**Question 3 – Consommation du service en Java – Seconde version**

Utilisez maintenant l'outil wsimport pour générer les classes chez le client, et reprenez le client pour y déclarer un objet IBonjour. Obtenez le port pour utiliser la méthode proposée.

**Question 4 – Consommation du service en C#**

Ecrivez un client C# qui déclare un objet BonjourService et qui appelle la méthode proposée. Vous aurez à générer les fichiers nécessaires au client avec wsdl.

**Question 5 – Consommation de service sur le net**

Retrouvez les services de TextCasing proposés sur le net, testez-en un en ligne et ensuite à travers un client C# puis un client Java.

**Question 6 – Consommation de service sur le net**

Dans cet exercice, on souhaite parcourir les résultats XML d'un service SOAP à la recherche d'un sous-élément retourné. Vous pouvez pour cela utiliser ces sites :

https://www.crcind.com/csp/samples/SOAP.Demo.cls

http://wsgeoip.lavasoft.com/ipservice.asmx

Consultez les définitions wsdl du service qui renvoie la ville à partir d'un code postal aux USA (par exemple 02118), ou de celui qui permet d'obtenir le pays à partir d'une adresse IP (GetIpLocation de GeoIPService). Testez dans un navigateur.

Pour un client C# : utilisez la classe System.Xml.XmlNode et les propriétés FirstChild et InnerText pour accéder aux données.

Pour un client de GeoIP en Java : écrivez une classe (éventuellement dans le même fichier que le client), annotez-la avec @XmlRootElement (attribut name qui représente l'élément XML), et annotez les champs avec @XmlElement, et ajoutez les getters.

Dans le client, utilisez un Unmarshaller, créé avec la méthode createUnmarshaller appelée sur un objet de type JAXBContext. Un JAXBContext peut être initialisé via la méthode statique newInstance de cette même classe, avec en paramètre la classe vers laquelle vous souhaitez "unmarhaller" (on utilise la méthode unmarshall de l'Unmarshaller pour obtenir l'objet).