

Introspection en Java

Université de Montpellier
Java avancé (IG4)

2019

Introspection : une forme
de méta-programmation

Méta-programmation

- Un méta-programme a accès à la représentation d'un programme pour réaliser divers traitements
- Réflexion, réflexivité
 - Capacité d'un programme à s'observer ou à modifier son état ou son mode d'exécution
 - Pendant son exécution

Méta-programmation

Différents niveaux

- Examen, observation des structures, des objets
- Création d'objets
- Modification de l'état des objets
- Modification du programme lui-même
- Modification de la sémantique d'exécution

Introspection en Java

- Classes et méthodes permettant
 - Accès à l'information sur les classes
 - attributs
 - méthodes
 - constructeurs
 - Manipulation des objets de ces classes
 - création d'objets et appel de constructeurs
 - modification des valeurs des attributs
 - appel de méthodes
- Pendant l'exécution

Utilisation *réalisation de*

- Débogueurs
- Interprètes
- Inspecteurs d'objets
- Navigateurs de classes (class browsers)
- Services particuliers, ex.
 - Sérialisation (sauvegarde d'objets)
 - Editeurs d'objets
 - Intercession (interception d'appels)

Principales classes

- java.lang
 - Class<T> classe
- java.lang.reflect
 - Field attribut
 - Constructor<T> constructeur
 - Method méthode

Principales classes

- **Class<T>**
 - le type de `String.class` est `Class<String>`
 - ses *fields*, *constructors*, *methods*, *interfaces*, *classes*, ..
- **Field**
 - son type, sa classe, sa valeur pour un objet, ..
- **Constructor<T>**
 - ses paramètres, exceptions, ..
- **Method**
 - ses paramètres, type de retour, exceptions, ..

Contextes d'utilisation

Sans volonté d'exhaustivité, quelques exemples classiques d'utilisation

- Inspection des méthodes
- Inspection d'objets
- Création d'objets selon des types non connus au préalable
- Appel de méthodes

Cas 1.

Inspection des méthodes

- Éléments utilisés
 - Class, Method
 - String getName()
 - Class
 - static Class forName(String c)
 - retourne l'instance représentant la classe nommée c
 - Method[] getMethods()
 - retourne les méthodes publiques de la classe
 - Method
 - Class getReturnType()
 - Class[] getParameterTypes()

Inspection des méthodes

```
abstract class Produit{  
    private String reference,designation;  
    private double prixHT;  
    public Produit(){}  
    public Produit(String r,String d,double p)  
        {reference=r; designation=d; prixHT=p;}  
    public String getReference(){return reference;}  
    public void setReference(String r){reference=r;}  
    public String getDesignation(){return designation;}  
    public void setDesignation(String d){designation=d;}  
    public double getPrixHT(){return prixHT;}  
    public void setPrixHT(double p){prixHT=p;}  
    abstract public double leprixTTC();  
    public String infos(){return getReference()+" "+  
                                getDesignation()+" "+leprixTTC();}  
}
```

Inspection des méthodes

```
class ProduitTNormal extends Produit
{ public ProduitTNormal(){}
    public ProduitTNormal(String r,String d,double p)
        {super(r,d,p);}
    public double leprixTTC()
        {return getPrixHT() * 1.196;}
}

class Livre extends ProduitTNormal
{ private String editeur;
    public Livre(){}
    public Livre(String r,String d,double p,String e)
        {super(r,d,p);editeur=e;}
    public String getEditeur(){return editeur;}
    public void setEditeur(String e){editeur=e;}
    public String infos(){return super.infos()+" "+getEditeur();}
}
```

Inspection des méthodes

```
package Exemples;
```

```
import java.lang.reflect.*;
```

```
// Class est dans java.lang
```

```
// Method est dans java.lang.reflect
```

Inspection des méthodes

```
public class TestReflexion
{
    public static void afficheSignaturesMethodes(Class cl)
    {
        Method[] methodes = cl.getDeclaredMethods();
        for (int i=0; i<methodes.length; i++)
        {
            Method m = methodes[i];
            String m_name = m.getName();
            Class m_returnType = m.getReturnType();
            Class[] m_paramTypes = m.getParameterTypes();
            System.out.print(" "+m_returnType.getName()+
                            " "+m_name + "(");
            for (int j=0; j<m_paramTypes.length; j++)
                System.out.print(" "+m_paramTypes[j].getName());
            System.out.println(")");
        }
    }
    .....
}
```

Inspection des méthodes

```
public class TestReflexion
{ ....
    public static void main(String[] argv)
        throws java.lang.ClassNotFoundException
    {
        System.out.println("Saisir un nom de classe");
        Scanner s=new Scanner(System.in);
        String nomClasse = s.next();
        Class c = Class.forName(nomClasse);
        TestReflexion. afficheSignaturesMethodes(c);
    }
} //fin TestReflexion
```

Inspection des méthodes

Saisir un nom de classe

<< Exemples.Livre

>> java.lang.String getEditeur() *Livre*

>> void setEditeur(java.lang.String)

>> java.lang.String infos()

>> double leprixTTC() *ProduitNormal*

>> double getPrixHT() *Produit*

>> java.lang.String getReference()

>> int hashCode() *Object*

>> java.lang.Class getClass()

>> boolean equals(java.lang.Object)

>> java.lang.String toString()

Cas 2. Inspection des objets

- Éléments utilisés
 - Object
 - Class `getClass()`
 - retourne la classe de l'objet
 - Class
 - String `getName()`
 - Field `getField(String n)`
 - retourne l'attribut nommé *n*
 - Field
 - Object `get(Object o)`
 - retourne la valeur de l'attribut pour l'objet *o*

Inspection des objets

```
Produit p = new Livre("X23", "Paroles de  
Prévert", 25, "Folio");  
System.out.println(p.getClass().getName());
```

>> Exemples.Livre

```
p = new Aliment("A21", "Pain d'épices",  
12, "BonMiel");  
System.out.println(p.getClass().getName());
```

>> Exemples.Aliment

Accès aux attributs *public*

// editeur et prixHT ont été déclarés *public* pour cette partie

```
Livre p = new Livre("X23","Paroles de  
Prévert",25,"Folio");
```

```
Class p_class = p.getClass();
```

```
Field f1_p = p_class.getField("editeur");
```

```
Object v_f1_p = f1_p.get(p);
```

```
Field f2_p = p_class.getField("prixHT");
```

```
Object v_f2_p = f2_p.get(p);
```

```
System.out.println("v_f1_p="+v_f1_p+"  
v_f2_p="+v_f2_p);
```

```
>> v_f1_p=Folio v_f2_p=25.0
```

Accès aux attributs privés

On les récupère grâce à

Field[] getDeclaredFields

On les rend accessibles grâce à

myfield.setAccessible(true);

Méthode héritée de *AccessibleObject*

Cas 3. Créer des objets

- Éléments utilisés
 - Class
 - static Class `forName(String)`
 - Constructor `getConstructor()`;
 - retourne le constructeur sans paramètres
 - Constructor
 - Object `newInstance()`
 - retourne un objet construit avec le constructeur

Créer des objets

```
System.out.println("Livre ou Aliment ?");  
Scanner s=new Scanner(System.in);  
String nomClasse = s.next();  
Object np;  
  
// et maintenant on voudrait créer  
// un livre ou un aliment
```

Créer des objets

code classique

```
System.out.println("Livre ou Aliment ?");  
Scanner s=new Scanner(System.in);  
String nomClasse = s.nextLine();Object np;
```

```
if (nomClasse.equals("Exemples.Livre"))  
    np = new Livre();  
else if (nomClasse.equals ("Exemples.Aliment"))  
    np = new Aliment();  
else if ...
```

- **Pb extensibilité** - ajout de classe, modification de nom de classe implique :

modification de code

Créer des objets avec la réflexion

```
System.out.println("Livre ou Aliment ?");  
Scanner s=new Scanner(System.in);  
String nomClasse = s.next();  
Object np;  
  
Class c = Class.forName(nomClasse);  
Constructor constructeur=c.getConstructor();  
np = constructeur.newInstance();
```


Créer des objets avec la réflexion

Pour appeler un constructeur prenant des paramètres

Constructor constructeur =

```
c.getConstructor(String.class,  
                String.class,  
                double.class,  
                String.class);
```

np = constructeur.newInstance

```
("xx", "Paroles", 12, "Folio");
```

Cas 4. Appeler des méthodes

- Éléments utilisés
 - Class
 - Method `getMethod(String n)`
 - retourne la méthode nommée *n*
 - Method
 - Object `invoke(Object)`
 - appelle la méthode sur l'objet *o*

Appeler des méthodes

```
System.out.println("Méthodes existantes sur np");
```

```
TestReflexion.afficheMethodesPubliques(c);
```

```
>> ...
```

```
>> java.lang.String infos()
```

```
>> double leprixTTC()
```

```
>> double getPrixHT()
```

```
>> ....
```

```
System.out.println("Quelle méthode sans argument voulez-vous  
appeler ?");
```

```
String nomMeth = s.next();
```

```
<< leprixTTC
```

```
Method meth = c.getMethod(nomMeth);
```

```
Object resultat = meth.invoke(np);
```

```
System.out.println("resultat = "+resultat);
```

```
>> resultat = 14.352
```

Appeler des méthodes avec des paramètres

```
meth = c.getMethod("setEditeur",  
                    String.class);
```

```
resultat = meth.invoke(np, "Gallimard");
```

```
System.out.println("nouvel objet = "+np);
```

On peut aussi ...

- accéder aux *modifiers*
- connaître les super-classes, les interfaces
- créer et manipuler des tableaux
- créer des proxys de classes ou d'instances pour intercepter des appels et ajouter du comportement (ex. tracer automatiquement)