La classe Class

```
+ toString() : String
+ forName(className : String) : Class
+ newInstance() : Object
+ isInstance(obj : Object) : boolean
+ isInterface() : boolean
+ getName() : String
+ …
```

La classe Object

```
+ getClass() : Class
+ equals(o : Object) : boolean
# clone() : Object
+ toString() : String
+ getName() : String
+ notify()
+ notifyAll()
+ wait()
+ wait(timeOut : long)
# finalize()
```

On peut …

- Charger une nouvelle classe
  public static Class **forName**(String className)
  throws ClassNotFoundException
- Interroger un objet sur sa classe
  v.getClass()==Vetement.class
  String.class.getClass() == Class.class
- Interroger l'objet de classe Class lui-même
  boolean **isInstance**(Object obj)
- Créer un nouvel objet
  public Object **newInstance**()
  throws InstantiationException, IllegalAccessException

? type

- *instanceof*
  *if (v **instanceof** Vetement) { … }*
- isInstance
  if ((Vetement.class).isInstance(v)) { … }
- getClass
  if (v.getClass() == Vetement.class) { … }

Application

```
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);
    }
  }
}
```

? classe héritée

- Dans Class : **Class getSuperclass**()
  retourne l'objet
  représentant la super classe (class, interface, type primitif ou void)

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

? interfaces implantées

- Dans Class : **Class[ ] getInterfaces**()
- Quelles sont les interfaces implantées par cette classe ?
  - s.getClass().getInterfaces()[0]
  - s.getClass().getInterfaces()[1]
  - …

Interfaces implantées par un objet

```
static void printInterfaceNames(Object o) {
  Class<?> c = o.getClass();
  for (Class<?> i : c.getInterfaces() ) {
    String interfaceName = i.getName();
    System.out.println(interfaceName);
  }
}
```

? Méthodes d'une classe

- Dans Class :
  - **Method[ ] getDeclaredMethods**()
    renvoie un tableau d'objets Method qui est le reflet des méthodes déclarées par la classe ou l'interface de cette classe
  - **Method[ ] getMethods**()
    renvoie un tableau d'objets Method qui est le reflet de toutes les méthodes déclarées publiques par la classe ou l'interface de cette classe et les classes héritées

La classe `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

Invoquer une méthode

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

Invoquer une méthode

```
Object lancerMethode(Object o, Object[] args,
                     String nomMethode)
  throws Exception {
      Class[] paramTypes = null;
      if(args != null) {
            paramTypes = new Class[args.length];
            for(int i=0;i<args.length;++i) {
              paramTypes[i] = args[i].getClass();
            }
      }
      Method m = o.getClass().getMethod(nomMethode,
                                        paramTypes);
      return m.invoke(o,args);
}
```

? Paramètres d'une méthode

- **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();
  for (Method m : c.getMethods()){
    String methodString = m.getName();
    System.out.println("Name: " + methodString);
    String returnString = m.getReturnType().getName();
    System.out.println(" Return Type: " + returnString);
    System.out.print(" Parameter Types:");
    for (Class<?> p : m . getParameterTypes() ) {
      String parameterString = p.getName();
      System.out.print(" " + parameterString);
    } System.out.println(); }
  }
}
```

? constructeurs d'une classe

- Dans Class :
  Constructor[ ] getContructors()
  retourne un tableau d'objets de classe Constructor représentant les constructeurs de l'objet .class correspondant

Exemple

```
static void showConstructors(Object o) {
  Class c = o.getClass();
  for (Constructor cons : c.getConstructors()) {
    System.out.print("( ");
    for (Class p : cons.getParameterTypes()) {
      String parameterString = p.getName();
      System.out.print(parameterString + " ");
    }
    System.out.println(")");
  }
}
```

Créer un nouvel objet

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

Création par introspection

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

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);
```

? Attributs d'une classe

- Dans Class :
  - **Field[ ] getDeclaredFields**()
    tous les champs de cette classe
  - **Field[ ] getFields**()
    tous les champs publics accessibles

La classe Field

+getDeclaringClass(): Class
+getName(): String
+getModifiers(): int
+getType(): Class
+equals(obj: Object): boolean
+hashCode(): int
+toString(): String
+get(obj: Object): Object
+getBoolean(obj: Object): boolean
+getByte(obj: Object): byte
+getChar(obj: Object): char
+getShort(obj: Object): short
+getInt(obj: Object): int
+getLong(obj: Object): long
+getFloat(obj: Object): float
+getDouble(obj: Object): double
+set(obj: Object, value: Object)
+setBoolean(obj: Object, z: boolean)
+setByte(obj: Object, b: byte)
+setChar(obj: Object, c: char)
+setShort(obj: Object, s: short)
+setInt(obj: Object, i: int)
+setLong(obj: Object, l: long)
+setFloat(obj: Object, f: float)
+setDouble(obj: Object, d: double)

Affecter un attribut

```
void changeValeur(Object o, String nomChamp,
                  Object val) throws Exception {
  Field f = o.getClass().getField(nomChamp);
  f.set(o,val);
}
```

Si ce champ est privé c.f. `f.setAccessible(true)` !

Les tableaux

- Dans Class :
  boolean isArray()
- Dans Array :
  Object get(Object array, int index)
  retourne la valeur située à index dans array

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;
}
```