

## TP 4

### Introduction au framework OSGi

#### Exercice 1. Mini-tutoriel Equinox (et Apache Felix)

- Mettre en place l'environnement de travail :
  - Télécharger Equinox :
    - Sur le dépôt Github `ctiber/composants`, télécharger le fichier `org.eclipse.osgi_xxx.jar` se trouvant ici (v3.7)  
Depuis la version 3.8 le démarrage du framework en mode console nécessite d'autres jar (console gogo shell (gosh) de Felix) et une configuration particulière
  - Ce même TP peut être fait avec la dernière version d'Apache Felix : (solution recommandée)  
<http://felix.apache.org/downloads.cgi>
- Démarrer le framework Equinox :
  - Aller dans le répertoire où se trouve le JAR téléchargé et taper la commande :  
`java -jar org.eclipse.osgi_3.7.1.R37x_v20110808-1106.jar -console`  
Pour Felix (en se mettant dans le répertoire d'installation) :  
`java -jar bin/felix.jar`
  - Après quelques instants, vous aurez accès au prompt OSGi : `osgi>` ou `g!`
- Saisir la commande `help` pour tester l'environnement (affiche toutes les commandes possibles à partir de ce prompt) :  
`osgi> help` ou `g! help`
- Saisir la commande `ss` (qui veut dire « *short status* ») et observer ce qui se passe :  
`osgi> ss` (pour la dernière version de Felix : `g! lb`)  
Cette commande affiche la liste des bundles OSGi installés dans le conteneur, leur id et leur état (Active, Installed, Resolved, ...). La première fois que vous démarrez cet environnement, il y a déjà un bundle installé et actif ; il s'agit d'Equinox (ou bien dans Felix, plusieurs bundles sont installés pour démarrer le framework en mode console)
- Saisir la commande `headers <id-du-bundle>` pour afficher le contenu du Manifest du bundle dont l'identifiant est `<id-du-bundle>` : (même chose pour Felix)  
`osgi> headers 0`
- Saisir la commande `bundle <id-du-bundle>` pour afficher l'état détaillé d'un bundle :  
`osgi> bundle 0` (même chose pour Felix)
- Quitter l'environnement Equinox : `osgi> close` (pour Felix, Ctrl-c)

- Vous pouvez démarrer ce prompt en lançant Eclipse en mode console. En effet, lorsque vous démarrez Eclipse, il s'agit d'Equinox qui est démarré, et qui charge les bundles (plugins) correspondants à votre installation d'Eclipse :  
**eclipse -console**  
Le prompt **osgi>** s'affiche. Vous pouvez ainsi saisir : **osgi> ss** et voir les bundles qui constituent Eclipse déployés dans le container OSGi Equinox.  
Quitter ce prompt (**stop 0**, pour désactiver le bundle dont l'id est 0, c'est à dire Equinox) et revenez sur celui obtenu en démarrant Equinox/Felix
- Gérer le cycle de vie d'un bundle (l'installer, l'activer, l'arrêter,...) : à faire dans l'ex. 2
  - Installer un bundle : **osgi> install file:<repertoire/fichier.jar>**  
Même chose sur Felix
  - Consulter les bundles installés (vérifier l'identifiant du bundle) : **osgi> ss** (ou **g! lb**)
  - Activer un bundle : **osgi> start <id-du-bundle>** (même chose sur Felix)  
La méthode **start** de la classe « Activator » est invoquée, le cas échéant
  - Mettre à jour un bundle installé/activé (si vous corrigez des erreurs dans votre bundle) : **osgi> update <id-du-bundle>** (même chose sur Felix)  
Si le bundle est installé seulement, ce bundle va être dés-installer puis ré-installé (**uninstall** puis **install**). Si le bundle est actif, ce bundle va être ré-installé et ré-activé (**stop**, **uninstall**, **install** puis **start**).
  - Désactiver un bundle : **osgi> stop <id-du-bundle>** (même chose sur Felix)  
La méthode **stop** de la classe « Activator » est invoquée, le cas échéant
  - Désinstaller un bundle du container : **osgi> uninstall <id-du-bundle>** (même chose sur Felix)
- Consulter les logs dans le sous-répertoire **configuration** créé par Equinox dans le répertoire à partir duquel vous avez lancé le framework (ou **felix-cache**)

Utiliser votre éditeur de texte préféré (pourquoi pas Emacs !!!) pour écrire vos interfaces et classes Java, et le MANIFEST.MF (N.B. : Il faut toujours ajouter une ligne vide à la fin du Manifest)

Pour compiler vos interfaces et classes, il faut utiliser le compilateur du SDK en ligne de commande. Ne pas oublier d'ajouter au CLASSPATH le JAR d'Equinox : **org.eclipse.osgi\_xxx.jar** (ou **felix.jar** si vous utilisez Felix)  
Cela vous permet de compiler vos interfaces et classes qui, rappelons le, dépendent de l'API OSGi (BundleActivator, BundleContext, ...).

Pour composer vos bundles, il faut utiliser l'outil **jar** fourni par le SDK de la façon suivante :  
**jar cvmf META-INF/MANIFEST.MF <nom-du-jar> <fichiers-du-jar>**

Pour afficher sur la console le contenu d'un JAR : **jar tvf <fichier-jar>**

Pour extraire le contenu d'un JAR : **jar xvf <fichier-jar>**

Il vaut mieux travailler sur deux consoles : la première pour la compilation et la construction des Jars et la seconde pour saisir les commandes sur le prompt OSGi.

## **Exercice 2.** Application HelloWorld

Écrire le bundle OSGi (vu en cours) qui affiche « Hello World ! » au moment de son activation et « Goodbye World ! » au moment de sa désactivation. Utiliser une classe Activator (disponible dans le dépôt Github + le Manifest).

Essayer les commandes `install` puis `start`, puis `update`, puis `stop`, puis `uninstall` sur le console du framework (après chaque commande, lancer la commande `ss` sur Equinox ou `lb` sur Felix) et expliquer ce qui se passe.

Mettre en place deux bundles OSGi, un Helloer et un Printer. Le bundle Helloer fournit une méthode `sayHello()` pour dire « Hello World!!! ». Cette méthode est déclarée dans une interface `IHelloer`. Ce bundle exporte le package dans lequel se trouve l'interface et masque le package (`.impl`) dans lequel se trouve la classe qui implémente l'interface. Ce bundle dispose également d'une classe `Activator`, placée dans le package masqué, pour démarrer l'application (elle invoque la méthode `sayHello()` dans `start(...)`). Le bundle Printer fournit une méthode `print(String message)` qui affiche le message dans la sortie standard (`System.out.print(...)`). Cette méthode est déclarée dans une interface `IPrinter` qui se trouve dans un package exporté par le bundle (la classe qui implémente l'interface étant dans un package `.impl` qui n'est pas exporté). Connecter les deux composants de deux façons différentes :

1. une classe `factory` dans le bundle Printer, qui retourne un objet de type `IPrinter`
2. une architecture à services : le bundle Printer est un fournisseur de service `IPrinter` et le bundle Helloer est un consommateur de ce service. Utiliser la méthode manuelle vue en cours et ensuite les Declarative Services.

Pour les Declarative Services, se référer au site Web suivant :

<http://www.vogella.com/tutorials/OSGiServices/article.html>

Modifier le bundle Helloer pour dire « Bonjour à tous !!! » au lieu de « Hello World!!! ».

Faire un `update` du bundle, alors qu'il est actif, et voir ce que cela donne.

C. TIBERMACHINE