Programmation orientée objet

> ⚠ **Attention cette page est en cours de construction**

> La programmation orientée objet (POO) est un *paradigme* de programmation, c'est à dire une manière de programmer. Ce que l'on faisait précédemment est appellé la *programmation fonctionnelle*

L'un des pouvoir de la POO est nottament de pouvoir créer ses propres types appellées "classes", on les reconnais en Java car elle commencent avec une lettre majuscule (par exemple `String` est une classe mais `int` n'en est pas une)

Une classe est donc comme un nouveau type et est défini par des propriétés ainsi que des méthodes qui lui sont propre.

Un objet est une instance de cette classe (c'est une occurence de notre nouveau type).

```
// Ici on crée un "objet" de la classe "String"
String hello_string = "Hello World"

// .length() est une méthode de String qui permet de compter le nombre de caractère de la chaine
hello_string.length();

// println est une méthode statique qui n'a pas besoin d'un objet pour être exécutée
System.out.println(hello_string);
```

Voici comment créer une nouvelle classe :

```
// On crée donc la classe "Color".
// Cette définition de lcasse n'est pas précédée par "public"
// Donc seul les classes qui sont dans le même package peuvent l'utiliser
class Color {
```

```
  // On va définir un attribut de classe (qui sont des constantes)
  // public signifie qu'elle peut être utilisée de n'importe où
  // static signifie que c'est un attribut de classe (et pas d'objet) donc qui ne nécessite pas d'objet pour être accédée
  // final  signifie que c'est une constante
  // Par définition toutes les attributs de classe seront static final
  // Et seront des constantes car sinon cela va affecter TOUS les objets qui utiliseraient cette classe
  public static final float TOLERANCE = 0.001f;

  // On défini les attributs d'objet "r", "g" et "b"
  // Cela va définir l'état des objets de cette classe
  // En d'autres termes ça sera l'identité des objets,
  // ce qui les différenties des autre objets de la même classe
  // Le modificateur "private" signifie que l'on peut y accéder directement uniquement au sein de la classe
  private int r;
  private int g;
  private int b;

  // Ceci est le constructeur, qui défini comment on crée et assigne chaque attribut
  // Il sera utilisé avec le mot-clé "new"
  public Color(int r, int g, int b) {
    // "this" est utilisé pour faire référence à l'objet lui même
    // Il est seulement obligatoire quand on a d'autres variables du même nom qui existent en même temps
    // Par exemple ici les paramètres de la fonction ont le même nom, donc this est obligatoire
    this.r = max(min(r, 255), 0);
    this.g = max(min(g, 255), 0);
    this.b = max(min(b, 255), 0);
  }

  // Ici on crée une méthode qui va retourner une nouvelle instance de la classe Color
  // Etant donné que cette méthode est statique, on a pas besoin d'une instance de classe pour l'utiliser
  public static Color fromRGB(int r, int g, int b) {
    // On utilise donc le constructeur défini précédemment
    return new Color(r, g, b);
  }

  // Pour modifier et récupérer les attributs qui sont private et s'assurer que l'état de l'objet est toujours cohérent on va faire de "l'encapsulation d'attributs"
  // Ajout d'un "getter" (ou "accesseur") pour récupérer des attributs de la classe (on ne peut pas les récupérer directement car l'attribut est private)
```

```
  public int getRed() {
    return this.r;
  }

  // Ajout d'un "setter" (ou "mutateur") pour écrire dans des attributs private de la classe (en passant des tests)
  public void setRed(int r) {
    this.r = max(min(r, 255), 0);
  }

  // Ceci est une méthode d'objet (il n'y a pas le mot "static" dans sa définition)
  public float getLightness() {
    // ici on pourait aussi remplacer r g et b par this.r, this.g et this.b
    // mais ce n'est pas obligatoire car il n'y a pas d'autres variables du même nom
    final int maxComponent = Math.max(r, Math.max(g, b));
    final int minComponent = Math.min(r, Math.min(g, b));

    return (maxComponent + minComponent) / (2.0f * 255.0f);
  }
}
```

Maintenant si on veut utiliser cette classe `Color` :

```
// On a pas besoin d'utiliser "new" car c'est déjà dans le contenu de la méthode
Color blue = Color.fromRGB(0, 0, 255);

// ici en revanche on utilise le constructeur, donc on doit utiliser "new"
Color red = new Color(255, 0, 0);

// Maintenant on peut utiliser l'une de ses méthode d'objet
float blue_lightness = blue.getLightness();
float red_lightness = red.getLightness();
```

Pour résumer

La différence entre une classe et un objet :

- Créer une classe c'est comme créer un nouveau type avec ses propre propriétés et ses propres méthodes
- Un objet est une occurence d'une classe (c'est une instance de cette classe)

- On peut créer un objet à partir du constructeur de la classe en utilisant `new`

Ce qui différentie plusieurs objets d'une même classe :

- Les attributs définissent l'état d'un objet, c'est à dire son "ADN", ce qui va le différentier des autres objets de la même classe

Ce qui défini une méthode statique

- Une méthode statique est reconnaissable par le mot-clé `static` dans sa définition. (A savoir que `static` peut aussi être utilisé sur une variable pour devenir une variable de classe)
- Une méthode statique n'a pas besoin d'une instance de la classe (d'objet) pour fonctionner (exemple: println)

Ce qui défini une méthode d'objet

- Une méthode d'objet a besoin d'une instance de la classe (objet) pour fonctionner
- Une méthode d'objet peut récupérer les attributs de celui-ci

Comment utiliser le mot-clé `this` dans les méthodes d'objet :

- Le mot-clé `this` permet d'éviter la confusion entre les noms de variables et représente l'objet lui-même

Les modificateurs d'accès aux variables et classes :

- Tous les attributs ou méthodes ayant le modificateur `private` alors on ne pourra y accéder uniquement au sein de la classe
- Par défault toutes les classes, attributs et méthodes en Java peuvent être accédée par tous les membre du même package
- Tous les attributs, méthodes ou classes ayant le modificateur `public` peuvent être accédé par n'importe quel classe quelque soit le package

L'encapsulation d'attributs permet de ne pas accéder aux attributs directement pour ne pas avoir un état de l'objet qui soit invalide.

- On met donc le modificateur des attributs en `private`
- On crée des méthodes "getter" ("accesseurs") pour récupérer les attributs. Ces méthodes vont souvent commencer par `get` par convention.
- On crée des méthodes "setter" ("mutateurs") pour écrire dans ces attributs (en faisant passer des tests pour vérifier que la valeur que l'utilsiateur veut écrire est valide), ces méthodes vont souvent commencer par `set` par convention.
- **Note:** Si l'un des attributs de l'objet est un autre objet (tel qu'un tableau), il est obligatoire d'en faire une copie quand on le `set` ou le `get` sinon cela brise l'encapsulation des attributs. → copie défensive

---

Revision #1
Created 27 April 2023 06:31:37 by SnowCode

Updated 27 April 2023 06:35:12 by SnowCode