Annexe 8 - Notes de cours : comparaisons d'objets

1. Égalité entre des valeurs de types prédéfinis :

C'est simple, on emploie comme à l'habitude l'opérateur ==.

Ex. :
```
int a = 9;
int b = 9;
if ( a == b ) → return true
```

2. Égalité entre 2 objets (autres que String)

- il faut d'abord distinguer si on parle de deux objets ou de deux références à un objet; des références étant des adresses indiquant où se trouvent les variables et les méthodes d'un objet donné.

Ex.:
```
Point p1, p2;
p1 = new Point ( 100, 100 );
p2 = p1;
```

- dans cet exemple, un seul objet est créé mais il y a deux références à cet objet (p1 et p2) Ainsi, toute modification faite sur cet objet à partir d'une ou l'autre des références aura un impact sur l'objet.

Ex.:
```
p1.setX( 200 );
p1.getX() → 200
p2.getX() → 200 également puisqu'on parle d'un seul objet
```

- l'opérateur == retournera true uniquement si on compare des références à un même objet

Ex.: `if(p1==p2)` → return true car ce sont deux références à un même objet

Ex.:
```
Point p3 = new Point ( 15,15 );
Point p4 = new Point ( 15,15 );
if ( p3 == p4 ) → return false ( ce ne sont pas deux références à un même objet. )
```

- Pour comparer deux objets entre eux, il faut vérifier si toutes les variables d'instance sont égales. La méthode `equals` est définie dans certaines classes pour réaliser cette tâche (classes `String`, `Color`, `Font`, `Point`, `Calendar`, `Hashtable`, `Vector`, etc.)

Ex.:
```
Point p5 = new Point ( 20,20 );
Point p6 = new Point ( 20,20 );
if ( p5.equals ( p6 ) ) → return true . ( On a employé equals car on avait affaire à deux objets différents )
```

- Si le type des objets à comparer ne définit pas la méthode equals, on doit la redéfinir (la coder) nous-mêmes. Attention, l'appel de la méthode equals sur des objets où elle n'est pas redéfinie fonctionnera mais il s'agira de la méthode equals de la classe Object (héritage). L'emploi de cette méthode est équivalent à == dans ces cas.

Ex.:
```
Roi r1 = new Roi ("k", "noir" );
Roi r2 = new Roi ("k", "noir" );
```

`if ( r1.equals(r2) )` → return false car la méthode `equals` n'a pas été redéfinie dans la classe `Roi` ou `Piece`, on est donc en train d'utiliser la méthode `equals` de la classe `Object`.

3. Egalité entre 2 objets (`Strings`)

- Le cas des Strings est particulier. Comme on l'a vu, ce type est mi-prédéfini, mi-objet. Le choix d'utiliser == ou equals pour les comparer réside dans la définition des Strings.

Ex. :
```
String s = "bonjour"; // chaîne littérale
String t = new String ("bonjour" ); // forme objet
```

- Dans le cas des chaînes littérales, on peut utiliser l'opérateur == en autant qu'on veuille comparer deux références à des chaînes littérales. On peut utiliser == car les chaînes littérales sont automatiquement associées à un même objet à l'interne lorsque leurs valeurs sont égales (procédé de l'interning).

Ex. :
```
String a = "allo";
String b = "allo";
if ( a == b )
```
→ return true // 2 chaînes littérales donc associées au même objet

```
String a = "allo";
if ( a == "allo" )
```
→ return true // idem

```
String a = "allo";
String b = champTexte.getText(); // contenant "allo"
if ( a==b )
```
→ return false car b n'est pas une chaîne littérale

```
String a = "allo";
String b = new String ("allo");
if (a==b)
```
→ return false car b n'est pas une chaîne littérale

- pour comparer deux formes objets ou une forme objet avec une forme littérale, on doit utiliser la méthode `equals`.

Ex.:
```
String a = "allo";
String b = new String ("allo");
if ( a.equals(b) )
```
→ return true

- finalement, on peut faire de l'interning sur des formes objets String en appelant la méthode `intern`()

Ex.:
```
String a = "allo";
String b = new String ("allo");
b.intern();
if ( a.equals (b) ) → return true
if ( a == b ) → return true ( elle est devenue littérale )
```