École Polytechnique de Montréal

LOG8430

Architecture logicielle et conception avancée

Hiver 2015

Guide du développeur

Remis par :

Matricule	Prénom & Nom
1579757	Christophe Baudinet
1581658	Julien Saad
1576001	Nicolas St-Aubin

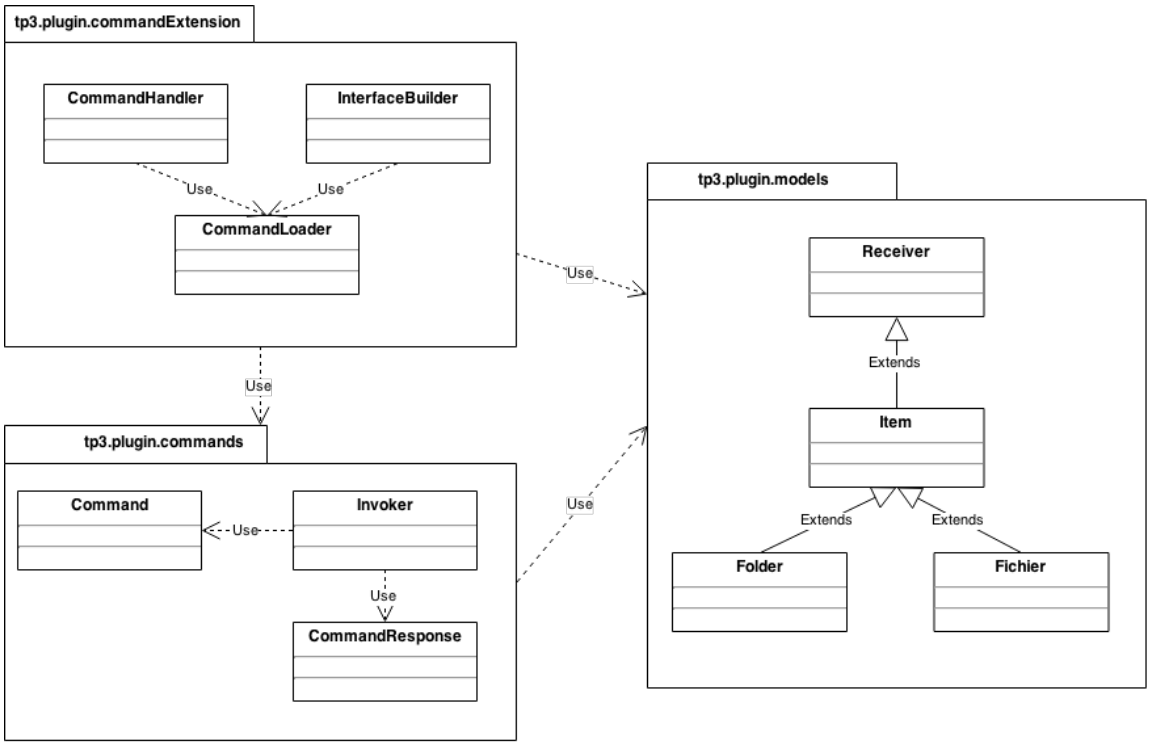
Remis à : Zéphyrin Soh et Yann-Gaël Guéhéneuc

Le 23 mars 2015

GUIDE DU DÉVELOPPEUR

ÉTAPE P1

Étant donné que le client souhaite qu'un nombre inconnu d'utilisateur puissent utiliser le point d'extension et ajouter des commandes par eux-mêmes, nous avons cru bon de conserver le patron de conception *Command* employé dans le cadre du TP1. Pour effectuer ceci, nous avons ajouté un paquetage *tp3.plugin.model* contenant les classes du modèle qui implémentent l'interface Receiver sur lesquels sont effectué les commandes. Ensuite, le paquetage *tp3.plugin.commands* contient l'Invoker et l'interface Command à implémenter. Finalement, le paquetage *tp3.plugin.commandExtension* contient les fichiers liés à la génération de l'interface du plugin pour effectuer l'extension des points d'extension d'Eclipse ainsi que la classe *CommandLoader* servant à charger dynamiquement les extensions. De plus, ce paquetage contient la classe *CommandHandler* qui constitue le *handler* du point d'extension et qui exécute les bonnes commandes selon l'événement appelé dans l'interface. Puisque plusieurs classes différentes font appel à la même liste de commandes du CommandLoader, cette classe est un singleton.



ÉTAPE P2 (P3)

a. *Utilisation* d'un point d'extension

L'utilisation d'un point d'extension permet d'ajouter une fonctionnalité à Eclipse à partir de points d'extension fournis, comme par exemple un bouton à la barre de menu en haut de l'écran.

Les étapes à suivre sont tout d'abord de trouver le point à étendre (à partir de la documentation d'Eclipse ou bien de la documentation d'un plugin déjà inclus dans Eclipse). Ensuite, il suffit de définir le point d'extension dans *plugin.xml*, un fichier généré lors de la création d'un projet servant d'extension. Dans l'exemple suivant, on voit une extension qui fait référence au point d'extension org.eclipse.ui.menus.

```
<plugin>
    <extension
         point="org.eclipse.ui.menus">
      <menuContribution
            allPopups="false"
locationURI="popup:org.eclipse.ui.navigator.ProjectExplorer#PopupMenu?after=additions"
             class="tp3.plugin.commandExtension.InterfaceBuilder">
      </menuContribution>
   </extension>
</plugin>
```

b. *Offre* d'un point d'extension

La démarche pour offrir un point d'extension est similaire à celle pour en utiliser un. En effet, c'est encore une extension qui doit être créée, mais cette fois, celle-ci ne sert que de point d'extension. Il faut donc ajouter des balises contenant les informations pour

```
<plugin>
<extension-point
       *INFORMATIONS SUR LE POINT D'EXTENSION*
 />
</plugin>
```

Une fois le point d'extension défini dans le fichier *plugin.xml,* il est primordial d'effectuer la recherche des extensions de ce point pour ensuite les afficher ou effectuer le travail associé aux dites extensions. Il faut aussi que les extensions à notre point d'extension aient un format

spécifique. Ainsi, un fichier .exsd contient la structure prévue pour les extensions. Finalement, en Java, un segment de code bouclant sur toutes les IConfigurationElement peut permettre de trouver les extensions faisant référence à l'ID du point d'extension créé.

ÉTAPE P4

Tout d'abord, afin de prototyper rapidement, nous avons ajouté dans notre point d'extension le code XML nécessaire pour afficher un menu et des sous-menu. Ceci nous a permis de voir la structure XML requise pour avoir le résultat désiré.

Ensuite, nous avons retiré le code statique contenant les éléments du menu et du sous-menu pour plutôt les générer avec une classe Java qui itère sur toutes les extensions disponibles pour le point d'extension créé.

ÉTAPE P5

Tel que mentionné en P1, en suivant le patron Command, nous avons déterminé l'interface de programmation requis pour le point d'extension. Nous avons conservé le format utilisé dans le TP1, qui contient la méthode *String execute(Receiver receiver);*. Celle-ci retourne une chaîne de caractères contenant le résultat de la méthode execute effectuée à partir d'un objet *Receiver.* L'interface *Receiver* contient les méthodes *getPath()* et *getClassName()*, qui permettent d'effectuer l'exécution de commandes à partir d'objets implémentant l'interface (Fichier et Folder).

ÉTAPE P6

Pour concevoir ce point d'extension, nous avons du adapter notre solution en P4. Tout d'abord, il a fallu ajouter une classe qui permet d'aller récupérer la liste d'extensions au point d'extension que nous avons créé.

```
public void loadCommands(){
        IExtensionRegistry extensionRegistry = Platform.getExtensionRegistry();
        IConfigurationElement[] configurationElements = extensionRegistry
                .getConfigurationElementsFor("tp3.plugin.command");
        for (IConfigurationElement configurationElement : configurationElements) {
            try {
                Command commandImpl = (Command) configurationElement
```

```
                    .createExecutableExtension("class");
            String name = configurationElement.getAttribute("name");
            commands.put(name, commandImpl);
        } catch (CoreException e) {
            e.printStackTrace();
        }
    }
}
```

Tel qu'expliqué en P4, le point d'extension doit avoir un une définition précise dans son fichier *plugin.xml*, ainsi qu'un format de données qu'il peut accepter dans un fichier *.exsd.* Les extensions devront suivre ce format dans leur propre fichier plugin.xml.