

# L'introspection

avec Java

# Introduction:

- Pour un objet, cela consiste à examiner par réflexion, la class d'implémentation de cet objet, en récupérant son nom à l'exécution et ce afin d'analyser sa structure (types et noms de ses attributs, méthodes, constructeur et bien plus encore).
- Le mécanisme de réflexion est fourni par l'API reflection (package `java.lang.reflect`), nous allons aussi utiliser la classe `Class` (`java.lang.Class`) et des méthodes de la classe `Object`.
- La réflexion est fonctionnalité fondamentale de beaucoup de Framework:
  - JEE, Hibernate, Spring, JUnit ...
  - Tous les Framework qui font des IHM Web.
  - Tous les services Rest/Web
  - Les IDE tel que Eclipse, IntelliJ IDEA ...
- La réflexion est donc utilisée dans beaucoup de concepts: Le monitoring, le débogage, le mapping d'objet avec une DB...

# Exemple d'un autre outil permettant d'analyser un .class

Javap est un outils du JDK qui permet de décompiler un fichier .class

```
$ javap -p Account.class
Compiled from "Account.java"
public abstract class fr.fms.entities.Account {
    private long accountId;
    private java.util.Date creationDate;
    private double balance;
    private fr.fms.entities.Customer customer;
    private java.util.ArrayList<fr.fms.entities.Transaction> listTransactions;
    public fr.fms.entities.Account(long, java.util.Date, double, fr.fms.entities.C
ustomer);
    public java.lang.String toString();
    public fr.fms.entities.Customer getCustomer();
    public void setCustomer(fr.fms.entities.Customer);
    public long getAccountId();
    public void setAccountId(long);
    public java.util.Date getCreationDate();
    public void setCreationDate(java.util.Date);
    public double getBalance();
    public void setBalance(double);
    public java.util.ArrayList<fr.fms.entities.Transaction> getListTransactions();
}
```

Liste des attributs de la classe Account.

Le constructeur

Liste des méthodes de la classe Account.

Notez bien qu'en paramètre, on passe un fichier compilé.


## Exemple d'une réflexion avec la classe IBankBusiness du projet UmlTPBank:

- Dans un premier temps: récupération du nom de la classe de notre objet:

```
IBankBusinessImpl banqueBusiness = new IBankBusinessImpl();

//Récupération du nom de la class d'un objet, grace a la methode .getClass() de la class Object.
System.out.println("Nom complet de la class :" + banqueBusiness.getClass().getName());


String nomDeMaClass = banqueBusiness.getClass().getName();
System.out.println("-----");
```



Retourne un objet de type Class

- Dans un second temps: on utilise la méthode .forName(String nomDeMaClass) de classe Class pour instancier un objet de la classe Class qui reflète la structure de la classe de notre objet d'études...

```
//Class c = Class.forName("fr.fms.business.IBankBusinessImpl");
Class c = Class.forName(nomDeMaClass); //est équivalent à la ligne du dessus
```



Objet de type class, modélisant la classe IBankBusinessImpl

```
<terminated> Introspection [Java Application] C:\Users\Stagiaires09\p2\pool\plu
Nom complet de la class :fr.fms.business.IBankBusinessImpl
```

## L'objet Field:

- La méthode `.getDeclaredFields()` retourne tous les attributs Field d'objet, peut importe leur visibilité.
- Les attributs retournés sont des objet de type Field, on peut alors récupérer, grâce à aux méthodes de la classe Modifier, leur accessibilité, leur nom et leur type.

```
//Recuperation de tous les champs de la class (et de leur modificateur d'accès), meme private ou protected.  
System.out.println("-----");  
System.out.println("Listes des attributs de la class : " + banqueBusiness.getClass().getSimpleName());  
Field[] attributes = c.getDeclaredFields();  
for(Field attribute : attributes) {  
    System.out.println(Modifier.toString(attribute.getModifiers()) + "\t" + attribute.getName() + " " + attribute.getType().getSimpleName());  
}
```

Retourne un tableau d'objet Field

accessibilité

Son nom

Son type

```
Listes des attributs de la class : IBankBusinessImpl  
private accounts HashMap  
private customers HashMap  
private numTransactions long
```

- La méthode `.getFields()` retourne tous les attributs public de c et de ses supers classes
- La méthode `.getField(« nomAttribut »)` retourne une seule instance Field, représentant l'attribut `nomAttribut`.

## Les objets Method et Type:

- La méthode `.getDeclaredMethods()` retourne toutes les méthodes déclarées dans la classe `c`. Les méthodes retournées sont des objets de type `Method`, on peut alors les inspecter.

```
System.out.println("Listes des methodes de la class : " + banqueBusiness.getClass().getSimpleName());  
//Recuperation de toutes les methodes (et de leur modificateur) de la class dans un tableau  
Method[] methods = c.getDeclaredMethods();  
for(Method method : methods) {  
    System.out.println(Modifier.toString( method.getModifiers()) + " : " + method.getName() + ":" + method.getReturnType());  
    Type[] types = method.getParameterTypes();  
    for(Type type : types)  
        System.out.println( " type du parametre : " + type.getTypeName() +"\t");  
}
```

Retourne un tableau d'objet de type Method

Son type

Son nom

Type de retour

Retourne un tableau d'objet se type Type

- La méthode `.getParameterTypes()` retourne un tableau d'objet de type `Type` des attributs des méthodes implémentées dans la classe `c`.

## Résultat:

```
Listes des methodes de la class : IBankBusinessImpl
private : addAccountToCustomer:void
  type du parametre : fr.fms.entities.Customer
  type du parametre : fr.fms.entities.Account
public : consultAccount:class fr.fms.entities.Account
  type du parametre : long
public : listTransactions:class java.util.ArrayList
  type du parametre : long
public : withdraw:boolean
  type du parametre : long
  type du parametre : double
public : listAccounts:class java.util.ArrayList
public : pay:void
  type du parametre : long
  type du parametre : double
public : addAccount:void
  type du parametre : fr.fms.entities.Account
public : transfert:void
  type du parametre : long
  type du parametre : long
  type du parametre : double
```

L'API reflection contient un grand nombres de méthodes permettant d'avoir accès à la structure des classe du application.

## Conclusion:

- L'API Reflection nous permet de chercher dynamiquement des informations sur une classe, mais aussi de les modifier, d'invoquer ses méthodes et ses constructeurs.
- Certaines de ses fonctionnalités peuvent être utilisées à des fins malveillantes. Afin de s'en prémunir, il existe une protection dans la JVM : le SecurityManager qui permet d'autoriser ou non l'utilisation des fonctionnalités de l'API Reflection.
- La réflexion est très gourmande en ressources et ne doit être utilisée que lorsqu'il n'existe aucune autre alternative.

### Sources:

[https://koor.fr/Java/Tutorial/java\\_reflexion\\_introduction.wp](https://koor.fr/Java/Tutorial/java_reflexion_introduction.wp)  
<https://www.jmdoudoux.fr/java/dej/chap-introspection.htm>

### Vidéos:

[Java Reflection\\_cours-en-ligne\\_Paumard](#)