Cours de programmation objet Java

Ahmed Zidna, Bureau : B37.
Département Informatique de l'IUT
Univerisité de Lorraine, Ile du Saulcy, F-57045 METZ
ahmed.zidna@univ-lorraine.fr

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

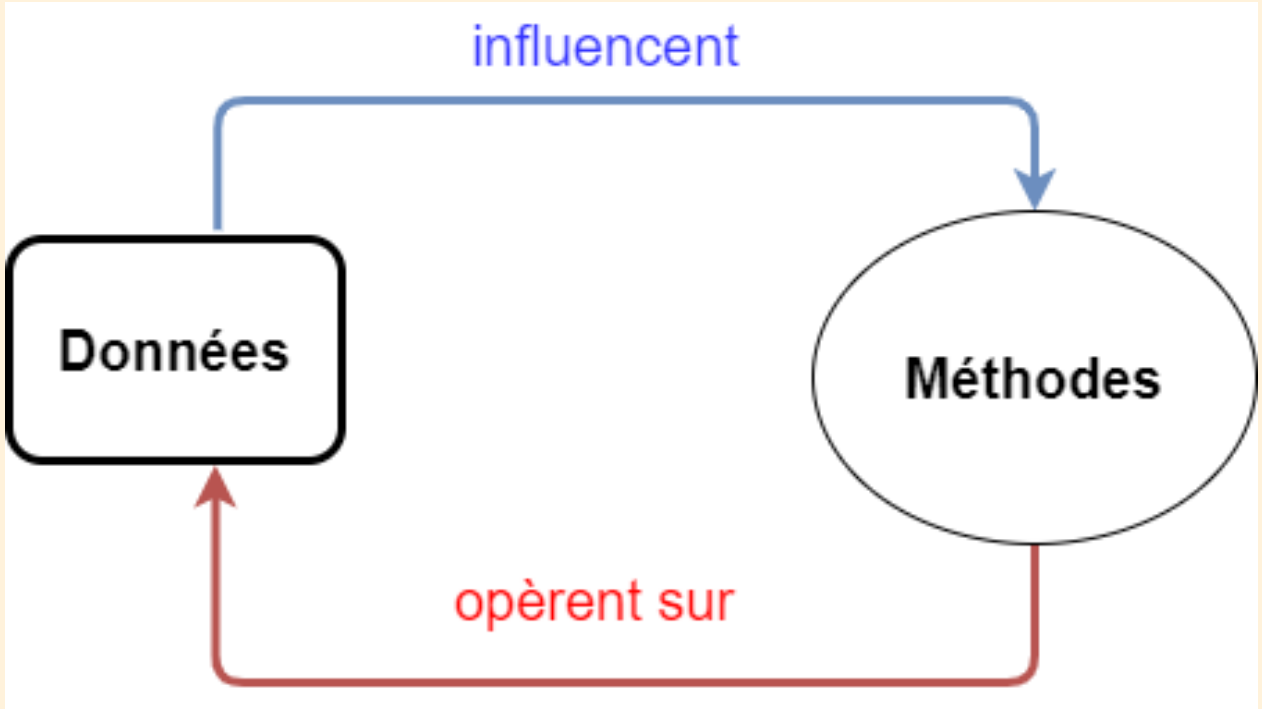
Sommaire

Sommaire

Sommaire

De la structure à la classe I

- En programmation procédurale, il y a séparation complète entre les
- données et les méthodes.



- Exemple de programme C++ pour saisir et afficher un employé.

```
int main(){
string nom="Robert";
int anneeNaissance=1956;
double salaire=3500;
afficher(nom,anneeNaissance,salaire);
}
```

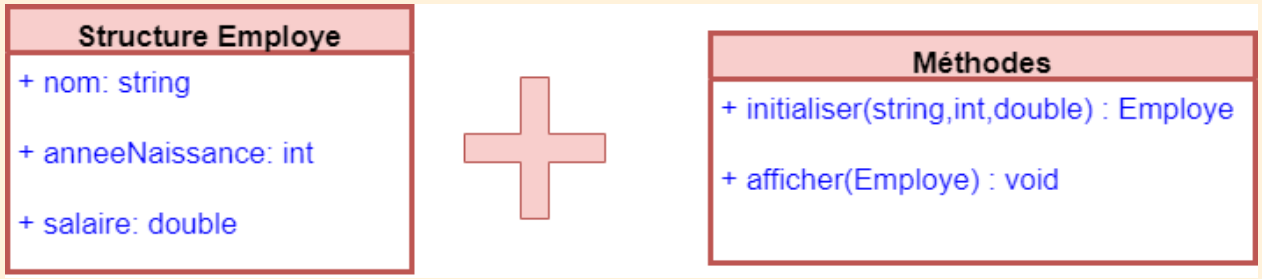
- La méthode afficher ne voit pas qu'elle affiche un employé

```
void afficher(string n,int a, double s){
cout<<n<<a<<s<<endl;
}
```

- Abscence de lien sémantique entre les variables nom, anneeNaissance et salaire alors qu'il s'agit d'un employé

De la structure à la classe I

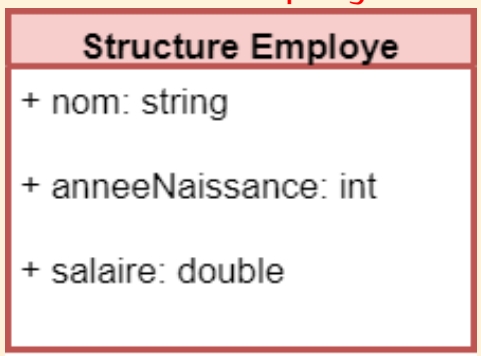
- En programmation procédurale, il y a séparation complète entre les
- données et les méthodes.



- **Amélioration** :La structure Employé réunit les données

De la structure à la classe I

- La structure Employe en UML et implémentation en C++ :




```
1  struct Employe{
   string nom;
3  int annee_naissance;
   double salaire;
5  };
```

- Voici un programme qui teste Employé

```
1  int main(){
   Personne p;
3  p=initialiser("Dupont",1986,2300);
   afficher(p);
5  return 0;
   }
```

De la structure à la classe I

- Méthode initialiser.

```
Employe initialiser(string ch,int a,double s){
Employe p;
p.nom=ch;
p.annee_naissance=a;
p.salaire=s;
return p;
}
```

- Méthode afficher.

```
void afficher(Employe p){
cout<<"Personne :"<<endl;
cout<<p.nom<<p.anneeNaissance<<p.salaire<<endl;
}
```

Inconvénient de la programmation procédurale I

- On ne voit pas que les méthodes initialiser et afficher sont intimmement liées à la structure Employe.
- L'utilisateur peut modifier de manière brutale les champs de la structure.

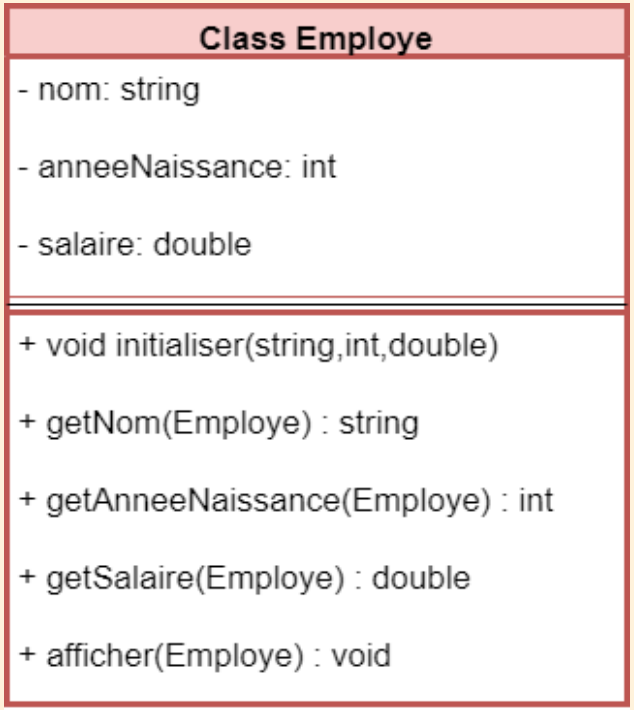
```
int main(){
p.nom="Durand";
p.salaire=4500;
return 0;
}
```

- **Solution** : la notion de Classe offre la possibilité d'encapsuler les données et les méthodes.

Encapsulation et Abstraction I

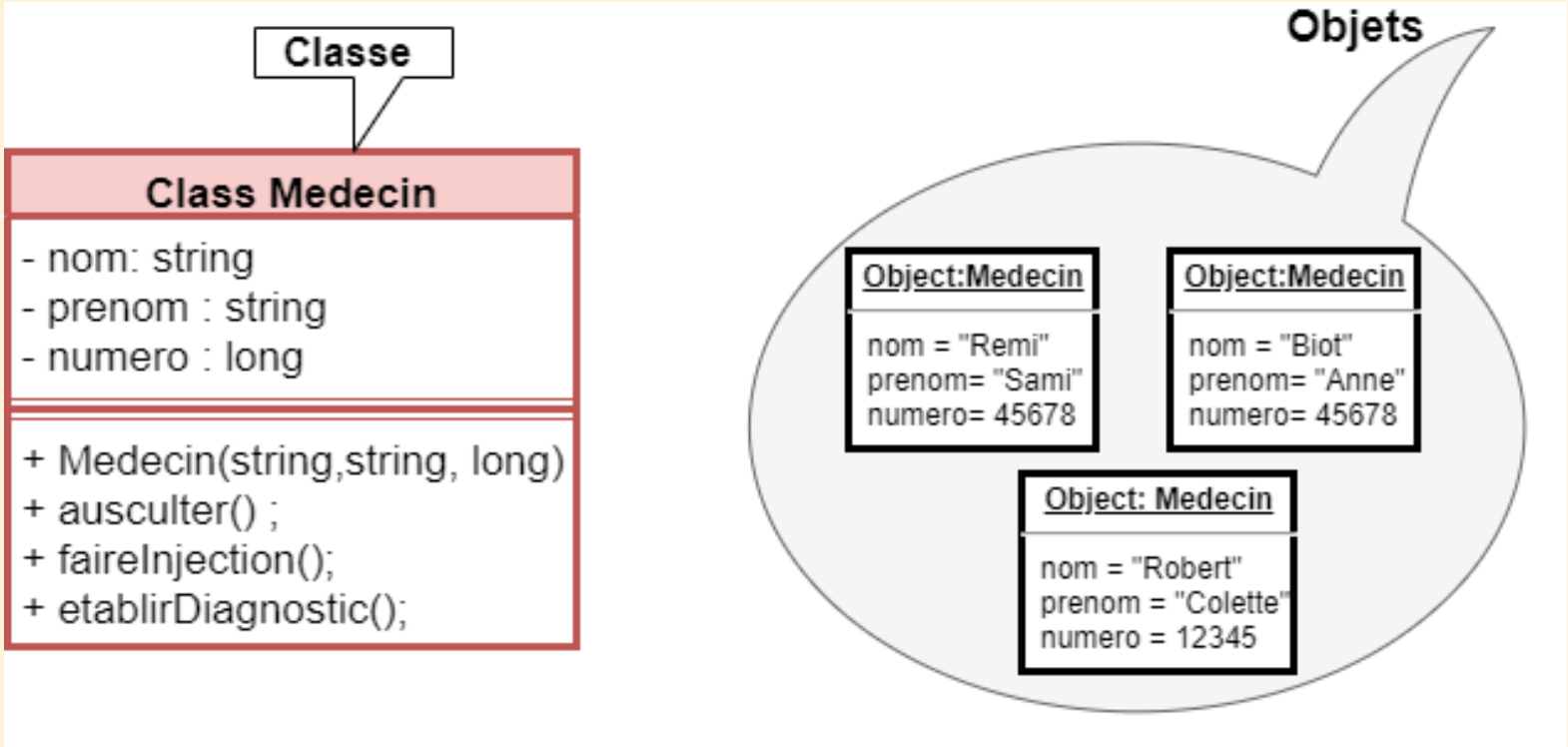
La classe offre deux concepts clé des LOO :

1. Encapsulation
   ⇒ regrouper données membres (attributs de la classe)
   ⇒fonctions membres (méthodes de la classe)



Encapsulation et Abstraction II

2. Abstraction
   - ⇒ Regrouper des objets ayant le même comportement
   - ⇒ Identifier les propriétés communes des objets



Concepteur/Utilisateur de la classe I

Il y a deux niveaux de perception de la classe :

1. externe : visible par le programmeur utilisateur.
   $\Rightarrow$ L'utilisateur ne "voit" que ce que la classe "veut bien lui montrer"
2. interne : implémenté par le programmeur concepteur.
   $\Rightarrow$ La classe permet de Masquer en partie le fonctionnement interne d'une classe
   $\Rightarrow$ Cette partie est transparente (n'est pas utile) pour l'utilisateur

Visibilité dans une classe I

- La classe offre la possibilité de protèger l'information contenue dans un objet
  $\Rightarrow$ Protèger les attributs des objets :
- Trois niveaux de "visibilité" pour les attributs et les méthodes :
  1. public :
     $\Rightarrow$ accessibles à l'extérieur de la classe
  2. private :
     $\Rightarrow$ accessibles seulement au sein de la classe.
  3. protected :
     $\Rightarrow$ accessibles seulement aux classes dérivées.

Quelques avantages I

La notion de classe permet d'écrire des programmes :

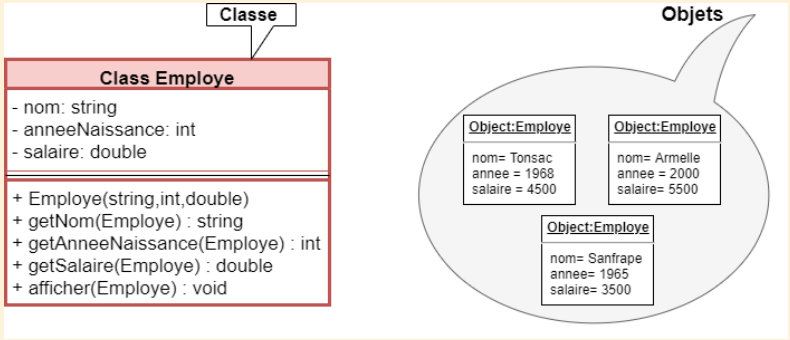
1. lisibles :
   ⇒ Chaque classe est définie par des attributs et des compétences uniques.
2. modulaires :
   ⇒ mécanisme basée sur la notion de classe.
3. robustes :
   ⇒ Garantie de l'intégrité de l'objet (cohérence de type, meilleur contrôle des erreurs)
   ⇒ Adapté pour les modifications et les changements.

⇒ Conséquence : Facilité de la maintenance du code.

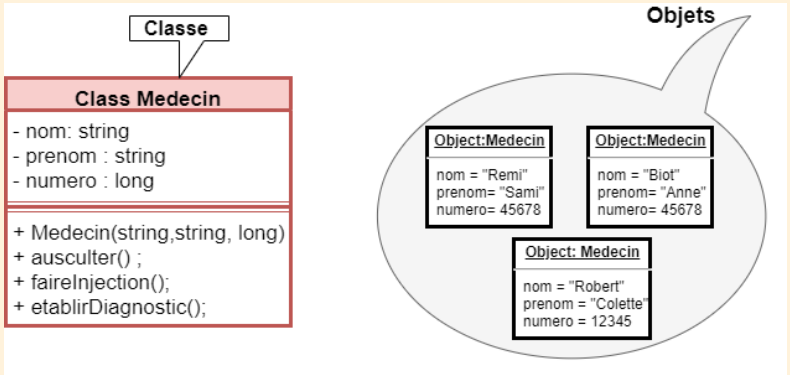
classe et objet I

⇒ Une classe peut décrire une infinité d'instances

1. Les instances de Employe sont des objets concrets



2. Les instances de Medecin sont des objets concrets



Objet I

Definition

Objet = Etat + Comportement + identité

1. Etat : Ensemble des valeurs des attributs de l'objet à un instant donné
2. Comportement : Regroupe les compétences d'un objet et décrit les actions et réactions de cet objet.
3. Identité : Ce qui identifie l'objet de manière non-ambigue
   - Attribuée implicitement à la création de l'objet
   - Indépendante de l'état ! 2 objets différents peuvent avoir le même état.

Comportement d'un objet I

1. Exemple de comportement d'un objet Employe

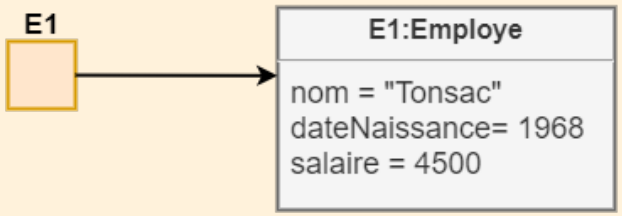
```
1  Employe E1= new Employe("Tonsac", 1968,4500);
   E1.afficher();
3  E1.setSalaire(7000);
```

2. Exemple de comportement d'un objet Medecin

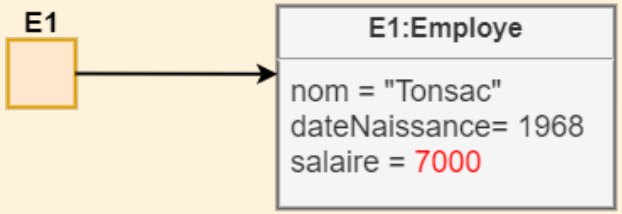
```
1  Medecin M1=new Medecin("Biot","Anne",45678);
   M1.etablirDiagnostic();
3  M1.faireInjection();
   M1.ausculter();
```

Comportement d'un objet I

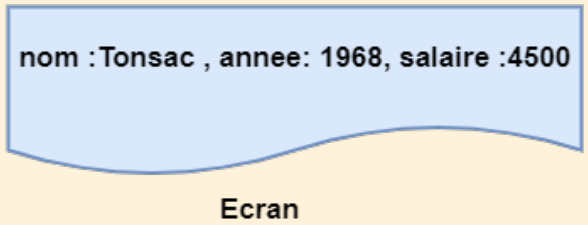
1- E1=new Employe("Tonsac",1968,4500);



2- E1.setSalaire(7000)



3- E1.afficher()



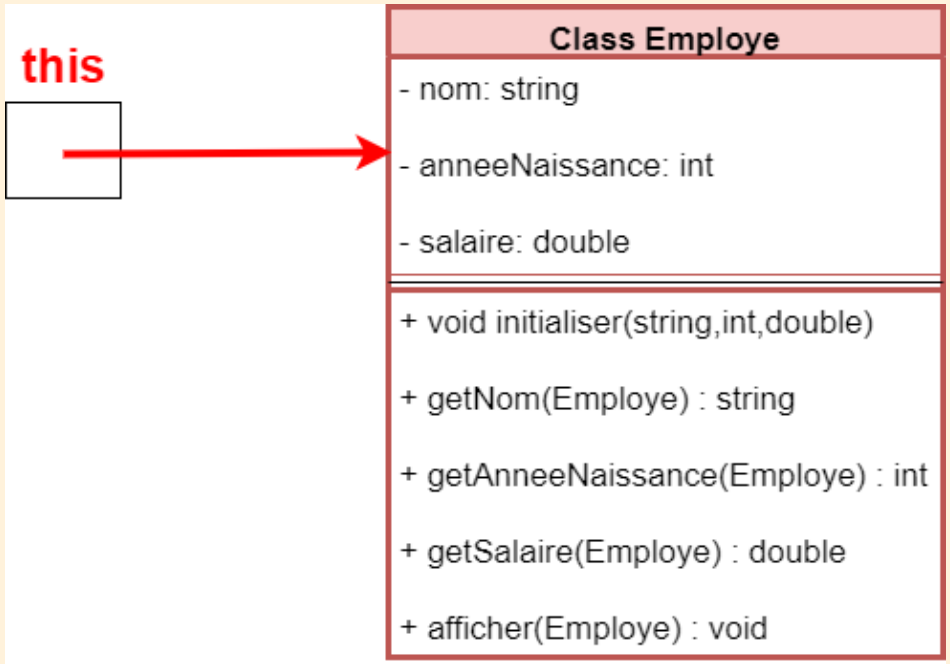
Classe Employe I

- Voici la classe java Employe

```
class Employe{
private string nom;
private int annee_naissance;
private double salaire;
public Employe(string ch,int a,double s)
{nom=ch; annee_naissance=a; salaire=s;}
public string getNom()
{return nom;}
public int getAnnee()
{return anne_naissance;}
public int  getSalaire()
{return salaire;}
public void afficher()
{System.out.println(nom+annee_naissance+salaire);}
}
```

Pointeur this I

- L'objet pour lequel une fonction membre est appelée constitue toujours un argument explicite de celle-ci.



- Il est toujours accessible et un pointeur est toujours associé à cet objet : **this**

Pointeur this I

- Voici la fonction main pour tester la classe Employe

```
1  public static main(String args[]){
   Employe E=new Employe("Dupont",1986,2300);
3  E.afficher();
   return 0;
5  }
```

- Les attributs de l'objet E sont accessibles avec this

```
1  public Employe(string ch,int a,int s){
   this.nom=ch;
3  this.annee_naissance=a;
   this.salaire=s;
5  }
```

```
1  public void afficher(){
   System.out.println(this.nom+this.anneNaissance+this.salaire);
3  }
```

Constructeur d'une classe I

- Un constructeur est chargé d'initialiser une instance de la classe au moment de la création de l'objet.
- Il porte le nom de la classe
- Une classe peut avoir plusieurs constructeurs :
  - constructeur sans paramètre,
  - constructeurs avec des paramètres,

```
1  public Employe(string n, int a, double s){
   this.nom=n;
3  this.anneeNaissance=a;
   this.salaire=s;
5  }
```

Surcharge du constructeur Employe I

1. constructeur à trois paramètres

```
1  public Employe(String n,int a,int s){
   this.nom=n; this.anneeNaissance=a; this.salaire=s;
3  }
```

2. constructeur à deux paramétres

```
1  public Employe(String n,int a){
   this.nom=n; this.anneeNaissance=a;this.salaire=1200;
3  }
```

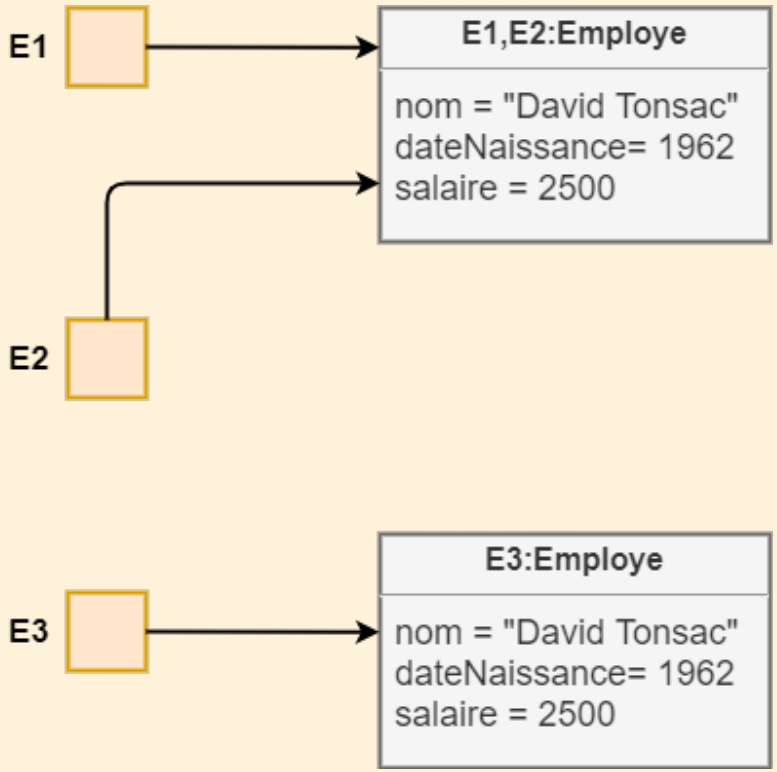
3. Constructeur sans paramètres

```
1  public Employe(){
   this.nom="toto";this.dateNaissance=2000; this.salaire
        =20;}
```

Constructeur de copie I

- Le constructeur de copie permet de dupliquer un objet.

```
Employe E1("DavidTonsac", 1962, 2500);
Employe E2=E1; //initialisation de E2 avec E1
Employe E3(E1); //constructeur de copie E3 avec E1
```



Variable de classe : variable statique I

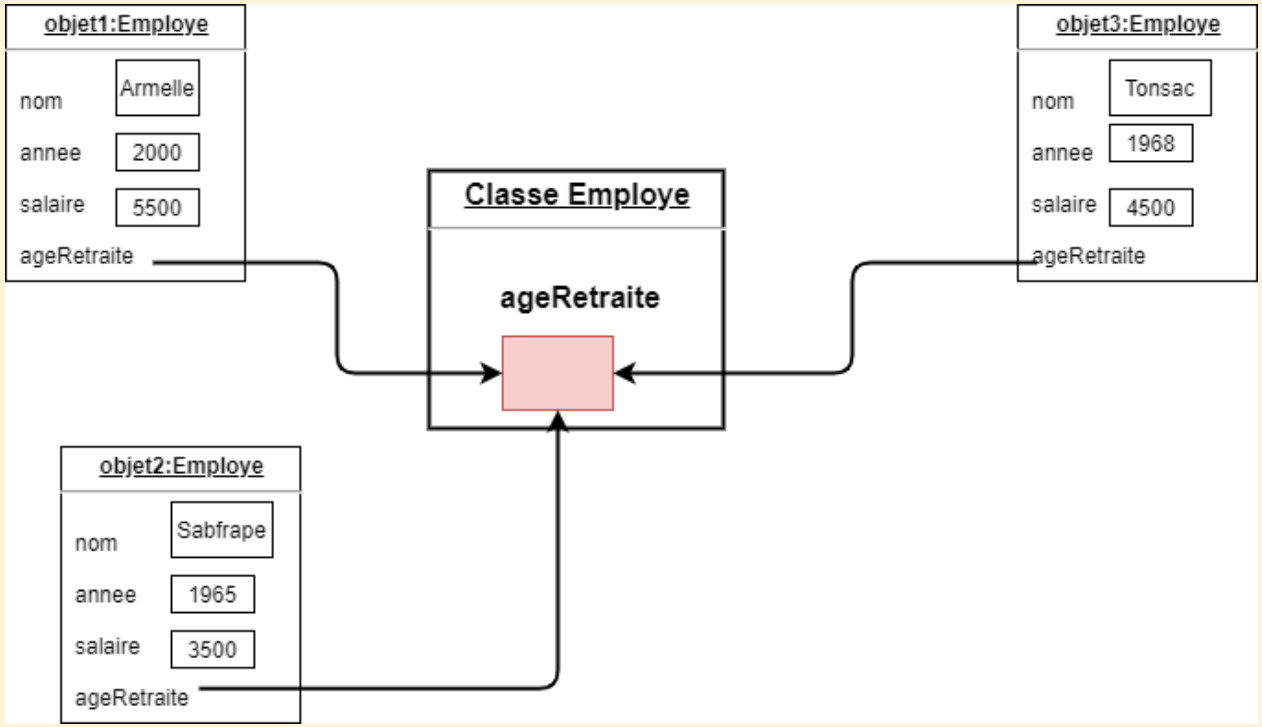
- Une variable statique = attribut commun à tous les objets.

```
1  class Employe{
   private string nom;
3  private int annee_naissance;
   private double salaire;
5  private static int ageRetraite=65;
   public :
7  };
```

- L'attribut ageRetraite est paratagé par tous les employés.

Variable de classe : variable statique I

- Tous les objets font référence à un unique espace mémoire ageRetraite.



Méthode statique I

Une méthode statique sert à définir des méthodes utilitaires

- Une méthode statique n'est pas accessible avec un objet.
  ⇒ N'est liée à aucun objet :
  ⇒ Ne peut pas accèder à des variables d'instances
  ⇒ Ne peut pas accèder à des méthodes d'instances
- Une méthode statique est accessible avec le nom de la classe.
  ⇒ Peut accèder à des variables statiques
  ⇒ Peut accèder à des méthodes statiques.

Méthode de classe : Méthode statique I

- Exemple de méthode statique qui calcule le nombre d'employés créés

```
1   class Employe{
    private string nom;
3   private int annee_naissance;
    private double salaire;
5   private static int nb=0;
    public Employe(string a, int d, double s){
7   this.nom=a; this.annee_naissance=d; this.salaire=s;
    nb++;// j'incremente a chaque creation d'un employe
9   }
    public static int nbEmployes(){
11  return nb;
    }
```

Méthode de classe : Méthode statique I

- Voici un programme qui affiche le nombre d'employés créés

```
    public static void main(String args[]){
2    Employe p1("DavidTonsac", 1962, 2500);
     Employe p2("AndreSanfrape", 1970, 4500);
4    Employe p3("SandraFaran", 2000,3500);
     p1.afficher();
6    p2.afficher();
     p3.afficher();
8    System.out.println("Nombredepersonnes :"+ Employe ::nbEmployes());
     }
```

- nbEmployes est statique ⇒ Employe ::nbEmployes()

Classe Fraction I

Une fraction est un nombre sous la forme a/b où a, b sont des entiers avec $b \neq 0$.

Class Fraction
- num : int - den : int
+ Fraction(int, int) + somme(Fraction): Fraction + getNum() : int + getDen() : int + setNum(int): void + setDen(int): void + afficher() : void

Classe Fraction I

- Voici l'interface de la classe Fraction

```
1  class Fraction{
   private int num;
3  private int den;
   public Fraction(int,int)
5  public int getNum()
   public int getDen()
7  public void setNum(int)
   public void setDen(int)
9  public Fraction somme(Fraction )
   public void afficher()
11 }
```

Classe Fraction I

- Voici le constructeur de la classe Fraction

```
1  Fraction ::Fraction(int a, int b){
       this.num=a;
3      this.den=b;
   }
```

- Voici la fonction somme des deux fractions this et r

```
   Fraction Fraction ::somme(Fraction r){
2      return new Fraction(this.num*r.den+this.den*r.num,this
           .den*r.den);
   }
```

Classe Fraction I

- Voici la fonction main pour tester

```
1  public static void main(String args[]){
   Fraction r1, r2, r3;
3  r1=new Fraction(2,3);
   r2=new Fraction(1,4);
5  r3=r1.somme(r2);
   }
```

- Question : Pourquoi n'a -t-on pas appelé la fonction somme ?

```
   public static void main(String args[]){
2     .........
   r3=somme(r1,r2);
4    .....
   }
```