Université de Montpellier
Faculté des Sciences
Département Informatique

Filière : Master Informatique Concepteur **:** Chouki TIBERMACINE

TP EJB 1

Objectifs du TD

- Écrire un composant EJB session sous l'IDE Eclipse
- 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 composant 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 de JBoss/Redhat
- Tester l'application sur votre navigateur Web.

Installations nécessaires.

1. Sur les postes de la salle de TP à la FDS :

Télécharger l'archive ZIP du serveur d'application WildFly disponible sur Moodle.

Démarrer une session sous Linux et aller au répertoire où vous avez placé l'archive.

Désarchiver l'archive pour obtenir un répertoire que je vais appeler WildFly_Home dans la suite du sujet.

Éditer le fichier standalone.xml qui se trouve dans le dossier :
WildFly_Home/standlone/configuration/

Remplacer le port HTTP 8080 (déjà utilisé par un serveur Tomcat lancé par défaut sur le poste) par le port 8081

Télécharger eclipse-installer et lancer l'installation d'« *Eclipse IDE for Java EE Developers* » comme indiqué ici :
https://moodle.umontpellier.fr/mod/page/view.php?id=131074

2. Sur vos machines personnelles :
Les trois outils suivants doivent être installés sur votre machine personnelle si vous voulez réaliser le TP chez vous :

- le JDK (1.8 minimum) :
  http://www.oracle.com/technetwork/java/javase/downloads/index.html
- l'IDE Eclipse Java EE :
  https://www.eclipse.org/downloads/packages/release/2019-09/r/eclipse-ide-enterprise-java-developers
- le serveur d'application Java EE WildFly copié et désarchivé précédemment, téléchargeable ici : http://wildfly.org/downloads/

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

- Lancer eclipse (ajouter un alias eclipse-jee à votre .bashrc par exemple)

- Créer un nouveau projet Java EE :
  - cliquer sur File -> New -> Enterprise Application Project
  - donner un nom à votre projet (Converter), choisir un répertoire pour votre projet (répertoire dans votre workspace Eclipse où vos sources et le bytecode vont être placés)
  - choisir le serveur d'application où vont être déployés vos composants : lors du premier démarrage, ajouter un serveur d'application (*Target Runtime*)
    - cliquer sur New...
    - choisir « *JBoss Community* », puis « *WildFly xx* », et ensuite cliquer sur Next
    - donner lui un nom, qui va être affiché dans la liste des cibles potentielles de déploiement
    - choisir le chemin d'installation du serveur d'application (cliquer sur Browse, puis aller vers le répertoire où vous avez désarchivé WildFly)
    - laisser les paramètres par défaut, puis cliquer sur Finish.
    - choisir un répertoire de base pour déployer votre application : aller vers le répertoire standalone (c'est le type de serveur WildFly qu'on va utiliser) et sélectionner le fichier standalone.xml, que vous avez édité précédemment, comme « *configuration file* »
  - cliquer sur Finish
    Eclipse va créer un répertoire pour l'application d'entreprise/principale (Converter) avec plusieurs sous-répertoires de configuration
  - créer deux projets correspondants à deux composants (un composant EJB ConverterEJB et un composant Web ConverterWeb). Associer à chaque fois chacun des projets au projet Converter (cocher la case « Add projet to an EAR »).
    Laisser les paramètres par défaut, sauf lors de l'étape de configuration du composant EJB, décocher la case « Create an EJB Client JAR ... ».
    Eclipse créera pour chacun des deux composants un répertoire comportant plusieurs sous-répertoires contenant les sources et les fichiers de configuration.

- Implémenter le bean session :
  - cliquer avec le bouton droit de la souris sur l'icône du composant EJB dans le menu à gauche (ConverterEJB)
  - choisir New -> Session Bean...
  - donner des noms au package (converter) et à la classe du bean (ConverterBean)
  - s'assurer que le type de bean session est bien Stateless
  - dans la rubrique « Create Business Interface », cocher « Remote » et décocher

« No-Interface View ». Donner à cette interface le nom : converter.IConverter
Laisser les paramètres par défaut et Eclipse va créer une interface Java (IConverter) et une classe (ConverterBean)

- ajouter la déclaration de la méthode suivante dans l'interface IConverter :

```
public double euroToOtherCurrency(double amount, String currencyCode);
```

- Éditer la classe du bean :
  - 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 télécharger depuis le site suivant (voir l'extrait de code donné plus loin pour commencer à utiliser jDOM) :
    http://www.jdom.org/downloads/

  - décompresser l'archive ZIP téléchargée dans le répertoire de votre choix
  - créer un répertoire lib dans le projet de l'application Enterprise (Converter) : cliquer avec le bouton droit de la souris sur le répertoire EarContent de votre projet Converter, et choisir New > Folder. Donner un nom au répertoire : lib
    Seul le contenu de ce répertoire (dans lequel on met la plupart du temps des bibliothèques) est par défaut déployé dans le serveur
  - importer le fichier jdom-xxx.jar dans le répertoire lib créé plus haut
  - ajouter le JAR dans le CLASSPATH du composant EJB :
    cliquer sur le bouton droit de la souris sur le composant EJB, choisir « Build Path » et ensuite « Configure Build Path... ». Dans l'onglet « Libraries », cliquer sur « Add JARs... » et indiquer le chemin jusqu'à « jdom-xxx.jar »
  - Voici un extrait de code Java pour commencer (ajouter les import nécessaires depuis les packages javax.net et org.jdom2, et les try-catch 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 composant Web (ConverterWeb)
  - choisir New -> JSP File
  - donner un nom au fichier JSP (index.jsp), puis cliquer sur Finish
  - 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 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 prête à être déployée et exécutée

- Déployer et exécuter votre application : cliquer avec le bouton droit sur l'icône Converter, puis Run As > Run on Server

- Pour dé-déployer votre application :
  - choisir la vue Servers dans la partie inférieure d'Eclipse
  - aller dans : WildFly xxx Runtime Server
  - cliquer avec le bouton droit sur l'icône de votre application (Converter), puis choisir Stop (pour dé-déployer) ou Remove (pour supprimer l'application de la configuration de (futurs) déploiements automatiques)

- Pour effacer les messages affichés dans les consoles de sortie d'Eclipse :
  - cliquer au centre de la console avec le bouton droit de la souris, puis choisir Clear

- A la fin de la séance de TP, ne pas oublier de :
  - dé-déployer vos applications
  - arrêter le serveur d'application :
    - choisir la vue Servers dans la partie inférieur d'Eclipse
    - cliquer avec le bouton droit sur WildFly xxx Runtime Server ..., puis choisir Stop
  - fermer Eclipse

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 avons déjà prévue 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 GlassFish
  - Nous allons désormais travailler soit en dehors d'Eclipse (ne pas le fermer) ou bien dans un projet Java (standard) qu'il faudra créé dans Eclipse.
  - 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 WILDFLY_HOME/bin/client/jboss-client.jar dans un sous-répertoire lib de votre répertoire client, et l'ajouter dans le Classpath (ou le build path si vous travaillez sur Eclipse) avant de compiler et exécuter votre programme client
  - La méthode main de la classe Client doit effectuer un lookup du bean en utilisant les instructions suivantes :
    ```
    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.
    (Ne pas oublier d'ajouter les imports nécessaires : javax.naming.InitialContext, ...)
  - 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:·...:8080
```

  Ne pas oublier de remplacer « ... » 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).

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
```

C. TIBERMACINE