Plateformes de Développement d'Applications Distribuées

Travaux Dirigés N° 02

Enterprise JavaBeans (EJB)

Préliminaires : installation de J2SE et de NetBeans

L'installation de l'environnement de développement **NetBeans 6.7.1 + J2SDK 1.6** sur les machines de l'ENIS se fera par les étudiants sur les postes Ubuntu. Il s'agit simplement de récupérer une archive précompilée préparée pour les TDs. Cette archive peut aussi être utilisée sur tout PC exécutant le système d'exploitation Ubuntu (8.04, 8.10 ou 9.04).

Récupération et extraction de l'archive

Après avoir récupéré l'archive `ENIS_J2EE_Package.tgz`, il faudra l'extraire. Ensuite, dans un terminal, saisir les commandes suivantes :

```
~% tar xzvvf /chemin/vers/ENIS_J2EE_Package.tgz
~% cd  ENIS_J2EE_Package
~% ./install.sh
```

À l'issue des deux commandes ci-dessus, on disposera d'un répertoire **`${HOME}/J2EE`** contenant l'installation de **NetBeans** ainsi que des outils de développement et d'exécution **JAVA 6** de SUN. On disposera aussi d'un fichier nommé **`env.sh`** qui permet de placer correctement les variables d'environnement nécessaires pour la compilation, le déploiement et l'exécution des applications réparties. Ce fichier doit être **"chargé" chaque fois** qu'un nouveau terminal (shell) est ouvert. Le chargement du fichier **`env.sh`** s'effectue, **dans le nouveau terminal**, à l'aide des commandes suivantes :

```
~% source ${HOME}/J2EE/env.sh
```

Pour lancer **NetBeans**, il suffit de saisir la commande `netbeans&` dans un terminal.

Exercice 01 : Convertisseur de devises

L'objectif de cet exercice est la prise en main des EJBs session sans état. Il s'agit de concevoir une application J2EE permettant au client de consulter un EJB session pour demander différents types de conversion de devises :

- dinars tunisien vers euros et vice versa
- dinars tunisien vers dollars américain et vice versa

On donne les deux taux de change suivants :

1. dinar vers euro : **0.51881282**
2. dinar vers dollar américain : **0.77863428**

Question 1 : Conception du squelette de l'application et de l'EJB

L'outil **Netbeans** permet de faciliter considérablement la création d'applications J2EE utilisant les EJBs. Une grande partie du code de l'application va être générée automatiquement. Il suffit ensuite de remplir les esquisses de code générées par le code métier.

Q1.1 : Créer un nouveau projet **Netbeans** de type "*Enterprise Application*" appelé `coursdevises` et choisir d'y incorporer les deux modules suivants :

- Un module EJB appelé `coursdevises-ejb`
- Un module client Java (**et non pas client Web**) appelé `coursdevises-app-client`

Q1.2 : Dans le module `coursdevises-ejb`, ajouter un nouvel EJB session **sans état** appelé `CoursDTN` et choisir de produire une interfaces de type **Remote** pour cetEJB.

Constater la quantité de code généré automatiquement.

Q1.3 : Dans la classe `CoursDTNBean`, utiliser la commande : *Click Droit → Insert Code → Add Business Method* pour ajouter les 4 méthodes métier suivantes :

- `public double toEuro (double d);`
- `public double fromEuro (double e);`
- `public double toDollar (double d);`
- `public double fromDollar (double s);`

Remarquer que ces 4 méthodes sont ajoutées à la fois dans l'interface Remote et dans la classes de l'EJB.

Q1.4 : Compléter la définition de l'EJB en implantant les 4 méthodes et en lui choisissant le nom JNDI "`ejb/CoursDTN`". Le choix du nom JNDI permet au client de pouvoir se connecter à l'EJB en utilisant le service de nommage J2EE. Cette action peut se faire grâce à l'ajout d'un paramètre à l'annotation `@Stateless` :

```
@Stateless (mappedName="...")
```

Q1.5 : Compiler et déployer l'EJB `CoursDTN`

Question 2 : Construction du client Java de l'application

Q2.1 : S'inspirer du code incomplet ci-dessous pour construire et peupler le client Java de l'application. L'annotation `@EJB`, appelée aussi *injection de dépendances*, permet au conteneur du client de se connecter à l'EJB. Elle remplace le code utilisant le service de nommage que l'utilisateur aurait du saisir manuellement dans les version antérieurs à 3.0 de la spécification EJB.

```
@EJB (mappedName="...")
private static CoursDTNRemote cours;

public static void main(String[] args) {
    try {
         double euro = cours.toEuro(1000);
         System.out.println("DTN" + m + " = " + euro + " €.");
         /* Tester toutes les méthodes métier */
         System.exit(0);
    } catch (Exception ex) {
         System.err.println("Exception non prévue!");
         ex.printStackTrace();
    }
}
```

Q2.2 : Compiler, déployer et tester le bon fonctionnement de l'application.

Exercice 02 : Gestion de compte bancaire

Dans cet exercice, on construira une application J2EE permettant la gestion **très simplifiée** d'un compte bancaire (consultation du solde, ajout et retrait d'argent). Pour ce faire, on utilisera un EJB session **avec état**.

Question 1 : Squelette de l'application et EJB

Q1.1 : Construire une nouvelle application J2EE appelée `gestioncompte` contenant les deux modules suivants :

- Un module EJB appelé `gestioncompte-ejb` (nom JNDI : `"ejb/GestionCompte"`)
- Un client **Web** appelé `gestioncompte-war`

Q1.2 : Ajouter un nouvel EJB appelé `GestionCompte` avec une interface **Remote** et contenant les 3 méthodes métier suivantes (et les attributs nécessaires) :

- `public double solde ()` : retourne le solde du compte (initialisé à 0)
- `public void crediter (double montant)` : ajoute la somme donnée dans le compte
- `public boolean debiter (double montant)` : si le solde existant le permet, retire la somme donnée et retourne `true`. Sinon, retourne `false`.

Q1.3 : Compiler et déployer l'EJB

Question 2 : Client Web

Q2.1 : Construire un client **Web** contenant les caractéristiques suivantes (voir les imprimés-écran à la fin de ce document) :

- Le fichier d'accueil principal `index.html` contenant les trois formulaires suivant :
  - Un simple bouton permettant la consultation du solde (traité par `solde.jsp`)
  - Un champ de texte et un bouton " Créditer " permettant l'ajout d'une somme donnée et l'affichage du nouveau solde (`credit.jsp`)
  - Un champ de texte et un bouton " Débiter " permettant le retrait d'une somme donnée (si possible) et l'affichage du nouveau solde. Dans le cas d'un retrait impossible, un message adéquat doit être affiché.
- Une bibliothèque de balises `utilities.tld` définit les 3 fonctions suivantes devra être implantée et utilisée dans les 3 fichiers JSP :

```
<function>
   <description>Retourne le solde bancaire</description>
   <name>solde</name>
   <function-class>gestioncompte.utils.GestionCompteUtils</function-class>
   <function-signature>double solde()</function-signature>
</function>
<function>
   <description>Crédite le compte bancaire</description>
   <name>crediter</name>
   <function-class>gestioncompte.utils.GestionCompteUtils</function-class>
   <function-signature>void crediter(double)</function-signature>
</function>
<function>
   <description>Débite le compte bancaire</description>
   <name>debiter</name>
   <function-class>gestioncompte.utils.GestionCompteUtils</function-class>
   <function-signature>boolean debiter(double)</function-signature>
</function>
```


Q2.2 : Construire la classe utilitaire `gestioncompte.utils.GestionCompteUtils` contenant les 3 méthodes `solde`, `crediter` et `debiter` déclarées dans la bibliothèque de fonctions `utilises.tld`. Cette classe effectue la connexion vers l'EJB session et, dans chacune des méthodes, elle invoque la méthode métier correspondante de l'EJB.
Remarque importante : Vu que les méthodes de la classe `GestionCompteUtils` doivent être statiques et qu'aucune instance de cette classe ne va être créée, on ne peu pas utiliser l'injection de dépendances `@EJB` pour se connecter à l'EJB. Il faut utiliser la méthode classique de résolution de nom JNDI :

```
InitialContext ctx;
GestionCompteRemote compte;
try {
   ctx = new InitialContext();
   compte = (GestionCompteRemote) ctx.lookup("ejb/...");
} catch (NamingException ex) {
   /* … */
}
```

On pourra factoriser le code ci-dessus dans une méthode de la classe `GestionCompteUtils` et l'appeler dans les 3 autre méthodes de la classe avant l'invocation des méthodes métier. On pourra aussi utiliser un attribut booléen pour éviter les initialisations multiples du compte.