Constructeurs et héritage

Lors de l'instanciation d'une sous-classe, il faut initialiser :

- les attributs *propres à la sous-classe*
- les attributs *hérités des super-classes*

MAIS...

...il ne doit pas être à la charge du concepteur des sous-classes de réaliser lui-même l'*initialisation des attributs hérités*

L'accès à ces attributs pourrait notamment être interdit ! (`private`)

L'initialisation des attributs hérités doit donc se faire au niveau des classes où ils sont explicitement définis.

Solution : l'initialisation des attributs hérités doit se faire en **invoquant les constructeurs des super-classes**.

Constructeurs et héritage : appel explicite

L'invocation du constructeur de la super-classe se fait au tout début du corps du constructeur au moyen du mot réservé `super` .

Syntaxe :

```
SousClasse(liste de paramètres)
{
     /* Arguments : liste d'arguments attendus par
      * un des constructeurs de la super-classe de SousClasse
      */
    super(Arguments);
    // initialisation des attributs de SousClasse ici
}
```

Lorsque la super-classe admet un constructeur par défaut, l'invocation explicite de ce constructeur dans la sous-classe n'est pas obligatoire

☞ le compilateur se charge de réaliser l'invocation du constructeur par défaut

Constructeurs et héritage : exemple 1

Si la classe parente n'admet pas de constructeur par défaut, l'**invocation explicite** d'un de ses constructeurs **est obligatoire** dans les constructeurs de la sous-classe

☞ La sous-classe doit admettre *au moins un constructeur explicite*.

Exemple :

```
class FigureGeometrique {
  private Position position;
  public FigureGeometrique(double x, double y) {
    position = new Position(x,y);
  }
  // ...
}
class Rectangle extends FigureGeometrique {
  private double largeur;
  private double hauteur;
  public Rectangle(double x, double y, double l, double h) {
    super(x,y);
    largeur = l;  hauteur = h;
  } // ...
}
```

Constructeurs et héritage : exemple 2

Autre exemple (qui ne fait pas la même chose) :

```
class FigureGeometrique {
   private Position position;
   public FigureGeometrique() { position = new Position(0.0, 0.0); }
}
class Rectangle extends FigureGeometrique {
   private double largeur;
   private double hauteur;
   public Rectangle(double l, double h) {
      largeur = l;
      hauteur = h;
   }
  // ...
}
```

Encore un exemple

Il n'est pas nécessaire d'avoir
des attributs supplémentaires...

```
class Carre extends Rectangle {
  public Carre(double taille) {
    super(taille, taille);
  }
  /* Et c'est tout !
     (sauf s'il y avait des manipulateurs, il
     faudrait alors sûrement aussi les redéfinir) */
}
```

Constructeurs et héritage : résumé (1)

1. Chaque constructeur d'une sous-classe *doit* appeler `super(...)`
2. Les arguments founis à `super` doivent être ceux d'au moins un des constructeur de la super-classe.
3. L'appel doit être la toute **1re instruction**
4. Erreur si l'appel vient plus tard ou 2 fois
5. Aucune autre méthode ne peut appeler `super(...)`

Constructeurs et héritage : résumé (2)

Et si l'on oublie l'appel à `super(...)` ?

- Appel automatique à `super()`
- Pratique parfois, mais erreur si le constructeur par défaut n'existe pas

Rappel : le constructeur par défaut est particulier

- Il existe par défaut pour chaque classe qui n'a aucun autre constructeur
- Il disparaît dès qu'il y a un autre constructeur

Pour éviter des problèmes avec les hiérarchies de classes, dans un premier temps :

- Toujours déclarer au moins un constructeur
- Toujours faire l'appel à `super(...)`

Ordre d'appel des constructeurs