Structure d'un plug-in

Un plug-in est un fichier JAR classique contenant, en plus de ses classes Java, deux fichiers manifestes :
- le fichier META-INF/MANIFEST.MF
- le fichier plug-in.xml

Le fichier MANIFEST.MF est exploité par le noyau d'Eclipse pour obtenir des informations sur le plug-in (version, liste des classes visibles, ...) qui serviront notamment à gérer le cycle de vie du plug-in et ses relations avec les autres plug-ins.

Le fichier plugin.xml sert à concrétiser les fonctionnalités d'extensibilité d'Eclipse. Via ce fichier, optionnel, des plug-ins vont déclarer des points d'extension et d'autres se brancher sur ces points d'extension.

Créer un plug-in

Pour simplifier le développement de plug-ins, l'IDE Eclipse (Eclipse SDK) intègre un outillage spécifique : le PDE (Plug-ins Development Environment). Basé sur l'outillage Java, le PDE complète l'environnement de développement Java avec des outils prenant en compte les étapes spécifiques au développement de plug-ins (édition des fichiers manifestes, lancement d'un environnement de test, ...).

Tester un plug-in

Les plug-ins obéissent à un cycle de vie particulier. Notamment, les fichiers manifestes sont analysés pendant la phase de démarrage d'Eclipse. Pour que ce cycle de vie soit correctement respecté lors des tests, le PDE propose de lancer une deuxième instance d'Eclipse à partir de celle servant au développement des plug-ins.

Notion de dépendance

Les plug-ins permettent de modulariser une application. Il est naturellement très fréquent de devoir permettre à un plug-in d'appeler le code contenu dans un autre.

Par défaut chaque plug-in est isolé : il ne peut accéder aux classes des autres plug-ins et les autres plug-ins ne peuvent accéder à ses classes. En modifiant le fichier MANIFEST.MF d'un plug-in, il est possible de rendre accessibles tout ou partie de ses classes (cf . `Export-Package`) Les autres plug-ins qui souhaitent appeler ces classes accessibles devront l'indiquer explicitement dans leur fichier MANIFEST.MF (cf. `Require-Bundle`)

Mise en oeuvre d'une dépendance entre plug-ins

Imaginons que l'on crée un deuxième plug-in que l'on appelle « com.eclipsetotale.tutorial.horloge.texte ».
Et que l'on ajoute une classe nommée HorlogeTexte au plug-in.

Rendre les classes accessibles par d'autres plug-ins

Par défaut, la classe précédemment créée ne peut pas être invoquée par un autre plug-in. Un plug-in doit déclarer explicitement dans le fichier MANIFEST.MF les classes qu'ils souhaitent rendre visibles. La granularité choisie est celle du package, la liste des packages visibles est déclarée via la variable 'Exported-package' du fichier MANIFEST.MF.

En gros, pour ‘exporter’ la classe « com.eclipsetotale.tutorial.horloge.texte.HorlogeTexte », on va rajouter dans le MANIFEST.MF la ligne suivante :
Export-Package: com.eclipsetotale.tutorial.texte

Utiliser les classes à partir d’un autre plug-in

Pour que notre premier plug-in puisse appeler la classe HorlogeTexte du second plug-in, il est nécessaire de déclarer explicitement dans le fichier MANIFEST.MF du premier plug-in qu'il y a une dépendance avec le deuxième plug-in.

En gros, on modifie la ligne suivante du fichier MANIFEST.MF :
Require-Bundle: org.eclipse.ui,
org.eclipse.core.runtime,
com.eclipsetotale.tutorial.horloge.texte