Où en est-on ?

Dans les vidéos précédentes, nous avons vu comment déclarer des classes et des objets.

On peut par exemple déclarer une instance de la classe `Rectangle` :

```
Rectangle rect;
```

Une fois que l'on a fait cette déclaration, comment faire pour donner aux attributs de `rect` *des valeurs autres que* `0.0`

```
class Rectangle {
  private double hauteur;
  private double largeur;

  public double surface()
    { return hauteur * largeur; }
  public double getHauteur()
    { return hauteur; }
  public double getLargeur()
    { return largeur; }
  public void setHauteur(double h)
    { hauteur = h; }
  public setLargeur(double l)
    { largeur = l; }
}
```

Initialisation des attributs

Première solution : **affecter individuellement une valeur** à chaque attribut

```
Rectangle rect;

System.out.println("Quelle hauteur ? ");
lu = clavier.nextDouble();
rect.setHauteur(lu);

System.out.println("Quelle largeur ? ");
lu = clavier.nextDouble();
rect.setLargeur(lu);
```

Ceci est une *mauvaise solution* dans le cas général :

- elle implique que tous les attributs fassent partie de l'interface (`public`) ou soient assortis d'un manipulateur ☞ casse l'encapsulation
- oblige le programmeur-utilisateur de la classe à initialiser explicitement tous les attributs ☞ risque d'oubli

Initialisation des attributs (2)

Deuxième solution : définir une **méthode dédiée à l'initialisation** des attributs

```
class Rectangle {
  private double hauteur;
  private double largeur;

  public void init(double h, double l)
  {
    hauteur = h;
    largeur = l;
  }
  //...
}
```

Pour faire ces initialisations, il existe en Java des méthodes particulières appelées **constructeurs**.

Les constructeurs

Un constructeur est une méthode :

- invoquée *systématiquement* lors de la déclaration d'un objet
- chargée d'effectuer toutes les opérations requises en « début de vie » de l'objet (dont *l'initialisation des attributs*)

Syntaxe de base :

```
NomClasse(liste_paramètres)
{
 /* initialisation des attributs
    en utilisant liste_paramètres */
}
```

Exemple :

```
Rectangle(double h, double l)
{
  hauteur = h;
  largeur = l;
}
```

Les constructeurs (2)

Les constructeurs sont des méthodes presque comme les autres. Les différences sont :

- *pas* de type de retour (pas même `void`)
- *même nom* que la classe
- invoqués *systématiquement* à chaque fois qu'une instance est créée.

```
Rectangle(double h, double l)
{
  hauteur = h;
  largeur = l;
}
```

Comme les autres méthodes :

- les constructeurs peuvent être surchargés

(exemples dans la suite)

Une classe peut donc avoir **plusieurs constructeurs**, pour peu que leur liste de paramètres soit différente.

Notre programme (1/3)

```
class Rectangle {
  private double hauteur;
  private double largeur;

  public Rectangle(double h, double l)
  {
    hauteur = h;
    largeur = l;
  }
  public double surface()
    { return hauteur * largeur; }
  // accesseurs/modificateurs si nécessaire
  // ...
}
```

Initialisation par constructeur

La *déclaration avec initialisation* d'un objet se fait selon la syntaxe suivante :

Syntaxe :

NomClasse instance = new *NomClasse*(*valarg1*, ..., *valargN*);

où *valarg1*, ..., *valargN* sont les valeurs des arguments du constructeur.

Exemple :

```
Rectangle r1 = new Rectangle(18.0, 5.3); // invocation du constructeur à 2 paramètres
```

Notre programme (2/3)

```
class Rectangle {
  private double hauteur;
  private double largeur;

  public Rectangle(double h, double l)
  {
      hauteur = h;
      largeur = l;
  }
  public double surface()
    { return hauteur * largeur; }
  // ...
}

class Exemple {
  public static void main(String[] args) {
    Rectangle rect1 = new Rectangle(3.0, 4.0);
      // ...
  }
}
```

Constructeur par défaut

Le constructeur par défaut est un constructeur qui **n'a pas de paramètre**

Exemple :

```
// Le constructeur par defaut
Rectangle() { hauteur = 1.0; largeur = 2.0; }

// 2ème constructeur
Rectangle(double c) { hauteur = c; largeur = 2.0*c; }

// 3ème constructeur
Rectangle(double h, double l) { hauteur = h; largeur = l; }
```

Constructeur par défaut par défaut

Si aucun constructeur n'est spécifié, le compilateur *génère automatiquement* une **version minimale du constructeur par défaut**
qui :
initialise les attributs avec les valeurs par défaut : `0`, `0.0`, et `false` pour les types de base et `null` pour les objets.

Dès qu'**au moins un constructeur a été spécifié**, ce constructeur par défaut par défaut *n'est plus fourni*.

Si donc on spécifie *un* constructeur sans spécifier de constructeur par défaut, on ne peut plus construire d'objet de cette classe sans les initialiser explicitement (ce qui est voulu !) puisqu'il n'y a plus de constructeur par défaut.

A :

```
class Rectangle {
  private double hauteur;
  private double largeur;
  // suite ...
}
```

B :

```
class Rectangle {
  private double hauteur;
  private double largeur;

  public Rectangle()
  {
    hauteur = 0.0;
    largeur = 0.0;
  }
  // suite ...
}
```

C :

```
class Rectangle {
  private double hauteur;
  private double largeur;

  public Rectangle(double h,
                   double l)
  {
    hauteur = h;
    largeur = l;
  }
  // suite ...
}
```

Constructeur par défaut : exemples

	constructeur par défaut	`Rectangle r1 = new Rectangle();`	`Rectangle r2 = new Rectangle(1.0,2.0);`
(A)	constructeur par défaut par défaut	0.0 | 0.0	Illicite !

```
class Rectangle {
}
```

Constructeur par défaut : exemples

	constructeur par défaut	`Rectangle r1 = new Rectangle();`	`Rectangle r2 = new Rectangle(1.0,2.0);`
(A)	constructeur par défaut par défaut	0.0 | 0.0	Illicite !
(B)	constructeur par défaut explicitement déclaré	0.0 | 0.0	Illicite !

```
class Rectangle {
  public Rectangle()
  {
    hauteur = 0.0;
    largeur = 0.0;
  }
}
```

Constructeur par défaut : exemples

	constructeur par défaut	`Rectangle r1 = new Rectangle();`	`Rectangle r2 = new Rectangle(1.0,2.0);`
(A)	constructeur par défaut par défaut	0.0 | 0.0	Illicite !
(B)	constructeur par défaut explicitement déclaré	0.0 | 0.0	Illicite !
(C)	pas de constructeur par défaut	Illicite !	1.0 | 2.0

```
class Rectangle {
  public Rectangle(double h, double l)
  {
    hauteur = h;
    largeur = l;
  }
}
```

Appel aux autres constructeurs

Java autorise les constructeurs d'une classe à appeler n'importe quel autre constructeur de cette même classe

Syntaxe : `this(...);`

```
class Rectangle {
  private double hauteur;
  private double largeur;

  public Rectangle(double h, double l)
  {
      hauteur = h;
      largeur = l;
  }
  public Rectangle() {
    // appel du constructeur à deux arguments
    this(0.0, 0.0);
  }
  // suite ...
```

Initialisation par défaut des attributs

Java permet de donner directement une valeur par défaut aux attributs.

Si le constructeur appelé ne modifie pas la valeur de cet attribut, ce dernier aura alors la valeur indiquée.

```
class Rectangle {
  private double hauteur = 1.0;
  private double largeur = 1.0;

  public Rectangle() { }

  public Rectangle(double h, double l)
  {  //...
  }
  //...
}
```

Conseil : préférez l'utilisation des constructeurs.

Constructeur de copie

Java offre un moyen de créer la **copie** d'une instance :
le *constructeur de copie*

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

`r1` et `r2` sont deux *instances* **distinctes**
mais ayant des mêmes valeurs pour leurs attributs
(au moins juste après la copie).

Constructeur de copie (2)

Le constructeur de copie permet d'initialiser une instance
en *copiant* les attributs d'une *autre instance* du même type.

Syntaxe :

```
NomClasse(NomClasse autre) { ... }
```

Exemple :

```
public Rectangle(Rectangle autreRectangle)
{
  hauteur = autreRectangle.hauteur;
  largeur = autreRectangle.largeur;
}
```

Constructeur de copie (3)

- En Java, il n'y a pas de constructeur de copie généré automatiquement.

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

- Le constructeur de copie n'est pas la seule façon de créer une copie d'objet.
  Le moyen le plus usuel est d'avoir recours à la méthode `clone()` que nous verrons un peu plus tard.

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

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