

classe	Une méthode d'une classe est obligatoirement <u>déclarée</u> et <u>définie</u>	
	<pre>class Toto{ public void methode1(){ /*instruction*/ } public int methode2(){ /*instruction*/ } }</pre>	<pre>class Tata extends Toto{ }</pre>
classe abstraite	MIXTE : les méthodes peuvent être juste déclarées ou déclarées et définies Dans la classe héritière, on définit toutes les méthodes qui étaient juste déclarées <pre>abstract class nomClasseAbstraite{ }</pre>	
interface	Les fonctions sont juste déclarées → Aucune fonction n'est définie. Pour être utilisé et donc (instancié), Une interface, doit être héritée (implémentée). C'est uniquement dans la classe héritière qu'on définit TOUTES les méthodes avant même d'en utiliser une. <pre>interface nomClasseInterface{ }</pre>	
	<pre>interface Titi{ public void methode1(); public int methode2(); }</pre>	<pre>class Tata implements Titi{ public void methode1(){ } public int methode2(){ } }</pre>

1. Exercice : classe hérite d'une INTERFACE

1. Ecrire une *interface* **Animal** qui contient deux méthodes : **marcher()** et **courir()**

marcher() permet d'afficher **Je marche**
courir() permet d'afficher **Je cours**

Déclarer et définir la classe **Chat**, qui hérite de l'interface **Animal** et affiche en plus:
Je miaule

2. Affiche juste :

Je marche
Je miaule

2. Mot clé this

```
class Toto{
    int age ;    // variable membre ou attribut age
    public void saisirAge(int age){    // argument age
        this.age=age+3;    // pour enlever l'ambigüité de age=age+3 le mot clé this se met devant la variable
    }
}
```

Le mot clé **this** permet de différencier une variable membre (attribut) et d'un argument passé en paramètre à une fonction.

3. Les accesseurs : getter et setter → private

1.1.1 Le getter : à créer dans la classe elle- même

MISE EN PLACE

```
class Individu{
    private String nom = "DUBOIS" ;    //variable globale
    public String getNom(){    //permet de récupérer
        return nom ;
    }
}
```

APPEL DE METHODE

```
Individu ind = new Individu();
String nomDubois = ind.getNom();    //retourne le nom
```

2.1.1 Le setter : : Créer dans la même classe

Un setter est une méthode qui peut ne rien retourner et qui permet de **modifier** le contenu d'une variable encapsulée en **private**.

MISE EN PLACE

```
class Individu{
    private String nom = "DUBOIS";    //variable globale
    public void setNom(String nom ){
        this.nom = nom;
    }
}
```

APPEL DE METHODE

```
Individu ind = new Individu();
ind.setNom("DUPONT");    //mise à jour du nom à DUPONT
```

2. Exercice

Ecrire le programme **GestionEtudiantPrivate**.

Dans la classe Etudiant

1. Ecrire la classe Etudiant de variables globales encapsulées **private** : **nom - age - note**.
On initialisera le nom à **DUBOIS**.
- Mettre en place les méthodes de getter et de setter afin d'accéder à ces 3 variables private

Dans la classe principale GestionEtudiantPrivate

1. Créer un objet **etud2** avec le constructeur sans paramètre (en instanciant la classe),
 - Appeler la méthode **getNom()** qui permet de récupérer le nom de l'étudiant1 (voir **RESULTAT**)
 - Appeler la méthode **setNom()** (*attention, cette méthode a un argument*) qui permet de rajouter le nom **DUPONT - 23 ans - note à 12**
 - Appeler la méthode **getNom()** qui permet de récupérer les caractéristiques de l'étudiant2
 - Reprendre les **2 méthodes** pour les restes des variables et afficher la 3ème phrase.

RESULTAT

```
le 1er étudiant s'appelle: DUBOIS
le 2ème étudiant : DUPONT a 23 ans a une note de 12 sur 20
le 3ème étudiant s'appelle: PEREZ a 35 ans a une note de 18 sur 20
```

Constructeur

Lorsqu'on crée une classe,

- ✚ Java crée une **fonction spéciale** (un **constructeur sans argument**) de même nom que la classe qui permet de construire des objets,

Avec une fonction (constructeur), on peut avoir le même nom plusieurs fois,

On les différencie juste par le nombre d'argument(**surcharge**),

- ✚ Le système saura les différencier selon le nombre d'arguments.
- ✚ Lors de l'appel du constructeur, java va proposer les constructeurs selon le nombre d'arguments

1. **Constructeur** : pas de type de retour - this

Mise en place d'un constructeur : avec paramètre

```
class Individu{  
    String nom ;           //variable globale  
    public Individu (String nom ){  
        this.nom = nom;  
    }  
}
```

Appel (utilisation) d'un constructeur → Objet

```
Individu indDubois = new Individu("DUBOIS");
```

1. Exercice

Ecrire le programme **GestionEtudiantConstructeur**.

Dans la classe Etudiant

1. Ecrire la classe Etudiant de variables globales (dans la classe directement) : **nom - age - note**
- Créer les constructeurs ci-dessous qui nous permettra de créer des objets.
 - a. Constructeur **Etudiant** avec 1 paramètre : **nom**
 - b. Constructeur **Etudiant** avec 3 paramètres : **nom, age** et **note**

Dans la classe principale GestionEtudiantConstructeur

Manipulation avec un constructeur avec paramètre

2. Créer un objet **etud1** avec le constructeur à 1 paramètre, qui permet de préciser le nom de l'étudiant1 à **DUBOIS**

Afficher la première phrase du **RESULTAT** ci-dessous

3. Créer un objet **etud2** avec le constructeur à 3 paramètres, qui permet de préciser l'étudiant 2 à **PEREZ - 23 ans - 12 sur 20**

o o o

RESULTAT

```
le 1er étudiant s'appelle: DUBOIS
le 2ème étudiant : PEREZ a 23 ans a une note de 12 sur 20
le 3ème étudiant s'appelle: DUPONT a 35 ans a une note de 18 sur 20
```

2. Exercice

Ecrire le programme **GestionEtudiantTableau**. Les variables sont private, mettre en place un getter et setter pour chaque variable.

1. Créer un tableau d'objet de taille 3, dont les valeurs sont de type **Etudiant** (classe à importer s'il faut). En instanciant le type.
2. Mettre en place un constructeur qui permet de créer les 3 étudiants ci-dessous.
3. Rajouter les objets étudiant créés, dans le tableau.
4. Afficher le nom PERREZ dans une phrase (mettre en place un getNom())

Nom	Age	Note
DUBOIS	23	13
DUPONT	21	8
PERREZ	20	9

Boucle for each :

```
for(type variable : varTableau){    //différentes valeurs dans variable
    System.out.println(variable);
    System.out.println(variable.getNom());
}
```

3. Exercice

Même exercice en mettant les valeurs (objets) dans un tableau dynamique

```
import java.util.ArrayList;

public class GestionObjet {
    public static void main(String[] args) {
        //création de liste d'objet de type Personne
        ArrayList<Personne> listPersonne = new ArrayList<Personne>();

        //instance pour créer un objet
        Personne pers1 = new Personne("Dubois");
        Personne pers2 = new Personne("Dupont");

        // Ajout des objets dans la liste
        listPersonne.add(pers1);
        listPersonne.add(pers2);

        // boucle for each
        for (Personne perso:listPersonne){
            System.out.println(perso.getNom());
        }
    }
}

class Personne{
    String nom;
    public String getNom(){
        return nom;
    }
    public Personne(String nom){
        this.nom=nom;
    }
}
```