La Programmation Orienté Objet – 1ère partie

Commençons par énoncer trois problématiques de développement logiciel.

1. Comment permettre à notre programme de représenter le monde réel ?
2. Comment avoir un code facilement modifiable sans effet de bords ?
3. Comment avoir un code permettant facilement l'ajout de nouvelles fonctionnalités sans effet de bords sur les fonctionnalités existantes ?

Pour faire une comparaison, développer une application c'est comme construire une maison. Et nous souhaitons que notre maison :

1. soit conforme aux besoins de ses habitants (problématique 1)
2. permette d'être entretenue et maintenue sans difficulté : changer un carreau de carrelage n'implique pas de tous les changer par exemple (problématique 2)
3. permette des évolutions : ajouter une extension à la maison sans tout reconstruire par exemple (problématique 3)

Pour résoudre ces problématiques, l'architecte devra sans aucun doute faire des choix avisés quant à la structure de la maison. De la même façon, pour avoir une application qui répond aux problématiques, un développeur devra correctement **structurer son code**.

C'est à ce moment-là que la programmation orientée objet devient une solution car elle nous donne un outil clé pour structurer le code : **les objets !**

Un objet est un code qui regroupe des **informations** et des **traitements**.

Comparons un objet à une pièce de la maison. Une pièce se définit par des informations comme sa longueur, sa largeur, sa hauteur ou encore la couleur des murs. Et elle se définit aussi par des capacités, par exemple permettre de cuisiner ou bien de dormir ou de se laver (ou les 3 pour un studio).

Comment définir du coup ces informations et traitements dans le cas d'un objet ?

De la même façon que l'architecte écrit le plan de chaque pièce de la maison, nous devons écrire le plan de chaque objet.

Pour définir ce plan, on écrit une **classe**. Une classe va donc définir des informations et des traitements.

On nomme les informations d'une classe :

- variables de classe
- ou attributs
- ou propriétés

On nomme les traitements d'une classe :

- fonctions de classe
- ou méthodes

Il est intéressant de noter que la programmation orientée objet s'appuie sur deux concepts élémentaires, les variables et les fonctions, tout en structurant leur utilisation au sein de classes.

Je choisis ici d'avoir une **classe** nommée **Personne** qui possède comme information **un nom, un prenom et un age** et comme **traitement** la capacité d**'afficher ces informations** dans la console :

Personne.java

```
package com.epf;

public class Personne
{
        String nom ;
        String prenom ;
        int age ;

   void afficherDescription()
    {
      System.out.println(this.nom + " " + this.prenom + " " + this.age);
    }
}
```

Les points à retenir de ce code sont :

- Le mot-clé pour définir une classe est donc tout simplement **class**.
- Au sein des accolades { } qui suivent la déclaration de la classe, les informations (attributs) et les traitements (méthodes) sont définis.
- Au sein d'une méthode, je peux accéder à un attribut grâce à la syntaxe *this.[nom de l'attribut]*, par exemple : this.nom

En Java, le nom d'une classe doit toujours commencer par une majuscule. Le nom du fichier Java doit correspondre au nom de la classe.

Et c'est tout ? Car pour l'instant ça ne change pas grand-chose à ce qu'on a déjà vu.

C'est vrai, je vais donc vous montrer ce que l'on peut faire de plus grâce aux objets !

Mettez en œuvre le processus d'instanciation à travers les constructeurs

Tout d'abord tout comme il est nécessaire de construire les pièces d'une maison une fois qu'on en a le plan, il est nécessaire de construire l'objet à partir de la classe. Cela revient à donner vie à l'objet pour pouvoir l'utiliser.

Cette opération de construction est nommée **instanciation**. Elle permet d'obtenir une **instance, c'est-à-dire de créer vraiment un objet**.

« Démonstration du prof »

Que pouvons-nous retenir ?

- L'instanciation se déclenche grâce au mot-clé **new** et exécute automatiquement une fonction particulière nommée **constructeur**.
- Java ajoute un constructeur par défaut (qui ne fait rien) à toutes les classes qui n'ont pas de constructeur déclaré.
- Un objet instancié peut être stocké dans une variable typée par l'objet.
- Le caractère **.** permet d'appeler une méthode de l'objet à partir de la variable qui contient l'instance.
- Il est possible d'instancier plusieurs fois la même classe.

Un constructeur a le même nom que la classe et peut être paramétré.

Voici le code du fichier Personne.java

Personne.java

```
package com.epf;

public class Personne
{
        String nom ;
        String prenom ;
        int age ;

  Personne()
  {
    this.nom="CRUISE" ;
    this.prenom="Tom" ;
    this.age = 60 ;
  }

  void afficherDescription()
  {
    System.out.println(this.nom + " " + this.prenom + " " + this.age);
  }
}
```

Et voici le code du fichier Main.java

Main.java

```
package com.epf;

public class Main
{
   public static void main(String[] args)
   {
      Personne acteur1 = new Personne(); // déclaration et instanciation => appel du cosntructeur
      acteur1.afficherDescription(); // appel d'une méthode de l'objet
   }
}
```

Contrairement à la partie précédente, les fonctions n'ont pas le mot **static** dans la signature. Pour utiliser une méthode non statique, il faut avoir une instance de la classe (donc un objet) tandis qu'**une méthode statique est indépendante du processus d'instanciation.** Vu que l'on n'avait pas encore appris à instancier un objet, c'est pour cela que jusqu'à présent on avait utilisé uniquement des méthodes statiques.

Mine de rien, cela fait pas mal d'informations d'un coup ! Allez, ne lâchez rien, continuons.

Tout comme à partir d'un même plan on peut construire plusieurs maisons sur des terrains différents, à partir d'une même classe on peut instancier plusieurs objets.

Et tout comme ces différentes maisons peuvent varier sur certains points (par exemple la couleur des murs), les instances peuvent avoir des données différentes.

Comment faire ?

L'une des façons est d'utiliser **un constructeur paramétré**. Un constructeur étant simplement une fonction particulière, comme toute fonction, il peut avoir des paramètres.

Prenons un exemple :

Personne.java

```
package com.epf;

public class Personne
{
   String nom ;
   String prenom ;
   int age ;

   Personne (String lenom, String leprenom, int lage)
   {
      this.nom = lenom;
      this.prenom = leprenom;
      this.age = lage;
   }

   void afficherDescription()
   {
      System.out.println(this.nom + " " + this.prenom + " " + this.age);
   }
}
```

Main.java

```
package com.epf ;

public class Main
{

   public static void main(String[] args)
   {
      Personne acteur1 = new Personne(); // création du 1er acteur
      acteur1.afficherDescription(); // appel d'une méthode de l'objet
      Personne acteur2 = new Personne(); // création du 2eme acteur
      acteur1.afficherDescription(); // appel d'une méthode de l'objet
   }
}
```

Contrairement au premier exemple où les nom, prenom et age étaient définis en dur dans la classe, ici ils sont passés en paramètre du constructeur lors de l'instanciation. Ainsi chaque instance peut varier bien que la structure de l'objet soit conservée.

Renforcez l'intégrité de vos classes avec l'encapsulation

Voyons une autre caractéristique des objets. Dans le chapitre sur les variables, j'ai indiqué que ces dernières ont une portée. **Pour rappel, la portée est la zone de code où une variable est accessible.** Et la règle que nous avons vue jusqu'à présent est la suivante : une variable est accessible dans le bloc d'instructions où elle est déclarée.

Pour rappel : un bloc est repéré par 2 accolades

{

…….

……

}

La programmation orientée objet inclut un concept nommé **encapsulation**. Grâce à l'encapsulation, la portée des attributs d'une classe ne sera pas forcément limitée à la portée du bloc d'instructions associées (à savoir la classe).

« Démonstration du prof »

Retenons les points suivants :

- La visibilité concerne les attributs, les méthodes et les classes.
- La visibilité **public** permet à un code en dehors de la classe d'accéder et modifier l'attribut ou d'appeler une méthode de la classe.
- La visibilité **private** interdit à tout code en dehors de la classe d'accéder ou modifier un attribut ou d'appeler une méthode de la classe.

Mais quel est l'intérêt ?

L'encapsulation permet de protéger l'intégrité de l'objet. Imaginez une maison où l'interrupteur pour allumer la lumière d'une chambre soit sur la façade extérieure. Cela serait illogique car n'importe qui pourrait alors allumer ou éteindre la lumière.

De la même façon, on ne veut pas que n'importe quel code puisse accéder aux informations ou appeler n'importe quelle méthode de l'objet. On risquerait de rencontrer des comportements non souhaités !

Il existe une bonne pratique de programmation qui consiste à mettre constamment tous les attributs en visibilité privée.

Mais comment faire pour qu'un code à l'extérieur de la classe puisse y accéder ou le modifier dans le cas où ce serait nécessaire ?

Dans ce cas-là, on ajoutera deux fonctions :

- un accesseur (*getter* en anglais) pour accéder à l'attribut
- un mutateur (*setter* en anglais) pour modifier l'attribut

Voici un code d'exemple :

Personne.java

```
package com.epf;

public class Personne
{
   private String nom ;
   private String prenom ;
   private int age ;

   Personne (String lenom, String leprenom, int lage)
   {
      this.nom = lenom;
      this.prenom = leprenom;
      this.age = lage;
   }

   void afficherDescription()
   {
      System.out.println(this.nom + " " + this.prenom + " " + this.age);
   }

   public String getNom()
   {
      return this.nom;
   }

   public void setNom(String lenom)
   {
      this.nom = lenom;
   }

// Faite pareil pour prenom et age

}
```

Les getters et les setters sont tellement courants que les IDE proposent de les générer automatiquement. Également, la terminologie française accesseur et mutateur est très peu utilisée, il faudra s'habituer à l'anglais !

À vous de jouer !!

En résumé

- Un objet est un ensemble d'attributs et de méthodes.
- Une classe permet de décrire la structure de l'objet.
- L'instance d'un objet est obtenue en instanciant une classe.
- L'instanciation appelle automatiquement le constructeur.
- L'encapsulation permet de protéger l'intégrité des objets.