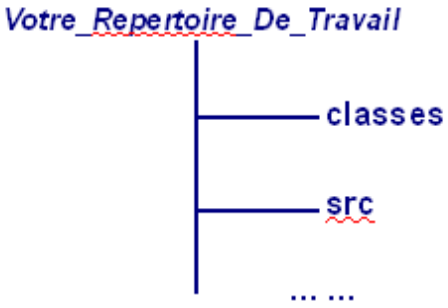
TP Web Services

Les parties ci dessous écrite en vert, gras sont les parties pour une correction.
Dans ce TP le sigle WS signifie Web Services.

Installation d'Axis

Créer un répertoire de travail. Par la suite, on conseille d'avoir une arborescence (classique) de répertoires comme :



où `src` est le répertoire contenant les fichiers sources de vos programmes Java et `classes` les fichiers compilés, byte-code des fichiers sources.

Récupérer le fichier qui se trouve à l'URL `http://cedric.cnam.fr/~farinone/SMB111/ADistribuer.zip`. Ouvrer cette archive `.zip` dans votre répertoire de travail pour ce TP. Dans cette archive, vous trouverez : `axis.war` (qu'on peut aussi récupérer à `http://ws.apache.org/axis/`), `log4j.properties`, des fichiers de scripts (`1posAxisVariables.bat`, ...) à adapter à votre machine pour exécuter et compiler, `HelloClient.java`, ... et `Reponse.jws`. dans `src\hello`.

REMARQUE IMPORTANTE :
Pour lancer une compilation ou une exécution d'un client web services dans une fenêtre, il faut toujours avoir lancé au préalable une fois le script `1posAxisVariables.bat`.

1°) Axis est une application web. Déployer cette application web c'est à dire le fichier `axis.war` fourni dans tomcat (cf. le TP sur les servlets/JSP). Vérifier le bon fonctionnement en vous connectant sur
`http://localhost:8080/axis/` puis en cliquant sur le lien `Validation`. Quel type de page le lien `Validation` amène t il ?

L'URL `http://localhost:8080/axis/Validation` amène à une page JSP.

Développement du client et d'un serveur WS

2°) Ecrire une service `.jws` qui est un compte bancaire proposant les méthodes :

```
public void debiter(double montant) {
}
public void crediter(double montant) {
```

```
}
public double getSolde() {
}
```
Ecrire le code de ces méthodes dans la classe `CompteBancaire.jws`.

Il faut évidemment construire une classe `CompteBancaire.jws` qui pourrait être :

```
public class CompteBancaire {
     private double solde;

     public CompteBancaire() {solde = 300.00;}

     public void debiter(double montant) {
          solde -= montant;
     }

     public void crediter(double montant) {
          solde += montant;
     }

     public double getSolde() {
          return solde;
     }

     public String toString() {
          return "" + solde;
     }
}
```

3°) Déployer ce service dans axis. Plus précisément on copiera ce fichier `CompteBancaire.jws` dans le répertoire `%TOMCAT_HOME%\webapps\axis` où `%TOMCAT_HOME%` est le nom du répertoire d'installation de tomcat.
Il faut bien faire ce qui est indiqué ci dessus. Par la suite on pourra travailler directement sur ce fichier `CompteBancaire.jws` installé coté serveur.

Vérifier que ce WS a bien été déployé en vous connectant à l'URL `http://localhost:8080/axis/CompteBancaire.jws`. Lisez sa description WSDL.

En ce connectant à cette URL `http://localhost:8080/axis/CompteBancaire.jws`, on arrive de proche en proche à une description WSDL du service web. Cette description est en XML (évidemment cf. le cours) et est relativement lisible et compréhensible.

4°) Ecrire un client WS qui interroge ce service. On pourra utiliser le client proposé ci dessous :
`fichier CompteBancaireWSClient.java`
`----------`

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import org.apache.axis.utils.Options;
import javax.xml.rpc.ParameterMode;

public class CompteBancaireWSClient
{
  public static void main(String [] args) throws Exception {
    Options options = new Options(args);
    String endpoint = "http://localhost:" + options.getPort() +
      "/axis/CompteBancaire.jws";
    args = options.getRemainingArgs();
    if (args == null) usage();
    else if (args.length == 1) {
      if (!args[0].equals ("getSolde")) usage();
      else {
        // A Completer comme ci dessous
        Service  service = new Service();
        Call     call    = (Call) service.createCall();
        call.setTargetEndpointAddress( new java.net.URL(endpoint) );
        call.setOperationName( "getSolde" );
        call.setReturnType( XMLType.XSD_DOUBLE );
        Double soldeDouble = (Double)call.invoke( new Object [] { });
        System.out.println("le solde est : " + soldeDouble);
      }
    } else if (args.length == 2) {
      if (args[0].equals ("debiter") || args[0].equals ("crediter")) {
        String nom = args[0];
        Service  service = new Service();
        Call     call    = (Call) service.createCall();
        call.setTargetEndpointAddress( new java.net.URL(endpoint) );
        if (nom.equals ("debiter")) {
          call.setOperationName( "debiter" );
        } else if (args[0].equals ("crediter")) {
          call.setOperationName( "crediter" );
        }
        call.addParameter("montant", XMLType.XSD_DOUBLE, ParameterMode.IN );
        call.setReturnType( XMLType.AXIS_VOID );
        Double argumentDouble = new Double(Double.parseDouble(args[1]));
        call.invoke( new Object [] { argumentDouble });
      } else usage();
    }
  }

  public static void usage() {
      System.err.println("Usage: java CompteBancaire [debiter | crediter |
solde] [valeur]");
    return;
  }
}
```

De même ce client WS est relativement compréhensible pour peu qu'on est déjà vu ce type de client (cf. le cours).

5°) Compiler ce client dans une fenêtre où a été lancé le script `1posAxisVariables.bat` ! Il faudra d'ailleurs adapté le contenu du fichier à votre environnement essentiellement en mettant les noms de vos répertoires et plus précisément en ajustant correctement la variable

AXIS_HOME de ce script pour que %AXIS_HOME%\lib soit un répertoire valide contenant des .jar. Voir pour cela ce que vous avez obtenu lorsque vous avez ouvert le fichier ADistribuer.zip.
Pour compiler, on pourra, après avoir créé le répertoire classes, utiliser la commande :

```
javac -d classes src\CompteBancaireWSClient.java
```

lancé dans le répertoire (et la fenêtre) adéquate !

En suivant les indications ci dessus, on ne devrait pas avoir de problème.

6°) Lancer ce client par la commande :

```
java CompteBancaireWSClient getSolde
```

Cette commande permet simplement d'avoir le service getSolde. On lancera l'exécution dans un répertoire contenant le fichier CompteBancaireWSClient.class et le fichier log4j.properties qui est un fichier récupéré de l'archive ADistribuer.zip. Ne pas se soucier des messages de warnings (WARN).

La commande `java CompteBancaireWSClient getSolde` permet d'atteindre le service web `CompteBancaireWSClient` et lui demande de lancer sa "méthode" (terme Java) `getSolde`. Cette méthode n'a pas d'argument. Voir le code du client pour s'en rendre compte.

Amélioration du serveur WS `CompteBancaire.jws`

Si les méthodes retirer() et crediter() du WS CompteBancaire.jws ont été codées simplement comme :

```
public void debiter(double montant) {
     solde -= montant;
}
```

et

```
public void crediter(double montant) {
     solde += montant;
}
```

elles n'assurent aucune persistance et donc ne conviennent pas pour garder trace de transactions sur ce compte bancaire : essayer avec le client précédent.

Dans cette section on est parti pour construire une véritable architecture 3-tiers : un client (la classe Java `CompteBancaireWSClient`) qui interroge un service distant WS (la classe `CompteBancaire`) qui utilise un service de persistence qu'il a lui même codé. On est proche d'une notion comme un EJB entité Bean Managed, mais ceci n'a pas été vu en cours et est un autre domaine (les Entreprise Java Bean).

7°) Coder "proprement " ces deux méthodes en assurant un service de persistence. On pourra utiliser les classes XMLEncoder et XMLDecoder du paquetage java.beans de Java SE.
On pourra, par exemple, utiliser le code ci dessous qui code correctement les 4 méthodes crediter(), debiter(), getSolde() ainsi que le constructeur de la classe CompteBancaire.jws pour qu'il ne réinitialise pas le solde à chaque invocation du WS à une valeur fixe mais qu'il obtienne ce solde à partir de la donnée persistante :

```
import java.beans.*;
import java.io.*;

public class CompteBancaire {
  // private double solde;
  private static boolean premiereFois = true;

  public CompteBancaire() {
    System.out.println("debut de public CompteBancaire()");
    try {
      if (premiereFois == true) {
        System.out.println("premiereFois == true");
        double leSolde = 300.00;
        XMLEncoder e = new XMLEncoder(
            new BufferedOutputStream(
                new FileOutputStream("compte.xml")));
        e.writeObject(new Double(leSolde));
        e.close();
        premiereFois = false;
    } else { // premiereFois == false
      System.out.println("premiereFois == false");
      // et on ne fait rien !!
    }
    } catch (Exception exp) {
       System.out.println ("Pb exception dans public CompteBancaire() : " +
exp);
    }
  }

  public void debiter(double montant) {
    System.out.println("debut de public void debiter(double montant)");
    try {
      XMLDecoder d = new XMLDecoder(
          new BufferedInputStream(
              new FileInputStream("compte.xml")));
      Double soldeDouble = (Double)d.readObject();
      double leSolde = soldeDouble.doubleValue() - montant;
      d.close();

      XMLEncoder e = new XMLEncoder(
          new BufferedOutputStream(
              new FileOutputStream("compte.xml")));
      e.writeObject(new Double(leSolde));
      e.close();
      // solde -= montant;
    } catch (Exception exp) {
       System.out.println ("Pb exception dans    public void debiter(double
montant) : " + exp);
    }
  }

  public void crediter(double montant) {
    System.out.println("debut de public void crediter(double montant)");
    try {
      XMLDecoder d = new XMLDecoder(
          new BufferedInputStream(
              new FileInputStream("compte.xml")));
      Double soldeDouble = (Double)d.readObject();
      double leSolde = soldeDouble.doubleValue() + montant;
```

```
      d.close();

      XMLEncoder e = new XMLEncoder(
          new BufferedOutputStream(
              new FileOutputStream("compte.xml")));
      e.writeObject(new Double(leSolde));
      e.close();
    } catch (Exception exp) {
       System.out.println ("Pb exception dans public void crediter(double
montant) : " + exp);
    }
  }

  public double getSolde() {
    System.out.println("debut de public double getSolde()");
    try {
      XMLDecoder d = new XMLDecoder(
          new BufferedInputStream(
              new FileInputStream("compte.xml")));
      Double soldeDouble = (Double)d.readObject();
      d.close();
      return soldeDouble.doubleValue();
    } catch (Exception exp) {
       System.out.println ("Pb exception dans public double getSolde() : " +
exp);
      return -111.00;
    }
  }
}
```

Le code ci dessus est complet et pas difficile à comprendre. Par faute de temps il est donné entièrement mais un bon exercice serait de le réécrire (où d'en écrire un équivalent) en masquant cette solution !

9°) Déployer ce WS dans Axis (cf. question 3°)).

On passe par le lien Tomcat Manager du conteneur d'application web tomcat (cf. le cours et le TP sur les servlets/JSP).

10°) Lancer le client par diverses commandes comme :

```
java CompteBancaireWSClient debiter 100.00
java CompteBancaireWSClient crediter 150.00
```

et vérifier régulièrement le solde par la commande :

```
java CompteBancaireWSClient getSolde
```

Tout devrait alors bien fonctionner ici y compris (et surtout) le service de persistence (que l'on avait pas si on utilise la simple classe donnée en solution à la question 2°)

Remarque :
Si vous avez des erreurs, il faut les corriger (eh oui !). Plus sérieusement, il faut faire un cycle : édition, compilation (éventuelle) et déploiement du WS. Il est conseillé aussi de, parfois, redémarrer `tomcat`.

Et voilà ;-)