TP 4 : Utilisation du PDE

Modèles de greffon

Exercice 1 : Créer un greffon en utilisant le modèle *Hello, World* (ajout d'un menu et d'un item dans ce menu). Lancer Eclipse avec ce greffon pour vérifier la présence de la nouvelle « fonctionnalité ».

Afin de se familiariser avec la création de greffon :

- inspecter le contenu des fichiers `plugin.xml` et `MANIFEST.MF` (et comparer avec ce qui a été entré dans l'assistant de création),
- localiser et modifier le nom du menu (vérifier que les modifications réalisés via le formulaire sont propagées dans `plugin.xml`),
- localiser et modifier le nom de l'action,
- localiser et changer le message affiché,
- déplacer le menu[1] et l'action[2],
- vérifier que le descriptif XML du plugin (`plugin.xml`) correspond à la description du point d'extension utilisé (`org.eclipse.ui.actionSets`).

Exercice 2 : Créer un greffon en utilisant le modèle *Plug-in with a view*. Lancer Eclipse avec ce greffon pour vérifier que cette nouvelle vue est disponible. Localiser et modifier le contenu affiché dans la liste. Faire en sorte que ce contenu soit référencé dans la classe implémentant la vue et créé lorsque la vue est créée.

Greffon de recherche de code source

Google Code™ Search est un service[3] fourni par *Google* qui permet d'effectuer des recherches dans des listings de code source disponibles publiquement (archives sur des pages web, dépôts CVS ou Subversion de projets libres...) :

`http://www.google.com/codesearch`

Bibliothèques client et échantillon

La première étape consiste à récupérer les bibliothèques client fournies par *Google* pour accéder à leurs services ainsi que leurs dépendances et à lancer l'un des échantillons.

Exercice 3 : Les bibliothèques suivantes sont nécessaires :

- *Google Data APIs* :
  `http://code.google.com/apis/gdata/clientlibs.html`
  Dans *Java client library / Downloads page*, récupérer et désarchiver `gdata-src.java-1.16.2.zip` et `gdata-samples-1.16.2.zip` quelque part dans votre compte,
- *Sun JavaMail API* :
  `http://java.sun.com/products/javamail/downloads/index.html`
  JavaMail 1.4.1 / Download, récupérer et désarchiver `javamail-1_4_1.zip` quelque part dans votre compte,
- *Sun JavaBeans Activation Framework* :
  `http://java.sun.com/products/javabeans/jaf/downloads/index.html`
  Download, récupérer et désarchiver `jaf-1_1_1.zip` quelque part dans votre compte,
- *Java Servlet* :
  `http://java.sun.com/products/servlet/download.html`
  Dans *SPECIFICATIONS / Java Servlet / 2.3 Final release / Download class files*, récupérer *servlet-2_3-fcs-classfiles.zip*.

[1] Essayer `path="help/additions"`, par exemple.
[2] Essayer `menubarPath="help/helpEnd"`, par exemple.
[3] Pour plus de détails, consulter la FAQ : `http://www.google.com/intl/fr/help/faq_codesearch.html`.

En fait, les seules bibliothèques vraiment indispensables sont les premières, le reste n'est utile que pour des parties des bibliothèques client que nous n'utiliserons pas ou bien pour recompiler les bibliothèques client.

Remarquer dans le répertoire `gdata/java` la présence des fichiers `build-src.xml` et `build-samples.xml` ; il s'agit de fichiers de *build* pour l'outil *Ant*. Pour (re)compiler[4], il suffit d'entrer la commande :

`ant -f <nom du fichier de build>`

Dans le sous-répertoire `lib` figurent plusieurs bibliothèques dont les seules qui nous intéressent pour la suite sont :

- `gdata-codesearch-1.0.jar`,
- `gdata-client-1.0.jar`, et
- `gdata-core-1.0.jar`.

Parmi les échantillons fournis (dans le sous-répertoire `sample`), se trouve un exemple de requête au service *Google Code™ Search*. Lancer-le manuellement en invoquant la machine virtuelle Java (sans oublier de rajouter les bonnes bibliothèques au `CLASSPATH`). Essayer différentes requêtes et examiner la sortie du programme.

Conversion en projet Eclipse

Exercice 4 : À partir de cet échantillon, créer un projet Java *CodeSearch* dans Eclipse (dans un répertoire à part dans votre espace de travail), et en profiter pour supprimer la dépendance à la classe `SimpleCommandLineParser` (utiliser directement les arguments fournis sur la ligne de commande s'il y en a suffisamment). Supprimer aussi le message d'aide. Ne pas oublier de rajouter les bibliothèques client dont dépend l'échantillon au *Build Path* de votre projet[5]. Vérifier que l'échantillon fonctionne encore depuis Eclipse.

Exercice 5 : Lire le code de l'échantillon et essayer de le comprendre. En particulier, identifier les étapes suivantes :

- génération d'une adresse de requête,
- envoi de la requête,
- affichage des résultats.

Enlever tout le code générant du XML (cette partie ne sera pas utile) pour ne conserver que l'affichage (textuel) des résultats (nom du fichier, auteur, license, projet, lignes correspondantes...) de la requête.

Création d'un greffon pour Eclipse

Nous avons maintenant le code client suffisant pour construire un greffon pour Eclipse.

Le greffon que nous nous proposons d'écrire fournit dans le menu *Help* une action *Code Search...* qui ouvre une boîte de dialogue. Cette boîte de dialogue demande la requête à effectuer sur *Google Code™ Search* et en cas de confirmation ouvre une nouvelle vue *Code Search*. Cette vue *Code Search* affiche la liste des résultats de la requête, et en double-cliquant sur un résultat, l'utilisateur peut ouvrir le résultat (le code source) dans son navigateur.

On pourra rajouter l'option `-consoleLog` dans la configuration de lancement du greffon Eclipse. Cette option permet de récupérer toutes les erreurs d'Eclipse sur la console.

Exercice 6 : Créer un nouveau projet *CodeSearchPlugIn* dans votre espace de travail en utilisant le modèle de greffon pour `org.eclipse.ui.actionSets`. Copier les bibliothèques client requises dans un sous-répertoire `lib` du projet et les rajouter au *Runtime Classpath* du greffon. Supprimer le menu du modèle créé pour ne conserver que l'action et placer cette action dans le menu d'aide. Modifier enfin le code du modèle pour afficher non pas une simple boîte de message

[4] Il faudra peut-être éditer les fichiers `build-src/build.properties` et/ou `build-samples/build.properties` pour indiquer où se trouvent `mail.jar` de *JavaMail*, `actionvation.jar` de *JAF* et `servler-api-2.4.jar` de *Java Servlet*.

[5] *Build Path.../ Add External Archives...*

mais une boîte de dialogue avec entrée et récupérer le texte entré dans la boîte[6]. Vérifier le fonctionnement du greffon.

Exercice 7 : Rajouter une extension de type `org.eclipse.ui.views` en choisissant l'un des modèles (désactiver tout sauf la gestion du double-clic[7]).

Faire en sorte que la confirmation dans la boîte de dialogue fasse apparaître la vue à l'aide du code suivant :

```
1 PlatformUI.getWorkbench().
2                 getActiveWorkbenchWindow().
3                     getActivePage().
4                         showView("<identifiant de la vue>");
```

Vérifier le fonctionnement du greffon.

Exercice 8 : Copier le code du client pour *Google Code Search* dans le projet courant et le modifier pour que la méthode de requête mette à jour deux `ArrayList<String>` passés en paramètre). Chaque entrée du premier tableau va correspondre à un résultat affiché dans la vue et la même entrée dans le deuxième tableau contiendra le lien hypertexte vers le résultat. On pourra présenter chacun des résultats sur plusieurs lignes avec indentation des différents champs et des lignes correspondant à la requête.

La confirmation dans la boîte d'entrée de la requête doit non seulement afficher la vue mais aussi mettre à jour son contenu. Rajouter le code nécessaire dans la classe implémentant la vue[8] (penser à invoquer la méthode `refresh()` du `TableViewer` pour rafraîchir les données de la table). Vérifier le fonctionnement du greffon.

Exercice 9 : Le greffon est maintenant fonctionnel. Rajouter :

- la possibilité de naviguer dans plusieurs pages de résultats (utiliser à nouveau l'assistant de création de vue pour ajouter des boutons de navigations à la vue),
- l'ouverture du navigateur sur l'un des résultats avec le code suivant :

```
1 IWebBrowser browser =
2     PlatformUI.getWorkbench().
3                     getBrowserSupport().
4                         createBrowser("<identifiant>");
5 browser.openURL(new URL("<URL>"));
```

[6] Utiliser *Search / Java...* avec le bon mot-clé et se laisser guider par *Content Assist* pour trouver les méthodes à invoquer pour obtenir le texte entré et si l'utilisateur a bien confirmé l'action.

[7] Attention, le code sera incomplet pour la gestion du double-clic : consulter le code généré au début du TP pour le compléter.

[8] Pour récupérer une instance de cette classe depuis le code de la boîte d'entrée, utiliser le résultat renvoyé par la méthode `showView()` de la question précédente