Atelier LOOPS - Eclipse pour l'intégration d'outils de développement

Stéphane Lopes(stephane.lopes@prism.uvsq.fr)

23 mai 2013

L'objectif de cet atelier est de montrer comment, à partir d'Eclipse, il est possible de créer aisément un projet intégrant divers outils de développement. Un exemple simple de projet Java sera utilisé pour cela. Les aspects étudiés incluent les outils automatiques d'audit de code (Checkstyle, FindBugs), le build (Maven), la gestion de versions (Git) ainsi que les tests unitaires (JUnit). Cet atelier se déroulera dans l'environnement suivant : JDK 1.6 ou 1.7, Eclipse intégrant les plugins checkstyle (eclipse-cs), findBugs, maven (m2e), git (EGit) et JUnit (en standard + MoreUnit).

1 Paramétrage initial (Eclipse)

1. Ajoutez un dictionnaire français (*General* > *Editors* > *Text Editors* > *Spelling* en ISO-8859-1) ;
2. Modifiez l'encodage des caractères en *UTF-8* et les fins de ligne en *Unix* (*General* > *Worspace*) ;
3. Vérifiez que l'option *Build automatically* est cochée (*General* > *Workspace*) ; Faites de même avec l'option *Report problem as you type* (*Java* > *Editor*) ;
4. Vérifiez les versions du JDK installées sur la machine et sélectionnez la plus récente (*Java* > *Installed JRE*) ;
5. Attachez les sources du JDK à la bibliothèque `rt.jar` (*Source Attachment...* dans les propriétés du JRE) ;
6. Définissez les chemins des sources (`src/main/java`) et du bytecode (`target/classes`) pour les projets (*Java* > *Build Path*) ;
7. Vérifiez la version du compilateur utilisé et sélectionez la plus récente (*Java* > *Compiler*) ;
8. Activez la mise en forme du code lors des sauvegardes (*Java* > *Editor* > *Save Actions*).

2 Création du projet (Maven)

Le projet permet de représenter un système de fichiers (FS) très simplifié. Un fichier est représenté par son nom et sa taille. Un répertoire est défini par son nom et peut contenir des fichiers et/ou des répertoires. On veut pouvoir calculer la taille totale d'un élément du FS.

1. Définissez la configuration globale de Maven (*Maven* > *Installations* > *Global settings*) ;

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository>C:/loops/_m2/repository</localRepository> <interactiveMode/>
</settings>
```

2. Vérifiez que le *local repository* est bien paramétré dans Eclipse (*Maven* > *User settings*) ;
3. Créez un nouveau projet Maven en sélectionnant l'archétype `maven-archetype-quickstart` et en définissant le groupe à `loops.atelier` et l'artéfact à `fslib` ;
4. Assurez-vous que le projet respecte les conventions Maven pour l'arborescence des répertoires ;
5. Assurez-vous que l'environnement Java du projet est un JDK 1.7 ;

6. Importez l'archive `fslib-squel.zip` dans le répertoire des sources du projet ;
7. En consultant la documentation du plugin Maven Compiler Plugin, éditez le `pom.xml` pour utiliser une version 1.7 des sources et des classes compilées ;
8. Supprimer la classe `TestApp` du projet.

3 Explorer le projet (Eclipse)

1. En utilisant la vue *Package Explorer*, naviguez dans les sources pour ouvrir le fichier `File.java` dans un éditeur ;
2. Dans la vue *Outline*, utilisez la barre d'icones pour masquer/voir les différents éléments ; Quel impact la vue *Outline* a-t-elle sur l'éditeur ?
3. Dans l'éditeur, faites apparaître la vue *Quick outline* (*Ctrl+o*) puis les membres hérités (*Ctrl+o*) et sélectionnez la méthode `java.lang.Object.toString()` ;
4. Dans l'éditeur, sur le fichier `File.java`, placez le curseur sur le type `File` et faites apparaître la *hiérarchie des types* (*Ctrl+T* pour *Quick Type Hierarchy* ou *F4* pour la vue *Type Hierarchy*).

4 Visualiser les erreurs du projet (Eclipse)

1. Comment les erreurs dans le projet sont-elles signalées (vue et moyen) ?
2. Placez le pointeur de la souris sur différents symboles d'erreur dans l'éditeur ;
3. Placez le curseur sur l'erreur concernant le type `LogFactory`, faites apparaître la vue *Quick Fix* (*Ctrl+1*) et sélectionnez *Fix project setup...*.

5 Gérer les dépendances (Maven)

Dans le fichier `pom.xml` :

1. Ajoutez la dépendance vers la librairie de journalisation *Apache Commons Logging* (groupId = commons-logging, artifactId = commons-logging, version = 1.1.2) ;
2. Quel impact cette action a-t'elle sur la gestion des dépendances au niveau d'Eclipse ?
3. Compiler votre projet avec Maven (*Run As...*) pour générer un jar (*mvn package*).

6 Exécuter le projet (Eclipse)

1. Dans la méthode *App.main*, écrivez quelques instructions utilisant *File* et *Directory* et réalisant des affichages sur la console ;
2. Lancez le programme comme une application Java ;
3. Éditez la configuration d'exécution pour définir les propriétés systèmes suivantes (*VM arguments*) :

```
-Dorg.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog
-Dorg.apache.commons.logging.simplelog.defaultlog=trace
```

4. Lancez à nouveau le programme ; Que remarquez-vous ?

7 Éditez le code (Eclipse)

Dans la méthode *Directory.getSize()* :

1. Utilisez *Content Assist* pour saisir la ligne suivante (après *log.*, tapez *Ctrl+espace*, puis *tr*, ...) :

```
log.trace("Directory : " + getName() + " (" + getSize() + ")");
```

2. Saisissez le corps suivant en utilisant les *templates de code* (après `for`, tapez *Ctrl+espace*, puis sélectionner *foreach*, puis *Tab* entre les champs) :

```
int size = 0;
for (Item i : catalog) {
  size += i.getSize();
}
return size();
```

3. Organisez les importations de module :
   - dans la vue *Outline*, rendez visible les déclarations d'*import* (*View Menu* > *Filters...*),
   - dans la même vue, supprimez les déclarations d'*import* du fichier Java,
   - enfin, dans l'éditeur, ajoutez automatiquement les déclarations d'*import* (*Source* > *Organize Imports* ou *Shift+Ctrl+o*).
4. Apportez les modifications suivantes au projet (*Refactoring*) :
   - changez le nom du package *loops.atelier.fslib* en *loops.atelier.fslibrary*.

8 Générer le site du projet (Maven)

1. Modifiez le `pom.xml` pour fabriquer les rapports pour `Javadoc`, `checkstyle`, et `findbugs` ;
2. Générez le site du projet (*mvn site*).

9 Générer une archive du projet (Maven)

1. En utilisant le plugin `assembly`, générez une archive du projet contenant ses dépendances.

10 Initialiser la gestion de versions (Git)

1. Configurer EGit avec votre nom (*user.name*) et votre email (*user.email*) ;
2. Partagez le projet en créant un dépot local nommé `fslib.git` dans le répertoire de votre choix ;
3. Ajoutez le répertoire `target` aux fichiers ignorés par *git* (fichier `.gitignore`) ;
4. Dans la vue *Git Staging*, ajoutez le répertoire `src` aux fichiers suivis puis validez ;
5. Créez une étiquette pour marquer la version 1.0 du projet ;
6. Ouvrez la vue *Git Repositories* pour examiner la structure du dépôt.

11 Tester le module (JUnit, Git)

Pour chaque test, vous respecterez la démarche suivante :

1. Créez une branche nommée `testXXX` et commutez vers cette dernière ;
2. Ajoutez la méthode de test au projet, exécutez les tests (*CTRL+R*) et validez ;
3. Modifiez si nécessaire la méthode testée (*CTRL+J* pour passer du test au code), exécutez les tests et validez ;
4. Lorsque le test passe, commutez la copie de travail vers la ligne principale ;
5. Fusionnez la copie de travail avec la branche `testXXX` puis valider les changements ;
6. Supprimez la branche `testXXX`.

Réalisez la configuration et implémentez les tests suivants :

1. Ajoutez la dépendance vers la librairie JUnit 4 (groupId = junit, artifactId = junit, version = 4.11, scope = test) ;
2. Vérifiez que l'option *Decorate Classes with Test Case* est cochée (*Window* > *Preferences...* > *General* > *Appearance* > *Label Decorations*) ;

3. Vérifiez la configuration de *MoreUnit* (*Test source folder* = *src/test/java*, *Test type* = *JUnit 4*) ;
4. Créez la classe de test *FileTest* avec les importations adéquats (*Ctrl+J*) ;

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
```

5. implémenter le test *testGetSizeOK* (*New JUnit Test Case*) qui vérifie le calcul de taille pour un fichier de taille 10,

```
@Test
public void testGetSizeOK() {
  File f = new File("file.pdf", 10);
  assertEquals(10, f.getSize());
}
```

6. implémenter le test *testFileWithNegSize* qui vérifie que le constructeur de `File` lève une exception (*IllegalArgumentException*) si la taille du fichier est négative,

```
@Test(expected=IllegalArgumentException.class)
public void testFileWithNegSize() {
  File f = new File("fichier.pdf", -10);
}
```

7. Faites de même pour les tests suivants :
   (a) différentes situations pour la taille d'un répertoire,
   (b) on ne peut pas ajouter une référence null à un répertoire,
   (c) on ne peut pas ajouter dans un répertoire un élément de même nom qu'un élément déjà dans le répertoire,
   (d) un répertoire ne peut pas être ajouté à lui-même,
   (e) un répertoire ne peut pas être un sous-répertoire de lui-même (même indirectement).
8. Modifiez le `pom.xml` pour que le rapport des tests unitaires apparaisse sur le site du projet.

12 Partager le projet sur Github (Git)

1. Créez un compte sur Github ;
2. Créez un dépôt distant (DD) pour le projet ;
3. Définissez le lien entre le dépôt local (DL) et le DD (*git remote add...*) ;
4. Envoyer la branche principale (*master*) sur le DD (*git push...*) ;
5. Consulter les *commits* sur le site Github.