Université de Montpellier
Faculté des Sciences
Département Informatique

Filière : Master Informatique **Intervenant :** Chouki TIBERMACINE

TP 5
Introduction aux plugins Eclipse

Exercice 1. Mini-tutoriel PDE (*Plug-in Development Environment*)

Sur Eclipse :

- Pour créer un nouveau projet de plugin : (ci-dessous, version anglaise d'Eclipse)
  - Aller au menu File -> New -> Project...
  - Choisir Plug-in project puis cliquer sur Next
  - Donner un nom à votre projet (ça sera considéré comme le Bundle-Name et le Bundle-SymbolicName dans le Manifest du bundle)
  - Dans la section « This plug-in is targeted to run with: », sélectionner « Eclipse version : », et laisser la version indiquée par défaut, puis cliquer sur Next
  - L'assistant de création de projet (le wizard) vous propose de créer automatiquement l'activateur du bundle. Décocher cette case (nous n'avons pas besoin d'activateur dans l'exercice ci-dessous)
  - Laisser la case qui indique que votre plugin apporte des contributions à l'interface utilisateur (UI) d'Eclipse, puis cliquer sur Finish
  - Accepter la proposition d'Eclipse d'ouvrir la perspective PDE.

- Pour éditer le contenu du plugin Eclipse :
  - Eclipse ouvre automatiquement au centre de la fenêtre l'éditeur de PDE qui permet d'éditer graphiquement ou textuellement la configuration du plugin (Manifest du bundle + le fichier plugin.xml)
  - Aller dans l'onglet «Dependencies» et cliquer sur « Add... » (org.eclipse.ui), si celle-ci n'apparaît pas déjà dans les dépendances du plugin
  - Aller dans l'onglet « Extensions » et cliquer sur « Add... » pour définir les contributions (les extensions) du plugin
  - Pour réaliser l'exercice suivant (Plugin Helloer), il faudra choisir les points d'extension indiqués dans le cours. Vous pouvez aussi éditer directement plugin.xml dans l'onglet dédié

- Pour démarrer Eclipse en y intégrant votre plugin :
  - Activer le menu contextuel de votre projet de plugin, puis choisir « Run As » → « Eclipse Application »
    - L'instance courante d'Eclipse va démarrer une autre instance d'Eclipse (appelée « Target Runtime ») et les contributions de votre plugin seront intégrées à cette instance (ainsi que toutes les contributions des projets de plugins ouverts dans

la première instance d'Eclipse). Cette « Target Platform » peut être paramétrée dans « Run Configurations... » du menu contextuel « Run As » (pour sélectionner par exemple la liste des bundles/plugins qui seront déployés avec votre plugin)

Exercice 2. Plugin Helloer

Créer un plugin qui contribue au menu principal d'Eclipse en ajoutant une *menuContribution* « Helloer », qui comporte une commande Say Hello. En cliquant sur cette commande le plugin affiche une boite de dialogue avec le message « Hello World »

La configuration et le code de ce plugin (fichiers plugin.xml, MANIFEST.MF et la classe Handler de la commande) sont disponibles dans le dépôt Github

Exercice 3. Plugin de manipulation de texte (code source Java)

Créer un nouveau plugin qui contribue au menu contextuel (popup) activable sur le code source Java (id du point d'extension : popup:org.eclipse.jdt.ui.source.menu) pour ajouter une commande permettant d'insérer une ligne vide avant et après chaque instruction pour « aérer » un peu le code dans une zone de texte sélectionnée par l'utilisateur. Voilà la démarche à suivre :

Pour commencer, il faudra déclarer des dépendances aux plugins suivants. Cela peut se faire dans l'onglet « Dependencies », puis cliquer sur « Add ».

```
org.eclipse.ui
org.eclipse.ui.editors
org.eclipse.jdt.core
org.eclipse.jdt.ui
org.eclipse.jface.text
```

Ensuite, il faudra définir les extensions aux points ci-dessous. Cela peut se faire simplement dans l'onglet « Extensions » : cliquer sur « Add », choisir le point d'extension, le paramétrer (nom, id, ...), activer le menu contextuel (cliquer sur le bouton droit) de l'élément que vous venez de créer pour lui ajouter des éléments enfants (ajouter une commande à une contribution de menu par exemple : New > menuContribution)

Les points d'extension auxquels il faudra contribuer sont :

org.eclipse.ui.menus

locationURI="popup:org.eclipse.jdt.ui.source.menu?after=CleanUp"

org.eclipse.ui.commands

org.eclipse.ui.handlers

Dans la définition de la contribution au menu avec la nouvelle commande, vous pouvez entourer cette commande de 2 séparateurs/lignes (un(e) avant et un(e) après) : activer le menu contextuel de la contribution de menu, puis « New » > « separator » et paramétrer le séparateur.

Dans le handler de la commande associée au menu (classe qui étend AbstractHandler), afin de récupérer le texte sélectionné par l'utilisateur, il faudrait d'abord s'assurer que la commande a été activée depuis un éditeur de code source Java et que la sélection correspond à du texte, en écrivant :

```
public Object execute(ExecutionEvent event) throws ExecutionException {
  String activePartId = HandlerUtil.getActivePartId(event);
  if(activePartId.equals("org.eclipse.jdt.ui.CompilationUnitEditor")) {
    ISelection selection = HandlerUtil.getCurrentSelection(event);
    if(selection instanceof TextSelection) {
      TextSelection texte = (TextSelection)selection;
      MessageDialog.openInformation(null, "Info", texte.getText());
      // ... A compléter par le code expliqué ci-dessous ...
    }
  }
}
```

Ajouter les imports nécessaires et tester votre plugin avec ce code avant d'aller plus loin.

À l'endroit indiqué ci-dessus, remplacer dans le texte récupéré de la sélection chaque retour à la ligne par un double retour à la ligne :

```
String str = texte.getText();
str = str.replace("\n", "\n \n");
```

Réécrire la sélection dans le fichier de code source. Pour cela, il faudrait écrire :

```
IEditorPart part = HandlerUtil.getActiveEditor(event);
if(part instanceof ITextEditor) {
    ITextEditor editor = (ITextEditor)part;
    IDocumentProvider prov = editor.getDocumentProvider();
    IDocument doc = prov.getDocument(editor.getEditorInput());
    try {
        doc.replace(texte.getOffset(), texte.getLength(), str);
    } catch (BadLocationException e) {
        e.printStackTrace();
    }
}
```

Une façon plus propre de faire la même chose consiste à passer par les plugins JDT (*Java Development Tools*) et leur API pour parser le code sélectionné, le modifier (en indentant les instructions bien comme il faut à la fin) et réécrire l'arbre syntaxique dans le document. Mais comme la modification faite sur la sélection est très basique (un simple ajout de retours à la ligne), on peut se contenter de la solution décrite ci-dessus, en laissant JDT pour les modifications sur le code source, qui peuvent introduire des erreurs de lexique/syntaxe.

Exercice 4. Plugins Helloer et Printer

Reprendre le plugin de l'exo 2.

Lui ajouter deux points d'extension :
1. message : pour attendre des contributions d'autres plugins pour apporter un message de bienvenue à afficher en remplacement de « Hello World »
2. printer : pour attendre des contributions d'autres plugins pour apporter des classes implémentant une interface avec la méthode print, et qui peuvent afficher le message de différentes façons

Le modifier pour s'appuyer sur les potentiels plugins qui contribuent avec des

extensions à ces deux points. (Voir les transparents de cours pour voir comment exploiter ces contributions.)
Par défaut, s'il n'y a pas de contributions, le plugin affiche « Hello World » dans la console (System.out.println(…)).

Définir un plugin Message, qui apporte une extension au premier point d'extension : un message (« Bonjour à tous »)

Définir un autre plugin Printer qui contribue avec une classe qui affiche un message en utilisant MessageDialog.openInformation(...).

Démarrer Eclipse avec ces trois plugins et tester la commande « Say Hello ».