# Partage et gestion de collections musicales

Ce TD va vous préparer aux 3 séances de TP pendant lesquelles vous allez implémenter une application JavaEE 6 de gestion et de partage de collections musicales.

## Scénario

- Chaque utilisateur possède des fichiers musicaux.
- Les morceaux sont inclus dans des collections.
- Seul l'utilisateur qui possède une collection peut la modifier.
- Les autres utilisateurs ont une vue en lecture seule de la collection d'un autre.
- On peut écouter les morceaux de la collection d'un autre utilisateur.
- Si on est ami avec un utilisateur, on peut copier ses morceaux dans sa propre collection.

## Contraintes

1. Ce projet sera réalisé avec des EJBs session (*stateless, stateful* et *singleton*).
2. **Pas de base de données !**
3. Pas de gestion de sessions utilisateur, on la délèguera à une session EJB + web (2 fenêtre d'un navigateur web auront la même session, 2 navigateurs différents auront 2 sessions distinctes).

*Pour être précis, il y aura un répertoire sur le serveur géré par l'application dans lequel seront déposés les fichiers musicaux des différentes collections (upload web, …). Les fichiers seront accédés par leur nom depuis les collections, e.g., MaCollection = {lorie-ma-meilleure-amie.mp3, sabrina-boysboysboys.mp3}*

## Conception de l'interface graphique initiale

1. Concevoir une interface graphique qui permette la création d'une collection musicale (demande de votre nom pour une session nouvelle, puis accès à la collection), l'ajout/suppression de morceaux, la lecture, et fermer la session.
2. Proposez des EJBs (type, interface) pour supporter cette interface graphique.
3. Un EJB spécifique sera dédié à la gestion d'un dépôt commun de fichiers musicaux : TunesRepository(import(File), export(name) : File, list() : List<File>).
4. Ecrivez du code client de ces EJBs pour illustrer l'emploi des interfaces.

## Accéder aux collections des autres

1. Etendre l'interface graphique pour accéder aux collections des autres utilisateurs.
2. Proposez une solution à base d'EJBs pour permettre cela. Tenir compte du cycle de vie des instances d'EJB, et en particulier du fait qu'une session peut se terminer, et que vous devrez donc la gérer proprement vis-à-vis des autres utilisateurs.
3. Illustrez dans du code client.

## Votre réseau social

1. Ajouter la prise en compte d'amis dans l'interface, et la possibilité de copier dans sa collection les morceaux qui vous plaisent de vos amis.
2. Toujours à base d'EJBs, proposez une solution.
3. Illustrez dans du code client.

# Compléments JavaEE 6

## Singletons

Java EE 6 a introduit un nouveau type d'EJBs : les singletons. Ce sont des EJBs sessions particuliers pour lesquels 1 seule instance sera crée. Exemple :

```
package ejbs;

import javax.ejb.Local;

@Local
public interface GlobalCounter
{
    public int increment();
}
```

et :

```
package ejbs;

import javax.ejb.Singleton;

@Singleton
public class GlobalCounterBean implements GlobalCounter
{
    private int counter = 0;

    public int increment()
    {
        return counter++;
    }
}
```

Comme tout EJB session, ils bénéficient des services du conteneur d'EJB. Ainsi par défaut ils s'exécutent en contexte transactionnel.

## Injection de ressources JavaEE

Tout objet géré par le serveur d'application (connexion à une base de donnée, file de messagerie, EJB, etc) est disponible pour injection, ce qui remplace avantageusement les recherches via JNDI / InitialContext .

Une demande d'injection se fait en plaçant l'annotation @Resource, qui est un marqueur utilisé par le conteneur d'EJB pour repérer les points d'injection. On peut spécifier un nom JNDI pour la ressource à injecter. Dans le cas d'EJBs utilisant le nommage par défaut (nom de la classe avec la première lettre en minuscule), ce n'est pas la peine de le préciser.

```
@Stateless
public class MyBean
{
    @Resource // same as the next injection
    OtherEJB other1;

    @Resource(name="otherEJB")
    OtherEJB other2;

    @Resource(name="jms/TasksQueue")
```

```
    Queue tasksQueue;
}
```

Dans le cas très particulier des EJBs, on peut remplacer @Resource par @EJB qui a le même effet.

## Servlets

Le traitement de requêtes http se fait par des Servlets. Exemple :

```
package servlets;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class CleanServlet extends HttpServlet
{
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        request.setAttribute("path", request.getPathInfo());
        request.setAttribute("message", "coin coin");
        getServletConfig()
                .getServletContext()
                .getRequestDispatcher("/WEB-INF/views/sample.jsp")
                .forward(request, response);
    }
}
```

Cette Servlet traite des requêtes http GET, extrait le path info (« le reste de l'URL après le préfixe qui ammène à la servlet »), pose 2 valeurs dans le contexte de requête, et délègue le rendu HTML à la JSP suivante, placée dans WEB-INF/views/sample.jsp :

```
<!DOCTYPE HTML>
<html lang="en">

<head>
    <meta charset="utf-8">
    <title>Hello</title>
</head>

<body>

<h1>Hello <%= request.getAttribute("path") %></h1>
<p><%= request.getAttribute("message") %></p>

</body>
</html>
```

Pour déployer cette Servlet, il faut la déclarer dans WEB-INF/web.xml, et indiquer quelles URLs elle peut traiter :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
         xmlns="http://java.sun.com/xml/ns/javaee"
```

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <servlet>
        <servlet-name>Counter</servlet-name>
        <servlet-class>servlets.CounterServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Counter</servlet-name>
        <url-pattern>/counter</url-pattern>
    </servlet-mapping>

    <servlet>
        <servlet-name>Clean</servlet-name>
        <servlet-class>servlets.CleanServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Clean</servlet-name>
        <url-pattern>/clean/*</url-pattern>
    </servlet-mapping>

</web-app>
```

## CDI : Container Dependency Injection

Java EE 6 a introduit une nouvelle spécification pour l'injection de dépendances. Elle est particulièrement adaptée aux applications web.

Concrètement, une classe annotée par @javax.annotation.ManagedBean devient disponible pour injection dans les autres classes. La spécification d'un point d'injection se fait avec @javax.inject.Inject.

Pour activer CDI dans un conteneur web, il suffit d'ajouter un fichier (même vide) META-INF/beans.xml, ou WEB-INF/beans.xml.

Une classe qui peut être injectée peut se voir déclarer un scope, les plus utiles étant :

- @javax.enterprise.context.ApplicationScoped (global)
- @javax.enterprise.context.SessionScoped (associé à une session web)
- @javax.enterprise.context.RequestScoped (associé au traitement d'une requête http)

**En outre, CDI sait accéder au conteneur d'EJB**, ce qui permet d'injecter des références sur des EJBs locaux. **Attention, cela ne marche pas avec les EJBs remote !** Ceci étant, on peut rendre une interface d'EJB session à la fois distante et locale :

```
package ejbs;

import javax.ejb.Local;
import javax.ejb.Remote;

@Remote
@Local
public interface Counter
{
    public int get();
```

```
    public int increment();
}
```

On peut ainsi faire un EJB qui est aussi utilisé dans CDI :

```
package ejbs;

import javax.ejb.Stateful;
import javax.enterprise.context.SessionScoped;

@Stateful
@SessionScoped
public class CounterBean implements Counter
{
    private int counter = 0;

    @Override
    public int get()
    {
        return counter;
    }

    @Override
    public int increment()
    {
        counter = counter + 1;
        return counter;
    }
}
```

Utilisation dans une servlet :

```
package servlets;

import ejbs.Counter;
import ejbs.GlobalCounter;

import javax.inject.Inject;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class CounterServlet extends HttpServlet
{
    @Inject
    Counter counter;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        response.setContentType("text/plain");
        response.getWriter().write(
                String.format("Counter = %d, counter.increment())) ;
    }
}
```