

## TD8 : Introspection

*Un programme écrit en Java a cette particularité fort intéressante de pouvoir dynamiquement analyser le code d'un objet Java inconnu. Cette propriété, appelée **introspection** ou **réflexivité**, a de nombreux usages.*

### VUE D'ENSEMBLE

---

Nous n'allons pas tout expliquer ici mais plutôt vous donnez une première vue d'ensemble. Après quoi, nous vous renvoyons vers le tutoriel de SUN pour les détails.

- ✓ Il existe une classe nommée **Class** qui représente la classe d'un objet. Cette classe ne propose aucun constructeur public mais on peut obtenir un objet de classe **Class** de plusieurs façons :
  - ✓ via la méthode `getClass()` d'un objet quelconque (ex: `s.getClass()`);
  - ✓ via l'attribut `'class'` de la classe (ex: `String.class`) ;
  - ✓ via la méthode `'forName'` de la classe `'Class'` à laquelle on donne une chaîne contenant le nom de la classe (ex: `Class.forName(« java.lang.String »)`)
- ✓ La classe `Class` fournit de nombreuses méthodes pour se renseigner sur la classe en question. On peut ainsi connaître son **nom**, ses **modificateurs**, sa **classe mère** et les **interfaces** implémentées.
- ✓ On peut aussi avoir le détail de tous les membres (**attributs** et **méthodes**) publics et connaître quels sont les **constructeurs** disponibles.
- ✓ Non seulement on peut se renseigner sur un objet inconnu lors de l'écriture du code mais on peut aussi **manipuler dynamiquement des objets**. On peut ainsi créer un objet d'une classe inconnue au moment de la rédaction du code, accéder et modifier les valeurs des champs de cet objet et appeler ses méthodes.

### UTILITÉ

---

A priori l'introspection ne paraît guère utile mais elle a des applications nombreuses:

- ✓ Elle permet d'utiliser des objets dont la classe ne sera connue qu'au moment de l'exécution (nous en verrons un exemple dans le chapitre suivant lorsque nous écrirons du code manipulant les données d'un SGBD relationnel: ce code sera idéalement indépendant du SGBD cible et donc le driver utilisé ne sera connu qu'au moment de l'exécution, vous en avez déjà rencontré une application avec la sérialisation, ...)
- ✓ Elle permet à votre IDE favori de présenter les propriétés des JavaBeans qu'il propose dans sa palette
- ✓ Elle permet d'écrire des codes très génériques
- ✓ Elle permet de réaliser du débogage
- ✓ ...

Attention, veillez à ne mettre en œuvre l'introspection que lorsque cela s'avère indispensable ou fortement utile: de manière générale, son utilisation a tendance à dégrader les performances des applications.

## APPRENTISSAGE

---

Au total, le sujet est assez simple mais il existe beaucoup de classes à mettre en œuvre et des consultations fréquentes de l'API sont nécessaires. Nous vous proposons les lectures suivantes :

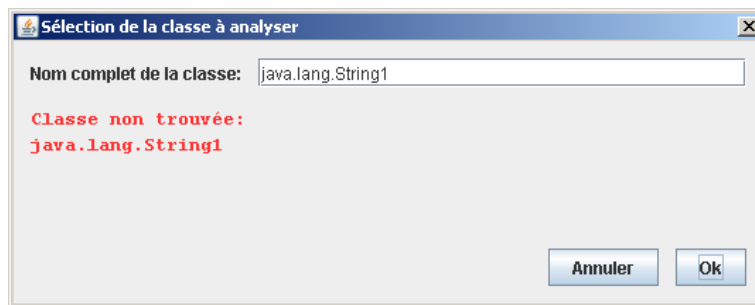
- [http://www.idiom.com/~zilla/Computer/java\\_reflection.html](http://www.idiom.com/~zilla/Computer/java_reflection.html) : pour un petit exemple
- <http://java.sun.com/docs/books/tutorial/reflect/index.html> : le tutoriel de SUN, plus complet

## APPLICATION

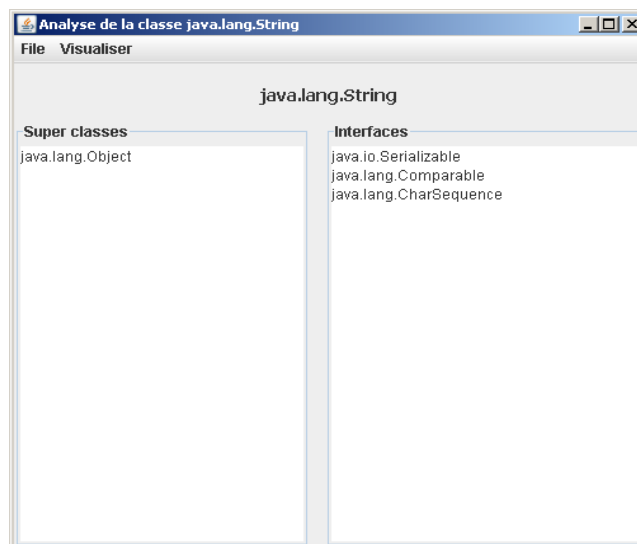
---

Nous allons réaliser une application permettant d'analyser partiellement une classe dont le nom sera fourni à l'exécution (attention, cette classe devra être présente dans le classpath de l'application).

Tout d'abord, réalisons un JDialog (`SelectClasse`) permettant de saisir le nom de la classe et de la charger en mémoire [si le chargement ne peut pas se faire, un message d'erreur apparaît]. Il offre un getter `getClasse()` retournant la classe chargée ou null si aucune sélection n'est faite. Nous veillerons à écrire un code permettant d'être réutilisé dans un autre contexte.

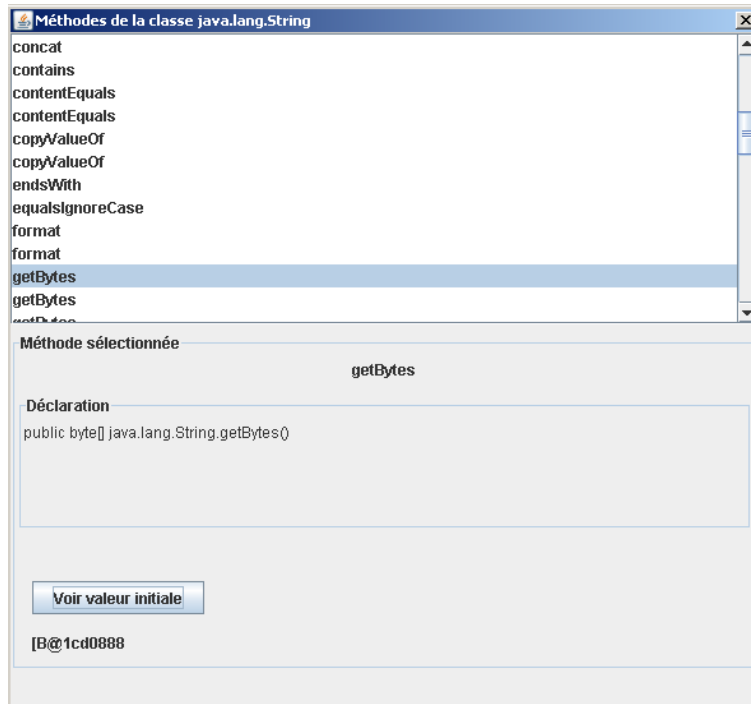


Ce dialogue apparaît au lancement du programme et, si la classe demandée est chargée correctement, la frame principale apparaît:



Le menu 'File' présente les items 'Choisir une classe' (permettant de changer de classe à analyser) et 'Quitter'.

Le menu 'Visualiser' présente les items 'Constructeurs', 'Méthodes' et 'Attributs' offrant des fonctionnalités comparables dont nous vous présentons la deuxième:



Vous présentez la liste des méthodes accessibles pour la classe sélectionnée et, à chaque fois que vous en sélectionnez une, son nom et sa déclaration [ceci provient du toString() de l'objet représentant la méthode] apparaissent dans un panel de détail.

Dans le cas où la classe choisie propose un constructeur par défaut, que la méthode sélectionnée n'a pas de paramètre et qu'elle fournit une valeur de retour, le bouton 'Voir valeur initiale' permet de donner la valeur retournée par la méthode sur l'objet obtenu en utilisant le constructeur par défaut.

Remarquez qu'il est possible via l'introspection – à vous de prévoir ce qu'il faut – de faire exécuter une méthode private!