Projet de Java

Ingénieur 2000 – Informatique Réseau – 1ière année

Rémi Forax, Philippe Finkel
(forax@univ-mlv.fr, pfinkel@cantor.fr)

Architecture :

L'architecture est contrainte par un ensemble d'interfaces à implanter.
De manière générale, obfusquer un JAR consiste en 3 étapes :

1. lire le JAR et stocker les classes dans le **ClassInfoManager** en utilisant un **JarReader**
2. Appeler la méthode obfuscate() de l'objet **Obfuscator** avec une politique d'embrouillage (**ScramblerPolicy**) pour effectuer l'obfuscation sur les classes en mémoire contenues dans le **ClassInfoManager**
3. écrire le JAR à partir des classes du **ClassInfoManager** en utilisant un **JarWriter**

Les interfaces sont à télécharger ici http://www.forax.org/ens/java/ir04-05/darkproject.zip.

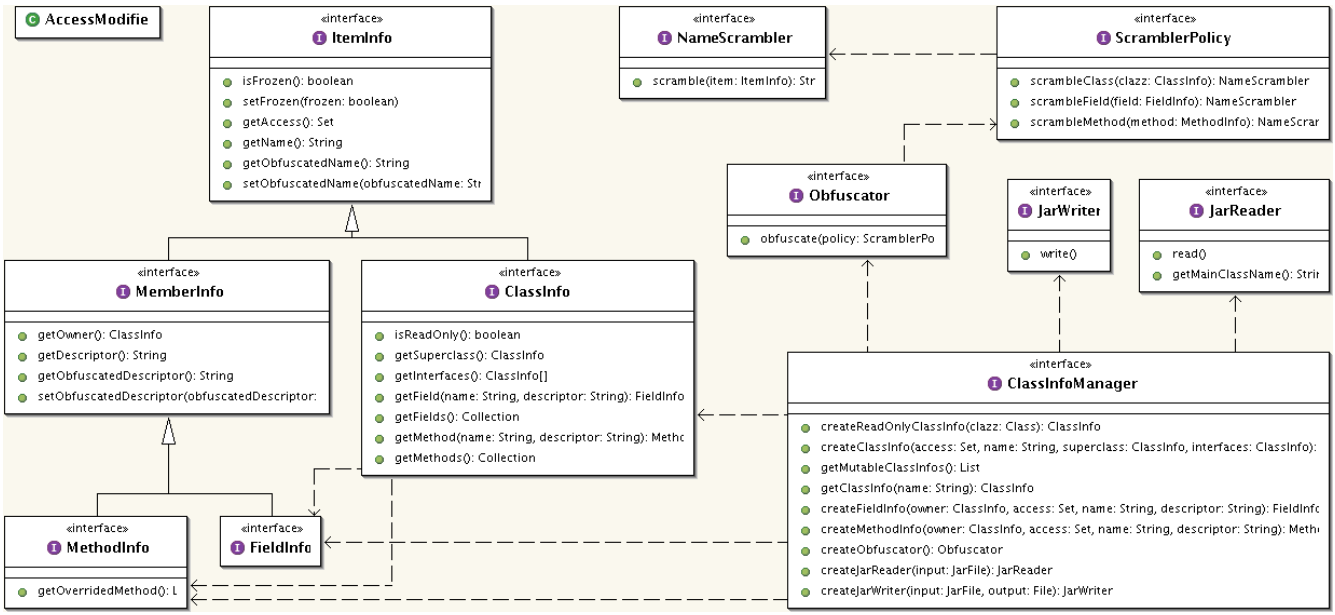
ClassInfoManager :

Gestionnaire des classes contenues dans le JAR, lors de la lecture du JAR, l'ensemble des classes contenues dans celui-ci sont stockées dans cet objet.
En plus des classes du JAR, pour pouvoir effectuer l'algorithme d'obfuscation
le gestionnaire devra contenir des classes qui ne n'appartiennent pas au JAR initial comme par exemple **java.lang.Object**. Ces classes non modifiables sont nécessaire pour savoir si une méthode est *redéfinie* ou non.
Quoi qu'il arraive, même après renommage, les classes sont toujours accessible en utilisant leurs nom avant renommage pour permettre de garder la cohérence.
De plus, le **ClassInfoManager** joue un rôle centrale dans l'application, toutes les objets nécessaires sont crées en utilisant ses méthodes create*

ClassInfo, **FieldInfo** et **MethodInfo** :

ClassInfo décrit les informations liées à une classe. Une classe possède une super-classe, des interfaces (aussi des **ClassInfo**) ainsi que des champs (**FieldInfo**) et des méthodes (**MethodInfo**). Les champs et les méthodes possède une interface commune (**MemberInfo**)

qui elle-même possède une interface commune avec **ClassInfo** (**ItemInfo**).
Le descripteur des champs correspond au type encodé suivant le format du byte-code.
De même, le descripteur des méthodes correspond à la signature de la méthode encodé suivant le format du byte-code. La classe **Util** permet de gérer se codage.
L'interface MethodInfo possède une méthode **getOverridedMethod()** qui indique les méthodes que la méthode courante redéfinie.



JarReader et **JarWriter** :

Ils permettent respectivement de lire et d'écrire un JAR.
Notons que **JarReader** possède aussi une méthode pour déterminer qu'elle la classe principale d'un JAR exécutable.

Obfuscator, **ScramblerPolicy** et **NameScrambler** :

Obfuscator correspond à l'algorithme d'obfuscation proprement dit. Il consiste à parcourir l'ensemble des classes modifiables du **ClassInfoManager** et pour chaque classe, renommer celle-ci ansi que l'ensemble des champs et méthodes la consistuant.
Il y a quelques restrictions :

- en mode application, il ne faut pas renommer la classe principale ni la méthode main de celle-ci.
- en mode bibliothèque seule les méthodes privées ou de paquetage doivent être renommé
- en aucun cas, les méthode redéfinissant une méthode d'une classe ou d'une interface qui n'est pas dans le JAR ne doit être renommée sinon le polymorphisme ne pourra

s'effectuer.

ScramblerPolicy permet de choisir le mode de fonctionnemeent en indiquant si le renommage doit se faire pour une classe, une méthode ou un champs particulier et si il y a renommage quel nouveau nom atttribuer. Le nouveau nom est attribué par l'intermédiaire de l'interface **NameScrambler**.

Détail du second rendu :

Il devra être rendu pour le 27 fevrier avant minuit par mail aux deux enseignants susnommés, dont le sujet sera Projet Darkproject : suivi des noms des membres du binome. En fichier attaché, une archive zip dont le nom est projet_ suivi des noms des membres du binome séparés par des soulignés (_).

Voici le nom des répertoires et fichiers qui doivent être contenus dans l'archive zip.

1. un fichier **readme.txt** indiquant comment compiler et exécuter le programme, où se trouve la doc, *etc.*
2. un fichier Ant **build.xml** permettant de compiler les sources du programme, et de créer le jar exécutable **darkproject.jar**.
3. un répertoire **src** contenant l'ensemble des sources (.java) sous forme de plusieurs paquetages. Les interfaces ne doivent pas être modifiées.
4. un répertoire **classes** contenant l'ensemble des classes (.class) correspondant aux sources sous forme de plusieurs paquetages.
5. un répertoire **lib** contenant l'ensemble des JARs correspondant aux bibliothèques externes utilisées.
6. un répertoire **bin** contenant deux fichiers, **darkproject.sh**, script shell démarrant le logiciel sous linux (en fait unix) et **darkproject.bat** démarrant le logiciel sous Windows.
7. un répertoire **docs** contenant deux documents au format PDF :
    1. La documentation utilisateur user.pdf contenant en plus des informations classiques (comment compiler, exécuter, etc.) une description de l'application, et comment l'utiliser
    2. La documentation développeur dev.pdf contenant :
       Une description détaillé et au niveau de chaque classe que vous avez implantée. (évitez les copier/coller de la javadoc)
       Une liste des bugs connus (s'il y en a) détaillant le scénario permettant de générer le bug ainsi que la raison pour laquelle celui-ci se produit.

    En plus des deux documents, le répertoire **docs** devra contenir un sous-répertoire **api** contenant la documentation complète du logiciel au format javadoc.