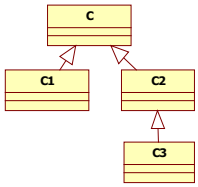
NFP121 an 7

cours n°7 --- Réflexivité

Avant compilation: des classes !



Des classes et des relations entre classes sont définies par édition de texte

Ce texte sera compilé

Après compilation: des objets !

v0 : C

v : C1

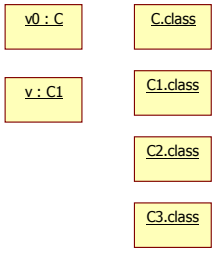
Au 'runtime'

Il n'existe que des objets de ces classes

C0 v0 = new C1();

C1 v;

Dont un objet spécifique pour chaque classe chargée !



Parmi les objets figure un objet-Class pour chaque classe chargée.

Cela vaut pour

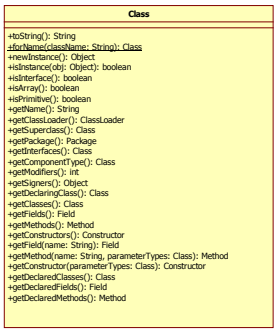
String.class par exemple

ou

Object.class

-4-

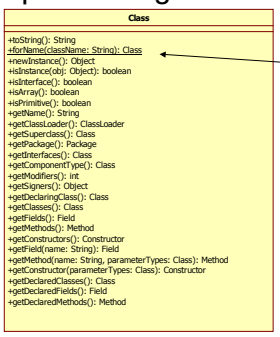
Cet objet est de classe Class



Il est construit
au chargement d'une classe
dans la JVM

-5-

Une méthode de classe pour charger de nouvelles classes



public static Class
forName(String className)
throws ClassNotFoundException

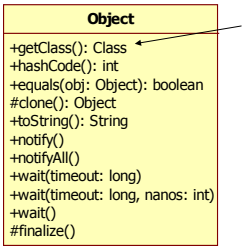
Retourne l'objet-Class associé au type dont le nom est fourni.
(nom qualifié)

Exemple

```
Class t =
    Class.forName("java.lang.Thread")
```

-6-

Tout objet peut être interrogé sur sa classe



A l'exécution il est possible de demander à tout objet quelle est sa classe

Exemple

v.getClass()==C1.class

String.class.getClass() == Class.class

Informations sur le type à l'exécution

- instanceof

```
if (g instanceof Graphics2D)
{

}
```

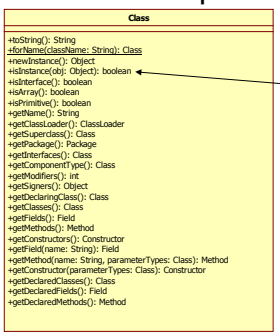
- isInstance

```
if ((Graphics2D.class).isInstance(instanceObject))
{

}
```

- getClass

```
if (g.getClass() == Graphics2D.class)
{

}
```


L'objet de classe Class lui-même peut-être interrogé



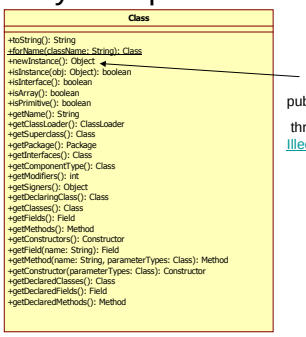
boolean isInstance(Object obj)

Cette méthode permet de savoir si obj est d'une sous-classe de la classe dont l'objet-Class est ici interrogé.

Exemple

v0.getClass().isInstance(v)

Il peut aussi servir à créer dynamiquement un nouvel objet



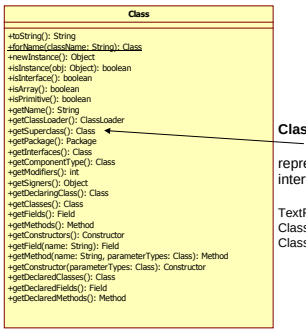
public Object **newInstance**()

throws InstantiationException, IllegalAccessException

Simuler l'opérateur instanceOf

```
class A {}

public class instance1 {

    public static void main(String args[]) {
        try {
            Class cls = Class.forName("A");
            boolean b1 = cls.isInstance(new Integer(37));
            System.out.println(b1);
            boolean b2 = cls.isInstance(new A());
            System.out.println(b2);
        } catch (Throwable e) {
                System.err.println(e);
        }
    }

}
```

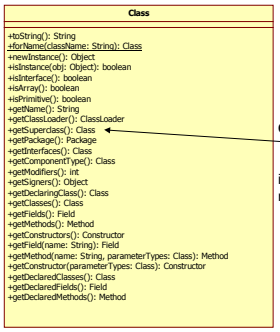

Récupérer les super classes



Class getSuperclass()
retourne l'objet class représentant la super classe (class, interface, primitive type or void)

```
TextField t = new TextField();
Class c = t.getClass(); // TextField
Class s = c.getSuperclass(); //TextComponent
```

Récupérer les interfaces implémentés



Class[ ] getInterfaces()

Determines the interfaces implemented by the class or interface represented by this object.

s.getClass().getInterfaces()[0]

-13-

Afficher les interfaces implémentées par un objet

```
static void printInterfaceNames(Object o) {
    Class c = o.getClass();
    Class[] theInterfaces = c.getInterfaces();
    for (int i = 0; i < theInterfaces.length; i++) {
        String interfaceName = theInterfaces[i].getName();
        System.out.println(interfaceName);
    }
}
```

-14-

pourquoi introduire plus de capacités introspectives ?

- Une industrie du composant logiciel
- Des ateliers d'intégrations de composants
- Des outils de 'customisation' d'applications

-15-

Qu'est-ce que la réflexivité ?

C'est ce qui permet à un objet d'obtenir des informations sur sa propre structure
et sur le traitement qu'il subit

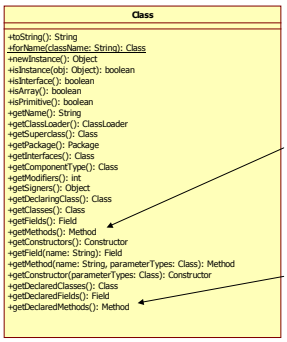
Lorsqu'un langage supporte la réflexivité des méta données sont générées à la compilation et mises à jour à l'exécution.

C'est un outil puissant permettant d'assembler du code à l'exécution sans que soit nécessaire la disposition du code source.

java.lang.reflect.*

- java.lang: Class and Object
- java.lang.reflect: Constructor, Field, Method, Array
  - Constructeurs, Attributs, et Methodes
  - Array contient un ensemble de méthodes statiques pour la création

Les méthodes d'une classe



Method[ ] getMethods()
Returns an array containing Method objects reflecting all the public *member* methods of the class or interface represented by this Class object, including those declared by the class or interface and those inherited from superclasses and superinterfaces.

Method[ ] getDeclaredMethods()
Returns an array of Method objects reflecting all the methods declared by the class or interface represented by this Class object.

java.lang.reflect.Method

Method

+getDeclaringClass(): Class
+getName(): String
+getModifiers(): int
+getReturnType(): Class
+getParameterTypes(): Class
+getExceptionTypes(): Class
+equals(obj: Object): boolean
+hashCode(): int
+toString(): String
+invoke(obj: Object, args: Object): Object

Appeler une méthode

Method

+getDeclaringClass(): Class
+getName(): String
+getModifiers(): int
+getReturnType(): Class
+getParameterTypes(): Class
+getExceptionTypes(): Class
+equals(obj: Object): boolean
+hashCode(): int
+toString(): String
+invoke(obj: Object, args: Object): Object

Object invoke(Object obj, Object[] args)
appelle cette méthode sur obj avec les paramètres args

Récupérer les paramètres

Method

+getDeclaringClass(): Class
+getName(): String
+getModifiers(): int
+getReturnType(): Class
+getParameterTypes(): Class
+getExceptionTypes(): Class
+equals(obj: Object): boolean
+hashCode(): int
+toString(): String
+invoke(obj: Object, args: Object): Object

Class[ ] getParameterTypes()

retourne un tableau d'objets de classe Class représentant le type des paramètres formels de cette méthode

Exemple

```
static void showMethods(Object o) {
    Class c = o.getClass();
    Method[] theMethods = c.getMethods();
    for (int i = 0; i < theMethods.length; i++) {
        String methodString = theMethods[i].getName();
        System.out.println("Name: " + methodString);
        String returnString = theMethods[i].getReturnType().getName();
        System.out.println(" Return Type: " + returnString);
        Class[] parameterTypes = theMethods[i].getParameterTypes();
        System.out.print(" Parameter Types:");
        for (int k = 0; k < parameterTypes.length; k++) {
            String parameterString = parameterTypes[k].getName();
            System.out.print(" " + parameterString);
        } System.out.println(); }
    }
}
```

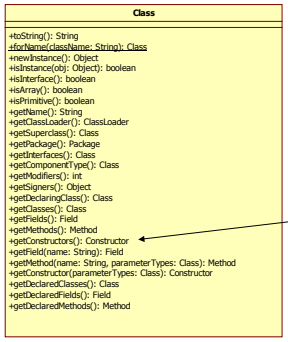
-22-

Incrementing a JavaBean property by reflection

```
public int incrementProperty(String name, Object obj) {
    String prop = Character.toUpperCase(name.charAt(0)) + name.substring(1);
    String mname = "get" + prop;
    Class[] types = new Class[] {};
    Method method = obj.getClass().getMethod(mname, types);
    Object result = method.invoke(obj, new Object[0]);
    int value = ((Integer)result).intValue() + 1;
    mname = "set" + prop;
    types = new Class[] { int.class };
    method = obj.getClass().getMethod(mname, types);
    method.invoke(obj, new Object[] { new Integer(value) });
    return value;
}
```

-23-

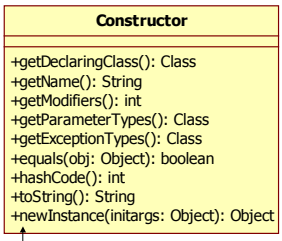
Obtenir les constructeurs



Retourne le
ou les constructeurs l'objet .class
correspondant

-24-

Créer un nouvel objet



Object newInstance(Object[] initargs)
crée un objet et l'initialise avec les arguments

Listing 1.

```
public class TwoString {
      private String m_s1, m_s2;
      public TwoString(String s1, String s2) {
            m_s1 = s1; m_s2 = s2;
      }
}
```

Le code du Listing 2 récupère le constructeur et l'utilise pour creer une instance de la classe TwoString utilisant les Strings "a" et "b":

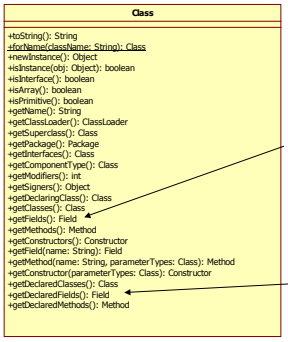
Listing 2. Appel du constructeur par réflexion

```
Class[] types = new Class[] { String.class, String.class };
Constructor cons = TwoString.class.getConstructor(types);
Object[] args = new Object[] { "a", "b" };
TwoString ts = cons.newInstance(args);
```


Exemple

```
/** * Affiche les paramètres des différents constructeurs publiques d'une classe * */
static void showConstructors(Object o) {
      Class c = o.getClass();
      Constructor[] theConstructors = c.getConstructors();
      for (int i = 0; i < theConstructors.length; i++) {
            System.out.print("( ");
            Class[] parameterTypes = theConstructors[i].getParameterTypes();
            for (int k = 0; k < parameterTypes.length; k++) {
                  String parameterString = parameterTypes[k].getName();
                   System.out.print(parameterString + " ");
            } System.out.println(")");
      }
}
```

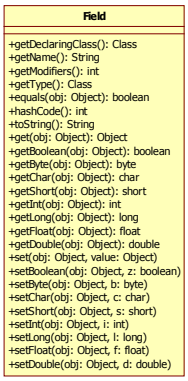
Obtenir les attributs d'une classe



Field[ ] getFields()
Returns an array containing Field objects reflecting all the accessible public fields of the class or interface represented by this Class object.

Field[ ] getDeclaredFields()
Returns an array of Field objects reflecting all the fields declared by the class or interface represented by this Class object.

-28-



java.lang.reflect.Field

Récupérer la valeur d'un champ de obj

Affecter la valeur d'un champ de obj

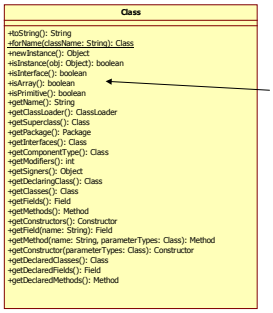
-29-

Incrémenter un attribut

```
public int incrementField(String name, Object obj) throws... {
    Field field = obj.getClass().getDeclaredField(name);
    int value = field.getInt(obj) + 1;
    field.setInt(obj, value);
    return value;
}
```

-30-

Les tableaux



boolean isArray()
Determines if this Class object represents an array class.

-31-

Array

Object get(Object array, int index)

retourne la valeur située à index dans array

-32-

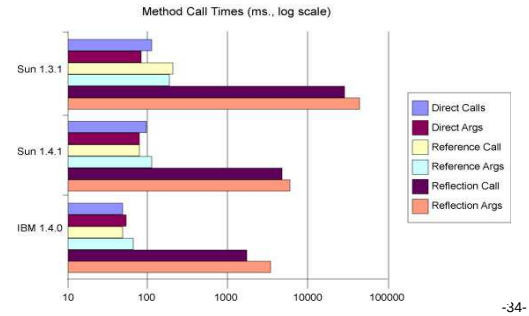
Agrandir un tableau

```
public Object growArray(Object array, int size) {
    Class type = array.getClass().getComponentType();
    Object grown = Array.newInstance(type, size);
    System.arraycopy(array, 0, grown, 0,
                     Math.min(Array.getLength(array), size));
    return grown;
}
```

-33-

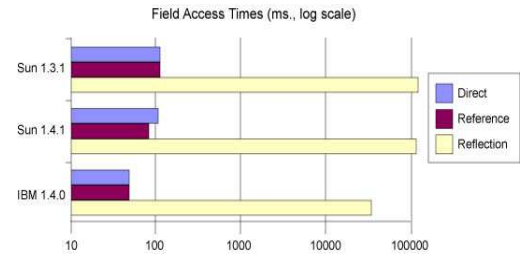
Performance

Beaucoup d'avantages mais pas efficace car interprété (au second niveau !)



-34-

Performance



-35-