Fin de vie d'un objet

Un objet est en fin de vie lorsque le programme n'en a plus besoin

☞ la référence qui lui est associée n'est plus utilisée nulle part

Exemple

```
class Exemple
{
   public static void main(String[] args) {
         // autres traitements
         afficherUnRectangle();
         //...
   }

   static void afficherUnRectangle() {
         Rectangle r = new Rectangle(3.0, 4.0);
         System.out.println(r);
   }
}
```

☞ Récupération de la mémoire associée à l'objet

Garbage Collection

- **Garbage collection** = processus qui récupère la mémoire occupée par les objets qui ne sont plus référencés par aucune variable
  - Egalement appelé « ramasse-miettes »
  - Nécessaire pour éviter les fuites de mémoire : indisponibilité de zones mémoires qui ne sont plus utilisées
- Le *garbage collector* est lancé régulièrement pendant l'exécution d'un programme Java

☞ De façon générale, le programmeur Java n'a pas à libérer explicitement la mémoire utilisée.

Affectation & copie d'objets

Supposons que l'on souhaite :

- créer un objet `b` à partir d'un autre objet `a` du même type;
- assurer que `a` et `b` soient deux objets **distincts** en mémoire

```
Rectangle r1 = new Rectangle(12.3, 24.5);
Rectangle r2 = r1;
```

`r1` et `r2` référencent ici le **même objet**
et non pas deux copies distinctes du même objet :
toute modification via `r2` modifiera également `r1`.

Affectation & copie d'objets (2)

Si l'on veut que l'objet référencé par `r2` soit une copie distincte de celui référencé par `r1`, il ne faut pas utiliser l'opérateur `=` mais plutôt un constructeur de copie (ou la méthode `clone` qui sera vue plus tard) :

```
Rectangle r1 = new Rectangle(12.3, 24.5);
Rectangle r2 = new Rectangle(r1);
```

Affichage d'objets

La portion de code suivante :

```
Rectangle rect = new Rectangle(1.0, 2.0);
System.out.println(rect);
```

afficherait la valeur de la référence. Par exemple : `A@29c2fff0` !

Que faire si l'on souhaite faire afficher le contenu de l'objet en utilisant exactement le même code ?

Java prévoit que vous fournissiez une méthode qui retourne une représentation de l'instance sous forme d'une `String`.

Il est prévu que vous donniez un entête particulier à cette méthode :

`String toString()`

La méthode `toString` est invoquée automatiquement par `System.out.println`

Affichage d'objets : la méthode `toString`

Exemple :

```
class Rectangle
{
  private double hauteur;
  private double largeur;
  //...
  public String toString()
  {
    return "Rectangle " + hauteur + " x " + largeur;
  }
}
class Exemple {
  public static void main(String[] args) {
      System.out.println(new Rectangle(4.0, 5.0));
  }
}
```

affiche : `Rectangle 4.0 x 5.0`

Comparaison d'objets

```
//.. par exemple dans la méthode main()

Rectangle r1 = new Rectangle(4.0, 5.0);
Rectangle r2 = new Rectangle(4.0, 5.0);

if (r1 == r2) {
 System.out.println("Rectangles identiques");
}
```

Comparaison d'objets (2)

L'opérateur `==` appliqué à deux objets compare les références de ces objets.
Souvenez-vous des `String` :

```
String s1 = "Rouge";
String s2 = "Rou" + "ge";

if (s1.equals(s2)) {
 System.out.println("Chaînes identiques");
}
```

Que faire si l'on souhaite comparer le contenu de deux objets de type `Rectangle` ?

☞ il faut fournir une méthode qui fasse le test selon les critères qui semblent sensés pour les objets de type `Rectangle`.

Java prévoit que vous puissiez définir cette méthode, par exemple avec l'entête suivant : `boolean equals(Rectangle arg)`

Comparaison d'objets : méthode `equals`

Exemple :

```
class Rectangle
{
  private double hauteur;
  private double largeur;
  //...
  public boolean equals(Rectangle autre)
  {
    if (autre == null) {
      return false;
    } else {
      return (   hauteur == autre.hauteur
              && largeur == autre.largeur);
    }
}
```

Comparaison d'objets : méthode `equals` (2)

Exemple (suite) :

```
//.. par exemple dans la méthode main()

Rectangle r1 = new Rectangle(4.0, 5.0);
Rectangle r2 = new Rectangle(4.0, 5.0);

if (r1.equals(r2)) {
 System.out.println("Rectangles identiques");
}
```

Comparaison d'objets : méthode `equals` (3)

Attention, deux entêtes sont possibles pour la méthode `equals` :

- `boolean equals(`*`UneClasse`*` c)`
- `boolean equals(Object c)`

☞ Nous reviendrons sur cela en temps voulu