La Programmation Orientée Objet: Java

Dr AKOHOULE

Chapitre I: CONCEPTS DE BASE DE LA PROGRAMMATION ORIENTEE OBJET

Objectifs spécifiques

- ✈ Introduire les facteurs de naissance de la POO
- ✈ Introduire la définition de la POO
- ✈ Introduction au concept de l'approche OO

Eléments de contenu

I.De la programmation classique vers la POO

II.Définition

III.Concepts de base de la POO

I.1 De la programmation classique vers la programmation orientée objet

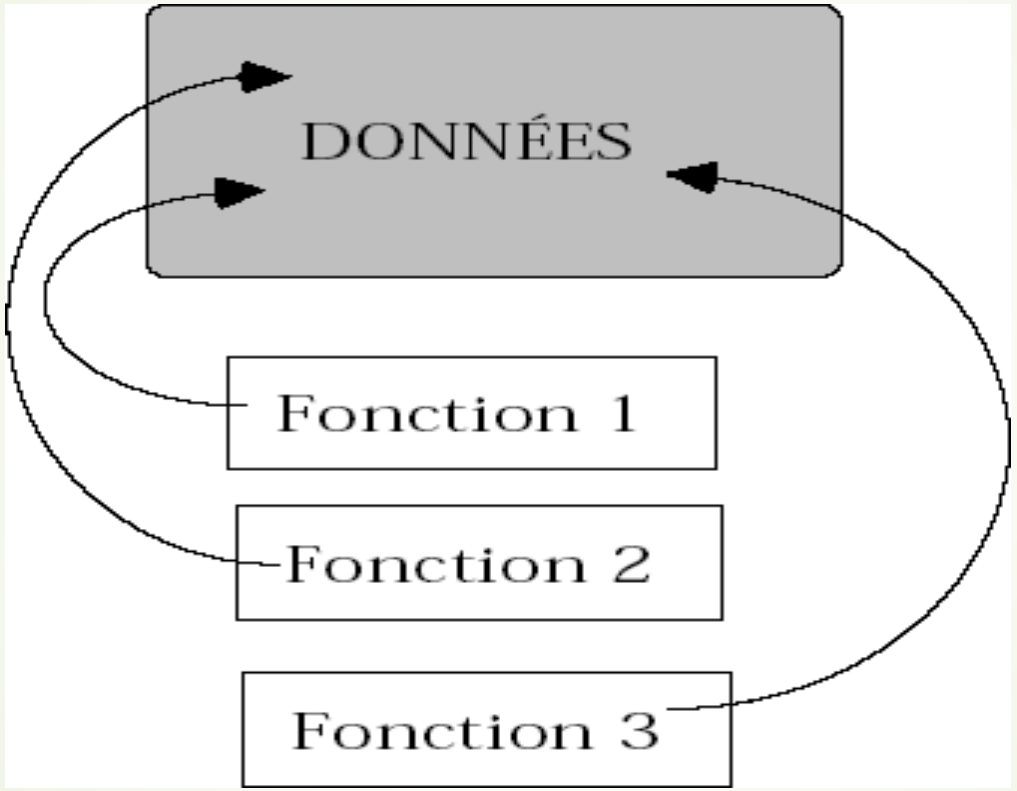
La programmation classique telle que étudiée au travers des langages C, Pascal… définie un programme comme étant un ensemble de **Données** sur lesquelles agissent des **Procédures** et des **Fonctions**.

Les **Données** constituent la partie passive du programme. Les **Procédures** et les **Fonctions** constituent la partie active.

Programmer dans ce cas revenait à :

- Définir un certain nombre de variables (structures, tableaux…)
- Ecrire des Procédures pour les manipuler sans associer explicitement les unes aux autres.

Exécuter un programme se réduit alors à appeler ces Procédures dans un ordre décrit par le séquençage des instructions et en leur fournissant les données nécessaires à l'accomplissement de leurs tâches.



Dans cette approche, **Données** et **Procédures** sont traitées indépendamment les unes des autres sans tenir compte des relations étroites qui les unissent.

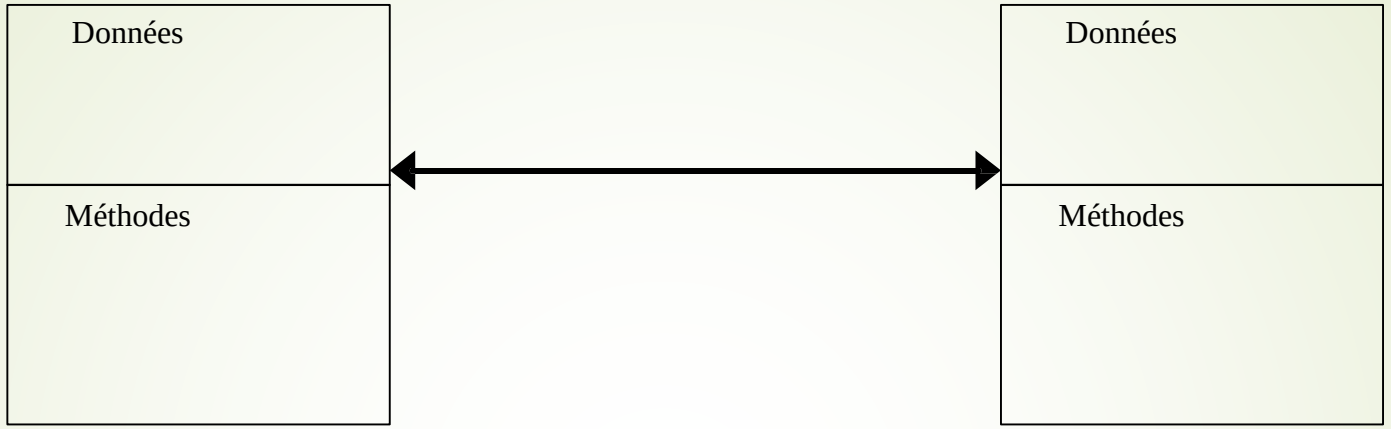
Les questions qu'on peut poser dans ce cas :

1. Cette séparation (**Données, Procédures**) est elle utile ?
2. Pourquoi privilégier les **Procédures** sur les **Données** (Que veut-on faire ?) ?
3. Pourquoi ne pas considérer que les programmes sont avant tout des ensembles objets informatiques caractérisés par les opérations qu'ils connaissent ?

Les langages objets sont nés pour répondre à ces questions. Ils sont fondés sur la connaissance d'une seule catégorie d'entités informatiques : les objets.

Un objet incorpore des aspects statiques et dynamiques au sein d'une même notion.

Avec les objets ce sont les données qui deviennent prépondérantes. On répond tout d'abord à la question « De quoi parle-t-on ? »



Un programme est constitué d'un ensemble d'objets chacun disposant d'une partie **Procédures** et d'une partie **Données**. Les objets interagissent par envoie de messages.

I.2 Définitions

I.2.1 POO

La POO est une méthode d'implémentation dans laquelle les programmes sont organisés sous formes de collections coopératives d'objets, dont chacun représente une instance d'une classe quelconque et dont toutes les classes sont membres d'une hiérarchie de classes unis à travers des relations d'héritage.

I.2.2 Algorithmique Objet

Dans l'approche orienté objet un algorithme sera essentiellement vu comme un ensemble d'objets auxquels l'utilisateur envoie des messages et qui s'en envoient pendent le fonctionnement.

Ces objets seront toujours pour l'utilisateur des boites noires et qui contiendront des variables locales, inconnues de l'environnement, et qui ne s'y intéressera d'ailleurs pas. Le seul moyen d'accéder à ces objets sera l'envoie des messages qu'ils sont capables de comprendre.

Remarque : La spécification d'un système dans l'approche OO va s'axer principalement sur la détermination des objets à manipuler. Une fois cette étape réalisé le concepteur n'aura plus qu'à réaliser les fonctions de haut niveau qui s'appuient sur les objets et les familles d'objets définis.

1.2.3 Objet et classe

Un objet est une entité logicielle :

- Ayant une identité
- Capable de sauvegarder un état c'est-à-dire un ensemble d'informations dans des variables internes.
- Répondant à des messages précis en déclenchant des activations internes appropriés qui changent l'état de l'objet. Ces opérations sont appelées méthodes. Ce sont des fonctions liées à des objets et qui précisent le comportement de ces objets.

1.2.4 Attributs

Les attributs d'un objet sont l'ensemble des informations se présentant sous forme de variable et permettant de représenter l'état de l'objet.

1.2.6 Méthodes

Une **Méthode** est une **Fonction** ou **Procédure** liée à un objet qui est déclenchée à la réception d'un message particulier

La méthode déclenchée correspond strictement au message reçu. La liste des méthodes définies au sein d'un objet constitue l'interface de l'objet pour l'utilisateur : ce sont les messages que l'objet peut comprendre si on les lui envoie et dont la réception déclenche les méthodes correspondantes.

1.2.7 Signature

La signature d'une méthode représente la précision de son nom, du type de ses arguments et du type de données retournées.

I.3 Concept de base de la POO

La POO se base sur les notions clés suivantes :

- Encapsulation
- Abstraction
- Classe et objets
- Héritage
- Polymorphisme

1.3.1 Encapsulation

Lors de la conception d'un programme orienté-objet, le programmeur doit identifier les objets et les données appartenant à chaque objet mais aussi des droits d'accès qu'ont les autres objets sur ces données. L'encapsulation de données dans un objet permet de cacher ou non leur existence aux autres objets du programme.

On précise trois modes d'accès aux attributs d'un objet.

- Le mode ***public*** avec lequel les attributs seront accessibles directement par l'objet lui même ou par d'autres objets. Il s'agit du niveau le plus bas de protection.
- Le mode ***private*** avec lequel les attributs de l'objet seront inaccessibles à partir d'autres objets : seules les méthodes de l'objet pourront y accéder. Il s'agit du niveau le plus fort de protection.
- Le mode ***protected :*** cette technique de protection est étroitement associée à la notion d'héritage.

1.3.2 Abstraction

C’est le fait de se concentrer sur les caractéristiques importantes d’un objet selon le point de vue de l’observateur.

Exemple : Voiture

L’abstraction est un principe qui consiste à ignorer certains aspects d’un sujets qui ne sont pas importants pour le problème dans le but de se concentrer sur ceux qui le sont.

I.3.3 Classes, Objets, Instances

Classe

- Une **Classe** est un ensemble d'**objets** qui ont en commun :
  - les mêmes méthodes
  - les mêmes types d'attributs

Classe = attributs + méthodes + instanciations

Objet

Un objet est une instance d'une classe. La création d'objets s'appelle donc l'Instanciation.

Cette instanciation se fait grâce à l'opérateur **new** suivi du nom de la classe à instancier et des parenthèses contenant les paramètres d'instanciation (parenthèses vides s'il n'y a pas de paramètres).

Soit la classe **Personne** suivante:

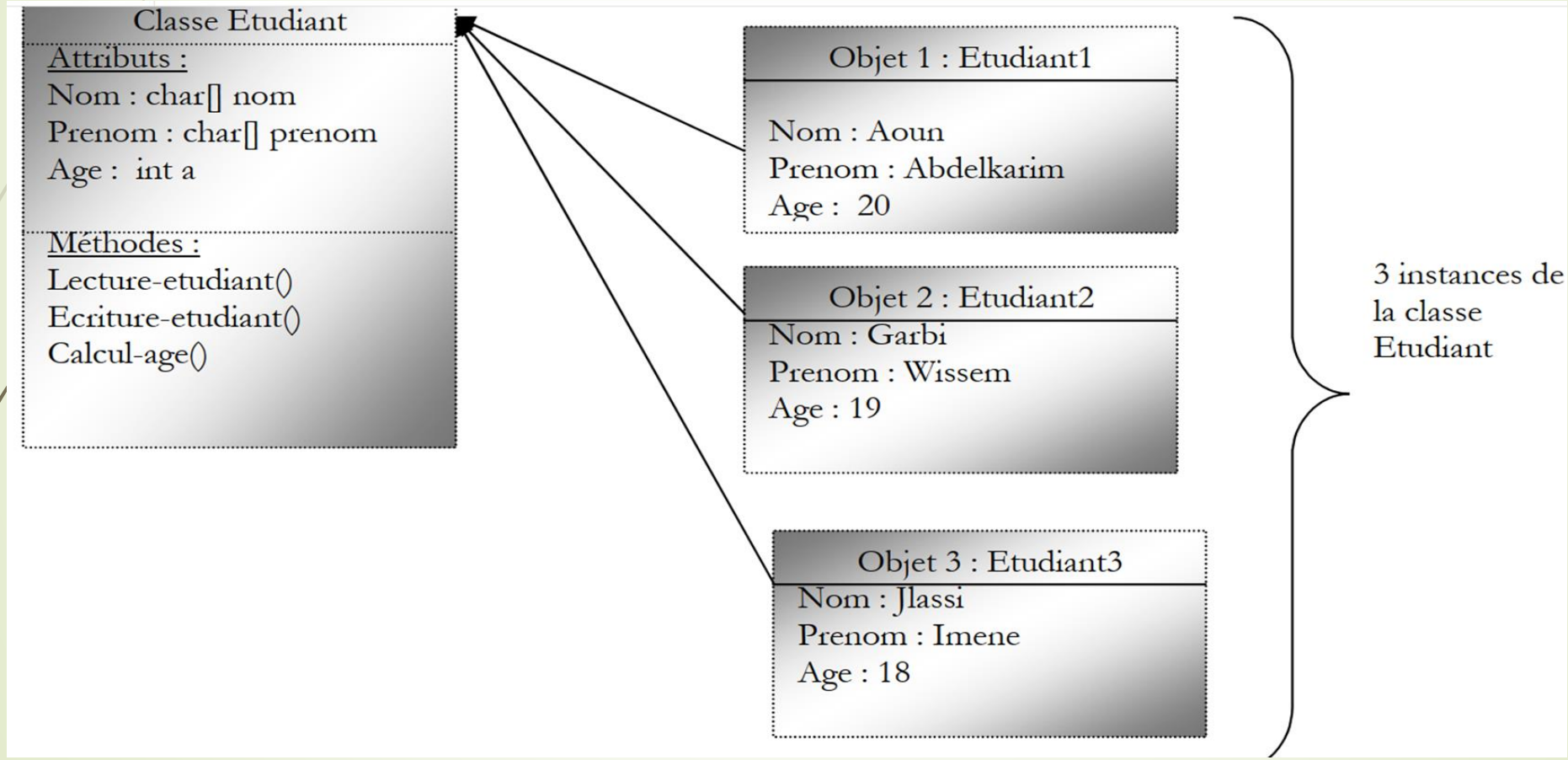
```
Class Personne{
    int age;
}
```

L'instanciation de cette classe (rudimentaire et actuellement inutile….) se fait de la façon suivante:

```
new Personne ();
```

Instance

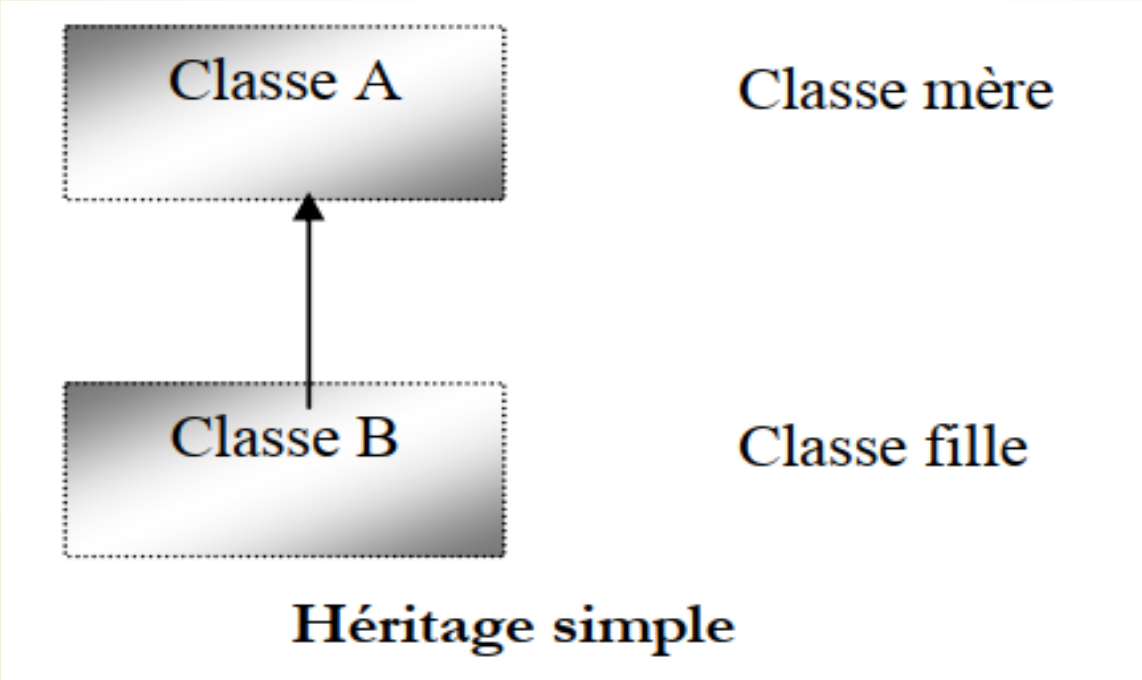
Une **Instance** d'une Classe est un **objet** particulier d'une classe qui peut activer les méthodes de la classe et qui a des valeurs particulières de ses attributs.



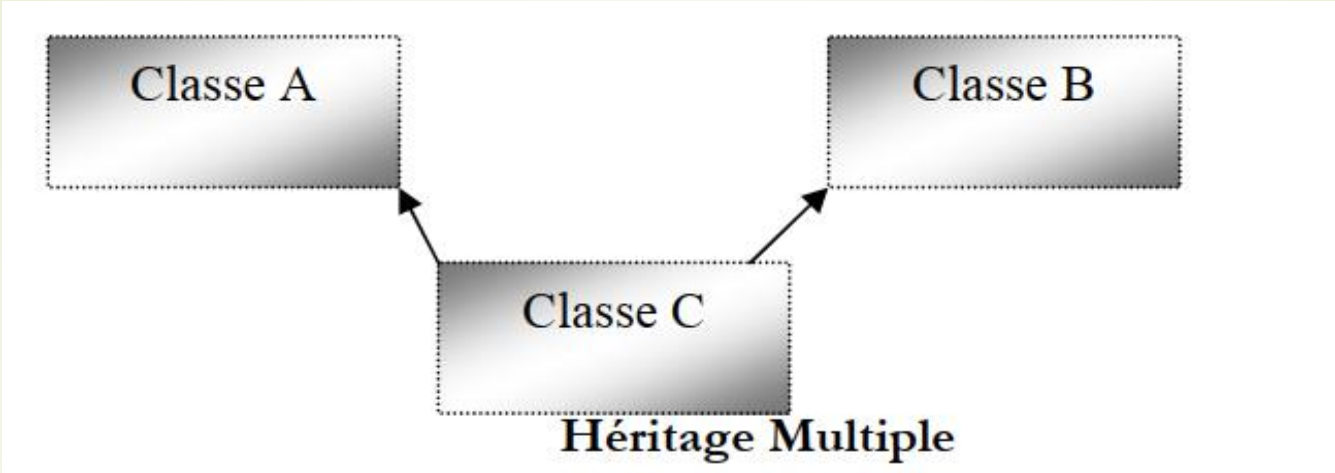
I.3.4 Héritage

La notion d'héritage est une relation entre différentes classes permettant de définir une nouvelle classe en se basant sur les classes existantes. On parle d'héritage simple lorsqu'une classe fille ne possède qu'une classe mère.

On parle d'héritage multiple lorsqu'une classe fille possède plusieurs classes mères.



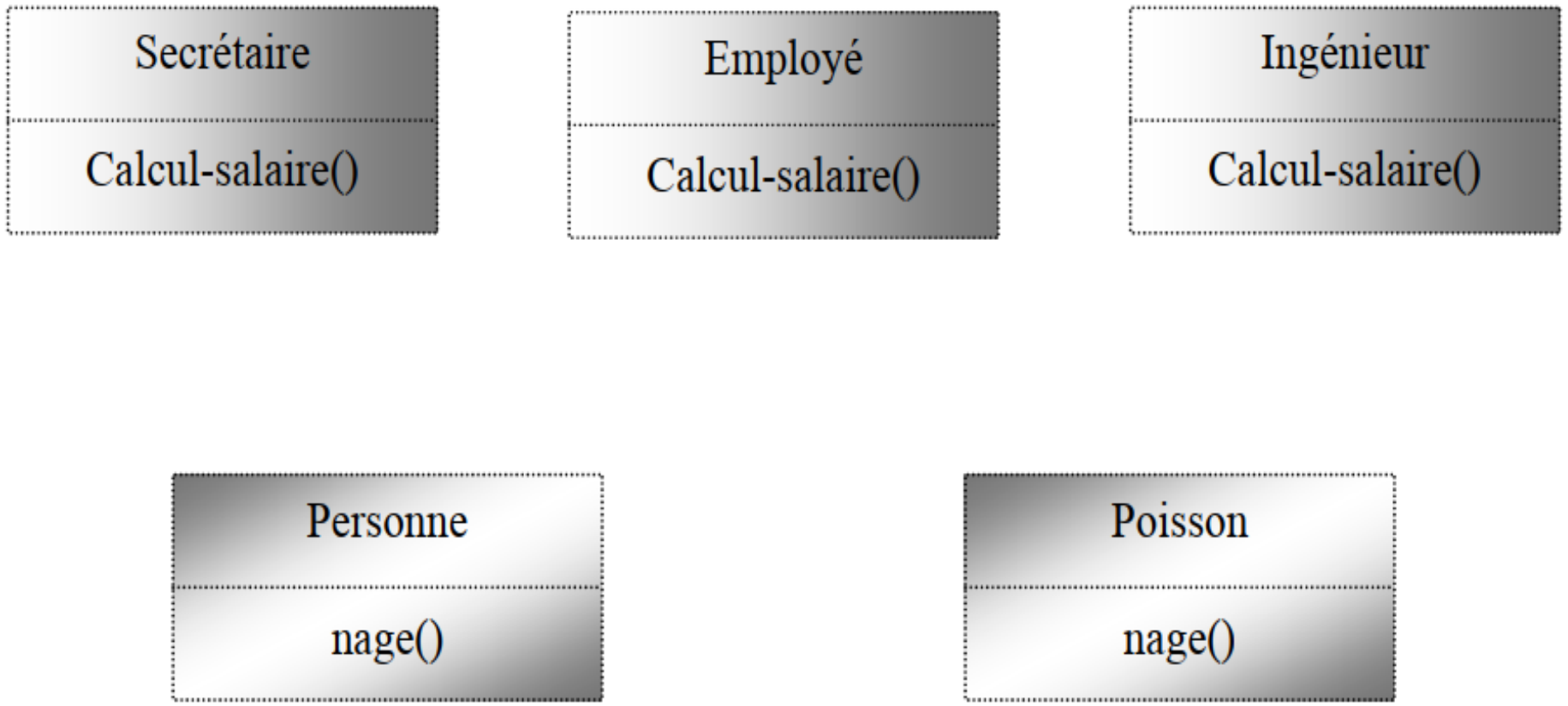
Héritage simple



I.3.5 Polymorphisme

Le terme polymorphisme issu du grec signifie la faculté de prendre plusieurs formes. Une entité est polymorphe si à l'exécution elle peut se référer à des instances de classe différentes.

Exemples :



Le polymorphisme est un mécanisme qui permet à une sous classe de redéfinir une méthode dont elle a hérité tout en gardant la même signature de la méthode.

Chapitre II: LE LANGAGE DE PROGRAMMATION JAVA

INTRODUCTION

Java est un langage de programmation originellement proposé par Sun Microsystems et maintenant par Oracle depuis son rachat de Sun Microsystems en 2010.

Java a été conçu avec deux objectifs principaux :

- Permettre aux développeurs d'écrire des logiciels indépendants de l'environnement hardware d'exécution.
- Offrir un langage orienté objet avec une bibliothèque standard riche

Le langage Java est un langage généraliste de programmation synthétisant les principaux langages existants lors de sa création en 1995 par Sun Microsystems. Il permet une programmation orientée-objet et reprend une syntaxe très proche de celle du langage C.

Outre son orientation objet, le langage Java a l'avantage d'être:

- **Modulaire:** On peut écrire des portions de code génériques,
- **Rigoureux:** La plupart des erreurs se produisent à la compilation et non à l'exécution,
- **Portable:** Un même programme compilé peut s'exécuter sur différents environnements.

II.1 L'Environnement JAVA

L'indépendance par rapport à l'environnement d'exécution est garantie par la machine virtuelle Java (Java Virtual Machine ou **JVM**). En effet, Java est un langage compilé mais le compilateur ne produit pas de code natif pour la machine, il produit du **bytecode** : un jeu d'instructions compréhensibles par la JVM qu'elle va traduire en code exécutable par la machine au moment de l'exécution.

Pour qu'un programme Java fonctionne, il faut non seulement que les développeurs aient compilé le code source mais il faut également qu'un environnement d'exécution comprenant la JVM soit installé sur la machine cible.

Il existe ainsi deux environnements Java qui peuvent être téléchargés et installés depuis le **site d'Oracle**:

JRE - Java Runtime Environment

Cet environnement fournit uniquement les outils nécessaires à l'exécution de programmes Java.

Il fournit entre-autres la machine virtuelle Java.

JDK - Java Development Kit

Cet environnement fournit tous les outils nécessaires à l'exécution mais aussi au développement de programmes Java. Il fournit entre-autres la machine virtuelle Java et le compilateur.

Java est un langage interprété, c'est à dire qu'un programme compilé n'est pas directement exécutable par le système d'exploitation mais il doit être interprété par un autre programme, qu'on appelle interpréteur.

Un programmeur Java écrit son code source, sous la forme de classes, dans des fichiers dont l'extension est **.java**. Ce code source est alors compilé par le compilateur **javac** en un langage appelé **bytecode** et enregistre le résultat dans un fichier dont l'extension est **.class**. Le bytecode ainsi obtenu n'est pas directement utilisable. Il doit être interprété par la machine virtuelle de Java qui transforme alors le code compilé en code machine compréhensible par le système d'exploitation. C'est la raison pour laquelle Java est un langage portable : le bytecode reste le même quelque soit l'environnement d'exécution.

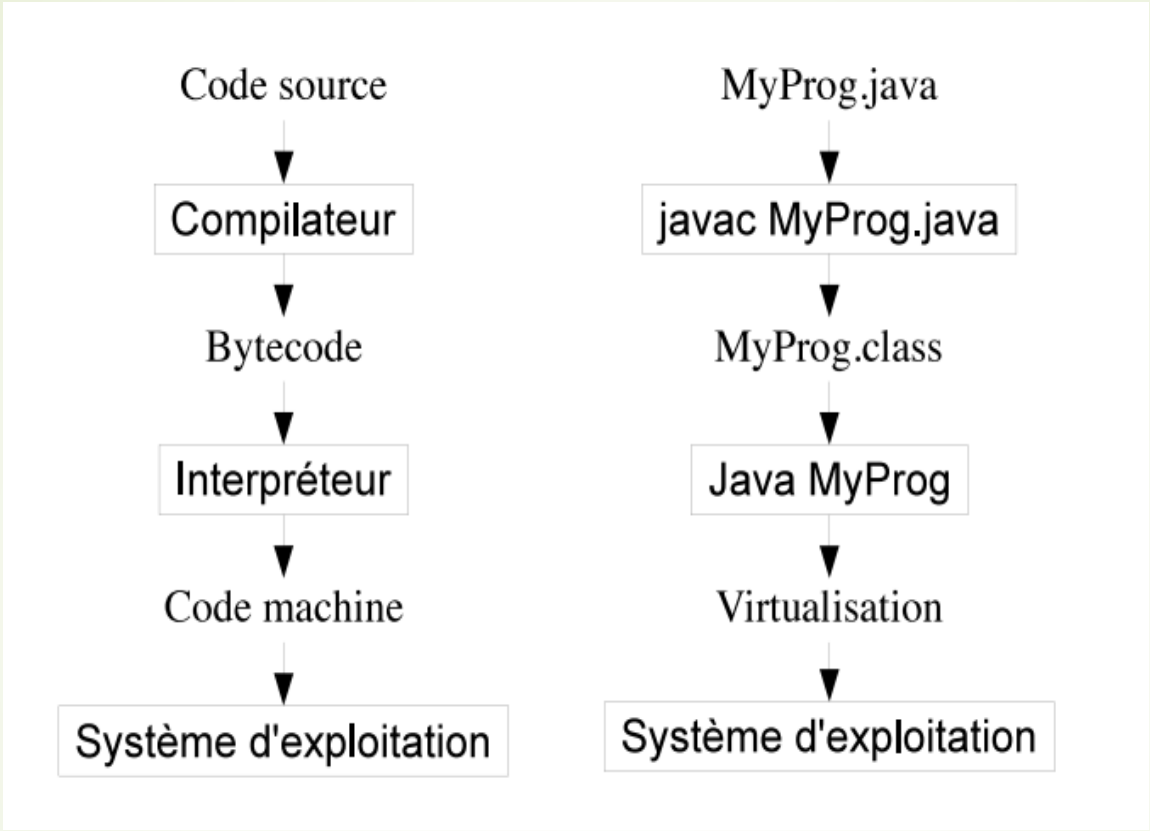


Figure 1: Interprétation du langage

II.1.1 Compilation

La compilation s'effectue par la commande **javac** suivie d'un ou plusieurs nom de fichiers contenant le code source de classes Java. Par exemple, **javac MyProg.java** compile la classe **MyProg** dont le code source est situé dans le fichier **MyProg.java**. La compilation nécessite souvent la précision de certains paramètres pour s'effectuer correctement, notamment lorsque le code source fait référence à certaines classes situées dans d'autres répertoires que celui du code compilé.

II.1.2 Interprétation

Le bytecode obtenu par compilation ne peut être exécuté qu'à l'aide de l'interpréteur. L'exécution s'effectue par la commande java suivie du nom de la classe à exécuter (sans l'extension **.class**). Comme lors de la compilation, il se peut que des classes d'autres répertoires soient nécessaires.

II.2 La syntaxe du langage

Le langage C a servi de base pour la syntaxe du langage Java :

- le caractère de fin d'une instruction est " ;"

```
a = c + c;
```

- les commentaires (non traités par le compilateur) se situent entre les symboles "/*" et "*/"
ou commencent par le symbole "//" en se terminant à la fin de la ligne

```
int a; // ce commentaire tient sur une ligne
int b;
```

Ou

```
/*Ce commentaire nécessite
 2 lignes*/
int a;
```

● les identificateurs de variables ou de méthodes acceptent les caractères {a..z}, {A..Z}, $, _ainsi que les caractères {0…..9} s'ils ne sont pas le premier caractère de l'identificateur. Il faut évidemment que l'identificateur ne soit pas un mot réservé du langage (comme **int** ou **for**).
Ex : **mon_entier** et **ok4all** sont des identificateurs valides mais **mon-entier** et **4all** ne sont pas valides pour des identificateurs.

II.3 Les types de données

II.3.1 Tableaux et matrices

Une variable est déclarée comme un tableau dès lors que des crochets sont présents soit après son type, soit après son identificateur. Les deux syntaxes suivantes sont acceptées pour déclarer un tableau d'entiers (même si la première, non autorisée en C, est plus intuitive) :

```
int[] mon_tableau;
int mon_tableau2[];
```

Un tableau a toujours une taille fixe qui doit être précisée avant l'affectation de valeurs à ses indices, de la manière suivante :

```
int[] mon_tableau = new int[20];
```


On peut également créer des matrices ou des tableaux à plusieurs dimensions en multipliant les crochets.
Exemple : **int[][] ma_matrice;**

II.3.2 Chaîne de caractères

Les chaînes de caractères ne sont pas considérées en Java comme un type primitif ou comme un tableau.
On utilise une classe particulière, nommée String, fournie dans le package **java.lang**.
Les variables de type **String** ont les caractéristiques suivantes :

- leur valeur ne peut pas être modifiée

```
String s1 = ”hello”;
String s2 = ”world”;
String s3 = s1 + ” ” + s2;
//Après ces instructions s3 vaut ”hello world”
```

- l’initialisation d’une chaîne de caractères s’écrit :

```
String s = new String(); //pour une chaine vide
String s2 = new String(”hello world”);
// pour une chaîne de valeur ”hello world”
```

- un ensemble de méthodes de la classe **java.lang.String** permettent d’effectuer des opérations ou des tests sur une chaîne de caractères

II.4 Structure de contrôle

Les structures de contrôle permettent d'exécuter un **bloc d'instructions** soit plusieurs fois (instructions itératives) soit selon la valeur d'une expression (instructions conditionnelles ou de choix multiple). Dans tous ces cas, un bloc d'instruction est

- soit une instruction unique ;
- soit une suite d'instructions commençant par une accolade ouvrante **"{"** et se terminant par une accolade fermante **"}"**.

II.4.1 Instruction conditionnelle

Syntaxe :

if (<condition>) <bloc1> [else <bloc2>]

ou

<condition>?<instruction1>:<instruction2>

<condition> doit renvoyer une valeur booléenne. Si celle-ci est vraie c'est <bloc1>

(resp <instruction1>) qui est exécuté sinon <bloc2> (resp <instruction2>) est exécuté.

La partie else <bloc2> est facultative.

Exemple:

```
if (a == b) {
    a = 50;
    b = 0;
} else {
    a = a - 1;
}
```

II.4.2 Instruction Itératives

Les instructions itératives permettent d'exécuter plusieurs fois un bloc d'instructions, et ce, jusqu'à ce qu'une condition donnée soit fausse. Les trois types d'instruction itératives sont les suivantes :

TantQue...Faire...

L'exécution de cette instruction suit les étapes suivantes :

1. La condition (qui doit renvoyer une valeur booléenne) est évaluée. Si celle-ci est vraie on passe à l'étape 2, sinon on passe à l'étape 4;

2. Le bloc est exécuté;

3. Retour à l'étape 1;

4. La boucle est terminé et le programme continue son exécution en interprétant les instructions suivant le bloc

Syntaxe :

while (<condition>) <bloc>

Exemple :

```
while (a != b) a++;
```

Pour...Faire

Cette boucle est constituée de trois parties : (i) une initialisation (la déclaration de variables locales à la boucle est autorisée dans cette partie) ; (ii) une condition d'arrêt ; (iii) un ensemble d'instructions à exécuter après chaque itération (chacune de ces instructions est séparée par une virgule)

L'exécution de cette instruction suit les étapes suivantes :

1. les initialisations sont effectuées ;

2. la condition (qui doit renvoyer une valeur booléenne) est évaluée. Si celle-ci est vraie on passe à l'étape 2, sinon on passe à l'étape 6 ;

3. le bloc principal est exécuté ;

4. les instructions à exécuter après chaque itération sont exécutées ;

5. retour à l'étape 2 ;

6. la boucle est terminée et le programme continue son exécution en interprétant les instructions suivant le bloc principal

Syntaxe :

for (<init>;<condition>;<instr_post_itération>) <bloc>

Exemple :

```
for (int i = 0, j = 49; (i < 25) && (j >= 25); i++, j--)  {
   if (tab[i] > tab[j]) {
       int tampon = tab[j];
```

Chapitre III: ÉLÉMENTS DE PROGRAMMATION JAVA

- Un programme écrit en Java est un ensemble de classes représentant les éléments manipulés dans le programme ainsi que les traitements associés.

- Exécuter le programme commence par l'exécution d'une classe qui doit implémenter une méthode particulière **"public static void main(String[] args)".**

- Les classes implémentant cette méthode sont appelées classes exécutables.

3.1 Classe HelloWorld

Une classe Java **HelloWorld** qui affiche la chaîne de caractères "Hello world" s'écrit :

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello world");
   }
}
```

L’exécution de cette classe va donner :

C:\>java HelloWorld
Hello world
C:\>

3.1 Création d’un objet: instancier une classe

Il est nécessaire de définir la déclaration d'une variable ayant le type de l'objet désiré.
La déclaration est de la forme **classe nom_de_variable**

Exemple :

```
MaClasse m;
```


L'opérateur **new** se charge de créer une instance de la classe et de l'associer à la variable

Exemple :

```
m = new MaClasse();
```

Il est possible de tout réunir en une seule déclaration

Exemple :
`MaClasse m = new MaClasse();`

Chaque instance d'une classe nécessite sa propre variable. Plusieurs variables peuvent désigner un même objet.

En Java, tous les objets sont instanciés par allocation dynamique.

Dans l'exemple ci dessus, la variable **m** contient une référence sur l'objet instancié (contient l'adresse de l'objet qu'elle désigne : attention toutefois, il n'est pas possible de manipuler ou d'effectuer des opérations directement sur cette adresse comme en C).

Si **m2** désigne un objet de type MaClasse, l'instruction **m2 = m** ne définit pas un nouvel objet mais **m** et **m2** désignent tous les deux le même objet.

- L'opérateur **new** est un opérateur de haute priorité qui permet d'instancier des objets et d'appeler une méthode particulière de cet objet : **le constructeur**.
- Il fait appel à la machine virtuelle pour obtenir l'espace mémoire nécessaire à la représentation de l'objet puis appelle le **constructeur** pour initialiser l'objet dans l'emplacement obtenu.
- Il renvoie une valeur qui référence l'objet instancié.

Remarque sur les objets de type String : Un objet String est automatiquement créer lors de l'utilisation d'une constante chaine de caractères sauf si celle ci est déjà utilisée dans la classe. Ceci permet une simplification dans l'écriture des programmes.

Exemple:

String chaine = '' bonjour''

Et

String chaine = new String ('' bonjour'')

Sont identiques

3.2 La durée de vie d'un objet

Les objets ne sont pas des éléments statiques et leur durée de vie ne correspond pas forcément à la durée d'exécution du programme.

La durée de vie d'un objet passe par trois étapes :

- La déclaration de l'objet et l'instanciation grâce à l'opérateur **new**

Exemple :
`nom_de_classe nom_d_objet = new nom_de_classe( ... );`

- L'utilisation de l'objet en appelant ces méthodes

- La suppression de l'objet : elle est automatique en java grâce à la machine virtuelle. La restitution de la mémoire inutilisée est prise en charge par le récupérateur de mémoire (garbage collector).

3.2 La création d'objets identiques

Exemple:

```
MaClasse m1= new MaClasse ();

MaClasse m2= m1 ;
```

m1 et m2 contiennent la même référence et pointent donc tous les deux sur le même objet : les modifications faites à partir d'une des variables modifient l'objet.
Pour créer une copie d'un objet, il faut utiliser la méthode clone() : cette méthode permet de créer un deuxième objet indépendant mais identique à l'original. Cette méthode est héritée de la classe Object qui est la classe mère de toute les classes en Java.

Exemple:

```
MaClasse m1= new MaClasse ();

MaClasse m2= m1.clone () ;
```

m1 et m2 ne contiennent plus la même référence et pointent donc sur des objets différents.

3.3 Les variables de classes

Elles ne sont définies qu'une seule fois quelque soit le nombre d'objets instanciés de la classe. Leur déclaration est accompagnée du mot clé **static**

Exemple:

```
Public class MaClasse (){
        static int compteur = 0 ;
}
```

L'appartenance des variables de classe à une classe entière et non à un objet spécifique permet de remplacer le nom de la variable par le nom de la classe.

Exemple:

```
MaClasse m =new MaClasse ();
int c1 = m.compteur ;
int c2= MaClasse.compteur
```

C1 et c2 possèdent la même valeur

Ce type de variable est utile pour par exemple compter le nombre d'instanciation de la classe qui est faite

3.4 La variable this

Cette variable sert à référencer dans une méthode l'instance de l'objet en cours d'utilisation. **this** est un objet qui est égale à l'instance de l'objet dans lequel il est utilisé.

Exemple :

```
private int nombre;
public maclasse(int nombre) {
   nombre = nombre; // variable de classe = variable en paramètre du constructeur
}
```

Il est préférable d'écrire

this.nombre = nombre ;

Cette référence est habituellement implicite :

Exemple :

```
class MaClasse() {
  String chaine = « test » ;
  Public String getChaine() { return chaine) ;
  // est équivalent à public String getChaine (this.chaine);
}
```

This est aussi utilisé quand l'objet doit appeler une méthode en se passant lui même en paramètre de l'appel.