Chapitre 2 : Programmation Orientée Objet (POO)

La POO est un paradigme de développement d'applications informatiques. Les objets du monde réel sont représentés de manières abstraites sous formes de classes. Chaque classe créée sera utilisée pour la création d'instance ou d'objet de cette classe. On peut alors dire qu'une classe est un modèle d'objet et que l'objet est un élément particulier de la classe.

Il existe diffèrents types de classe :

- Les classes abstraites *(qu'on ne peut pas instancier)*
- Les classes de types ***final*** *(qu'on ne peut pas hériter)*
- Les classes concrètes

La POO permet d'implémenter également la notion d'héritage entre classes qui est modélisé par UML.

Lors de la conception d'une application, le développeur doit concevoir les fonctionnalités de l'application dans un diagramme d'UML appelé diagramme de classe de conception. Dans ce diagramme, le programmeur définit les **classes** mais également les **interfaces** du système. Une interface est en général utilisée pour décrire les fonctionnalités qui s'appliquent aux objets d'une classe. Elle est aussi utilisée en java pour l'utilisation de l'inversion de contrôle avec les injections de dépendances. En java ces interfaces seront implémentées par des classes encore appelées classes d'implémentations.

La finalité de la création des classes ainsi que les interfaces est l'instanciation.

1) Notion de classe

a) Attributs

L'attribut fait partie des éléments qui caractérisent une classe. En java un attribut est précédé d'une portée ou visibilité qui peut être privée (***private***), publique (***public***) ou protégée (**protected**)

- public : sa définition est précédée de public (+), et il peut être utilisé par tout utilisateur de la classe.

00221 77 180 05 10 *layegaye001@gmail.com*

- **privé** : sa définition est précédée de private (-), et il ne peut être utilisé qu'à l'intérieur de la classe.
- **protégé** : sa définition est précédée de protected (#), et il ne peut être utilisé qu'à l'intérieur de la classe, ou des classes dérivées.
- **paquetage** : l'attribut peut être utilisé dans toute classe du même paquetage.

Un attribut qui n'a pas de visibilité dans une classe est par défaut à public. Tous les caractères du nom d'un attributs sont en minuscules.

b) Constructeur

Il permet de créer les objets d'une classe. Il permet aussi d'initialiser les attributs de la classe c'est-à-dire leur donner des valeurs. Il existe deux types de constructeurs :

- Le constructeur avec argument(s)
- Le constructeur sans argument

Le constructeur sans argument est encore appelé constructeur par défaut.

Syntaxe :

- Constructeur sans argument

```
public NomClasse(){

}
```

- Constructeur avec argument(s)

```
public NomClasse(type attribut1, type attribut2){
        this.attribut1 = attribut1 ;
        this.attribut2 = attribut2 ;
}
```

NB : Le mot clé this

Par convention, **this** désigne l'objet courant (c'est une référence sur l'objet courant) et il permet de lever l'ambiguïté de nom dans le cas ou une méthode a un nom de paramètre identique à un attribut.

```
//Constructeur sans argument
public Personne() {
}

//Constructeur avec arguments
public Personne(String nom, String prenom) {
    this.nom = nom;
    this.prenom = prenom;
}
```

c) Méthodes

Une méthode définit le comportement d'une classe et est toujours définie en public sauf dans les cas où elle est utilisée par une autre méthode de la même classe. Dans ce cas elle est définie en privée. Les méthodes sont en général publique pour interagir avec les autres objets. Dans une classe, on peut sentir le besoin d'avoir des méthodes qui permettent de faire une tache spécifique par rapport aux besoins d'une autre méthode publique. Dans ce cas celles-ci sont privées.

```
private String genererMatricule(int numero){
    String mat;
    mat = "Mat-"+this.nom.charAt(0) +
          this.prenom.charAt(0) +
          numero;
    return mat;
}
```

Il existe les méthodes de classes ainsi que les méthodes d'instances.

Les méthodes les plus utilisées dans les classes sont les getters et les setters.

- ✓ ***getter ou accesseur*** : c'est une méthode (**fonction**) publique spéciale permettant de récupérer la valeur d'un attribut **private**. Par convention, son nom est composé du préfix **get**, suivi du nom de l'attribut avec la première lettre en majuscule.

  Exemple :

```
public String getNom() {
    return nom;
}
```

- ✓ ***setter ou mutateur ou modificateur*** : c'est une méthode (**procédure**) publique spéciale permettant de modifier la valeur d'un attribut **private**. Un mutateur prend en paramètre la nouvelle valeur de l'attribut en question. Par convention, son nom est composé du préfix **set**, suivi du nom de l'attribut avec la première lettre en majuscule.

  Exemple :

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

Remarque :

Les attributs ou les méthodes d'une classe peuvent être static, et on parle de méthode ou d'attribut de classes, alors que pour les méthodes et attributs non statiques, on parle d'attribut et de méthode d'instances.

00221 77 180 05 10 *layegaye001@gmail.com*

2) L'objet

L'objet ou instance est un élément particulier d'une classe. En effet, il permet d'affecter des valeurs aux attributs d'une classe. Il permet aussi d'utiliser les méthodes d'instance définies dans une classe.

Il est créé grâce au mot clé ***"new"***, sa classe d'origine ou bien sa classe de base dans le cas de l'héritage.

```
//Instanciation d'objets
Personne p1 = new Personne();
Personne p2 = new Personne( nom: "GAYE", prenom: "Abdoulaye");
```

3) Héritage

L'héritage en POO permet de factoriser un code en évitant la répétition de l'écriture des attributs et de quelques méthodes.

Une classe qui hérite d'une autre classe est appelée ***classe dérivée*** ou ***sous classe*** ou encore ***classe fille***.

La classe qui permet l'implémentation de l'héritage est appelée ***classe de base*** ou ***super classe*** ou encore ***classe mère***.

En java si une classe fille veut faire appel à des fonctionnalités de sa classe de base, on utilise le ***mot clé super***.

Pour implémenter l'héritage en java, on utilise le mot clé ***extends*** (étendre)

Syntaxe :

```
public class BaseClass{
        //attributs
        //constructeurs
        //méthodes
}
```

```
public class SubClass extends BaseClass{
        //attributs
        //constructeurs
    public SubClass(){
        super() ;
    }
    public SubClass(typeBaseClass attributB, typeSubClass attributS){
        super(attributB);
    this.attributS = attributS;
    }
    public void
    m1(){
    super.m1() ;
        //code
    }
}
```

Exemple :

```
public class Etudiant extends Personne{
    1 usage
    private String matricule;

    public Etudiant() {
        super();
    }

    public Etudiant(String nom, String prenom, String matricule) {
        super(nom, prenom);
        this.matricule = matricule;
    }
```

4) Constante

Les constantes sont définies par le mot clé **final**.
Une constante ne peut être initialisée qu'une seule fois même si elle est publique.

Exemple:

```
public final int maxNotes = 10;
```

5) Attribut de classe (statique)

Par défaut, chaque instance de chaque classe occupe son propre espace mémoire.
Deux objets instances d'une même classe occupent donc deux espaces mémoire qui ne se recouvrent pas. Lorsque que l'on déclare un membre statique (**static**), il est au contraire placé dans un espace mémoire commun à tous les objets de la classe. Si un des objets modifie la valeur d'un champ statique (par exemple), tous les objets verront la valeur de ce champ modifiée.
On appelle attribut statique d'une classe (ou attribut de classe) tout attribut attaché à cette classe plutôt qu'à l'une de ses instances.
Un attribut statique peut exister, être référencé même si aucune instance de cette classe n'existe.
Un attribut statique ne peut référencer par le mot **this**.

```
public static int nbEtudiants = 0;
```

6) Notion d'interface

Une application informatique doit être fermée à la modification et ouverte à l'extension. Tout cela, les fonctionnalités ne doivent pas être codées dans les classes d'analyse mais elles doivent être d'abord définies dans les interfaces java avant de les implémenter. Une interface est définie grâce au mot clé ''interface'' à la place de ''class''.

Les classes qui implémentent les interfaces sont appelées classes d'implémentation. Pour définir l'implémentation d'une classe, on utilise le mot clé ''implements''

Syntaxe :

00221 77 180 05 10 *layegaye001@gmail.com*

```
public interface NomInterface{
        public Type methode1([parametres]) ;
        public Type methodeN([parametres]) ;
    }
public class NomClass implements NomInterface{
                //code ici
        }
```

La méthode toString()

■ La méthode toString est définie dans la classe Object ; en conséquence toutes les classes Java en hérite.

■ La méthode toString définie dans la classe Object ne fait pas grand-chose : elle renvoie le nom de la classe de l'objet concerné suivi de l'adresse de cet objet.

■ Cette méthode est utilisée par la machine Java toutes les fois où elle a besoin de représenter un objet sous forme d'une chaîne de caractères.

```
@Override
public String toString() {
    return this.nom+ " " +
            this.prenom + " "+
            this.anneeNaissance;
}
```

TP3 :

■ Créer une classe **Etudiant**

– nom, prénom : **chaînes de caractères**

– année de naissance : **entier**

– tableau de 10 (constante) notes : **double**

– nbNotes : **entier (indiquant le nombre actuel)**

■ Écrire deux constructeurs :

– un avec nom et prénom

– un avec nom, prénom et année de naissance

NB : Les constructeurs initialiseront tout le tableau de notes à 0 ainsi que la variable nbNotes.

■ Écrire des accesseurs et mutateurs pour les attributs : nom, prénom et année de naissance.

■ Écrire les méthodes suivantes :

– **moyenne()** : pour calculer et retourner la moyenne des notes

– **max()** qui retournera la meilleure note

– **min()** qui retournera la note la plus faible

– **ajouterNote(double note)** : pour ajouter la note n à l'étudiant

– **age()** : méthode qui retourne l'age de l'étudiant

■ Modifier la méthode **ajouterNote()** pour qu'elle affiche la nouvelle moyenne d'un étudiant à chaque entrée d'une nouvelle note

■ Écrire un programme principal qui à partir de deux étudiants dit lequel est le plus vieux

■ Redéfinir la méthode **toString** pour afficher le nom, le prénom et l'âge de l'étudiant

■ Utilisation des attributs de classe (**static**) : utiliser un attribut de classe pour conserver le numéro d'ordre de l'étudiant (étudiant n°1, n°2, ...)

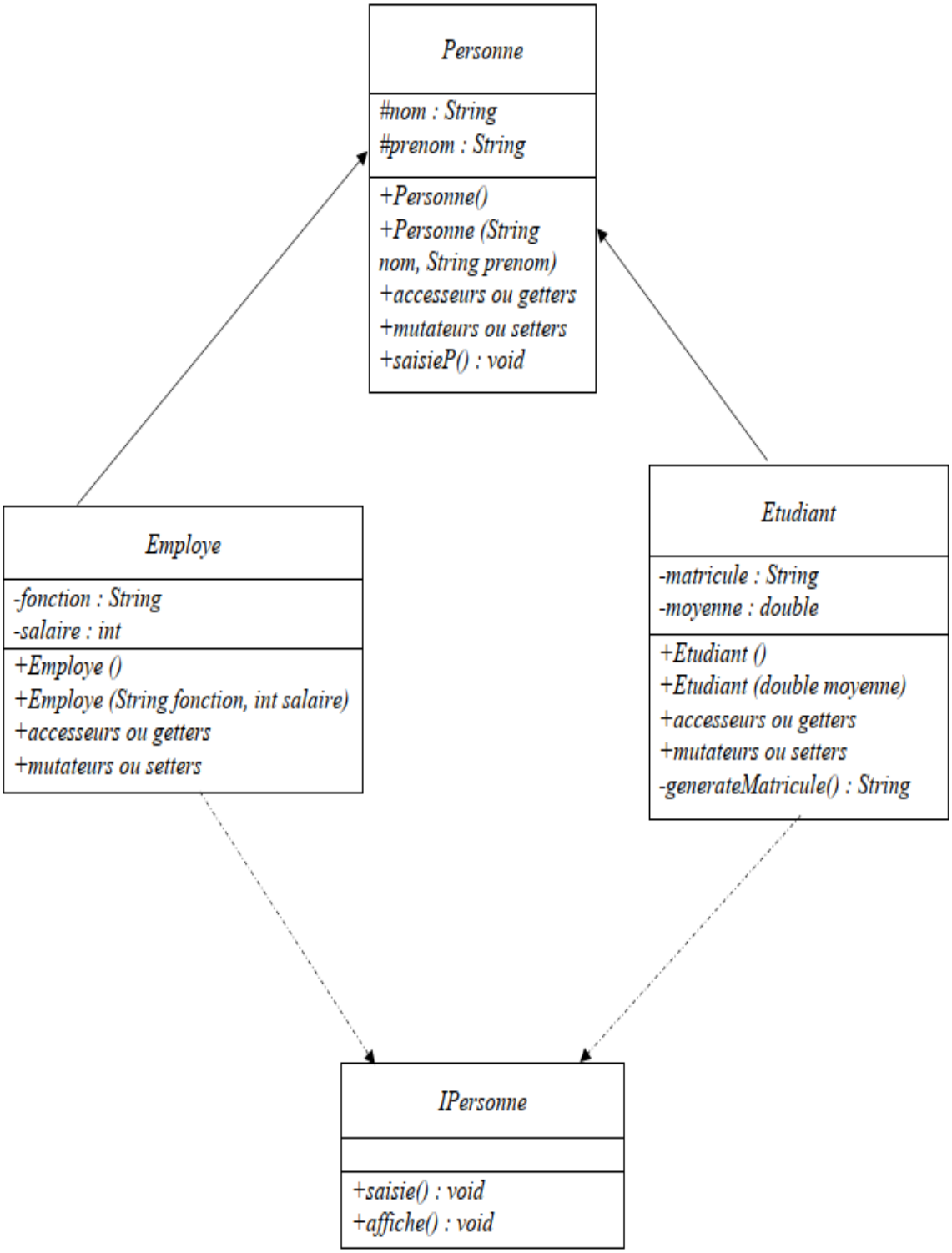
■ Modifier la classe Etudiant pour y ajouter l'attribut **matricule** : **chaine de caractères**.

■ Créer une méthode **genererMatricule()** qui permet de retourner un matricule unique pour chaque étudiant : Mat-suivi de la première lettre du nom suivi de la première lettre du prénom suivi du numéro d'ordre de l'étudiant.

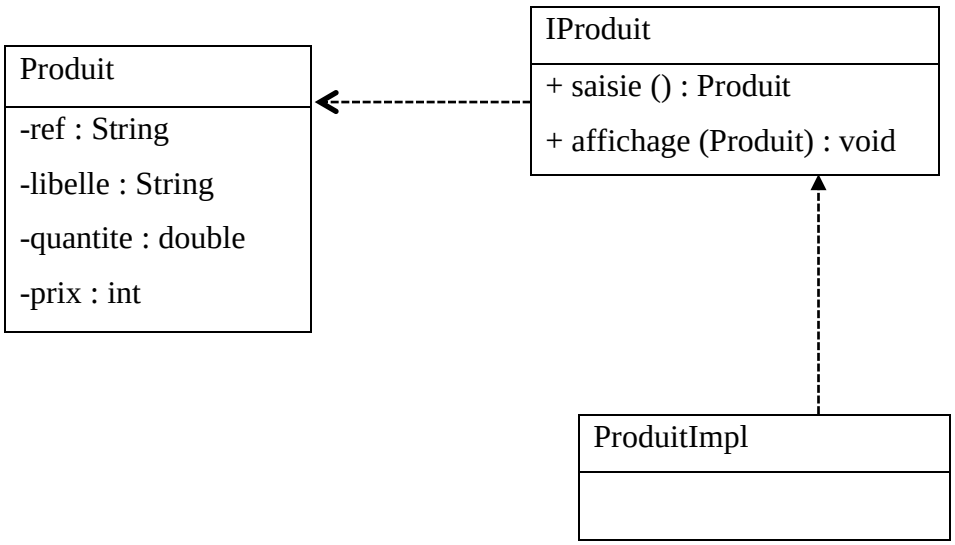
NB : Cette méthode n'est accessible qu'à l'intérieur de la classe.

00221 77 180 05 10 *layegaye001@gmail.com*

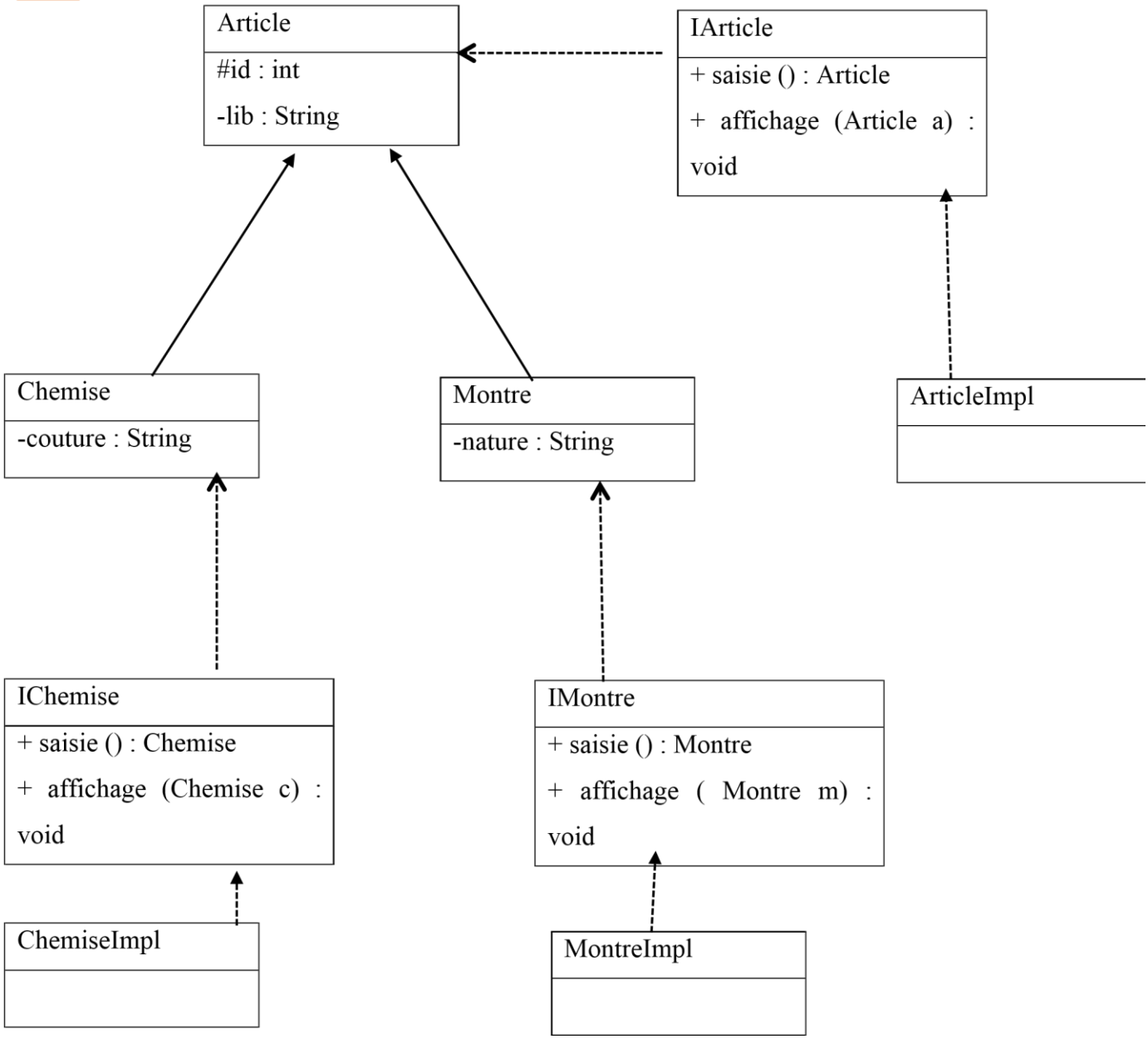
TP4 :


Personne
#nom : String
#prenom : String
+Personne()
+Personne (String nom, String prenom)
+accesseurs ou getters
+mutateurs ou setters
+saisieP() : void
Employe
-fonction : String
-salaire : int
+Employe ()
+Employe (String fonction, int salaire)
+accesseurs ou getters
+mutateurs ou setters
Etudiant
-matricule : String
-moyenne : double
+Etudiant ()
+Etudiant (double moyenne)
+accesseurs ou getters
+mutateurs ou setters
-generateMatricule() : String
IPersonne
+saisie() : void
+affiche() : void

TP4 :


Produit
-ref : String
-libelle : String
-quantite : double
-prix : int
IProduit
+ saisie () : Produit
+ affichage (Produit) : void
ProduitImpl

TP5 :


Article
#id : int
-lib : String
IArticle
+ saisie () : Article
+ affichage (Article a) : void
Chemise
-couture : String
Montre
-nature : String
ArticleImpl
IChemise
+ saisie () : Chemise
+ affichage (Chemise c) : void
IMontre
+ saisie () : Montre
+ affichage (Montre m) : void
ChemiseImpl
MontreImpl