TP n°2 : Développement de Web Services avec l'API JAX-WS

Le but de ce deuxième TP est d'apprendre à manipuler l'API JAX-WS pour le développement de Web Services étendus à partir de la plateforme de développement Java. La leçon insiste sur le développement d'un Web Service suivant les approches Bottom / Up et Top / Down puis sur le développement de la partie cliente d'un Web Service et enfin sur la manipulation de Handler.

But pédagogique : Transformation d'un POJO Java en Web Service, génération des artifacts à partir d'une description WSDL, utilisation de l'outillage fourni par JavaSE 6, mise en place d'un intercepteur (handler), clients Web Service en mode synchrone et asynchrone, outils wsgen et wsimport.

Prérequis techniques :

- Java Development Kit 6 (jdk1.6uXX) téléchargeable sur à l'adresse URL suivante: http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Eclipse IDE for Java EE developers v3.X téléchargeable à l'adresse URL suivante : http://www.eclipse.org/downloads/
- Apache Tomcat 6.XX téléchargeable à l'adresse URL suivante :
- http://tomcat.apache.org/download-60.cgi
- L'API JAX-WS 2.2.5 téléchargeable à l'adresse URL suivante : http://jax-ws.java.net/2.2.5/
- L'application SOAP-UI téléchargeable à l'adresse URL suivante : http://www.soapui.org/soapUI-News/soapui-401-the-bug-fix-release.html

Installation de l'API JAX-WS dans le serveur d'application Tomcat

But : Nativement, Tomcat n'est pas configuré pour exécuter des Web Services contrairement à d'autres serveurs d'application comme GlassFish. Pour pouvoir exécuter les applications développées durant ce TP, il faut installer l'API JAX-WS dans Tomcat.

Étapes à suivre :

1. Décompresser l'archive « *JAXWS2.2.5-20110729.zip* »précédemment téléchargée.
2. Aller dans le dossier « jaxws-ri\lib ».
3. Copier toutes les librairies (*.jar) contenues dans ce dossier.
4. Aller dans le dossier d'installation d'Apache Tomcat. Par défaut : « *C:\Program Files\Apache Software Foundation\Tomcat 6.0* ».
5. Aller dans le dossier « lib » et y coller toutes les librairies copiées précédemment.

Exercice n°1 : Développer un Web Service suivant une approche montante « Bottom/Up »

But : Décrire un Web Service à partir d'une interface Java, implémenter le Web Service, déployer sous Tomcat, tester le Web Service via SOAP-UI.

Description : Le Web Service de ce premier exercice consiste à fournir des opérations pour la gestion d'un carnet d'adresses simplifié. Une opération pour ajouter une personne, une autre pour récupérer la liste complète et enfin une dernière opération pour récupérer une personne par un nom. Une personne est décrite par un nom (String) et une adresse (String).

Étapes à suivre :

1. Ouvrir l'environnement de développement intégré Eclipse.
2. Vérifier que vous êtes bien sur la perspective Java EE (regarder en haut à droite).
3. Créer un nouveau projet web « *File→new→Dynamic Web Project* », nommer le « *NotebookWebServiceExercice1* », et dans le champ « *Target Runtime* », sélectionner « *Apache Tomcat v6.0* ».
4. Cocher l'option « *Generate web.xml deployment descriptor* ».
5. Créer le package « *soa.jaxwslabs.notebookwebserviceexercice1* ».
6. Dans ce package, créer une classe nommée « *Person* ».
7. Dans cette classe, créer un attribut privé « *name* » de type String et un attribut privé « *address* » de type String. Ensuite, définir un constructeur par défaut, et un constructeur avec deux paramètres permettant d'initialiser les deux attributs. Enfin, créer les getters/setters des deux attributs.
8. Dans le même package, créer une interface nommée « NotebookService ». A l'intérieur de celle-ci, déclarer les trois méthodes suivantes :
    - void addPerson(Person p)
    - ArrayList<Person> getPersons()
    - Person getPersonAt(String name)

    Sur l'interface, ajouter l'annotation « *@WebService* » dans laquelle il faut insérer les attributs suivants :
    - name = NotebookService
    - targetNamespace = http://soa.jaxwslabs.notebookwebserviceexercice1
9. Toujours dans le même package, créer une classe nommée « NotebookServiceImpl » implémentant l'interface « *NotebookService* » et l'implémenter comme ci-dessous :

```java
package soa.jaxwslabs.notebookwebserviceexercice1;

import java.util.ArrayList;

import javax.jws.WebService;

@WebService(endpointInterface = "soa.jaxwslabs.notebookwebserviceexercice1.NotebookService",
                  serviceName="NotebookService",
                  portName="NotebookPort")
public class NotebookServiceImpl implements NotebookService {

      @Override
      public void addPerson(Person p) {
            if (p == null) {
                  throw new NullPointerException("L'objet 'Person' est null");
            }
            try {
                  Thread.sleep(5000);
            } catch (InterruptedException e) {
                  e.printStackTrace();
            }
            System.out.println("La méthode 'addPerson' a été invoquée : " + p.toString());
      }

      @Override
      public ArrayList<Person> getPersons() {
            ArrayList<Person> myPerson = new ArrayList<Person>();

            myPerson.add(new Person("Nesim FINTZ", "Cergy"));
            myPerson.add(new Person("Houcine SENOUSSI", "Cergy"));
            myPerson.add(new Person("Hervé DE MILLEVILLE", "Franconville"));

            System.out.println("La méthode 'getPersons' a été invoquée");

            return myPerson;
      }

      @Override
      public Person getPersonAt(String name) {
```

```
            if (name == null || name.equals("")) {
                  throw new NullPointerException("La chaine de caractères 'name' est null ou vide");
            }

            System.out.println("La méthode 'getPersonAt' a été invoquée");

            return new Person("Nesim FINTZ", "Cergy");
      }
}
```

10. Dans « Web Resources : Web Content/WEB-INF », ouvrir le fichier « *web.xml* » et y insérer le code suivant :

```
<?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" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      id="WebApp_ID" version="2.5">
      <listener>
            <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
      </listener>
      <servlet>
            <servlet-name>notebook</servlet-name>
            <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
            <load-on-startup>1</load-on-startup>
      </servlet>
      <servlet-mapping>
            <servlet-name>notebook</servlet-name>
            <url-pattern>/notebook</url-pattern>
      </servlet-mapping>
      <session-config>
            <session-timeout>120</session-timeout>
      </session-config>
</web-app>
```

11. Dans ce même dossier, créer un fichier XML nommé « *sun-jaxws.xml » et y insérer le code suivant :*

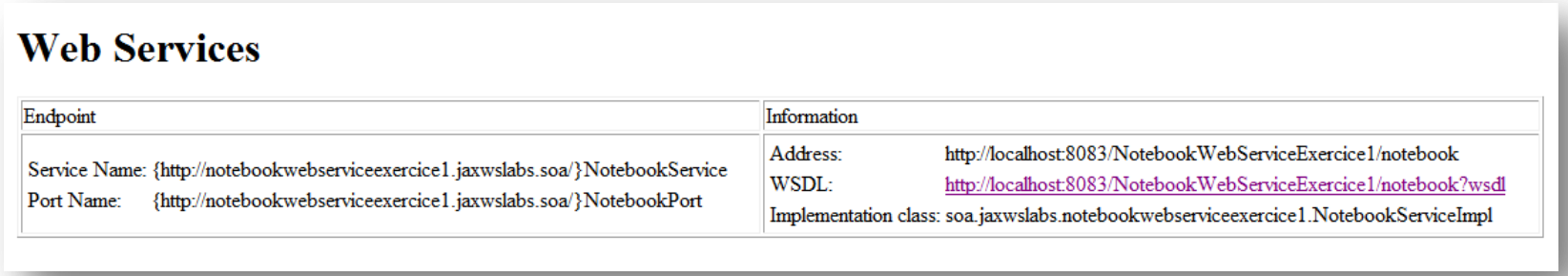
```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime'
```

```
      version='2.0'>
      <endpoint name='NotebookService'
            implementation='soa.jaxwslabs.notebookwebserviceexercice1.NotebookServiceImpl'
            url-pattern='/notebook' />
</endpoints>
```

12. Exécuter l'application. Dans votre navigateur d'accès préféré insérer l'URL suivante :
    http://localhost:8083/NotebookWebServiceExercice1/notebook

    Attention : le port 8083 n'est pas forcément le port utilisé par votre serveur d'application Tomcat. Veuillez insérer celui que vous avez défini lors de l'installation de Tomcat.

    Vous devriez avoir la page web suivante :



    En cliquant sur le lien WSDL, vous pourrez visualisez celui et vérifier la présence des informations définies par vos annotations.
13. Effectuer différents tests sur ce web service à l'aide l'application SOAP-UI.
14. Félicitation vous venez de créer votre premier web service !

Exercice n°2 : Développer un service web suivant une approche descendante « Up/Down »

But : Générer les artifacts d'un Web Service à partir d'une description WSDL, implémenter le Web Service, déployer sous Tomcat, test le Web Service, wsimport.

Description : Dans cet exercice nous développons un Web Service à partir de sa description WSDL. Pour cela, nous nous basons sur la description WSDL du Web Service obtenue à la fin de l'exercice 1. L'intérêt est de montrer que la génération des classes (artifacts) ne donne pas exactement les mêmes classes que celles construites et utilisées dans l'exercice 1.

Étapes à suivre :

1. Créer un nouveau projet web nommé « *NotebookWebServiceExercice2* ».
2. Réduire eclipse
3. Ouvrir l'invite de commande (raccourcis touche windows + R, et saisir cmd puis appuyer sur entrée), aller dans le dossier « src » de votre projet précédemment créé via la commande :
   ```
   cd chemin_du_dossier_src_de_votre_projet
   ```
4. Saisir la commande :
   ```
   wsimport -s . adresse_url_du_fichier_wsdl_de_l_exercice1
   ```
5. Restaurer eclipse, et appuyer sur la touche « F5 » pour rafraichir le projet.
6. Observer les différentes classes générées.
7. Créer la classe « *NotebookServiceImpl* » implémentant l'interface « *NotebookService* » (utiliser la même implémentation que dans l'exercice n°1).
8. Mettre à jour le « *web.xml* », et créer le fichier « *sun-jaxws.xml* » (s'inspirer de l'implémentation de ces fichiers dans l'exercice n°1).
9. Exécuter le projet et observer le fichier descriptif WSDL. Est-il identique à celui de l'exercice n°1 ?
10. Effectuer les mêmes tests que dans l'exercice n°1 via l'application SOAP-UI.

Exercice n°3 : Développer un client de Web Service en mode synchrone

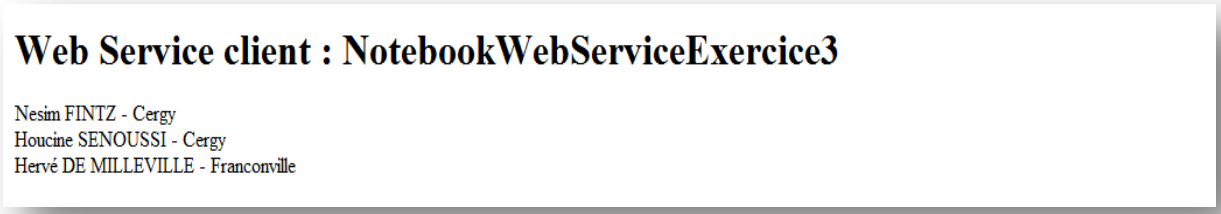
But : Développer un client Web Service, appel synchrone, outil wsimport, client Web via une JSP

Description : Cet exercice consiste à appeler le Web Service défini dans l'exercice n°1. Un client léger via une JSP est utilisé pour appeler l'opération getPersons() définie par le Web Service. Cette opération est invoquée en mode synchrone, c'est-à-dire que le client est en attente de la réponse pour continuer son traitement.

Étapes à suivre :

1. Créer un nouveau projet web nommé « *NotebookWebServiceClientExercice3* ».
2. Effectuer les mêmes manipulations que dans l'exercice n°2 (étapes 3 à 6) pour générer les différentes classes du Web Service de l'exercice n°1.

3. Editer le fichier « *index.jsp* » pour afficher la liste des personnes renvoyées par le web service de l'exercice n°1. S'inspirer du cours (cour4, à partir du slide 31), pour implémenter le traitement java dans la JSP afin de récupérer la liste des personnes.
4. En titre de votre page, vous afficherez « *Web Service client : NotebookWebServiceExercice3* ». En dessous du titre, vous afficherez la liste des personnes de la façon suivante :
   Personne_nom – Personne_adresse
5. Exécuter votre projet et vous devriez voir afficher une page ressemblant à celle-ci :



Exercice n°4 : Développer un client de Web Service en mode asynchrone

But : Développer un client d'un Web Service, appel asynchrone, outil wsimport, client lourd via une interface graphique SWING.

Description : Cet exercice consiste à appeler le Web Service défini dans l'exercice n°1. Un client lourd défini via une interface graphique SWING est utilisé pour invoquer l'opération addPerson(…) en mode asynchrone. L'intérêt de cet exercice est de montrer comment paramétrer la génération des artifacts via wsimport pour le mode asynchrone.

Étape à suivre :

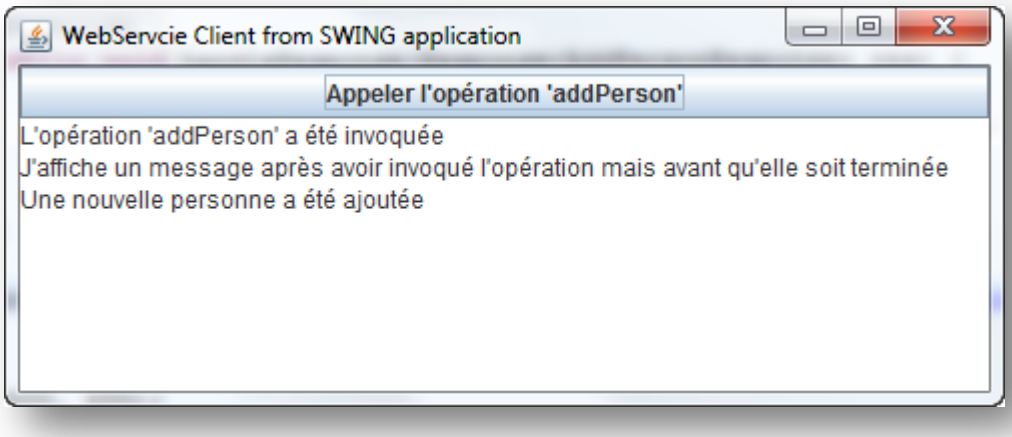
1. Créer un nouveau projet d'application client lourd « *File →New →Project… →Java Project* » et nommé le « *NotebookWebServiceExercice4* ».
2. Créer un package « *soa.jaxwslabs.notebookwebserviceclientexercice4* ».
3. Dans ce package, créer une classe (Main) nommée « *WebServiceClient* ».
4. Dans ce même package, créer un fichier XML nommé « binding.xml », dans lequel vous devez implémenter le code suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<bindings xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns="http://java.sun.com/xml/ns/jaxws">
      <enableAsyncMapping>true</enableAsyncMapping>
</bindings>
```

5. Effectuer les mêmes manipulations que dans l'exercice n°2 (étapes 3 à 6) pour générer les différentes classes du Web Service de l'exercice n°1.
6. Sur l'étape 4, vous devez saisir la commande suivante :

```
wsimport -b
soa\jaxwslabs\notebookwebserviceclientexercice4\binding.xml -s
. adresse_url_du_fichier_wsdl_de_l_exercice1
```

7. Dans la classe « *WebServiceClient* » implémenter l'interface graphique permettant de faire appel à l'opération « *addPerson* » du web service de l'exercice n°1 (cf. cours4, à partir du slide n°37). L'interface devra ressembler à celle-ci :



Exercice n°5 : Développer un Web Service en utilisant l'outillage de JavaSE 6

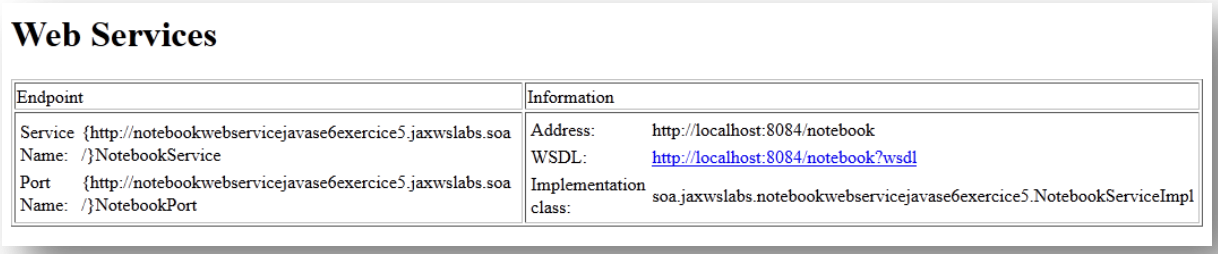
But : Développer un Web Service avec JavaSE 6, EndPoint, méthode publish, serveur intégré.

Description : L'objectif de cet exercice est de développer et publier un Web Service en utilisant les fonctionnalités offertes par JavaSE 6. L'exemple utilisé par l'exercice n°1 est utilisé.

Étapes à suivre :

1. Créer un nouveau projet d'application client lourd et le nommer : « *NotebookWebServiceExercice5* ».
2. Créer un package nommé : « *soa.jaxwslabs.notebookwebservicejavase6exercice5* ».
3. Dans ce package, créer une classe (Main) nommée « *JavaSE6WebService* ».
4. Dans ce même package, copier/coller les classes « *NotebookService* », « *NotebookServiceImpl* » et « Person » du projet de l'exercice n°1. Effectuer les modifications nécessaires au niveau des annotations.
5. Implémenter la classe « JavaSE6WebService », permettant de publier le web service sur le serveur intégré via une interface graphique (cf. cours4, à partir du slide n°54). L'interface graphique ressemblera à celle-ci :

6. Tester la publication, en entrant dans votre navigateur d'accès, l'adresse URL que vous aurez spécifiée précédemment. Vous devriez voir afficher cela :



Web Services

Endpoint	Information
Service Name: {http://notebookwebservicejavase6exercice5.jaxwslabs.soa /}NotebookService Port Name: {http://notebookwebservicejavase6exercice5.jaxwslabs.soa /}NotebookPort	Address: http://localhost:8084/notebook WSDL: http://localhost:8084/notebook?wsdl Implementation class: soa.jaxwslabs.notebookwebservicejavase6exercice5.NotebookServiceImpl

7. Effectuer divers tests sur votre application via l'application SOAP-UI.

Exercice n°6 : Ajouter un intercepteur (handler) à un Web Service

But : Implémentation handler, configuration du Web Service, filtrage par opérations.

Description : Dans cet exercice un intercepteur est ajouté au Web Service décrivant le carnet d'adresses. L'intercepteur a pour fonction de filtrer les messages SOAP de telle sorte que le traitement de l'opération getPersons() ne soit pas réalisé.

Étapes à suivre :

1. Créer un nouveau projet Web nommé « *NotebookWebServiceExercice6* ».
2. Créer un package nommé « *soa.jaxwslabs.notebookwebserviceexercice6* ».
3. Dans ce package, copier/coller les classes « *NotebookService* », « *NotebookServiceImpl* » et « *Person* » à partir du projet de l'exercice n°1.
4. Effectuer les modifications nécessaires sur les annotations.
5. Mettre en place un handler (cf. cours4, à partir du slide n°47). Il devra déterminer à partir de la requête SOAP, si l'opération « *getPersons()* ». Dans ce cas, il devra bloquer le processus et renvoyer une réponse SOAP vide.

Vous trouverez ci-dessous le code source de la méthode permettant d'extraire le nom de l'opération invoqué :

```
/**
 * Cette méthode extrait du message SOAP, le nom de l'opération invoquée
 * @param context Le message SOAP
 * @param isRequest Spécifie si le message est de type requête
 * @return le nom de l'opération sous la forme d'une String
 */
private String getMethodName(SOAPMessageContext context, boolean isRequest) {
	try {
		Field field = context.getClass().getSuperclass().getDeclaredField("packet");
		field.setAccessible(true);
		Packet packet = (Packet) field.get(context);

		if (isRequest) {
			return ((StreamMessage) packet.getMessage()).getPayloadLocalPart();
		}

		return ((JAXBMessage) packet.getMessage()).getPayloadLocalPart();
	} catch (SecurityException e) {
		e.printStackTrace();
	} catch (IllegalArgumentException e) {
		e.printStackTrace();
	} catch (NoSuchFieldException e) {
		e.printStackTrace();
	} catch (IllegalAccessException e) {
		e.printStackTrace();
	}
	return null;
}
```

6. Exécuter le projet
7. Avec l'application SOAP-UI, vérifier que l'opération « *getPersons* » retourne une réponse vide.