IN329 : JVM SI

<garion@supaero.fr>

L'objectif de ce TP est de construire un *class loader* personnalisé pour comprendre le chargement dynamique dans la JVM.

---

1 Objectifs

Les objectifs de ce TP sont les suivants :
- comprendre le fonctionnement des *class loaders* ;
- redéfinir son propre *class loader*.

2 Redéfinir son propre *class loader*

Nous allons écrire notre propre *class loader*. Celui-ci devra pouvoir charger des classes dont le *bytecode* est contenu dans un fichier ne se terminant pas par `.class` mais par `.impl`. Pour cela, écrire une classe `MonClassLoader` qui étend `ClassLoader` et qui possède les méthodes suivantes[1] :
- `Class loadClass(String nom)` qui appelle la méthode `loadClass(nom, true)`
- `Class loadClass(String nom, boolean true)` qui :
  - vérifie que la classe n'a pas déjà été chargée. On utilisera un attribut de type `java.util.HashTable` pour stocker les classes déjà chargées par notre *class loader* ;
  - essaye d'utiliser le *class loader* du système (c'est le *class loader* parent par défaut) ;
  - sinon :
    - charge le *bytecode* depuis un fichier se terminant par `.impl` et ne se situant pas dans un répertoire du `CLASSPATH` (on choisira arbitrairement un répertoire) ;
    - construit une instance de `Class` à partir du *bytecode* via la méthode `defineClass` ;
    - résoud la classe si besoin est ;
    - place la classe dans la table de hachage en utilisant son nom comme clé ;
    - renvoie l'objet de type `Class` ainsi construit.
- une méthode `byte getClassImplFromDataBase(String className)` qui lit un fichier contenant le *bytecode* de la classe. On pourra utiliser le code suivant :

```
FileInputStream fi = new FileInputStream("monChemin"+className+".impl");
result = new byte[fi.available()];
fi.read(result);
return result;
```

3 Écrire une application utilisant le nouveau *class loader*

Dans un premier temps, nous allons écrire une classe que nous allons charger via notre *class loader* particulier. Cette classe possédera une méthode `start` qui fera les opérations suivantes :
- créer un objet de type `java.util.Random` ;
- stocke plusieurs nombres aléatoires dans un vecteur de type `java.util.Vector` ;
- crée de nouveau un objet de type `java.util.Random` ;

[1] On affichera sur la console les opérations effectuées pour tracer les appels.

– affiche les éléments du vecteur.

Le *bytecode* de cette classe ne sera pas disponible dans le `CLASSPATH`. Il suffit pour cela de changer l'extension du fichier.

Créer ensuite une classe de test utilisant `MonClassLoader` pour charger la classe développée précédemment et exécuter la méthode `start` de cet objet. Qu'observe-t-on au niveau des traces ?