INSA
CENTRE VAL DE LOIRE

POO Avancée : Java

Introspection

Salwa SOUAF

Outline

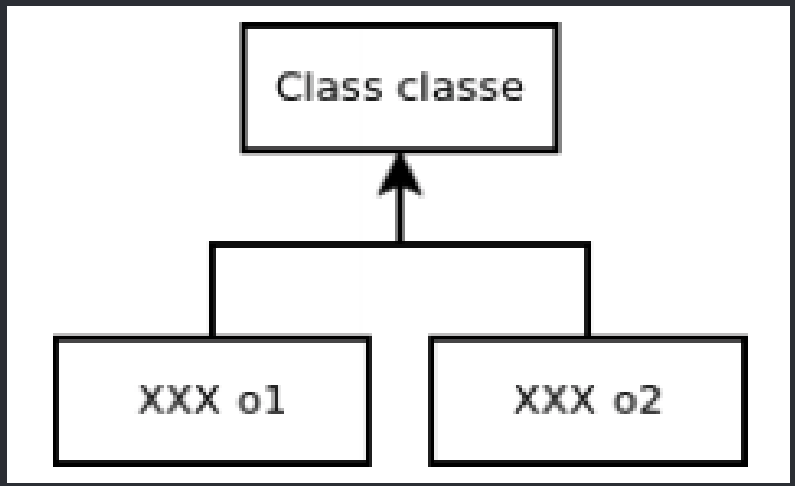
Introduction

L'introspection (ou aussi traduit reflexion pour reflective) est une capacité du langage Java à permettre l'accès à l'information sur les classes chargées, leurs attributs, méthodes ou constructeurs. Cette fonctionnalité du langage est très utile pour un certain nombre de programme qui analysent dynamiquement des classes du programme, comme par exemple les débuggueurs, les inspecteurs d'objets, les services de sérialisation du type Serializable ou Bean, les IDEs.

Outline

La classe Class

Chaque objet o instancié possède une référence vers un autre objet def de type Class. Il s'agit d'un objet contenant un certain nombre d'informations à propos de la classe de l'objet o. Evidemment, cet objet def est statique puisque toutes les instances de classe comme o partagent la même définition de classe. Visuellement, on peut représenter cela ainsi:



La classe Class

La récupération de l'objet Class se fait directement sur l'objet ou en appelant la méthode statique de la classe concernée:

```
Class classe = o.getClass();
Class classe = Class.forName("java.lang.String");
```

La classe Class est en fait un type paramétré: Class Class<T>. Lors de l'introspection d'une classe, le programmeur ne sait pas forcément quel est le type de l'objet o (sinon à quoi sert de fait de l'introspection ?). L'écriture précédente signifie donc en pratique:

```
Class<?> classe = o.getClass();
```

Outline

Inspecter une classe

Un certain nombre de méthodes sont disponibles sur la classe Class afin d'inspecter les différentes méthodes, attributs, constructeurs, localisation, classes mères, classes agrégées de la classe.
D'autres classes du package java.lang permettent de récupérer les informations de la classe Class:

- Constructor<T>: la classe représentant un constructeur
- Field: la classe représentatant un attribut de classe
- Method: la classe représentant une méthode de classe
- Type: la classe représentant les interfaces implémentées par une classe
- Package: la classe représentant le package d'une classe
- Annotation: la classe représentant les annotation de la classe

Inspecter une classe

Why ?

Le but de l'inspection (ou introspection) est de répondre à des questions du type:

- Est-ce que cette classe est publique ?
- Quels sont les constructeurs disponibles pour cette classe ?
- Cette classe possède-t-elle un attribut nommé truc ?
- Y a-t-il une méthode X(String t) dans cette classe ?

Le but secondaire de l'inspection est de réaliser des actions dynamiquement, en fonction des informations collectées précédemment:

- Construction d'un nouvel objet
- Appel de la méthode X(String t)
- Affichage de l'attribut truc

Inspecter une classe

Modificateurs de classe

Les méthodes suivantes renseignent sur la définition de la classe :

- isAnonymousClass() Retourne vrai si et seulement si la classe est une classe anonyme.
- isArray() Détermine si l'objet représente un tableau.
- isAssignableFrom(Class<?> cls) Détermine si la classe ou l'interface représenté par ces objet est une super classe ou super interface de la classe cls passée en paramètre.
- isEnum() Retourne vrai si la classe est une énumération.
- isInstance(Object obj) Détermine si l'objet spécifié en paramètre est compatible avec la classe interrogée pour un possible assignement.
- isInterface() Détermine si l'objet de type Classe est une interface.
- isPrimitive() Détermine si l'objet de type Classe est un type primitif.

Inspecter une classe

Interfaces et classe mère

La recherche d'interfaces et d'une classe mère s'opère au travers des méthodes:

```
Class<? super T>      getSuperclass()
Class<?>[]    getInterfaces()
```

La classe mère peut être une classe, une interface, un type primitif ou void. Si la classe sur laquelle la méthode est appellée est Object ou une interface, un type primitif ou void, alors null est renvoyé. Les interfaces implémentées peuvent être multiple et sont renvoyées dans l'ordre de leur déclaration par le mot clef implements. Si aucune interface n'est implémentée, le tableau est de taille 0.

Inspecter une classe

Attributs

Les attributs de classes peuvent être récupérés à l'aide de Field[] getFields(). Cet attribut peut être modifié, par exemple à l'aide de setInt, setFloat, ... ou tout simplement la méthode set(Object obj, Object value) qui modifie l'attribut représenté par l'objet de type Field sur l'objet obj avec l'objet value.

```
Classe c = vehicule.getField("moteur");
Moteur m = new Moteur();
c.set(vehicule, m);
```

Inspecter une classe

Les constructeurs

L'ensemble des constructeurs sont renvoyés par la méthode suivante:

```
public Constructor<?>[] getConstructors() throws SecurityException
```

L'objet de type Constructor ainsi récupéré peut-être à son tour inspecté. Par exemple, ses paramètres d'appel peuvent être récupérés:

```
Class<?>[]   getParameterTypes()
```

Un constructeur peut aussi être invoqué, afin de construire une nouvelle instance de l'objet que décrit la classe Class. On utilise pour cela la méthode T newInstance(Object... initargs) qui demande à la classe Class une nouvelle instance de classe, avec, comme paramètres (à nombre variables) les paramètres passés à la méthode.
La difficulté de l'utilisation de newInstance réside dans la découverte des paramètres du constructeur. En effet, il n'y a pas toujours de constructeurs sans paramètres permetant d'appeler newInstance().

```
String str = new String("test");
Class c = str.getClass();
String str2 = c.newInstance();
```

Inspecter une classe

Les méthodes

Les méthodes peuvent être récupérées dans un tableau d'objets Method. Il est aussi possible de chercher une méthode particulière à partir de son nom et de la liste des classes de ses paramètres (le nom n'est en effet pas suffisant):

```
Method[]      getDeclaredMethods()
Method        getDeclaredMethod(String name, Class<?>... parameterTypes)
```

Une méthode peut ensuite être inspectée par exemple pour connaitre son type de retour à l'aide de Class<?> getReturnType(), ses paramètres à l'aide de Class<?>[] getParameterTypes(), les exceptions qu'elle est susceptible de lever à l'aide de Class<?>[] getExceptionTypes().
Une méthode peut ensuite être invoquée sur l'objet passé en premier paramètre à l'aide des paramètres (à nombre variable) dans les paramètres suivants:

```
Object invoke(Object obj, Object... args)
```