



Formation Architecture Logicielle

Exercices

Formateurs :

Frantz Degrigny,
frantz.degrigny@keyconsulting.fr

Méric Garcia, meric.garcia@keyconsulting.fr

Téléphone : 02.30.96.02.75

<http://www.keyconsulting.fr>

Sommaire

4. INTERAGIR AVEC LE SI	3
4.1. EXERCICE PATTERN : FACADE	3
4.2. EXERCICE 1 : REQUÊTE POUR WEBSERVICE SOAP AVEC SOAPUI	4
4.2. EXERCICE 2 : CRÉER UN MOCK DE SERVICE AVEC SOAPUI	4
4.3. EXERCICE 3 : PASSONS AU JAVA - LE SERVEUR	7
4.3.1. <i>Configuration</i>	7
4.3.2. <i>Implémentation</i>	7
4.3.3. <i>Vérification</i>	8
4.4. EXERCICE 4 : PASSONS AU JAVA - CÔTÉ CLIENT	9
4.4.1. <i>Configuration</i>	9
4.4.2. <i>Implémentation</i>	9
4.4.3. <i>Vérification</i>	9
4.4.4. <i>Evolution</i>	10
4.5. CAS D'USAGE : CRÉATION D'UN WEBSERVICE DE PERSISTANCE DES CALCULS.	11
4.5.1. <i>Nouveau Webservice</i>	11
4.5.2. <i>Nouveau client WebService</i>	11
4.6. JMS - PREMIER TESTS.	11
4.6.1. <i>Installation</i>	11
4.6.2. <i>Ecriture et lecture d'un message</i>	11
4.6. JMS - PASSAGE D'UN OBJET.	13
4.6.1. <i>Développement</i>	13
4.6. EXPOSER UN WEBSERVICE DANS UN ANNUAIRE UDDI.	14
4.6.1. <i>Installation</i>	14
4.6.2. <i>Ajout d'un service</i>	14
4.6.3. <i>Enregistrement du service dans l'annuaire</i>	16

Table des illustrations

4. Interagir avec le SI

4.1. Exercice Pattern : FACADE

Cloner l'urt git : <https://github.com/MericGarcia/soa-cours.git>

—> git clone <https://github.com/MericGarcia/soa-cours.git>

Implémenter la classe suivante :

```
public class LastService implements ILastService{

    PersistenceService filePersistence = new FilePersistenceService();
    PersistenceService ormPersistence = new ORMPersistenceService();

    @Override
    public Calcul getLastCalcul() {
        // get the last persistence from the two service
        return null;
    }

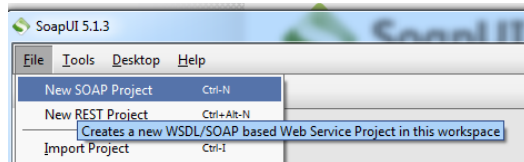
}
```

La méthode doit chercher l'ensemble des prix stocker dans le fichier et dans l'ORM et filtrer le plus récent.

4.2. Exercice 1 : Requête pour Webservice Soap avec SoapUI

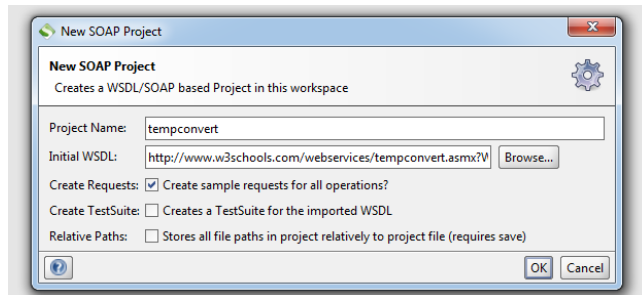
Télécharger et installer SoapUI : <http://www.soapui.org/>

Créer un nouveau projet Soap.

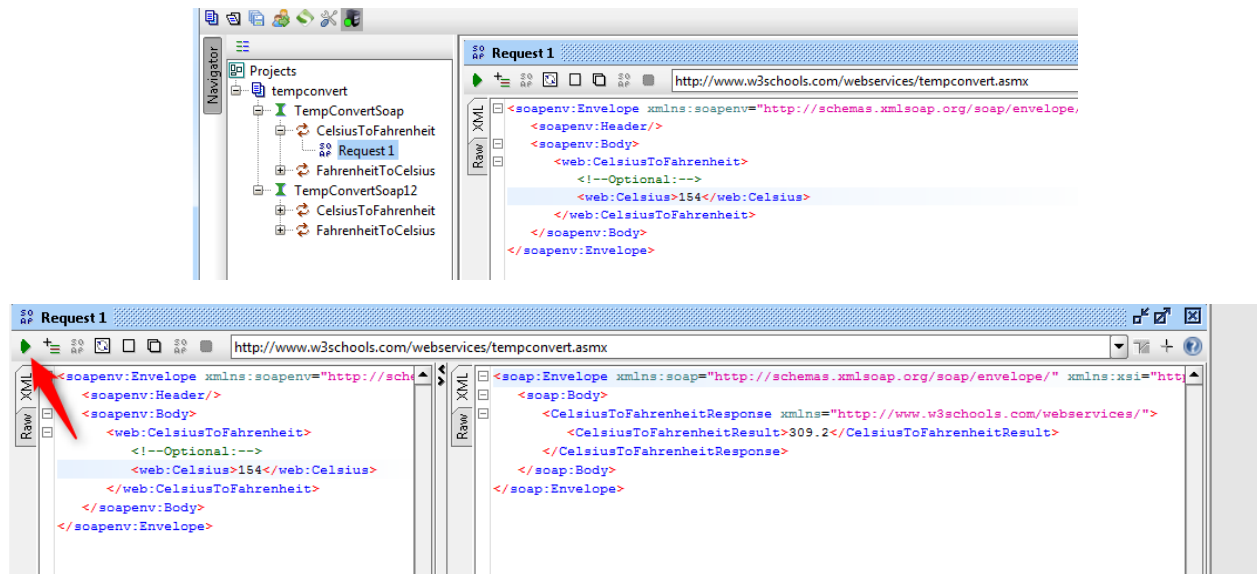


Importer la description d'un Webservice présent online :

<http://www.w3schools.com/webservices/tempconvert.asmx?WSDL>



Effectuer une requête et vérifier le résultat obtenu :



4.2. Exercice 2 : Créer un mock de service avec SoapUI

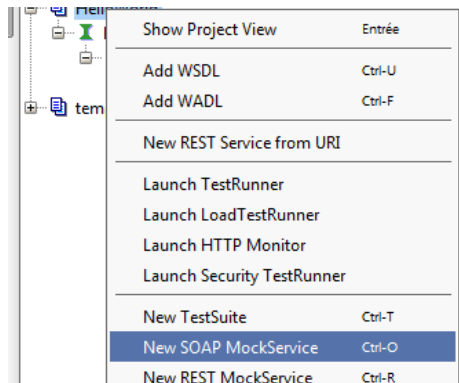
Créer un nouveau projet Soap.

Importer la WSDL : HelloWorld.wsdl présente sur le github dans Tools.

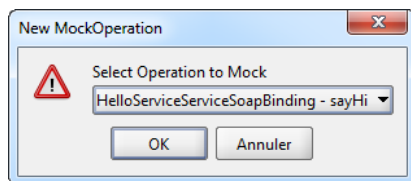
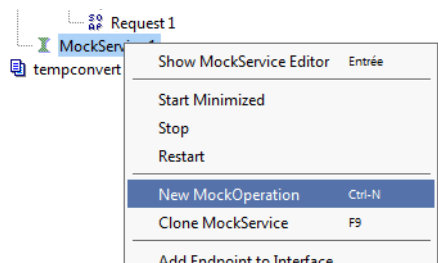
Nous allons créer un bouchon (mock) de ce service.

Pour cela créer un projet SoapUI en important cette WSDL :

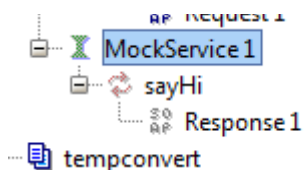
⇒ Créer un nouveau MockService



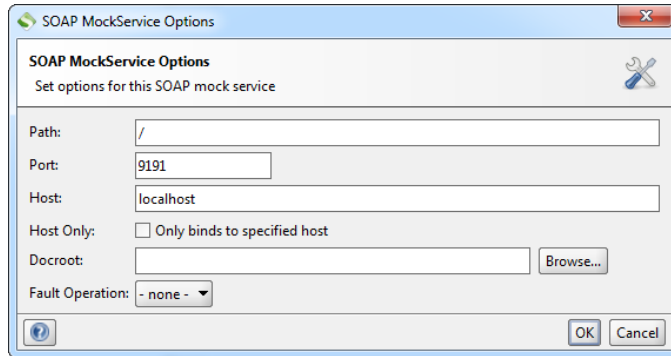
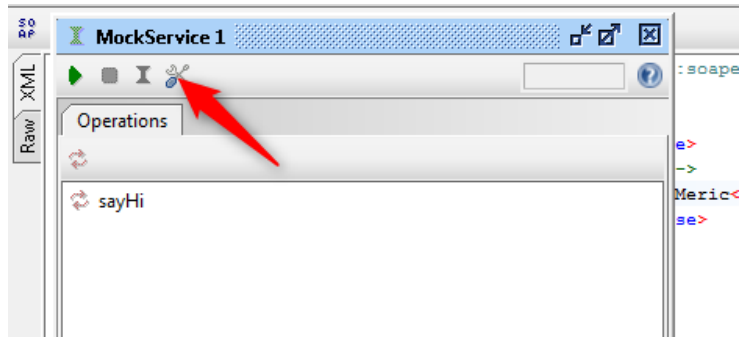
⇒ Ajouter une mock opération :



⇒ Double cliquer sur MockService1



⇒ Modifier les propriétés sur service



⇒ Le lancer (flèche verte).

⇒ Effectuer une requête pour vérifier que le mock est UP : <http://localhost:9191/webservice/services/HelloWorld>

Vérification : effectuer une requête avec le même soapUI sur ce web service.

4.3. Exercice 3 : Passons au Java - le serveur

Nous allons créer un serveur web proposant le service HelloWorld. Pour cela nous utiliserons la librairie apache CXF avec le framework Spring. Dans un premier temps, nous allons implémenter le serveur. Le projet java-ws permet de démarrer un serveur web (tomcat) configuré pour utiliser CXF et spring. Le plugin cargo de maven afin de démarrer un tomcat embarqué dans notre projet.

4.3.1. Configuration

Le fichier *Spring-context.xml* décrit les différents composants gérés par spring.

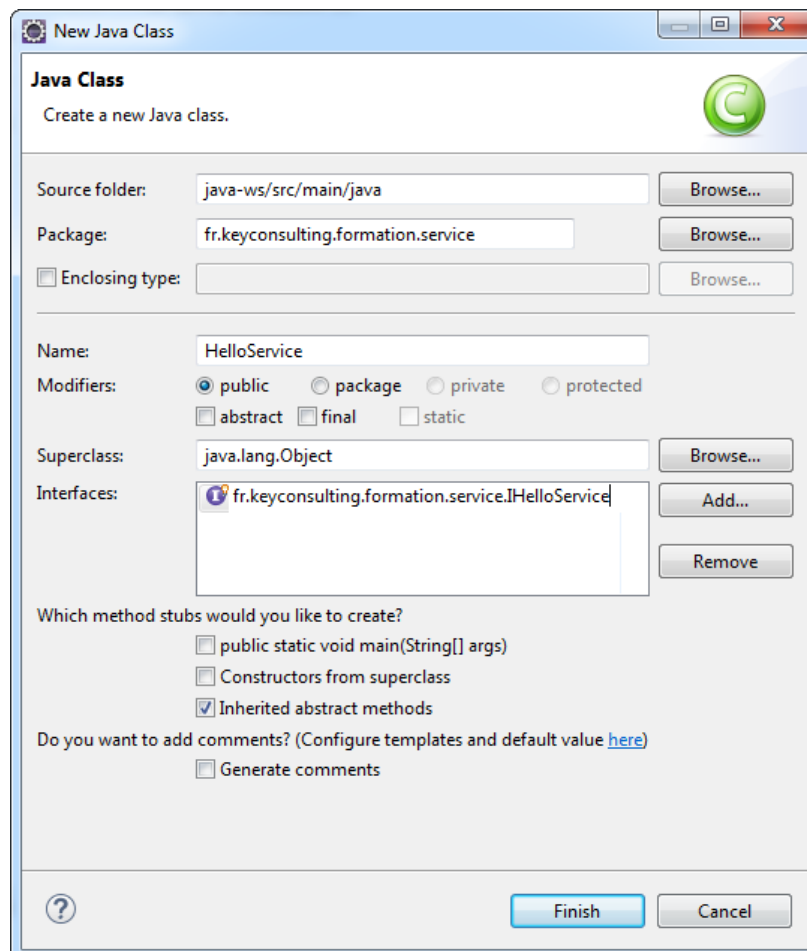
⇒ Ajouter la ligne suivante dans ce fichier :

```
<jaxws:endpoint id="helloWorld"
implementor="fr.keyconsulting.formation.service.HelloService" address="/HelloWorld"/>
```

4.3.2. Implémentation

Nous allons donc devoir créer l'implémentation du service : HelloService.

⇒ Implémenter l'interface IHelloService disponible dans le projet java-ws-interface.



Le WebService condient une méthode sayHi qui doit renvoyer « Hello » + l'argument reçu + « ! ».

⇒ Ecrire l'implémentation de la classe HelloService.

```
package fr.keyconsulting.formation.service;

public class HelloService implements IHelloService {

    @Override
    public String sayHi(String name) {
        //implémenter la méthode
    }

}
```

4.3.3. Vérification :

Lancer le serveur. Pour cela compiler le projet :

⇒ **mvn clean install**

puis, se placer dans le projet java-ws, et lancer via le plugin cargo :

⇒ **cd java-ws**

⇒ **mvn cargo:run**

Une fois le serveur démarré, la liste des services qu'il propose est disponible ici :

<http://localhost:9090/webservice/services>

Vérifier le bon fonctionnement du WebService avec SoapUI en important la WSDL générée à l'url : <http://localhost:9090/webservice/services/HelloWorld?wsdl> .

4.4. Exercice 4 : Passons au Java - côté client

Le projet `java-calculatrice-client-ws` est préconfiguré pour pouvoir créer un client `WebService`. Le fichier de configuration de `Spring-ws-client.xml` décrit les clients `WebService` qui seront rendu disponible grâce à `Spring`.

4.4.1. Configuration

⇒ Ajouter la ligne suivante au fichier `Spring-ws-client.xml` :

```
<jaxws:client id="helloClient" serviceClass="fr.keyconsulting.formation.service.IHelloService"
address="http://localhost:9090/webService/services/HelloWorld" />
```

4.4.2. Implémentation

Dans la classe `Controller` nous allons alors pouvoir récupérer le service.

⇒ Ajouter cette ligne au début de l'initialisation (méthode `initialize`):

```
service = (IHelloService) context.getBean("helloClient");
```

⇒ Et l'utiliser pour renseigner le texte en haut de la vue de la calculatrice (fin de la méthode `run`) :

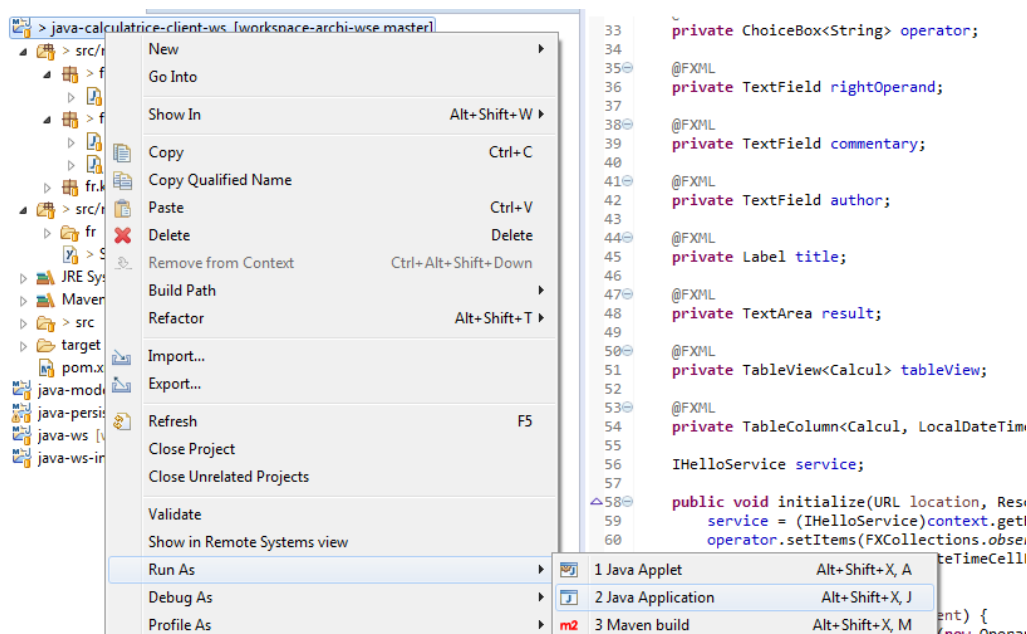
```
title.setText(service.sayHi("inconnu"));
```

4.4.3. Vérification

Relancer le webservice (serveur) s'il a été arrêté :

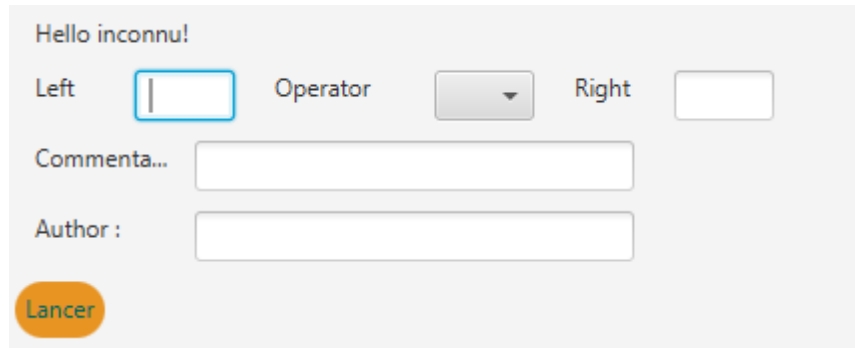
⇒ `mvn cargo:run` dans le répertoire `java-ws`.

Lancer l'application et vérifier le fonctionnement.



Si une classe main est demandée, elle se trouve ici : `fr.keyconsulting.formation.Main`

Résultat :

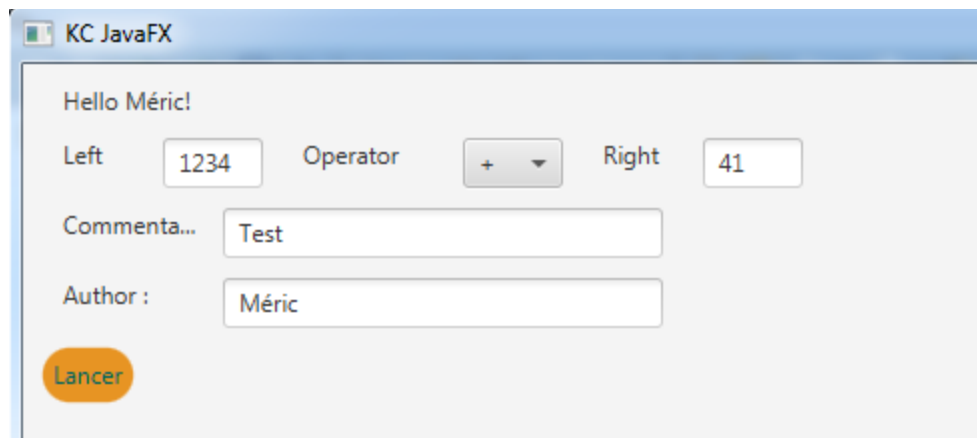


The screenshot shows a JavaFX application window with a light gray background. At the top, it says "Hello inconnu!". Below this, there are three input fields: "Left" (containing a vertical bar), "Operator" (a dropdown menu), and "Right" (empty). Below these are two more text input fields labeled "Commenta..." and "Author :". At the bottom left, there is an orange button labeled "Lancer".

4.4.4. Evolution

Modifier la classe Controller afin de changer le message « title » lorsque un author est renseigné, il doit alors afficher l'auteur utilisé lors du dernier calcul effectué. Puis recompiler et relancer.

Résultat :



The screenshot shows the same JavaFX application window, but now the title bar says "KC JavaFX". The "Hello" message has changed to "Hello Méric!". The "Left" field now contains "1234", the "Operator" dropdown shows a "+" sign, and the "Right" field contains "41". The "Commenta..." field now contains "Test" and the "Author :" field contains "Méric". The orange "Lancer" button is still present at the bottom left.

4.5. Cas d'usage : Création d'un Webservice de persistance des calculs.

4.5.1. Nouveau Webservice

Créer un Webservice CalculService dans le projet java-ws qui aura pour charge d'alimenter une base de données avec les calculs effectués.

Ajouter une méthode permettant la récupération de tous les calculs effectués. L'interface ICalculService présente dans java-we-interface définit le contrat du Webservice.

Utiliser l'implémentation ORMPersistenceService pour effectuer les enregistrement et les lectures.

Lancer des requêtes sur ce Webservice via SoapUi afin de vérifier son fonctionnement.

4.5.2. Nouveau client Webservice

Ajouter un client à ce Webservice dans le projet java-calculatrice-ws-client afin de pouvoir stocker les calculs effectués et d'afficher dans l'application javaFx l'ensemble des calculs au démarrage.

4.6. JMS - premier tests.

4.6.1. Installation

Récupérer apache ActiveMQ :

<http://activemq.apache.org/activemq-5120-release.html>

Décompresser l'archive.

Démarrer le service : en ligne de commande à la racine du dossier :

⇒ `./bin/activemq qrtart`

Vérifier que le service est bien démarré : <http://127.0.0.1:8161/admin/>

Login : admin, Mdp : admin

4.6.2. Ecriture et lecture d'un message :

La classe JmsServiceHelper permet d'accès aux méthode d'enregistrement et de lecture des files JMS.

public void send(String text) : permet d'envoyer un message sous forme de texte dans la première file

public void sendForListener(String text) : permet d'envoyer un message sous forme de texte dans la file sur laquelle écoute un listener

public String next() : récupère le message suivant disponible dans la première file.

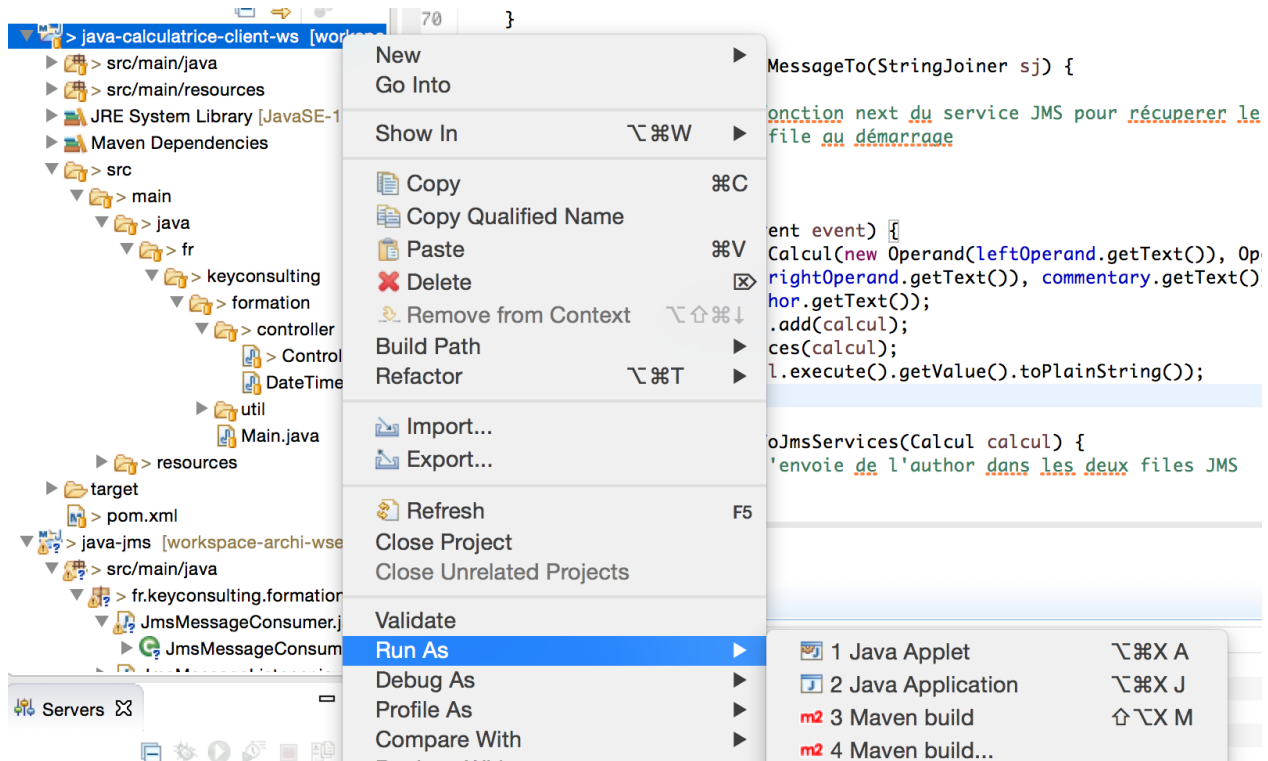
Les deux fonctions à modifier se trouvent dans le contrôleur (classe Controller) :

private void addEnqueuedMessageTo(StringJoiner sj)

et

private void sendAuthorToJmsServices(Calcul calcul)

Une fois les modifications effectuées, lancer le programme :



Si le développement est correct, le résultat sera le suivant après trois calculs effectués :

[Home](#) | [Queues](#) | [Topics](#) | [Subscribers](#) | [Connections](#) | [Network](#) | [Scheduled](#) | [Send](#)

Queue Name

Queues

Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
ListenQueue	0	0	3	3	Browse Active Consumers Active Producers atom rss	Send To Purge Delete
Queue	3	0	3	0	Browse Active Consumers Active Producers atom rss	Send To Purge Delete

4.6. JMS - passage d'un objet.

4.6.1. Développement

Des files ont été ajoutées dans la configuration afin de pouvoir gérer les calculs.

Il faut désormais modifier trois classes afin d'utiliser ses files :

`JmsMessageSender` : méthode `send(Calcul calcul)`

`JmsMessageConsumer` : méthode `getFollowingCalcul()`

`Controller` : méthode `sendCalculAndAuthorToJmsServices(Calcul calcul)` et `addEnQueuedCalculToList(List<Calcul> calculs)`

Les lignes à modifier sont indiquées par des commentaires.

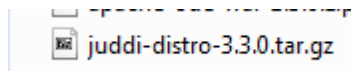
Une fois les modifications effectuées, lancer l'application comme dans l'exercice précédent et observer l'évolution des files JMS.

La liste des calculs de la session précédente doit se recharger à chaque redémarrage de l'application (la file JMS devient un moyen de persister temporairement les calculs).

4.6. Exposer un webservice dans un annuaire UDDI.

4.6.1. Installation

Décompresser le dossier juddi-distro-3.3.0.tar.gz.



Copier votre JDK 7 dans juddi-distro-3.3.0\juddi-tomcat-3.3.0 avec comme nom : jdk7-win sous windows et jdk7-lin sous linux.

Ajouter le fichier setenv.sh ou setenv.bat disponible dans le github.

Exécuter la commande bin/setenv.sh ou bin\setenv.bat.

Démarrer en ligne de commande : bin\startup.bat ou bin/startup.sh.

L'annuaire de service est lancé ! S'y connecter : <http://localhost:8080/juddi-gui/home.jsp>.

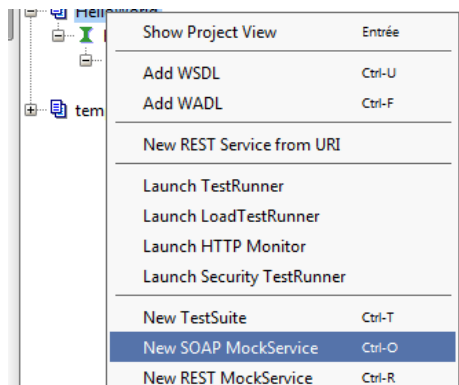
4.6.2. Ajout d'un service

Nous allons enregistrer un service généré par un mock SoapUI dans l'annuaire.

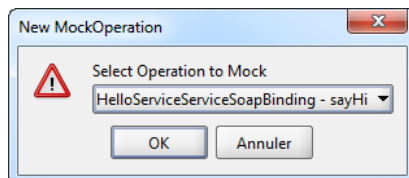
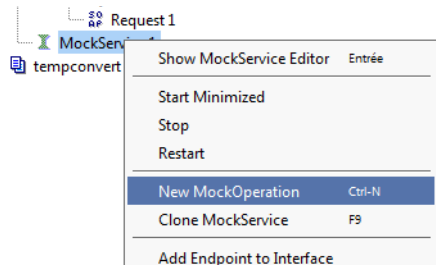
Démarrer un mock SoapUI généré à partir du WSDL HelloWorld.wsdl.

Pour cela créer un projet SoapUI en important cette WSDL :

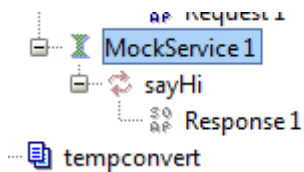
⇒ Créer un nouveau MockService



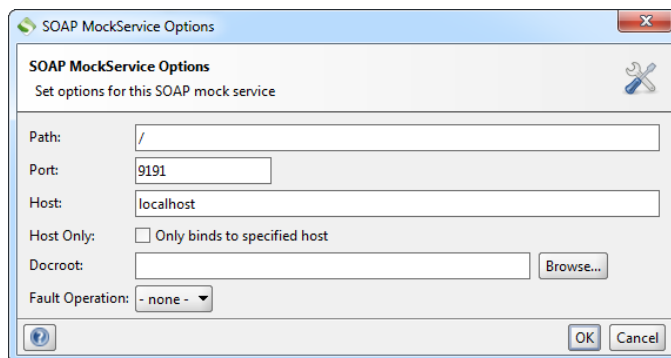
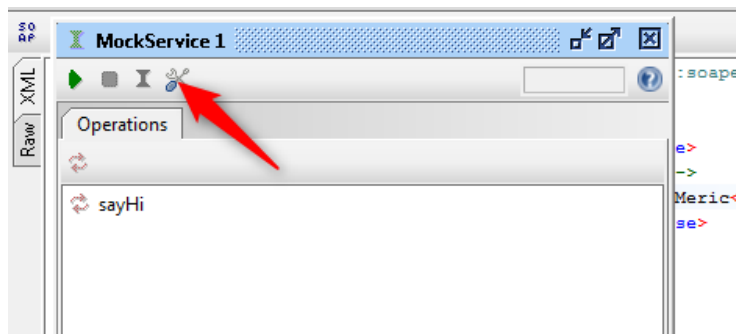
⇒ Ajouter une mock opération :



⇒ Double cliquer sur MockService1



⇒ Modifier les propriétés sur le service



⇒ Le lancer (flèche verte).

⇒ Effectuer une requête pour vérifier que le mock est UP : `http://localhost:9191/webbservice/services/HelloWorld`

Nous allons pouvoir enregistrer ce webservice dans l'annuaire. Sa WSDL est exposée ici : <http://localhost:9191/webService/services/HelloWorld?wsdl>

4.6.3. Enregistrement du service dans l'annuaire

L'interface d'admin se trouve ici : <http://localhost:8080/juddi-gui/home.jsp>.

Se logger en tant qu'admin (admin - admin).

Aller dans create -> Register Services from WSDL.

Enregistrer la wsdl.

Rechercher la dans discover -> tModels.

Quelles informations peut on récupérer ?

Quel est l'intérêt de stocker les services dans un annuaire ?