

# JAV TP4

## Introspection et réflexivité

Durée: 4h  
Titre: Réflexivité des classes et objets dans Java  
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 d'objets (`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 possibles 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 (code compilé uniquement), généré à l'aide d'un simulateur de belle-soeur top secret.

Pour les questions suivantes :

1. Un conseil : gardez la javadoc sous les yeux
2. Modifiez la classe `RunMe` et testez/vérifiez vous-mêmes les réponses
3. Soyez curieux et posez(-vous) des questions...

### 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 <code>toString()</code> de <code>Object</code> ?
---

- |             |   |
|-------------|---|
| Question 2: | Que vous apprend-elle sur le contenu du vecteur ? (Codez !)   |
| Question 3: | Y a-t-il des éléments particuliers à l'affichage ? Pourquoi ? |

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

La classe `java.lang.Class` fournit une représentation des classes. Nous verrons quelle est cette représentation plus loin. `Class` étant une classe, on peut l'instancier...

- |             |   |
|-------------|---|
| Question 4: | Mais alors, les classes sont-elles des objets ?<br>(question vache) |
|-------------|---|

Cette représentation interne des classes permet de les introspecter (découvrir leur composition interne).

- |             |  |
|-------------|--|
| Question 5: | A quelles informations peut-on accéder ? |
|-------------|--|

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 ? (Codez !)<br><br>A titre d'exemple, choisissez un des éléments du vecteur. Découvrez dynamiquement sa liste de méthodes, et appelez une de ces méthodes. Rappel : lorsque vous codez, vous ne connaissez pas à l'avance le nom de la méthode à invoquer ! |
|-------------|---|

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

- |             |   |
|-------------|---|
| Question 7: | Que pouvez-vous dire au sujet de l'encapsulation ?<br>Y a-t-il des incidences du point de vue sécurité ? Discutez ! |
| Question 8: | D'après vous, à quoi ces méthodes peuvent servir dans la vraie vie, pourquoi existent-elles ?                       |
| Question 9: | Pourquoi la classe <code>Class</code> est-elle <code>final</code> ?   |

## 5. Réflexivité

Nous avons vu qu'il est possible de récupérer énormément d'informations sur les classes et les objets, y compris ceux que le développeur ne connaît pas a priori. 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 public quelconque d'un des éléments contenus dans le vecteur. |
|--------------|--|

Jusqu'ici, cela ressemble à une utilisation normale :

```
monObjetDeClasseX.unAttributPublic = valeur;
```

La différence est que la classe et le nom de l'attribut sont découverts dynamiquement, sans connaissance préalable.

Etape suivante : par héritage, la classe `Field` dispose d'une méthode `setAccessible()`.

Question 11: Que fait cette méthode ?

Question 12: Utilisez-la en reprenant vos conclusions aux questions 6, 7 et 10.

## 6. Conclusion

Si vous avez bien suivi, vous venez de modifier la valeur d'un champ private appartenant à une instance d'une classe que vous ne connaissez même pas.

Cela signifie que, lorsque vous utilisez Java dans un environnement critique (serveur d'entreprise...), il faut prendre des précautions supplémentaires. Il est important d'évaluer les risques d'injection de code, de délimiter quel est le code de votre application que vos clients peuvent atteindre, etc.

Le (faible) risque est qu'une application cliente découvre et modifie l'état interne de votre application. Mais attention, le risque peut être corrigé, et ceci ne vous dispense pas de *concevoir* proprement votre application, à base d'objets proprement encapsulés.

Le même type de risque existe également pour Microsoft .NET, et pour n'importe quelle technologie capable d'introspection et de réflexivité. Le risque est encore plus grand pour les langages à typage dynamique (Ruby, Lisp...).

En Java, on peut se protéger de modifications extérieures, telles que celles vue dans ce TP, en utilisant le package `java.security`. De nombreux droits d'accès sont contrôlables via un « Security Manager » et des « permissions », comme par exemple restreindre à certaines classes précises le droit d'appeler les méthodes de `java.lang.reflect.Field`.

Mais ceci est une autre histoire !