ensiie

école nationale supérieure d'informatique pour l'industrie et l'entreprise

Classes et Interfaces utiles en Java

Documentation Java

- Sur le Web :
  - Java 5 (ou 1.5) : http://java.sun.com/j2se/1.5.0/docs/...
    - ...api/index.html
  - Java 6 (ou 1.6) : http://java.sun.com/javase/6/docs/
- à l'ENSIIE (1.5)
  - http://www.ensiie.fr/intranet/doc/java/docs/index.html

La classe Object

- En Java la classe Object est la classe mère de toutes les classes
  - Toute nouvelle classe dérive implicitement de la classe Object : Arbre d'héritage.
  - La classe Object peut donc être utilisée par la généricité grâce au polymorphisme d'héritage.
    - Exemple : la classe Vector stocke une collection de références vers des Object.
      - Avantages :
        - » généricité simple à gérer.
      - Inconvénients :
        - » Hétérogénéité possible des collections
        - » Obligation d'utiliser l'introspection pour contrôler les types si l'on souhaite n'avoir qu'un seul type d'instances dans une collection.
  - La classe Object ne contient pas de données mais définit un certain nombres de méthodes utiles.
    - Ces méthodes peuvent et/ou doivent être surchargées par les classes filles.

Les méthodes de la classe Object

- protected Object clone()
  - Pour créer une copie de soi-même au sens du contenu (deep copy)
  - `x.clone() != x` ⇒ true (deux instances distinctes)
  - `x.clone().equals(x)` ⇒ true (deux instances identiques en terme de contenu)
- public boolean equals(Object o)
  - Pour tester l'égalité de soi-même avec une autre instance
    - Au sens du contenu (deep compare) et non plus seulement au sens des références (shallow compare).
  - Réflexivité : `x.equals(x)` ⇒ true; (au sens du contenu)
  - Symétrie : `x.equals(y)` ⇒ true; (si x et y sont de même type effectif et que leur contenu est indentique)
  - Transitivité : si `x.equals(y)` == true et `y.equals(z)` == true alors `x.equals(z)` == true.

Les méthodes de la classe Object

- **`protected void finalize() throws Throwable`**
  - Permet le nettoyage de l'instance avant garbage collecting : appelée par le garbage collector quand il n'y a plus de références pointant vers cette instance.
- **`public Class<? extends Object> getClass()`**
  - Permet d'obtenir une instance de la classe Class correspondant au type effectif de l'instance interrogée (Introspection).
- **`public int hashCode()`**
  - Renvoie un code de hashage (unique) pour l'objet au sens du contenu.
  - Utilisé dans les `class Hashtable<K,V>`
  - Si `x.equals(y)==true` alors `x.hashcode()==y.hashCode()`.

Les méthodes de la classe Object

- **`public String toString()`**
  - Permet de créer une chaîne de caractères représentant l'objet.
  - Utilisé par les méthodes d'affichages comme `System.out.println(Object o)`.
- Méthodes spécifiques utilisées par les Threads
  - `public final void notify()`
  - `public final void notifyAll()`
  - `public final void wait()`
  - `public final void wait(long timeout)`
  - `public final void wait(long timeout, int nanos)`

La classe Class<T>

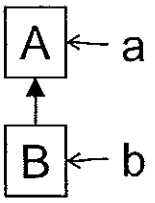
- Les instances de la classe Class représentent les classes et les interfaces dans une application Java en cours d'exécution.
  - Toute classe ou interface instanciée dans une application a donc sa propre instance de la classe Class.
  - Exemple :

```
MonObjet o = new MonObject();
Integer e = new Integer(3);
Class class = o.getClass();
class.equals(e.getClass()) ⇒ false
```

  - Les types simples (`boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`), et le mot clé `void` peuvent aussi être représentés par des instances de la classe Class.
  - Obtention des instances de la classe Class
    - Pas de constructeurs
    - Au travers de la méthode `getClass()` de la classe Object
    - Au travers du literal « class » : `Type.class;`

Utilisation de la classe Class

- A partir du langage lui-même
  - `if (b instanceof A) {...}` ⇒ true
  - `if (a instanceof B) {...}` ⇒ false



- En utilisant les méthodes de la classe Class
  -
```
Class ca = a.getClass();
Class cb = b.getClass();
if (ca.equals(cb)) { ⇒ false
  System.out.println("de la même classe " + ca.get[Simple]Name());
}
```
  - `if (ca.isInstance(b)) {...}` ⇒ true
  - `if (B.class.isInstance(a)) {...}` ⇒ false
  -
```
Object monObject = new Object();
if (Class.forName("java.lang.Object").isInstance(monObject))
```
    ⇒ true

Le package java.lang

- Interfaces
  - *Appendable*
    - xxxWriter(s), …, PrintStream, StringBuffer, …
  - *CharSequence*
    - CharBuffer, String, StringBuffer, StringBuilder
  - *Cloneable*
  - *Comparable<T>*
    - Toutes les classes représentant des types scalaires : Integer, Boolean, Double, etc
  - *Iterable<T>*
    - *Toutes les interfaces des collections*
    - Toutes les collections
  - *Readable*
    - xxxReader(s),
  - *Runnable*
- Classes
  - Types scalaires : Number et classes filles : Byte, Boolean, Character, Double, Float, Integer, Long…
  - Class<T>
    - Introspection
  - Math
    - Fonctions mathématiques
  - String, StringBuffer
    - Chaînes de caractères
  - System
  - Thread

Le package java.util

- Interfaces
  - *Collection<E>*
    - *List<E>, Queue<E>, Set<E>*
    - AbstractCollection<E>, …, Vector<E>
  - *Comparator<E>*
    - *comparaison*
  - *Enumeration<E>*
    - Énumération des éléments d'une collection
  - *Iterator<E>*
    - Itération sur une collection
  - *List<E>*
    - *AbstractList, ArrayList, …*
  - *Map<K,V>*
    - *AbstractMap, HashMap, TreeMap, HashTable*
  - *Queue<E>*
    - File d'attente
  - *Set<E>*
    - *Ensemble d'éléments sans doublons*
- Classes
  - Implémentations des interfaces
    - Abstract[Collection,List,Queue]<E>
    - [AbstractMap,Dictionary]<K,V>
  - Calendar, Date
  - Formatter
  - HashXxx
    - Hash[Map<K,V>,Set<E>,table<K,V>]
  - LinkedXxx
    - Linked[HashMap<K,V>, HashSet<E>, List<E>]
  - Locale
  - Observable
  - Stack<E>
  - Vector<E>