Concepts Fondamentaux de la POO

PLAN

Références

[1] https://bpesquet.gitbooks.io/programmation-orientee-objet-csharp/content/chapters/01-initiation-poo.html

[2] https://www.irisa.fr/triskell/members/pierre-alain.muller/teaching/objectsnclasses

1. Introduction

Parmi les critères de qualité d'un logiciel :

- l'exactitude : aptitude d'un programme à fournir le résultat voulu et à répondre ainsi aux spécifications
- la robustesse : aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation
- l'extensibilité : facilité avec laquelle un programme pourra être adapté pour répondre à l'évolution des spécifications
- la réutilisabilité : possibilité d'utiliser certaines parties du logiciel pour résoudre un autre problème
- la portabilité : facilité avec laquelle on peut exploiter un même logiciel dans différentes plates-formes
- l'efficience : temps d'exécution, taille mémoire

Parmi les objectifs de la programmation objet :

- gagner en productivité et abaisser les coûts
- augmenter la qualité des logiciels
- faciliter la maintenance des applications
- favoriser et simplifier la réutilisation
- rapprocher la solution informatique du problème réel
- encapsuler les traitements à fin de libérer le concepteur des détails d'implémentation

Parmi les langages objet : C++, C#, Objective C, VB.NET, Ada, Python, Smalltalk, Delphi, Fortran 2003, Php, Perl, java

2. Histoire de la programmation orientée objet

Les concepts de la POO naissent au cours des années 1970 dans des laboratoires de recherche en informatique. Les premiers langages de programmation véritablement objets ont été **Simula**, puis **Smalltalk**. [1]

A partir des années 1980, les principes de la POO sont appliqués dans de nombreux langages comme **Eiffel** (créé par le Français Bertrand Meyer), **C++** (une extension objet du langage **C** créé par le Danois Bjarne Stroustrup) ou encore **Objective C** (une autre extension objet du C utilisé, entre autres, par l'iOS d'Apple). [1]

Les années 1990 ont vu l'avènement des langages orientés objet dans de nombreux secteurs du développement logiciel, et la création du langage **Java** par la société Sun Microsystems. Le succès de ce langage, plus simple à utiliser que ses prédécesseurs, a conduit Microsoft à riposter en créant au début des années 2000 la plate-forme **.NET** et le langage **C#**, cousin de Java. [1]

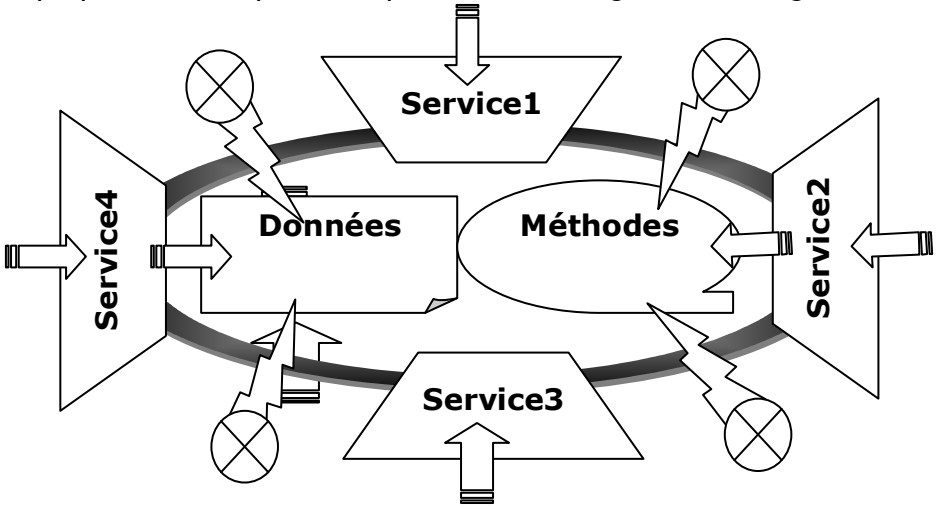
De nos jours, de très nombreux langages permettent d'utiliser les principes de la POO dans des domaines variés : Java et C# bien sûr, mais aussi **PHP** (à partir de la version 5), **VB.NET**, **PowerShell**, **Python**, etc. Une connaissance minimale des principes de la POO est donc indispensable à tout informaticien, qu'il soit développeur ou non. [1]

REMARQUE : la POO s'accompagne d'un changement dans la manière de penser les problèmes et de concevoir l'architecture des applications informatiques. C'est ce que l'on appelle l'analyse (ou la modélisation) orientée objet. Son principal support est le langage de modélisation **UML**. [1]

3.L'encapsulation

Encapsuler : cacher Capsule 

L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés. L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet.



L'encapsulation des données

L'encapsulation présente deux avantages :

- Les données encapsulées sont protégées des accès intempestifs, ce qui permet de garantir leur intégrité. [2]
- Les clients d'une abstraction ne dépendent pas de la réalisation de l'abstraction, mais seulement de sa spécification. [2]

4.Classe

2.1- Définition

Une classe est le support de l'encapsulation : c'est un ensemble de données et de fonctions regroupées dans une même entité. Une classe est une description abstraite d'un objet. Les fonctions qui opèrent sur les données sont appelées des méthodes. Instancier une classe consiste à créer un objet sur son modèle. Entre classe et objet il y a, en quelque sorte, le même rapport qu'entre type et variable.

On appelle classe la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet. Un objet est donc "issu" d'une classe, c'est le produit qui sort d'un moule. En réalité on dit qu'un objet est une instanciation d'une classe, c'est la raison pour laquelle on pourra parler indifféremment d'objet ou d'instance (éventuellement d'occurrence).

Remarque

Par convention (en java) le nom de la classe commence par une lettre majuscule.

Dans le cas des noms composés de plusieurs mots, pour chaque nouveau mot commencer par une majuscule.

2.2- Les Attributs (les données membres)

Les attributs (parfois appelés données membres): il s'agit des données représentant l'état de l'objet

Les données membres sont des variables stockées au sein d'une classe. Elles doivent être précédées de leurs types et (éventuellement) d'une étiquette précisant leur portée, c'est-à-dire les classes ayant le droit d'y accéder (private, public…).

Chaque attribut possède une *visibilité*, un *type* et un *nom*.

Remarque

Par convention (en java) le nom d'un attribut commence par une lettre minuscule.

Dans le cas des noms composés de plusieurs mots, pour chaque nouveau mot commencer par une majuscule.

2.3- Les méthodes (les fonctions membres)

Les méthodes (parfois appelées fonctions membres): il s'agit des opérations applicables aux objets

La définition d'une méthode se fait en définissant le prototype et le corps de la fonction à l'intérieur de la classe en une opération.

Chaque méthode possède : une *visibilité*, un *type de retour* (sauf les constructeurs), un *nom*, une *liste de paramètres* et un *corps*.

Une méthode peut être un constructeur, un modificateur, un observateur, un accesseur ou un interrogateur.

Remarque

Par convention (en java) le nom d'une méthode commence par une lettre minuscule.

Dans le cas des noms composés de plusieurs mots, pour chaque nouveau mot commencer par une majuscule.

2.4- Syntaxe

```
visibilité class NomClasse{
        //attributs
        visibilité type nom ;
        //constructeurs
        public NomClasse(listeDesParamètres){
                corps
        }
        //les autres méthodes
        visibilité typeDeRetour nomMethode(listeDesParamètres){
                corps
        }
}
```

2.5- Visibilité ou protection ou portée

Elle peut être appliquée à la classe, aux attributs et aux méthodes. On s'intéresse dans ce paragraphe à la protection des attributs et des méthodes.

Il existe quatre types de visibilité : public, private, (package) et protected (chapitre Héritage).

Classe / Visibilité	La classe	Les classes du même package	Les autres classes
Public	Oui	Oui	Oui
(package)	Oui	Oui	Non
Private	Oui	Non	Non

2.5.1- Visibilité des attributs

Pour un attribut donné les accès possible sont la lecture ou l'écriture (l'affectation).

Exemple

une classe A possède un attribut i.

```
A a = new A() ;
System.out.println(a.i) ;  //lecture
int j=a.i;                 //lecture
a.i=j;                     //écriture
```

Règle à appliquer

- Si l'attribut ne change pas de valeur après l'instanciation de l'objet alors cet attribut doit être private.
- Si l'attribut change la valeur, poser la question qui change la valeur ? si seulement la classe change la valeur alors il est private sinon il est public.

Exemple

- Couleur, marque et niveauEncre d'un Stylo
- Température de l'eau d'une machine à laver
- Nombre d'étage, étageEnCours, capacité et charge d'un ascenseur

2.5.2- Visibilité des méthodes

Règle à appliquer

Si la méthode est appelée seulement dans la classe alors elle est private sinon (c-à-d elle est appelée par les autres classes) elle est public.

Exemple

Pour une machine à laver :

- chafferEau(…)
- tourner(…)
- remplirEau(…)
- lancerProgramme(…)
- viderEau(…)

Remarque

La surcharge d'une méthode est le fait d'avoir dans une même classe plusieurs méthodes :

- de même nom
- de signatures (liste des paramètres) différentes.

2.6- Les constructeurs

2.6.1- Définition

Un constructeur est une méthode spéciale qui permet de créer des objets sur le modèle de la classe. Le nom du constructeur est celui de la classe. Un constructeur ne possède pas de type de retour (même pas void). Un constructeur permet d'initialiser tous les attributs d'un objet lors de sa création.

Exemples

```
public class A{
        public A1(){
        }
        public int A(){
        }
        public void A(){
        }
        public A(){
        }
}
```

2.6.2- this

This fait référence à l'objet en cours. Il existe deux utilisations de this :

- pour accéder aux attributs
- pour appeler un constructeur

A- Accéder aux attributs

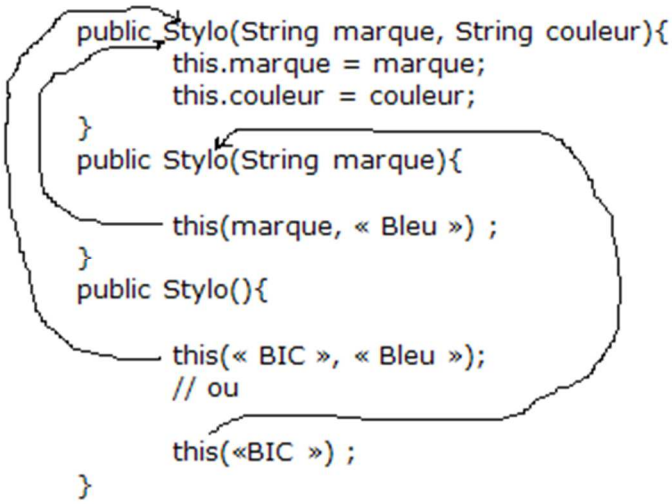
```
public class Stylo{
        private String marque;
        public Stylo(){
                marque = "BIC";
        }
        public Stylo(String marque){
                marque = marque;
                this.marque = marque;
        }
}
```

B- Appeler un constructeur

L'appel du constructeur doit être la première instruction. On peut appeler un constructeur sans ou avec paramètres.

```
public Stylo(){
        marque = « BIC » ;
        couleur = « Bleu » ;
}
public Stylo(String marque){
        this() ;                    //exécution des deux initialisations
        this.marque = marque;
}
public Stylo(String marque){
        this.marque = marque;
        this() ;        //l'appel de this doit être la première instruction
}
```

appel



2.6.3- Le choix des valeurs par défauts

Le choix des valeurs par défauts peut se baser sur des critères logiques ou statistiques. On peut aussi choisir la première valeur qui fonctionne (généralement zéro ou l'unité).

Exemples

couleur « Bleu » Statistique (les plus vendus)
niveauEncre 100 logique (c'est un nouveau stylo)
longueur 1 (première valeur)

2.6.4- Les types des constructeurs

A- Les constructeurs sans paramètres ou par défaut

Ce constructeur ne prend aucun paramètre.

```
public Stylo(){
    marque= « BIC »;
    couleur= « Bleu »;
    niveauEncre=100;
}
```

B- Les constructeurs avec paramètre(s)

Ce constructeur prend un ou plusieurs paramètres.

```
public Stylo(String marque){
    this.marque= marque;
    couleur= « Bleu »;
    niveauEncre=100;
}
public Stylo(String marque,String couleur){
    this.marque= marque;
    this.couleur= couleur;
    niveauEncre=100;

}
```

C- Les constructeurs par copie

Ce constructeur prend un paramètre de même type que l'objet à créer.

```
public Stylo(Stylo stylo){
    marque= stylo.marque;
    couleur= stylo.couleur;
    niveauEncre= stylo..niveauEncre; // ou 100
}
```

2.7- Les destructeurs

Un destructeur est une méthode spéciale qui est appelé automatiquement lors de la destruction de l'objet pour libérer les zones mémoires éventuellement réservées. En java ceci se fait en surchargeant la méthode *finalize()*.

2.8- Les modificateurs

Un modificateur est une méthode spéciale qui permet de changer la valeur d'un attribut. Le nom du modificateur commence par convention par le préfixe **set**. Le type de retour d'un modificateur est void. Un modificateur prend un seul paramètre de même type que l'attribut à modifier

```
public void setValeur(int valeur){
        this.valeur = valeur;
}
```

2.8- Les observateurs/ les accesseurs/ les interrogateurs (getters)

Ce sont des méthodes spéciales qui permettent de retourner la valeur d'un attribut. Leurs nom commence par le préfixe **is** si le type de l'attribut est booléen et **get** sinon. Leur type de retour est celui de l'attribut. Ils ne prennent aucun paramètre.

```
public String getValeur(){
        return couleur;
}
public boolean isEmpty(){
        return empty;
}
```

2.9- Choix entre public/private et get/set

Un attribut peut être accéder en écriture (par affectation a.i =10) ou en lecture (affichage System.out.println(a.i) ou affectation dans une autre valeur j=a.i).

Écriture / Lecture	**Autorisée**	**Interdite**
Autorisée	Solution1 : public Solution2 : private + set et get	private + get
Interdite	private + set	private

5.Objet

Un objet possède [Booch] :

- **un état** : l'état regroupe les valeurs instantanées de tous les attributs d'un objet, il évolue au cours du temps. L'état d'un objet à un instant donné est la conséquence de ses comportements passés. [2]
- **un comportement** : le comportement décrit les actions et les réactions d'un objet, il regroupe toutes les compétences d'un objet et il se représente sous la forme d'opérations. [2]
- **une identité** : l'objet existe indépendamment des valeurs de ses attributs. Tout objet possède une identité qui lui est propre et qui le caractérise. L'identité permet de distinguer tout objet de façon non ambigüe, indépendamment de l'état. [2]

Un objet est une instance d'une classe. La création d'un objet est appelé instanciation. L'instanciation se fait par le mot clé new suivi d'un constructeur de la classe. [2]

Exemple

```
Stylo s ;     //déclaration
s = new Stylo() ;  // instanciation
s.ecrire();
Stylo s1 = new Stylo(); // déclaration et instanciation
s1.ecrire();
Stylo s2;
s2.ecrire();  // erreur s2 n'est pas initialisé
```