JAV TP5
Introspection et réflexivité

Durée: 4h
Titre: Réflexivité des classes et objets
Objectif: Manipuler la classe Class, la classe Object, java.lang.reflect.*
Environnement: Java 1.4+

1. Introduction

Pour votre anniversaire, votre belle-soeur vous a offert une très belle instance de `java.util.Vector`, peuplée d'un certain nombre d'éléments. Merci belle-soeur !

C'est très gentil de sa part, seulement elle est un peu tête-en-l'air : elle a oublié de vous dire ce qu'il y avait dans le `Vector`. Pire, elle a perdu le code source et la javadoc du programme qui l'a généré. Et bien sûr, vous n'avez même pas le ticket de caisse pour vous faire rembourser.

Qu'allez-vous donc faire de ce `Vector` ? Vous pouvez récupérer le contenu sous forme de `java.lang.Object`, mais vous ne savez pas quoi en faire. Vous ne savez pas vers quelle classe effectuer le transtypage (« cast » en VO), comme vous faites d'habitude :

```
String s = (String) monVector.get(1);
```

Mais rassurez-vous, tout n'est pas perdu (ouf). En effet, Java vous fournit des mécanismes d'introspection et de réflexivité, respectivement pour découvrir la structure interne d'une classe/objet et pour agir dessus.

Vous allez dans ce TP découvrir la classe `java.lang.Class`, la classe `java.lang.Object` et le package `java.lang.reflect`. Il s'agit d'outils très puissants et « bas niveau » dans Java, qui rendent possible des technologies comme RMI ou JavaBeans (*cf.* sérialisation). Vous les utiliserez pour tout découvrir sur le contenu du `Vector` de votre belle-soeur.

2. Préparation

Récupérez le code fourni sur `/home/Partage/enseignants/Info/3tc-jav`.

Vous disposez du code source d'une classe `RunMe`, qui récupère le `Vector` décrit plus haut, ainsi que son contenu. Vous avez également le paquet cadeau (compilé uniquement), généré à l'aide d'un simulateur de belle-soeur.

3. Classe `Object`

En Java, toutes les classes héritent de la classe `Object`. Ceci permet d'une part d'adresser tous les objets de manière générique, par exemple pour les placer dans un `Vector`, et d'autre part de forcer des comportements génériques aux objets. Par exemple :

Question 1:	A quoi sert la méthode `toString()` ?
Question 2:	Que vous apprend-elle sur le contenu du vecteur ?
Question 3:	Y a-t-il des éléments particuliers ?

4. Classe `Class` et package `java.lang.reflect`

La classe `Class` fournit une représentation des classes.

Question 4:	Mais alors, les classes sont-elles des objets ?

Cette représentation permet alors d'introspecter la composition d'une classe.

Question 5:	A quoi peut-on accéder ? (voir méthodes get en particulier)

Repérez comment accéder à la liste des méthodes d'une classe, de ses constructeurs, de ses attributs. Les types retournés par ces méthodes appartiennent au package `java.lang.reflect`.

Question 6:	Quels sont les méthodes, constructeurs, attributs des éléments contenus dans le vecteur ? Effectuez un appel de méthode sur l'un de ces éléments.

Comparez et testez les méthodes `getFields()` et `getDeclaredFields()`.

Question 7:	Que pouvez-vous dire au sujet de l'encapsulation ? Y a-t-il des incidences du point de vue sécurité ?
Question 8:	Pourquoi ces méthodes existent-elles ?
Question 9:	Pourquoi la classe `Class` est-elle `final` ?

5. Réflexivité

Nous avons vu qu'il est possible de récupérer énormément d'informations sur les classes et les objets. Nous allons maintenant voir qu'il est possible d'agir dessus.

Observez la classe `java.lang.reflect.Field`. Les méthodes `getXX` permettent de lire les valeurs des champs, les méthodes `setXX` de les modifier.

Question 10:	Modifiez un attribut d'un des éléments contenus dans le vecteur.

Par héritage, la classe Field dispose d'une méthode `setAccessible()`.

Question 11:	Que fait cette méthode ?

Utilisez cette méthode en utilisant vos conclusions des questions 6 et 7.

Question 12:	Java est-il donc non sécurisé par design ?

C'est bien entendu une question piège !