Sujet de TP 1
Framework JEE
Session Beans

Chouki Tibermacine - chouki.tibermacine@umontpellier.fr

Objectifs du TP

- Écrire un composant EJB session
- Écrire un composant Web et le connecter au composant EJB
- Déployer les composants dans le serveur d'application WildFly de Redhat (jBoss)
- Écrire une application "client-lourd" du composant EJB et l'exécuter en dehors de la JVM du serveur

Exercice 1

Ci-dessous, l'énoncé de l'exercice, suivi du détail de la démarche à suivre.

- Écrire un module EJB session sans état qui permet de faire la conversion d'un montant en euros dans d'autres monnaies (Dollars américains/canadiens, Yen, ...),
- Écrire une page HTML qui contient un formulaire de saisie du montant et du choix de la monnaie cible
- Écrire une page JSP qui récupère les informations saisies dans le formulaire HTML et qui utilise le bean session pour effectuer la conversion de monnaie selon les taux de change en cours, disponibles dans le document XML à cet URL de la banque centrale européenne : `https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml`
- Déployer toute l'application dans le serveur d'application libre WildFly
- Tester l'application sur votre navigateur Web.

Installations nécessaires.

Télécharger l'archive ZIP de la dernière version du serveur d'application WildFly disponible en ligne : `http://wildfly.org/downloads/`

La désarchiver pour obtenir un répertoire que nous nommerons `WildFlyHome` dans la suite du sujet.

Vérifier sur votre navigateur s'il y a un serveur Web démarré sur le port 8080 (`http://localhost:`

8080). Si c'est le cas, éditer le fichier standalone.xml qui se trouve dans le dossier suivant :
`WildFlyHome/standlone/configuration/`
pour remplacer "`jboss.http.port :8080`" par "`jboss.http.port :8081`".

Au lieu de télécharger WildFly, on peut également réaliser le TP en utilisant Maven ou Gradle et en intégrant dans le projet le plugin WildFly, qui vient avec des goals/tasks qui permettent de démarrer un serveur d'application (téléchargé automatiquement pour nous), déployer des apps dedans, ... :
`https://mvnrepository.com/artifact/org.wildfly.plugins/wildfly-maven-plugin`

Télécharger et installer IntelliJ Idea version Ultimate. Avec votre compte @etu.umontpellier.fr vous avez la possibilité de créer un compte sur le site de Jetbrains et ensuite télécharger une version ultimate permanente (pas la version d'essai d'un mois). Dans la version ultimate, on retrouve les plugins nécessaires pour réaliser les TP.

Une fois l'environnement installé, voici la procédure à suivre pour réaliser le TP.

- Lancer IntelliJ IDEA
- Créer un nouveau projet Java Enterprise :
  - cliquer sur File > New > Project > Java Enterprise
  - choisir la « Full Platforme »
  - donner un nom à votre projet (Converter) et laisser le dossier par défaut
  - donner un group id (fr.m2.aigle) et artifact id (Converter), laisser le numéro de version par défaut
  - cliquer sur Finish
  - si vous n'avez pas intégré dans votre projet le plugin Maven/Gradle de WildFly, choisir le serveur d'application où vont être déployés vos composants : lors du premier démarrage, ajouter un serveur d'application (Menu Run > Edit Configurations...)
    - cliquer sur +, puis choisir « JBoss Server > local », puis indiquer l'emplacement WildFly-Home/
    - donner un nom à cette config
    - dans l'onglet Deployment, ajouter votre projet, puis cliquer sur OK
- Implémenter le bean session :
  - dans les réglages du projet (cliquer avec le bouton droit sur le projet, puis « open Module Settings », ajouter un nouveau module en cliquant sur « + » puis EJB
  - afficher la vue EJB : menu View > Tool Window > EJB
  - dans cette nouvelle vue, clique bouton droit sur votre projet puis > New > Create Stateless Session Bean...
  - donner un nom au module EJB (ConverterEjb)
  - choisir un nom pour le package (converter)
  - Créer une interface avec le nom : `converter.IConverter`
  - ajouter la déclaration de la méthode suivante dans l'interface IConverter :
    `public double euroToOtherCurrency(double amount, String currencyCode);`
- Éditer la classe du bean :

— ajouter l'annotation @Stateless à la classe, si ce n'est pas déjà fait
— ajouter la méthode de l'interface puis écrire son implémentation
— dans cette implémentation, utiliser l'URL suivant (qui a été donné dans la page précédente) pour rechercher le taux de change courant :
`https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml`
— pour analyser ce document XML, utiliser le parser jDOM 2 qu'il faudra intégrer dans votre projet. Ajouter la déclaration de dépendance suivante dans le script de build de votre projet (pom.xml) :

```
<dependency>
 <groupId>org.jdom</groupId>
 <artifactId>jdom2</artifactId>
 <version>2.0.6</version>
</dependency>
```

— Voici un extrait de code Java pour commencer l'analyse du fichier (ajouter les import nécessaires depuis les packages javax.net et org.jdom2, et les handlers d'exceptions qu'il faut) :

```
SAXBuilder sxb = new SAXBuilder();
URL url = new URL(
  "https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml");
HttpsURLConnection con = (HttpsURLConnection)url.openConnection();
Document document = sxb.build(con.getInputStream());
Element racine = document.getRootElement();
Namespace ns = Namespace.getNamespace(
  "http://www.ecb.int/vocabulary/2002-08-01/eurofxref");
Element elem = racine.getChild("Cube", ns);
```

Noter ici l'utilisation d'un espace de nom (ns) pour la navigation dans les éléments du document XML. Noter également la présence de plusieurs niveaux d'éléments « Cube ». Compléter le code ci-dessus pour naviguer jusqu'aux monnaies et leurs taux de change
— la documentation complète de l'API jDOM 2 est disponible ici :
`http://www.jdom.org/docs/apidocs/`

▶ Implémenter le composant Web :
— cliquer avec le bouton droit de la souris sur le dossier webapp/
— choisir New > JSP/JSPX
— donner un nom au fichier JSP (index.jsp), puis valider
— un fichier index.jsp est généré et pré-rempli : i) d'une directive JSP indiquant entre autres l'encodage du fichier, et ii) de code HTML de base
— éditer ce fichier en créant un formulaire où l'on peut saisir un montant à convertir et une monnaie cible
— on demande l'exécution du même script JSP (action="index.jsp") dans le formulaire, lorsque ce dernier est soumis au serveur
Du coup, il faudrait prévoir un test (if) qui englobe la production du formulaire HTML par le script :

```
if(request.getParameter("convert") == null){ ... }
```

Je suppose ici que vous avez un champ (hidden, par ex.) qui a un name="convert"

— ajouter dans cette page JSP le code qui permet de récupérer les données saisies par l'utilisateur et qui fait appel au composant EJB pour convertir la montant saisi (code à mettre dans le else du if précédent) :
Mais d'abord, ajouter quelque part dans le body ce qui suit :

```
<jsp:useBean class="converter.ConverterBean" id="beanConv" />
```

Ceci indique au serveur qu'il faut réaliser une injection de dépendance, qui permet de connecter le composant EJB au composant Web, lors du déploiement (nous allons désormais pouvoir référencer le composant EJB ayant une classe converter.ConverterBean, sous le nom beanConv)
Ajouter ensuite :

```
<%@page import="java.util.*" %>
```

Ceci sert à importer toutes les classes du package java.util
Ajouter dans cette directive, après le '*', les autres imports quand c'est nécessaire, en les séparant par des virgules.
Ex :

```
<%@page import="java.util.*,javax.jms.*,javax.naming.*,java.math.*" %>
```

<% Tout ce qui est placé ici c'est du code Java classique (jusqu'à "%>")

```
double amount = Double.parseDouble(request.getParameter("amount"));
```

Cette instruction permet de récupérer la valeur saisie dans l'élément du formulaire dont le nom est amount (c'est équivalent à $_POST['amount'] ou $_GET['amount'] de PHP)
Ajouter le code nécessaire pour récupérer le code de la monnaie cible

```
amount = beanConv.euroToOtherCurrency(amount,currency);
```

Ceci permet d'invoquer la méthode euroToOtherCurrency du bean

```
out.println("<h4>Le montant converti est : </h4>"+amount);
```

Ceci permet de générer le code HTML placé en paramètre (équivalent à `echo '<h4>...';` de PHP) ... %>

- Votre application est à présent prête à être déployée et exécutée
- Déployer et exécuter votre application : menu Run
- Pour dé-déployer votre application :
  — choisir la tool-window Services dans la partie inférieure de l'IDE
  — aller dans : WildFly Server
  — cliquer avec le bouton droit sur l'icône de votre application (Converter), puis choisir Stop (pour dé-déployer) ou Delete... (pour supprimer l'application de la configuration de (futurs) déploiements automatiques)
- Pour effacer les messages affichés dans les consoles de sortie de l'IDE :
  — cliquer au centre de la console avec le bouton droit de la souris, puis choisir Clear All

- A la fin de la séance de TP, ne pas oublier de :
  — dé-déployer votre application
  — arrêter le serveur d'application :
    — choisir la tool-window Services dans la partie inférieur d'IntelliJ
    — cliquer avec le bouton droit sur WildFly Server ..., puis choisir Stop
  — fermer IntelliJ

Exercice 2.

A la place du composant Web de l'application (considéré comme le client du composant EJB), nous allons écrire une application cliente Java indépendante (classe avec une méthode main qui va utiliser le composant EJB et qui va être déployée dans une JVM indépendante, pas celle utilisée par le serveur d'application) :

- Nous allons prévoir dans la classe du bean l'annotation suivante : `@Remote`
  Ceci indique au serveur d'application, lors du déploiement, que le bean session peut être utilisé par des clients distants.
- Déployer l'application Java EE dans le serveur d'application WildFly
- Nous allons désormais travailler soit en dehors de l'IDE (ne pas le fermer) ou bien dans un projet Java (standard) qu'il faudra créé dans votre IDE.
- Dans la suite, on explique ce qu'il faut faire dans le premier cas. On va utiliser un Terminal et un éditeur de texte (Emacs ;-) par exemple)
- Copier l'interface du bean (IConverter.class) dans un répertoire de votre choix (nommé : client/). Par contre, veiller à mettre ce fichier dans la même hiérarchie de répertoires que celle correspondant aux packages de l'interface, et ses éventuelles sous-packages (client/converter/IConverter.class par exemple, si l'interface est déclaré dans un package qui s'appelle converter)
- Écrire une classe Java (Client.java) avec une méthode main dans le même répertoire (client/)
- Copier l'archive WildFlyHome/bin/client/jboss-client.jar dans un sous-répertoire lib de votre répertoire client, et l'ajouter dans le Classpath avant de compiler et exécuter votre programme client
- Pour ceux qui utilisent Maven, copier ce JAR plutôt dans le dossier `src/main/resources/` puis ajouter dans le `pom.xml` la dépendance suivante :

```
<dependency>
 <groupId>jboss</groupId>
 <artifactId>jboss-client</artifactId>
 <version>1.0</version>
 <scope>system</scope>
 <systemPath>${project.basedir}/src/main/resources/jboss-client.jar</systemPath>
</dependency>
```

  Dans la tool-window Maven, appuyer sur le bouton d'actualisation pour faire le *build* du projet.
- La méthode main de la classe Client doit effectuer un lookup du bean en utilisant les instructions suivantes : (Ne pas oublier d'ajouter les imports nécessaires : javax.naming.InitialContext, ...)

```
IConverter converter = (IConverter) InitialContext.doLookup(
  "ejb:/Converter/ConverterEJB/ConverterBean!converter.IConverter");
```

La chaîne de caractères passée en argument à la méthode lookup ci-dessus est le nom JNDI donné automatiquement par le serveur d'application au bean lorsqu'il a été déployé (on peut le voir dans la console lors du déploiement de l'application Converter). Ce nom a été utilisé lors du déploiement pour enregistrer le bean auprès du serveur de noms JNDI.

- Pour que le bean soit retrouvé en dehors du serveur d'application, il faudra éditer quelques propriétés de configuration : créer un fichier texte, le nommer jndi.properties et le placer dans le même répertoire que l'emplacement à partir duquel vous allez lancer l'application Java (là où vous allez écrire la commande java : dans le répertoire client/ par exemple)
- Mettre dans ce fichier les 2 lignes suivantes

```
java.naming.factory.initial=org.wildfly.naming.client.WildFlyInitialContextFactory
java.naming.provider.url=http-remoting://<ip-ou-host>:8081
```

Ne pas oublier de remplacer `<ip-or-host>` par localhost. Dans l'exercice suivant, on va y mettre l'adresse IP de la machine de l'un de vos camarades, sur laquelle se trouve le serveur d'application dans lequel le composant EJB est déployé

- Ajouter à la méthode main le code qu'il faut pour demander à l'utilisateur un montant à convertir et le code de la monnaie cible (utiliser simplement la classe Scanner), puis afficher le résultat retourné par la méthode du bean
- Compiler la classe, puis l'exécuter :

```
export CLASSPATH=.:jndi.properties:./lib/jboss-client.jar
java Client
```

Le jar ajouté dans le CLASSPATH ci-dessus comporte les classes nécessaires que la JVM instancie pour accéder au serveur de noms JNDI (et son contexte initial indispensable au lookup du bean).
Les utilisateurs de Windows doivent remplacer `export` par `set`, les `:` par `;`
Ceux qui utilisent Maven n'ont pas besoin de taper les commandes ci-dessus. L'initialisation du CLASSPATH est faite automatiquement avec les dépendances dans `pom.xml`. Il y a juste à faire un *Run*.
Si jamais, la JVM vous indique qu'elle n'arrive pas à instancier la classe InitialContext (ce qui veut dire que le fichier jndi.properties n'a pas été lu correctement), remplacer la ligne de code `InitialContext.doLookup...` dans la méthode `main(...)` par :

```
Properties props = new Properties(); // Importer java.util.Properties
props.put("java.naming.factory.initial",
  "org.wildfly.naming.client.WildFlyInitialContextFactory");
props.put("java.naming.provider.url",
  "http-remoting://localhost:8081");
// Dans l'exercice 3, remplacer ci-dessous localhost par l'adresse IP
IConverter converter = (IConverter) new InitialContext(props).lookup(
  "ejb:/Converter-1.0-SNAPSHOT/ConverterEJB!converter.IConverter");
```

Exercice 3.

Les machines des salles de TP sont sur le même réseau. On va déployer le composant EJB sur une machine et lancer l'exécution de l'application Java cliente de cet EJB sur une autre machine.

- Demander à un voisin l'adresse IP de sa machine. Vous pouvez aussi ouvrir une session sur une machine voisine
- Éditer le fichier jndi.properties pour y mettre l'adresse IP à la place de localhost
- S'assurer que le nom JNDI du composant EJB recherché (passé en argument à la méthode lookup dans l'application client Java) correspond bien à celui qui a été donné par le serveur d'application là où le composant EJB a été déployé
- Si vous n'avez rien modifié dans le code Java de l'application cliente, pas besoin de re-compiler (seul le fichier jndi.properties, lu par la JVM à l'exécution, aura été modifié)
- Lancer l'exécution de votre programme comme précédemment (en supposant que le CLASSPATH comporte les mêmes informations que dans l'exercice précédent) :
  `java Client`