TP nº 6

Consignes les exercices ou questions marqués d'un ⋆ devront être d'abord rédigés sur papier (afin de se préparer aux épreuves écrites du partiel et de l'examen). En particulier, il est recommendé d'être dans les même conditions qu'en examen : pas de document autres que le cours ni de calculatrice. Tous les TPs se font sous Linux.

1 Préambule

Le but TP est le suivant :
— corriger le bug de conception du TP précédent.
— transformer simplement l'application en application réseau au moyen du *package* `java.rmi`.
— **il est interdit de renommer les classes et packages Java**
Le projet Eclipse contenant le code à compléter est récupérable sur la page du cours.

2 Gestion des IDs

Le TP précédent contient un bug dans la gestion des IDs de films. En effet, le générateur d'identifiants n'est pas correctement sauvegardé avec la base. Lors d'un nouveau lancement d'application, ce dernier est de nouveau à 0 alors que la base chargée contient des films.

Régler ce problème va nous permettre de régler un problème plus général : en présence d'applications client-serveur, les IDs doivent être uniques pour *tous les clients* qui, a priori, ne parlent qu'avec le serveur.

1. Ajouter à la classe `Movie` un constructeur par copie qui prend en argument un objet `Movie` et renvoie une copie de cet objet avec un nouvel identifiant.
2. ⋆ On considère la méthode `add` de la classe `MovieDB`. Que peut-il se passer si, une fois mise en réseau, cette méthode est appelée par deux clients indépendants qui fournissent chacun leur objet `Movie` ?

> **Réponse:** Si deux clients génèrent chacun un `Movie` avec le même `id` leur ajout créera un conflit (et seul l'objet ajouté en deuxième sera dans la base).

3. Modifier la méthode `add` de la classe `MovieDB` pour utiliser le constructeur par copie de la classe `Movie` et assurer ainsi que l'identifiant du `Movie` ajouté est unique.
4. ⋆ On considère la méthode `load` de la classe `MovieDB`. Que se passe-t'il si l'on a déjà ajouté des films et que l'on charge une base précédemment sauvée ?

> **Réponse:** Les films précédemment rentrés sont écrasés. Les films chargés ont des valeurs d'`id` correspondant à une ancienne exécution du programme et sans rapport avec l'objet `AtomicInteger` utilisé dans la classe `Movie`.

5. Modifier la méthode `load` pour *ajouter* à la base existante les films se trouvant dans la base chargée, en modifiant leur id si besoin.
6. Compléter la méthode `mainLoop` de la classe driver, pour les cas `"n"` (nouveau film), `"S"` (suppression de film) et `"f"` (recherche de film par identifiant). Deux méthodes `readString(BufferedReader in, String msg)` et `readInt(BufferedReader in, String msg)` ont été ajoutées à cette classe pour factoriser le code de saisie au clavier.

3 Mise en réseau

On souhaite maintenant mettre l'application en réseau, en utilisant le package `java.rmi`.

1. ⋆ Quelle classe fait office de serveur ?

> **Réponse:** La classe `MovieDB`.

2. Modifier l'interface `IMovieDB` lui faire : *(i)* étendre l'interface `Remote` et *(ii)* faire en sorte que chaque méthode de l'interface puisse lever l'exception `RemoteException`.
3. ⋆ Quelles sont les classes impactées par ce changement ?

> **Réponse:** La classe `MovieDB` compile sans changement. La classe `Driver` est presque inchangée (on doit rattraper `RemoteException` par endroit).

4. Modifier maintenant la classe `MovieDB` pour lui ajouter une méthode `public static void main()`. Comme indiquée en cours, cette dernière devra créer un `SecurityManager`, créer une instance de la classe `MovieDB` et l'ajouter sous un certain nom dans la *registry*.
5. Exécuter depuis Eclipse la classe `MovieDB` et constater une exception (`java.security.AccessControlException`) Modifier les propriétés d'exécution de la classe de la manière suivante :
   — clique droit sur le fichier `MovieDB.java`
   — sous-menu `Run As`
   — `Run Configurations...`
   — onglet `Arguments`
   — dans le champ de texte `VM arguments`, mettre :
   `-Djava.security.policy=default.policy`
   Se placer **dans le terminal** dans le répertoire `bin` se trouvant à la racine du projet eclipse du TP. Et exécuter la commande :

   ```
   rmiregistry -J-Djava.rmi.server.codebase=file:/`pwd`/
   ```

   La commande ne rend pas la main. Lancer maintenant de nouveau la classe `MovieDB` et vérifier qu'elle ne rend pas la main mais n'affiche pas non plus d'exception.
6. Modifier maintenant la classe `Driver` pour lui ajouter une méthode `public static void main()`. Comme indiquée en cours, cette dernière devra créer un `SecurityManager`, récupérer une instance de l'interface `IMovieDB` via la *registry* et la passer à l'objet `Driver`.
7. Ajouter la même option

   ```
   -Djava.security.policy=default.policy
   ```

   pour cette classe à la configuration d'exécution.
8. Tester le client et le serveur.