

Architecture d'Applications Java EE - EI - Sujet

0. Préliminaires

Installation de l'EI

- Ouvrez un navigateur web (par exemple, Google Chrome).
- Dans la barre d'adresse, tapez l'une des adresses suivantes :
 - `zou.ovh/ei794` (*lien principal*)
 - `tiny.cc/ei794` (*lien de secours, si le 1^{er} lien ne fonctionne pas*)
- Sur la page web qui s'affiche, cliquez sur le bouton **Télécharger**.
- Lorsque le fichier est téléchargé, ouvrez-le pour voir son contenu.
- Double-cliquez sur le fichier `2024-01-i3-archi-jee-a.bat` qu'il contient, afin de démarrer l'installation de l'EI.
- Si le panneau "Windows a protégé votre ordinateur" s'affiche :
 - Cliquez sur le lien **Informations complémentaires**.
 - Puis, actionnez le bouton **Exécuter quand même**.
- Attendez jusqu'à ce que l'exécution du traitement soit terminée.
Une fenêtre de l'explorateur de fichiers a dû s'ouvrir pour afficher le contenu du dossier de travail. Attention ! cette fenêtre est peut-être cachée derrière les autres fenêtres.
Ce dossier comporte 3 sous-dossiers :
 - Sujets-des-TPs contient les sujets des TPs que vous avez faits.
 - Supports-de-Cours contient les supports de cours.
 - workspace est le Workspace Eclipse dans lequel vous allez travailler.
- Double-cliquez sur le raccourci **Eclipse** pour démarrer Eclipse.
- Si tout est OK, on peut couper l'accès au réseau.

Changement de nom du projet ^eleve-

- Faites un clic-droit sur le nom du projet `^eleve-`, puis "Rename..."
- À la suite de `^eleve-`, mettez votre nom. Le résultat doit ressembler à `^eleve-Mon-Nom`.

Important ! : Ajoutez bien votre nom au projet `^eleve-`, car c'est le moyen qui permettra de vous identifier pour l'attribution de la note.
NB. Ce projet est vide et n'a pas d'autre utilité que de vous identifier.

Clean général

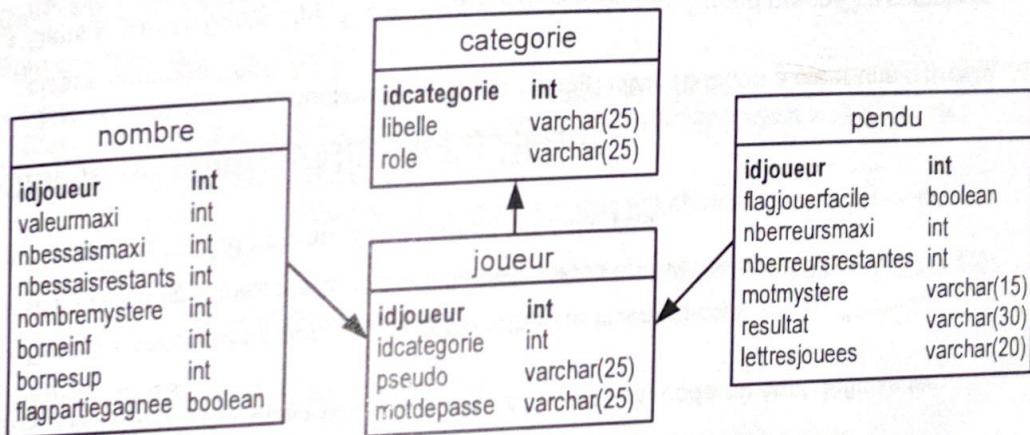
- Pour être sûr d'avoir un environnement de travail propre, effectuez l'opération clean sur l'ensemble du workspace :
 - Menu Project > Clean...
 - Sélectionnez la case : Clean all projects. Puis, appuyez sur le bouton **Clean**.

Thème de l'EI

Vous allez travailler sur une nouvelle version de l'application Jeux.
En plus de jouer au Nombre Mystère et au Jeu du Pendu, cette version permet de gérer une liste de joueurs qui ont chacun un pseudo et un mot de passe pour leur permettre de se connecter.

Il existe deux catégories de joueurs :

- Les joueurs "normaux" de type JOUEUR.
- Les joueurs qui ont le type ADMINISTRATEUR. En plus de pouvoir jouer, ils peuvent gérer les autres joueurs (c'est-à-dire ajouter, modifier et supprimer).



La base de données est composée de 4 tables.

- Un joueur appartient à une catégorie et à une seule (JOUEUR ou ADMINISTRATEUR).
- Les tables **nombre** et **pendu** contiennent l'état des parties.
- Chaque partie appartient à un joueur et un seul.

Mise en marche de l'application

Démarrage de PostgreSQL

- À l'aide de l'application UtilServeurs, démarrez PostgreSQL.
- Utilisez la vue **Data Source Explorer** d'Eclipse pour vérifier que la base de données contient bien les 4 tables **categorie**, **joueur**, **nombre** et **pendu**.

Exécution de l'application

- Exécutez l'application en démarrant la classe AppliJeux1 qui se trouve dans le projet jeux-appli.

La configuration fournie est celle qui accède à la base de données en utilisant JDBC.

Trois joueurs sont enregistrés et permettent de se connecter.

Pseudo	Mot de Passe	Catégorie
admin	admin	Administrateur
tom	tom	Joueur
lulu	lulu	Joueur

- Connectez-vous avec le compte admin et vérifiez que :
 - Vous pouvez jouer au Nombre Mystère et au Jeu du Pendu.
 - Vous pouvez créer, modifier et supprimer des joueurs.
 - Vous pouvez créer, modifier et supprimer des catégories.
Attention ! Ne modifiez pas les catégories existantes.

Structure de l'EI

Cette EI est découpée en 2 parties :

1. JPA (12 points)
2. EJB (8 points)

Ces 2 parties sont indépendantes. Vous pouvez les traiter dans l'ordre que vous voulez.

1. Accès à la base avec JPA

Dans cette partie, il vous est demandé d'effectuer les transformations nécessaires pour que l'accès aux données se fasse par l'intermédiaire de JPA.

Configuration du projet jeux-appli

Ajout de la dépendance à Hibernate

- Dans le projet **jeux-appli**, ouvrez le fichier **POM**.
- Choisissez l'onglet **pom.xml** et recherchez la partie du code où sont définies les dépendances.
- Faites le nécessaire pour ajouter la bibliothèque **hibernate-core**.
- Si besoin, supprimez les lignes inutiles (signalées en avertissement)
- Enregistrez la modification. Puis, fermez le fichier **POM**.

Classe de configuration pour Spring

- Dans le package **jeux**, dupliquez la classe **Config1Jdbc**, de façon à créer une nouvelle classe nommée **Config2Jpa**.
- Dans la classe **Config2Jpa**, faites le nécessaire pour que Spring prenne en compte le package **jeux.emb.dao.jpa** à la place du package **jeux.emb.dao.jdbc**.
- Cette classe contient une méthode **titre()**. Faites en sorte qu'elle retourne la chaîne : "Jeux JPA"
- Dans la classe **AppliJeux1**, faites le nécessaire pour que ce soit cette nouvelle classe qui soit utilisée comme classe de configuration de Spring.

Lorsque vous démarrez l'application, la trace doit permettre de vérifier que vous utilisez bien les classes du package **dao.jpa**.

CONFIG:
Couche DAO : jeux.emb.dao.jpa
Couche Service : jeux.emb.service.standard
Couche Model : jeux.gui.model.standard

Actuellement, seule la vue Connexion fonctionne (grâce à un code factice). Toutes les autres actions produisent des erreurs, car le code est entièrement à écrire.

- Revenez dans la classe **Config2Jpa**.
- Observez qu'elle contient la définition d'un objet **DataSource** qui correspond à la version JDBC de l'application.

- Cette définition n'est plus utile. Replacez-la par la définition des 2 objets dont a besoin JPA :
 - Un objet EntityManagerFactory qui peut être créé avec l'instruction

```
return Persistence.createEntityManagerFactory( "jeux" );
```
 - Un objet EntityManager qui peut être créé avec l'instruction

```
return emf.createEntityManager();
```Cela suppose que la méthode reçoit un paramètre de type EntityManagerFactory nommée emf.
- Exécutez l'application, connectez-vous et actionnez le bouton **Pendu**.
 - La vue du Jeu du Pendu doit s'afficher correctement.
 - Dans la vue **Console** d'Eclipse, vous devez voir les traces rouges typiques d'Hibernate, qui sont du type:

```
INFO: HHH000490: ...
```
 - Il n'est pas possible de jouer au Pendu, car une partie du code est manquante.

Transformation des classes en entités JPA

Ajout des références des classes dans le fichier persistence.xml

Les classes qui doivent être transformées en entités JPA sont celles qui se trouvent dans le package `jeux.emb.data`.

- Faites le nécessaire pour référencer les classes de ce package dans le fichier `persistence.xml`.
- Enregistrez les modifications.

Eclipse doit signaler des erreurs au niveau de chacune de ces classes du package, car il reste à les transformer en entités.

Vous allez commencer par la classe `Joueur`.

Mapping de la classe `Joueur`

- Ouvrez le code de la classe `Joueur`.
- Faites les modifications nécessaires pour que cette classe devienne une entité JPA correspondant à la table `joueur` (voir le schéma au début de ce document).
- Vous devez prendre en compte les particularités suivantes :
 - La clé primaire de la table `joueur` est générée automatiquement par le SGBD.
 - La table a une clé étrangère car un joueur appartient à une catégorie.
 - Il vous est demandé de mettre toutes les annotations `@Table` et `@Column`, même quand leur présence est facultative.

Rappel : Eclipse a tendance à signaler des erreurs lorsqu'on est en train de faire le mapping. La plupart de ces erreurs sont dues au fait que le travail n'est pas terminé. Elles disparaîtront lorsque vous aurez terminé le mapping.

Mapping de la classe Categorie

- Ouvrez le code de la classe Categorie.
- Faites les modifications nécessaires pour que cette classe devienne une entité JPA correspondant à la table categorie.
- Vous devez prendre en compte les particularités suivantes :
 - La clé primaire de la table categorie est générée automatiquement par le SGBD.
 - La classe Categorie gère la liaison réciproque vers la classe Joueur.
 - On souhaite que les traitements appliqués à un objet Categorie soient tous répercutés en cascade sur les objets Joueur qui lui sont rattachés.
 - Il est demandé que les joueurs soient triés par pseudo.
 - Il vous est demandé de mettre toutes les annotations @Table et @Column, même quand leur présence est facultative.

Mapping des classes Nombre et Pendu

Dans le package jeux.emb.data, il y a encore 2 classes à annoter : Nombre et Pendu.

Pour ces deux classes, la consigne est différente.

Il vous est demandé de mettre le moins possible d'annotations.

- Il vous est demandé de mettre le moins possible d'annotations.
C'est-à-dire, uniquement les deux annotations dont la présence est indispensable.

NB : Pour les 2 tables nombre et pendu :

- Les colonnes ont exactement le même nom que les champs correspondants dans les classes Java.
- La valeur de la clé primaire n'est pas générée automatiquement par le SGBD. Elle est fournie par le code Java.

Utilisation des classes de test

Des classes de test sont fournies pour vous permettre de tester votre code.

- Dans le projet jeux-emb, dépliez la branche src/test/java.
- Faites un clic-droit sur le package jeux.emb.dao.jpa.
Puis, Run As... > JUnit Test

La plupart des tests échouent, car le code n'est pas encore écrit. À la fin, tous les tests devraient réussir.

Vous n'êtes pas obligé d'exécuter à chaque fois tout le package.

Par exemple, si vous travaillez sur la classe DaoCategorie, vous pouvez la tester en exécutant uniquement la classe TestDaoCategorie.

Rappels sur l'utilisation de JUnit

- Échecs :

- Dans la vue **JUnit**, si un test est signalé avec un pictogramme bleu, il s'agit d'un test qui a échoué.

- Un double-clic sur la ligne où se trouve le pictogramme bleu permet d'afficher l'instruction qui a effectué le test en échec.
- À gauche, en bas, dans la section **Failure Trace**, un double-clic sur la 1^{re} ligne permet généralement d'afficher une explication.

- **Erreur :**

- Si le pictogramme est rouge, il s'agit d'une erreur. C'est-à-dire qu'un bug a généré une exception.
- En sélectionnant la ligne qui contient le pictogramme rouge, cela fait apparaître, en bas dans la section **Failure Trace**, le résumé de la trace de l'exception.
Dans la barre qui contient le titre **Failure Trace**, à droite, on peut cliquer sur la 1^{re} icône Show Stack Trace in Console View. Cela permet de faire afficher la trace complète de l'exception dans la vue **Console**.

Classe DaoCategorie

EntityManager

- Dans le package jeux.emb.dao.jpa, ouvrez le code de la classe **DaoCategorie**.
- Ajoutez-lui une variable privée, nommée **em**, pour représenter l'**Entity Manager**.
Cette variable doit être initialisée par injection de dépendance.

Opérations de base

- Écrivez le code des méthodes :
 - **insérer()**
 - **modifier()**
 - **supprimer()**
 - **retrouver()**

Rappel : la méthode **insérer()** doit retourner l'identifiant après que celui-ci ait été généré automatiquement par le SGBD.

- Test : Exécutez la classe **TestDaoCategorie**.
Les 5 premiers tests doivent réussir.

listerTout()

- Écrivez le code de la méthode **listerTout()**.
Les catégories doivent être triées par libellé.
- Test : Exécutez la classe **TestDaoCategorie**.
Le dernier test doit réussir.

Classe DaoJoueur

EntityManager

- Ouvrez le code de la classe **DaoJoueur**.

- Ajoutez-lui une variable privée, nommée `em`, pour représenter l'Entity Manager.
Cette variable doit être initialisée par injection de dépendance.

Opérations de base

- Écrivez le code des méthodes `insérer()`, `modifier()`, `supprimer()` et `retrouver()` ;
- Test : Exécutez la classe `TestDaoJoueur`.
Les 4 premiers tests doivent réussir.

`listerTout()`

- Écrivez le code de la méthode `listerTout()`.
Les joueurs doivent être triés par pseudo.
- Test : Exécutez la classe `TestDaoJoueur`.
Le test n°5 doit réussir.

`compterPourCategorie()`

- Écrivez le code de la méthode `compterPourCategorie()`.
Cette méthode retourne le nombre de joueurs qui sont rattachés à la catégorie dont l'identifiant est passé en paramètre.
Utilisez une requête JPQL pour obtenir le résultat souhaité en utilisant la fonction `COUNT()`.
- Test : Exécutez la classe `TestDaoJoueur`.
Les tests n° 6 et 7 doivent réussir.

`validerAuthentification()`

Actuellement, la méthode `validerAuthentification()` retourne un faux joueur `admin`, quel que soit le pseudo et le mot de passe saisis par l'utilisateur.

- Dans la méthode `validerAuthentification()`, supprimez le code factice.
- À la place, faites le nécessaire pour que la méthode retourne le joueur correspondant au pseudo et au mot de passe qui sont passés en paramètres.
La méthode doit retourner `null`, si aucun joueur ne correspond au couple pseudo + mot de passe.
Il vous est demandé d'utiliser `getSingleResult()`.
- Test : Exécutez la classe `TestDaoJoueur`.
Les tests n° 8 et 9 doivent réussir.

verifierUnicitePseudo()

- Écrivez le code de la méthode `verifierUnicitePseudo()`.

Cette méthode doit renvoyer "faux", si au moins un joueur dans la table vérifie les conditions suivantes :

- son pseudo est égal à celui reçu en paramètre
- et son identifiant est différent de celui reçu en paramètre.

Utilisez une requête JPQL spécifique pour obtenir le résultat souhaité.

Il vous est demandé d'utiliser `getSingleResult()`.

- Test : Exécutez la classe `TestDaoJoueur`.

Les 4 derniers tests doivent réussir.

2. Utilisation des EJBs

Le projet jeux-ejb contient une version de la couche Logique Métier qui utilise JDBC pour accéder à la base de données.

Pour pouvoir la déployer sur le serveur WildFly, il suffit de transformer en EJBs les classes Services et DAOs.

Classes DAOs

Annotations de base

- Dans le projet jeux-ejb, dépliez le package jeux.ejb.dao.jdbc.
- Ouvrez le code de la classe DaoCategorie.

Vous allez lui ajouter les 2 annotations de base qui vont la transformer en EJB.
Vous avez 4 possibilités. Faites le bon choix !

| | | | |
|-----------|------------|-----------|------------|
| @Stateful | @Stateless | @Stateful | @Stateless |
| @Remote | @Remote | @Local | @Local |

- Une fois que vous avez mis les annotations sur la classe DaoCategorie, procédez par copier-coller pour mettre les mêmes annotations sur les 3 autres classes DAOs

Annotation de la variable DataSource

Les classes DAOs contiennent une variable de type DataSource. Cette variable doit être annotée afin qu'elle soit initialisée par injection de dépendance.

- Dans les 4 classes DAOs, ajoutez l'annotation suivante à la variable DataSource :
`@Resource(lookup="java:/ds/jeux")`

Remarque : il ne faut pas mettre l'annotation habituelle pour l'injection de dépendance.
Ici, c'est l'annotation @Resource qui la remplace.

Classes DaoJoueur

- La classe DaoJoueur contient une variable de types IDaoCategorie qui doit être annotée afin d'être initialisées par injection de dépendance.
- Ajoutez l'annotation nécessaire (il s'agit de l'annotation habituelle).

Classes Services

Annotations de base

- Dans le projet jeux-ejb, dépliez le package jeux.ejb.service.standard.
- Ouvrez le code de la classe ServiceCategorie.

De nouveau, vous avez 4 choix possibles :

| | | | |
|-----------|------------|-----------|------------|
| @Stateful | @Stateless | @Stateful | @Stateless |
| @Remote | @Remote | @Local | @Local |

- Une fois que vous avez mis les annotations sur la classe `ServiceCategorie`, procédez par copier-coller pour mettre les mêmes annotations sur les autres classes Services

Annotation des variables

Chaque classe Service contient plusieurs variables. Celles dont le type commence par la lettre I (par exemple `IDaoCategorie`, `IMapper`, ...) doivent être initialisées par injection de dépendance.

- Faites le nécessaire, dans les 5 classes Services, pour que ces variables puissent être initialisées correctement par le serveur WildFly.
Attention à ne pas traiter, à tort, des variables qui ne sont pas concernées.

Optimisation des transactions

Actuellement, les EJBs ne comportent aucune indication sur la gestion des transactions. Tous les traitements sont donc exécutés au sein d'une transaction même lorsque ce n'est pas nécessaire. Cela entraîne un gaspillage des ressources du serveur.

Traitements non transactionnels

- Dans le projet jeux-ejb, dans le package `jeux.ejb.service.standard`, ouvrez le code de la classe `ServiceCategorie`.
Cette classe contient 5 méthodes publiques. Parmi elles, 2 méthodes n'ont pas besoin d'être exécutées au sein d'une transaction.
- Annotez ces 2 méthodes afin que WildFly ne mette pas en place de transaction lorsqu'elles sont exécutées.
- Ajoutez la même annotation, à toutes les méthodes concernées, dans les autres classes Services.

Précisions :

- Dans la classe `ServiceConnexion`, les 2 méthodes qui s'y trouvent n'ont pas besoin de transaction.
- Dans les classes `ServiceNombre` et `ServicePendu`, ce sont les méthodes `retrouver()` et `tricher()` qui n'ont pas besoin de transaction.

Traitements non transactionnels des EJBs DAOs

- Dans les 4 classes DAOs, ajoutez aussi la même annotation aux méthodes concernées.
Attention ! Le nombre de méthodes concernées n'est pas le même dans tous les DAOs.

Méthodes transactionnelles des EJBs DAOs

- Ouvrez le code de la classe `DaoCategorie`.

Cette classe contient 3 méthodes publiques qui nécessitent d'être exécutées au sein d'une transaction.

- À chacune de ces 3 méthodes, ajoutez l'annotation qui indique qu'elle doit être exécutée au sein d'une transaction qui existe au préalable. C'est-à-dire qu'elle nécessite une transaction qui a été créée au sein d'un autre EJB.
- Ajoutez cette même annotation aux méthodes concernées, dans toutes les classes DAOs.

Adaptation des classes DTOs

Pour pouvoir être déployées sur le serveur, les classes DTOs doivent être sérialisables.

- Dans le projet jeux-commun, dépliez le package jeux.commun.dto.
- Faites le nécessaire pour rendre sérialisable, chacune des classes dont le nom commence par **DTO**
- Si vous le souhaitez, vous pouvez ajouter une annotation `@SuppressWarnings` pour éviter qu'Eclipse ne signale un avertissement pour chacun de ces classes.

Configuration du projet EAR

Le projet jeux-ear vous est fourni, car sa création nécessite d'être connecté à Internet. Ce projet est dans l'état où il se trouve juste après avoir été créé par l'assistant d'Eclipse.

Adaptation du fichier POM

- Ouvrez le fichier POM du projet jeux-ear.
- Faites le nécessaire pour faire disparaître les avertissements.
- Vers la fin du fichier POM, recherchez la balise fermante `</configuration>`.
- Juste avant cette balise, insérez la ligne suivante :

```
<applicationName>jeux</applicationName>
```

(Attention au **N** majuscule)

Cela va indiquer au serveur WildFly que le nom de l'application est simplement jeux (et nom jeux-ear qui serait la valeur choisie par défaut).

Configuration des dépendances

- À la fin du code, juste avant la balise fermante `</projectct>`, insérez une ligne vide.
- Faites le nécessaire pour que, sur cette ligne, soit écrit le code :
`<dependencies></dependencies>`
- Puis, insérez une ligne vide entre ces 2 balises `<dependencies></dependencies>`.
- À cet endroit, utilisez la complétion de code (**Ctrl+Espace**), pour y ajouter la dépendance à la bibliothèque mapstruct (tout court).

- Il faut aussi y ajouter la dépendance au projet **jeux-commun**.

Comme la complétion de code ne fonctionne pas dans ce cas-là, vous pouvez récupérer cette dépendance dans un autre fichier POM, par exemple celui du projet **jeux-appli**.

- Enfin, il faut également insérer la dépendance au projet **jeux-ejb**.

Vous pouvez prendre modèle sur la dépendance à **jeux-commun** que vous venez d'insérer.

Mais, **attention !** Cette dépendance doit contenir la ligne :

<type>ejb</type>

- Enregistrez les modifications et fermez le fichier POM.

Configuration du nom du module EJB

- Dans le projet **jeux-ejb**, dépliez la branche **src/main/resources**
- Clic-droit sur le dossier **META-INF**, puis **New > Other... > XML > XML File**.
Bouton **Next >**
- Dans le champ **File Name**, indiquez : **ejb-jar.xml**
Bouton **Finish**.
- Au bas de la vue centrale, choisissez l'onglet **[Source]**.
- Faites le nécessaire pour que le contenu du fichier soit :

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar>
    <module-name>ejb</module-name>
</ejb-jar>
```

Cela indique à WildFly que le nom du module est simplement **ejb** (et nom **jeux-ejb** qui serait la valeur choisie par défaut).

- Enregistrez la modification et fermez le fichier.

Ajout du pilote JDBC à WildFly

- Dans Eclipse, démarrez le serveur WildFly.
- Dans un navigateur web, ouvrez la console d'administration web de WildFly (Web Management Console).
URL : <http://localhost:9990>
Nom d'utilisateur : admin
Mot de passe : admin
- Allez dans la section Deployments
- Faites le nécessaire pour ajouter le fichier **postgresql-xx.x.x.jar** qui se trouve dans le dossier **C:\dev24\jdbc-drivers\postgresql**
- Dans le champ Runtime Name, indiquez **postgresql** (tout court).

Configuration de la DataSource dans WildFly

- Toujours dans la Console d'Administration Web de WildFly., à l'aide du menu Configuration > Subsystems, ajoutez une nouvelle DataSource ayant les caractéristiques suivantes :
 - Name : Jeux
 - JNDI Name : java:/ds/jeux
 - Connection URL : jdbc:postgresql://localhost:5432/postgres?currentSchema=jeux
Supprimer db et bien mettre un S majuscule
 - User Name : postgres
 - Password : postgres
- Pensez à cliquer sur le lien **Reload required.**

Déploiement et test

- Revenez dans Eclipse.
- Déployez le projet jeux-ear sur le serveur WildFly.
Normalement, le déploiement devrait s'effectuer sans erreur.
- Exécutez l'application en démarrant la classe AppliJeux3 qui se trouve dans le projet jeux-appli-ejb.
Tout devrait fonctionner sans problème.

Classe TestEjb

- Dans le package jeux du projet jeux-appli-ejb, ouvrez le code de la classe TestEjb.
- Cette classe contient 2 instructions incomplètes : `ic.lookup("...")`
- Remplacez les trois petits points par les noms JNDI des 2 EJBs suivants :
 - ServiceCategorie
 - ServiceJoueur
- Exécutez la classe TestEjb.
Cela doit vous afficher la liste des catégories et celle des joueurs.