Environnements de développement

Examen de rattrapage

18 juin 2008

Durée : 3 heures. Tous les documents sont autorisés.

Avant de commencer : s'assurer que la machine virtuelle utilisée est celle de Sun et non la version GNU (dans *Window / Preferences.../ Java / Installed JREs*), et qu'Eclipse est configuré pour compiler du Java 5.0 (dans *Window / Preferences.../ Java / Compiler / JDK Compliance*).

Modalités de remise des copies : en plus de remettre une copie papier, envoyer par e-mail à l'adresse labatut@di.ens.fr les projets des deux problèmes sous forme d'archive compressée `zip` (sélectionner le projet ouvert à archiver dans le *Package Explorer* et choisir *File / Export.../ General / Archive File*).

Pour chaque question (pratique), décrire de manière succincte (quelques lignes *maximum*) comment Eclipse a été utilisé pour réaliser l'opération demandée.

Il faudra attacher une attention toute particulière à la présentation du code : indentation et (éventuels) commentaires.

1 Introduction

Exercice 1 : Indiquer quels sont les outils généralement inclus ou intégrés dans un environnement de développement (en donner quatre au minimum). Citer d'autres outils (ou bibliothèques) classiques en développement Java qui sont intégrés dans l'environnement Eclipse en particulier (en donner trois au minimum). Justifier l'utilisation d'un environnement de développement par rapport au développement classique à l'aide de plusieurs outils spécialisés distincts.

2 Utilisation du JDT

L'objectif de ce problème est de définir deux classes qui collectent des informations concernant les classes contenues dans les fichiers sources d'un utilisateur (nom complet, nombre de caractères, de champs et de méthodes) et qui retrouvent une classe satisfaisant un critère.

2.1 Projet

Exercice 2 : Créer un projet Java `ClassInfo` et rajouter un *package* `ci` à ce projet. Rajouter à ce *package* une classe `ClassInfo` avec comme champs privés : une chaîne de caractères `fullName` et trois entiers `numberOfCharacters`, `numberOfFields` et `numberOfMethods`.

Exercice 3 : Rajouter à la classe `ClassInfo` :

- un constructeur (à 5 paramètres : `projectName`, `name`, `numberOfCharacters`, `numberOfFields` et `numberOfMethods`), qui initialise le champs `fullName` à `projectName+"/"+name` [1], puis les trois autres champs de `ClassInfo` à l'aide des trois paramètres suivants,
- des accesseurs (mais pas d'affecteurs) pour chacun de ses champs.

Exercice 4 : Rajouter au *package* `ci` une classe `ClassInfoList` qui hérite de la classe `ArrayList` de `java.util` [2]. La classe `ArrayList` implémente un tableau dynamique avec l'interface `Collection` et dispose donc déjà de méthodes pour ajouter des éléments, en supprimer, etc.... On souhaite empêcher l'ajout d'information sur une classe déjà présente dans la liste [3]. Dans ce but, surcharger la méthode d'insertion `boolean add(ClassInfo ci)` dans la classe `ClassInfoList` :

1. afin d'identifier de manière unique les classes homonymes issus de projets différentes.
2. ou plutôt `ArrayList<ClassInfo>`...
3. la bonne façon de faire serait d'utiliser une classe implémentant l'interface `Set`...

la nouvelle méthode devra renvoyer `true` mais lancer une exception `IllegalArgumentException` si un élément avec le même champ `fullName` existait déjà dans la liste.

On souhaite maintenant récupérer le l'élément de la liste précédente réalisant le maximum ou le mininmum pour l'un des trois champs entiers de la classe `ClassInfo`.

Exemple : si l'on extrait l'élément de la liste à 3 éléments suivante d'après le champs `numberOfFields` :

```
{ { "Projet1/package.Classe1", 47, 3, 12},
  { "Projet1/package.Classe2", 31, 4, 11},
  { "Projet2/package.Classe1", 62, 2, 23}}
```

on doit obtenir l'élément suivant :

```
  { "Projet1/package.Classe2", 31, 4, 11}
```

Exercice 5 : Comment trouver les méthodes de recherche de maximum ou minimum disponibles dans le JRE depuis Eclipse ? Existe-t-il une méthode de recherche de maximum ou de minimum dans un tableau dans les packages standards (`java.*`) ?

On trouve dans la classe `java.util.Collections` une méthode statique `max()` qui extraît le maximum d'une `Collection` d'après une relation d'ordre indiquée par un objet d'une classe implémentant l'interface `java.util.Comparator`.

Exercice 6 : Implémenter dans la classe `ClassInfoList`, les méthodes *publiques* suivantes :
- `ClassInfo minByNumberOfCharacters()`,
- `ClassInfo maxByNumberOfCharacters()`,
- `ClassInfo minByNumberOfFields()`,
- `ClassInfo maxByNumberOfFields()`,
- `ClassInfo minByNumberOfMethods()` et
- `ClassInfo maxByNumberOfMethods()`

qui extraient de la liste d'information sur les classes, un élément dont le champ `numberOfCharacters`, `numberOfFields` ou `numberOfMethods` est minimum ou maximum.

Remarque : Comme les méthodes `Collections.min()` et `Collections.max()` peuvent utiliser un objet de type `java.util.Comparator` pour trier les objets, il faudra créer une classe implémentant cette interface pour les trois critères de sélection (par exemple, en se contentant d'invoquer la méthode `compareTo()` du type `java.lang.Integer`).

2.2 Tests unitaires

Exercice 7 : Ajouter la bibliothèque JUnit 3.8 au chemin d'accès aux classes du projet.

Exercice 8 : Rajouter un *package* `ci.test` au projet et dans ce *package*, ajouter une classe de tests unitaires `ClassInfoListTest` qui définit des méthodes de test pour chacun des six méthodes de sélection précédentes.

Exercice 9 : Rajouter à la classe de tests unitaires `ClassInfoListTest` une méthode de test vérifiant que le comportement de la méthode `boolean add(ClassInfo ci)` est bien conforme à ses spécifications.

Exercice 10 : Comment vérifier que la classe `ClassInfoList` passe les tests unitaires définis ?

2.3 Débogage

Exercice 11 : À l'aide du débogueur, inspecter à l'exécution le contenu d'un objet de la classe `ClassCodeInfoList` et identifier les différents champs de la classe `java.util.ArrayList` qui implémente un tableau dynamique (bien détailler la procédure). Quel est la taille initiale (avant ajout d'élément) du tableau interne à cette classe ? Rajouter 15 éléments à ce tableau et

(toujours en utilisant le débogueur) trouver quand le tableau interne est redimensionné et quelle est sa nouvelle taille. Vider le tableau à l'aider de la méthode `clear()`. Quel est la nouvelle taille du tableau interne ?

3 Utilisation du PDE

L'objectif de ce deuxième problème est de réutiliser les classes précédentes afin d'implémenter un *plug-in* Eclipse fournissant un nouveau bouton dans un menu séparé et affichant un message après activation du bouton : le message informera l'utilisateur du nombre de classes contenues dans les projets ouverts, et indiquera le nom des six classes avec le nombre minimum ou maximum de caractères, de champs ou de méthodes.

Exercice 12 : Créer un projet de *plug-in* intitulé `ClassInfoPlugIn` dans l'espace de travail. Quel est le *template* de *plug-in* à utiliser ? Quel est le nom exact du ou des points d'extension correspondants ?

Exercice 13 : Changer le nom du bouton en "Information about classes..." et le nom du menu en "Information".

Rajouter les *plug-ins* `org.eclipse.jdt.core` et `org.eclipse.core.resources` aux dépendances du *plug-in* (dans l'onglet *Dependencies* du descriptif), puis utiliser l'option *Update the classpath and the compiler compliance settings* dans l'onglet *Overview* pour mettre à jour le chemin d'accès aux classes.

Le morceau de code suivant génére un objet de type `ClassInfoList` d'après les fichiers sources Java présents dans les projets ouverts de l'espace de travail :

```
1  public ClassInfo createCI(IProject project, IType type) throws JavaModelException {
2      return new ClassInfo(project.getName(),
3                           type.getFullyQualifiedName(),
4                           type.getSourceRange().getLength(),
5                           type.getFields().length,
6                           type.getMethods().length);
7  }
8
9  public void updateCIList(IProject project, IJavaElement element, ClassInfoList list) {
10     try {
11         if (element instanceof IJavaProject) {
12             IPackageFragmentRoot[] roots = ((IJavaProject) element).getPackageFragmentRoots();
13             for (int i = 0; i < roots.length; i++)
14                 if (!roots[i].isArchive())
15                     updateCIList(project, roots[i], list);
16         } else if (element instanceof IPackageFragmentRoot
17                 || element instanceof IPackageFragment) {
18             IJavaElement[] children = ((IParent) element).getChildren();
19             for (int i = 0; i < children.length; i++)
20                 updateCIList(project, children[i], list);
21         } else if (element instanceof ICompilationUnit) {
22             ICompilationUnit unit = (ICompilationUnit) element;
23             IType[] types = unit.getTypes();
24             for (int i = 0; i < types.length; i++)
25                 if (types[i].isClass())
26                     list.add(createCI(project, types[i]));
27         }
28     } catch (JavaModelException e) {
29     }
30 }
31
32 public ClassInfoList createCIList() {
33     ClassInfoList list = new ClassInfoList();
34     IProject[] projects = ResourcesPlugin.getWorkspace().getRoot().getProjects();
35     for (int i = 0; i < projects.length; i++) {
36         if (projects[i].isOpen())
37             updateCIList(projects[i], JavaCore.create(projects[i]), list);
38     }
39     return list;
40 }
```

Ce code est disponible à l'adresse :

`http://www.di.ens.fr/~labatut/ED6/rattrapage/ClassInfoPlugIn.java`

Exercice 14 : Intégrer ce morceau de code ainsi que les deux classes du premier problème au *plug-in* pour le rendre fonctionnel. Comment rajouter (semi-)automatiquement les directives `import` adéquates pour supprimer les erreurs signalées dans le morceau de code fourni ? [4] Où modifier le code pour remplacer ce qui est affiché par la boîte de dialogue ?

4. Le seul `import` ambigu `org.eclipse.jdt.core.ICompilationUnit`.