Les concepts (encapsulation, composition, heritage, etc)

Initialisation, surcharges de constructeurs et méthodes de fabriques

```
class Matricule {
□□// Java va d'abord initialiser les attributs de classe une seule fois dans le programme
    public static final String DEFAULT_DEPARTMENT = "Software Development";
    private static int next = 1; // Cet attribut est static mais pas final. Ce qui veut dire qu'il va changer pour toute la classe et donc pour tous les objets aussi

    // Java va ensuite initialiser les attributs d'objets (autant de fois qu'il n'y a d'objets)
    private String department = DEFAULT_DEPARTMENT;
    private int year = 2022;
    private int sequenceNumber;

    // On va déifnir notre constructeur principal avec "public NomDeLObjet(paramètres)"
□□// Et on peut faire référence aux attributs de l'objet sous la forme de "this.attribut"
    public Matricule(String department, int year, int seq) {
        if(department != null && !department.isBlank()) {
            this.department = department;
        }
        this.year = Math.abs(year);
        this.sequenceNumber = Math.abs(seq);
        if(seq == next) {
            next++;
        }
```

```
    }

    // On peut ensuite définir des surcharges de constructeurs qui vont utiliser notre constructeur de base
    // Pour cela on doit utiliser "this(paramètres)"
    public Matricule(String department, int year) {
        // Celui ci fait appel au constructeur 1
        // ⚠ ceci doit être la première instruction du constructeur
        this(department, year, next);
    }

    public Matricule(String department) {
        // Et celui ci fait appel au constructeur 2
        this(department, 2023);
    }

    // TODO faire une méthode de fabrique
}
```

Composition vs héritage

Composition

La composition est une technique qui permet de mettre en relations plusieurs éléments entre eux pour créer des structures plus complexes.

Pour cela il suffit de mettre des objets dans les attributs d'autres objets. Ainsi un objet qui contient d'autres objets est appellé *composite* et les objets qui le compose sont appellé *composants*.

```
// Ici c'est un enum mais cela pourrait très bien être juste une classe
enum Level {
    HEADMASTER(1, "Headmaster"),
    PROFESSOR(3, "Professor"),
    GRADUATED(5, "Graduated"),
    STUDENT(7, "Student");

    // reste du code ici
}
```

```
// Wizard est un composite
// Level et String sont des composants de Wizard
class Wizard {
    private Level level;
    private String name;

    // reste du code ici
}
```

Héritage

> **Attention ⚠ :** L'héritage n'est plus considéré comme étant une bonne pratique et ne devrait donc être utilisé que dans le cas de maintenance d'une codebase legacy.

```
// Ici on dit que la classe "PlayingCard" va hériter de "BaseCard"
// C'est à dire qu'elle va hériter de toutes ses méthodes et attributs qui ne sont pas private
// Cela veut dire que la classe "PlayingCard" aura toutes les méthodes disponibles par BaseCard + de nouvelles
class PlayingCard extends BaseCard {

}
```

Revision #1
Created 27 April 2023 06:31:41 by SnowCode
Updated 27 April 2023 06:35:12 by SnowCode