

---

# Chapitre 8

## La réflexion

05/12/03

La réflexion

page 1

---

## Reflection

- La réflexion désigne la possibilité de manipuler objets, classes, méthodes et champs de façon dynamique.
- La classe centrale permettant la mise en œuvre de la réflexion est la classe :  
Class
- Dans cette classe, figurent toutes les méthodes permettant d'accéder aux (définitions de) membres de la classe.

05/12/03

La réflexion

page 5

---

## Plan

- Présentation
- Méta-objets
- Introspection
- Manipulation d'objets
- Manipulation de tableaux
- Run-Time Type Identification

05/12/03

La réflexion

page 2

---

## Package java.lang.reflect

- Les autres classes indispensables à la réflexion se trouvent dans le package java.lang.reflect.
- On y trouve les classes suivantes :
  - Array
  - Constructor
  - Field
  - Method
  - Modifier

05/12/03

La réflexion

page 6

---

## Références

- The Java™ Tutorial, The Reflection API  
<http://java.sun.com/docs/books/tutorial/reflect/index.html>
- Thinking in Java, Chapitre 12, Run-time Type Identification
- Java beans Tutorial, Part 4, Reflection and Introspection  
<http://developer.java.sun.com/developer/onlineTraining/Beans/Beans4/reflect00.html>

05/12/03

La réflexion

page 3

---

## Possibilités de la réflexion

- Avec la réflexion, vous pouvez :
  - introspecter les classes, i.e., examiner leur définitions (modificateurs, champs, méthodes, classe étendue, interfaces implémentées, classes internes) de classes ;
  - manipuler les objets, i.e., créer des instances, accéder aux champs, exécuter des méthodes dont les noms ne sont connus qu'au moment de l'exécution ;
  - manipuler les tableaux, i.e., identifier les tableaux et leurs types, créer des tableaux et accéder aux éléments de tableaux.

05/12/03

La réflexion

page 7

---

## Présentation

05/12/03

La réflexion

page 4

---

## Intérêts de la réflexion

- La réflexion sera particulièrement utile si vous projetez de développer des outils de développements tels que :
  - des debuggers
  - des navigateurs (browsers) de classes
  - des environnements de développement d'interfaces graphiques

05/12/03

La réflexion

page 8

## De la bonne utilisation de la reflection

- Il faut éviter d'utiliser la reflection lorsqu'il est possible de s'en passer naturellement.
- Par exemple :
  - il est possible de passer des fonctions en paramètre en utilisant la classe `Method`,
  - mais cette approche n'est pas naturelle en conception objet ;
  - l'approche naturelle est l'utilisation de l'héritage et du polymorphisme.
- Par ailleurs, la reflection rend plus difficile le débogage (les erreurs sont cachées).

05/12/03

La réflexion

page 9

## Et pour les types primitifs ?

- Il est possible d'obtenir un méta-objet pour les types primitifs par la première technique :

```
Class c1 = int.class;
Class c2 = double.class;
```
- De façon équivalente, on peut aussi écrire :

```
Class c1 = Integer.TYPE;
Class c2 = Double.TYPE;
```
- Cela peut être utile pour spécifier le type des arguments d'un constructeur ou d'une méthode que l'on souhaite utiliser (voir plus loin).

05/12/03

La réflexion

page 13

## Méta-objets

05/12/03

La réflexion

page 10

## Méthode `getName()`

- La méthode `getName` appliquée à un méta-objet retourne le nom de la classe qu'il représente.
- Dans le cas d'un tableau,
  - le nom est préfixé par `[]`
  - si le type des éléments est primitif alors un caractère spécifique est utilisé (voir diapositive suivante)
  - si le type des éléments n'est pas primitif, alors le nom de la classe des éléments est entouré par `L` et `;`
- Pour plus d'informations :
  - <http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html>

05/12/03

La réflexion

page 14

## Objets de la classe `Class`

- Pour toute classe utilisée par une application, un (seul) objet particulier instance de `Class` est créé par la JVM.
- Etant instance de la classe `Class`, il est possible d'utiliser toutes les méthodes définies dans cette classe.
- Remarque : ce « méta-objet » gère tous les champs et méthodes statiques de la classe qu'il représente.

05/12/03

La réflexion

page 11

## Encodage des noms de type

<b>B</b>	<code>byte</code>	<b>C</b>	<code>char</code>
<b>D</b>	<code>double</code>	<b>F</b>	<code>float</code>
<b>I</b>	<code>int</code>	<b>J</b>	<code>long</code>
<b>S</b>	<code>short</code>	<b>Z</b>	<code>boolean</code>
<b>V</b>	<code>void</code>		
<code>L</code> <code>className</code> ;	<code>class or interface</code>		
<code>[</code> <code>className</code>	<code>array</code>		

05/12/03

La réflexion

page 15

## Retrouver un méta-objet

- Si vous disposez du nom de la classe au moment de la compilation :

```
Class c = JButton.class;
```
- Si vous disposez du nom de la classe au moment de l'exécution :

```
Class c = Class.forName(name);
```
- Si vous disposez d'une instance d'une classe dont vous souhaitez retrouver le méta-objet :

```
Class c = mystery.getClass();
```

05/12/03

La réflexion

page 12

## Introspection

05/12/03

La réflexion

page 16

## Introspection

- L'introspection consiste à examiner la définition d'une classe en accédant à l'ensemble des caractéristiques de celle-ci :
  - modifieurs
  - champs
  - constructeurs
  - méthodes
  - super-classe
  - classe internes
  - ...

## Héritage

- Il est possible de connaître respectivement la superclasse et les interfaces implémentées d'une classe donnée avec les méthodes suivantes de Class :
  - public Class getSuperclass()
  - public Class[] getInterfaces()
- Il est possible de déterminer si un objet Class représente une interface, un type primitif ou une classe « normale » :
  - public boolean isInterface()
  - public boolean isPrimitive()

## Membres d'une classe

- Les champs, constructeurs et méthodes d'une classe sont obtenus (dans un tableau) en utilisant les méthodes suivantes de Class :
  - public Field[] getDeclaredFields()
  - public Constructor[] getDeclaredConstructors()
  - public Method[] getDeclaredMethods()
- Remarque : les méthodes getFields(), getConstructors() et getMethods() ne retournent que les membres déclarés « public ».

## Classes externe et internes

- Il est possible de connaître respectivement la classe externe et les classes internes d'une classe donnée avec les méthodes suivantes de Class :
  - public Class getDeclaringClass()
  - public Class[] getDeclaredClasses()
- Les éléments retournés peuvent désigner des interfaces.

## Modifieurs

- Il existe une méthode getModifiers dans les classes Class, Field, Constructor et Method.
- Cette méthode retourne un entier m qu'il est alors possible d'utiliser en argument de :
  - Modifier.isAbstract(m);
  - Modifier.isFinal(m);
  - Modifier.isPrivate(m);
  - ...
  - Modifier.toString(m)

## Manipulation d'objets

## Types

- Il est possible de connaître le type d'un champ (objet Field) :
  - public Class getType()
- Il est possible de connaître le type de retour d'une méthode (objet Method) :
  - public Class getReturnType()
- Il est possible de connaître le type des arguments d'une méthode (ou d'un constructeur) :
  - public Class[] getParameterTypes()

## Créer un objet

- Lorsque le type des objets à créer est connu à la compilation, il est possible d'utiliser l'opérateur new.
  - Rectangle r = new Rectangle(20, 50);
- Dans le cas contraire, il faut utiliser la méthode newInstance de la classe Class ou de la classe Constructor.

## Construction sans argument

- Dans ce cas précis, il est possible d'utiliser la méthode suivante de Class :
  - `public Object newInstance()`
- Il faut prendre en compte les deux types d'exceptions :
  - `InstantiationException`
  - `IllegalAccessException`

## Manipulation de tableaux

## Construction avec arguments

- Il est d'abord nécessaire de récupérer le bon constructeur en utilisant la méthode suivante de la classe Class :
  - `public Constructor getConstructor(Class[] types)`
- Il est ensuite possible de créer une instance en utilisant la méthode suivante de la classe Constructor :
  - `public Object newInstance(Object[] initargs)`

## Run-Time Type Identification

## Manipuler un champ

- Il est d'abord nécessaire de récupérer le bon champ en utilisant la méthode suivante de Class :
  - `public Field getField(String name)`
- Il est ensuite possible de consulter ou modifier ce champ pour un objet donné en utilisant les méthodes suivantes de la classe Field :
  - `public Object get(Object obj)`
  - `public void set(Object obj, Object value)`

## RTTI

- L'identification de type à l'exécution consiste à s'intéresser au type d'un objet ; ce type étant inconnu au moment de la compilation.
- Cette identification de type, ou plutôt cette gestion de type, permet d'effectuer des tests et des contrôles au moment de l'exécution.

## Exécuter une méthode

- Il est d'abord nécessaire de récupérer la bonne méthode en utilisant cette méthode de Class :
  - `public Method getMethod(String name, Class[] Types)`
- Il est ensuite possible d'exécuter cette méthode pour un objet donné en utilisant cette méthode de Method :
  - `public Object invoke(Object obj, Object[] args)`

## Gestion de type

- Les différentes formes de gestion dynamique de type sont les suivantes :
  - le transtypage (« cast »)
  - l'accès au méta-objet :
  - l'opérateur `instanceof`
  - la méthode `isInstance`

## Transtypage

- Lorsqu'on utilise l'héritage, il est parfois nécessaire d'effectuer une opération de transtypage.
- Par exemple :  
Vehicule v = ...  
... (Automobile)v ...
- Cette opération permet de considérer v comme étant non plus de type Vehicule mais plus précisément comme étant de type Automobile.
- Une ClassCastException est lancée si le transtypage est incorrect.

05/12/03

La réflexion

page 33

## Accès au méta-objet

- Comme vu précédemment, il est possible d'obtenir le méta-objet de toute classe.
- Les trois façons de procéder sont présentées à la section « méta-objets ».

[Retrouver un méta-objet](#)

05/12/03

La réflexion

page 34

## Opérateur instanceof

- Cet opérateur permet de déterminer si un objet est instance d'un type particulier.
- Par exemple :  
If (animal instanceof Oiseau)  
((Oiseau)animal).voler();
- Il n'est pas possible d'utiliser instanceof en considérant un méta-objet (objet de la classe Class) comme second argument.

05/12/03

La réflexion

page 35

## Méthode isInstance

- La méthode suivante existe dans la classe Class :  
– public boolean isInstance(Object obj)
- Elle permet de déterminer si l'objet passé en paramètre est instance de la classe dont le méta-objet (qui reçoit l'appel) est le représentant.

05/12/03

La réflexion

page 36