



Rappel de cours

Classe et objet

Les possibilités de Programmation Orientée Objet reposent sur le concept de classe.
Une classe est la généralisation de la notion de type défini par l'utilisateur, dans lequel se trouve associées à la fois des données (on parle de « membres donnée » ou attributs) et des fonctions (on parle de « fonctions membre » ou de méthodes). C'est une sorte de moule qui nous permettra par la suite de créer des objets

En P.O.O. pure, les données sont privées et « encapsulées », ce qui signifie que leur accès ne peut se faire que par le biais des méthodes.

1) Déclaration et définition des membres et méthodes d'une classe

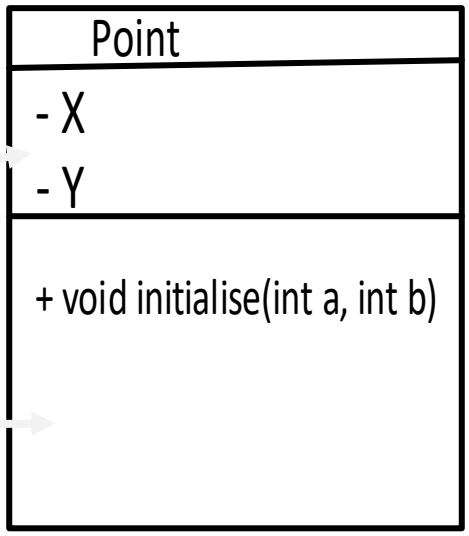
Une classe commence toujours par la **déclaration** qui précise quels sont les membres (données ou membres) publics (c'est-à-dire accessibles à l'utilisateur de la classe) et quels sont les membres privés (inaccessibles directement à l'utilisateur de la classe). On utilise pour cela les mots-clés public et private, comme dans cet exemple dans lequel la classe point comporte deux membres donnée privés x et y.
On déclare et on définit par la suite les méthodes de cette classe ici initialise
La **définition** d'une classe consiste à fournir les définitions des méthodes.

```
/* ------------ Déclaration de la classe point ------------- */

class point
{
/* déclaration des membres privés ou publics*/
private :
int x ;
int y ;

/* déclaration  et définition des méthodes publics */
public  void initialise (int abs, int ord)
{
x = abs ; y = ord ;
}

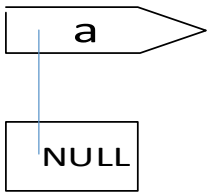
} ;
```



2) Utilisation d'une classe

On déclare un « objet » d'un type classe donné en faisant précéder son nom de celui de la classe, comme dans l'instruction suivante qui déclare deux objets a et b de type point :

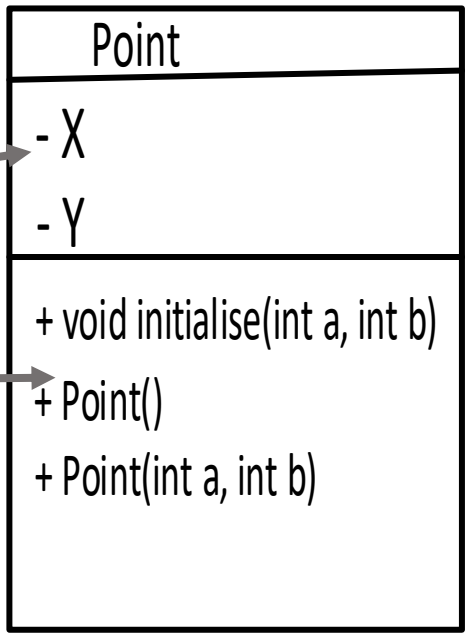
point a, b ;
A ce moment-là on a créé qu'une référence a et b NULL vers un objet du type point qui sera créé plus tard



Une méthode membre portant le même nom que sa classe se nomme un **constructeur**. Dès qu'une classe comporte un constructeur (au moins un), il n'est plus possible de déclarer un objet du type correspondant, sans fournir des valeurs pour les arguments requis par ce constructeur (sauf si ce dernier ne possède aucun argument constructeur par défaut). Le constructeur est appelé **après** l'allocation de l'espace mémoire destiné à l'objet (opérateur new).

```
class Point
{
/* déclaration des membres privés ou publics*/
private  int x ;
private int y ;

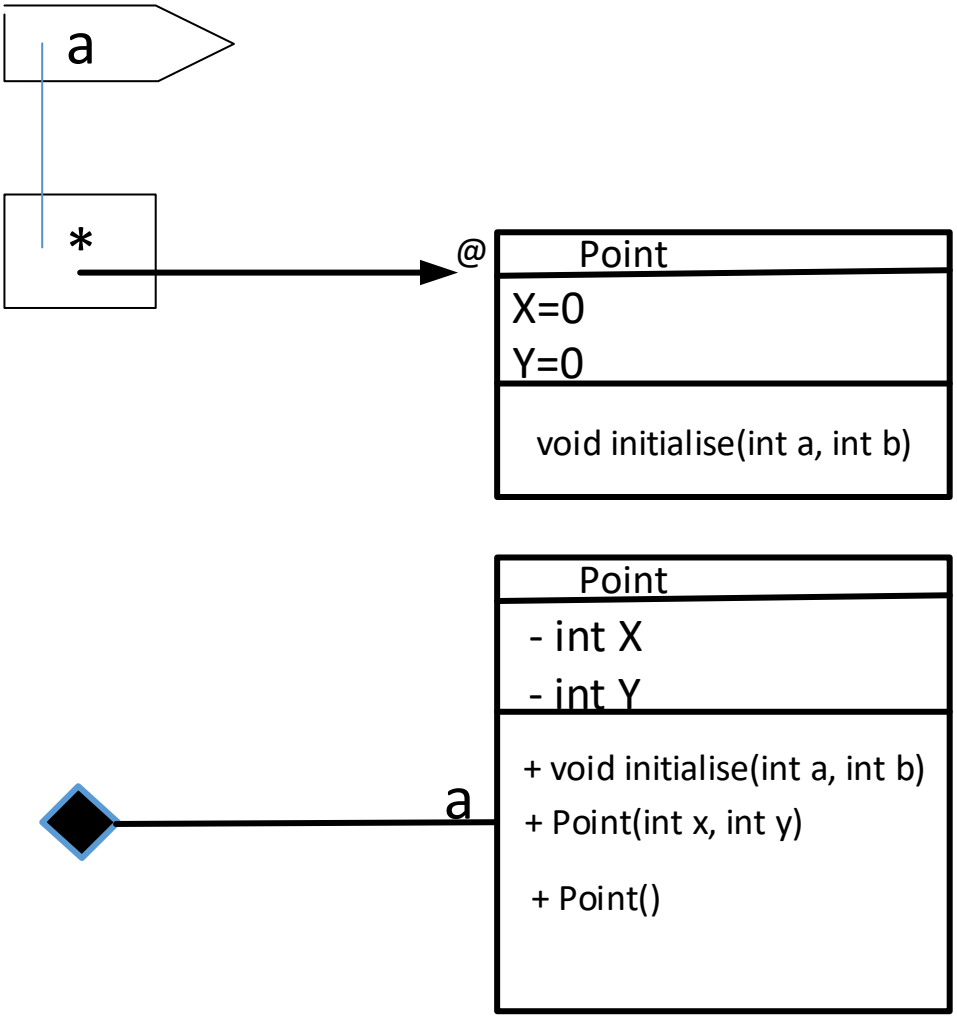
/* déclaration  et définition des méthodes publics */
public  void initialise (int abs, int ord)
{
x = abs ; y = ord ;
}
public  void Point ()
{
x = 0 ; y = 0 ;
}
public  void Point (int x, int y)
{
this.x = x ; this.y = y ;
}
} ;
```



Dans tous les cas s'il y a appel du **constructeur**, celui-ci a lieu après l'allocation de l'emplacement mémoire destiné à l'objet.
Les objets sont créés par l'opérateur new, auquel on doit fournir, le cas échéant, les valeurs des arguments destinés à un constructeur, comme dans cet exemple.

```
Point b ;
Point a = new Point( ) ; // appel du constructeur par défaut
b= new Point(2,3 ) ;
```

On a créé deux instances de la classe Point dont les références sont a et b
Pour l'objet de référence a on retrouve :



D'une manière générale on peut écrire :

```
Nomdelaclasse   Nomdelaréférence  = new Appelduconstructeur(.....) ;
```

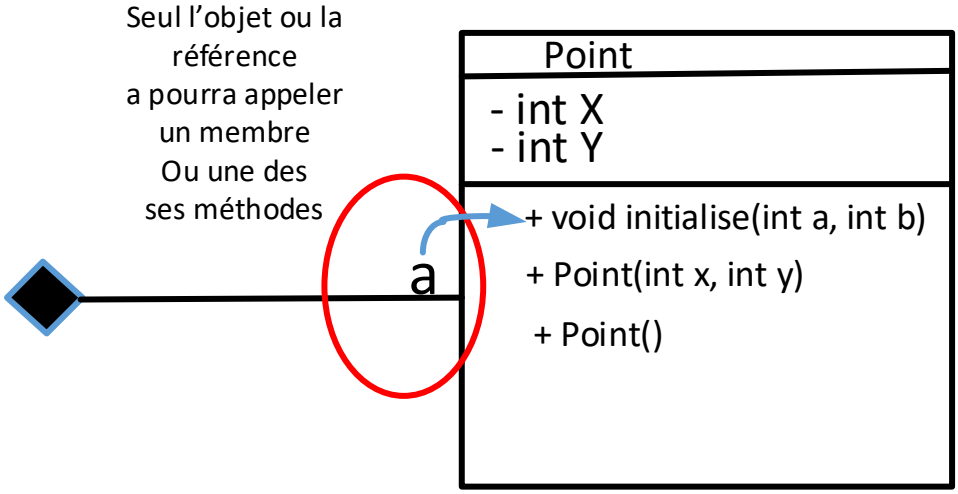
Nomdelaréférence est aussi baptisée nom de l'objet.

3) Appel des membres ou des méthodes.

On peut accéder à n'importe quel membre ou méthode public d'une classe en utilisant l'opérateur ".".

a. **initialise (2, 3) ;**
initialise (2,3) ;

Attention dans la majorité des cas l'appel d'un membre ou d'une méthode se fait toujours par un objet de la classe ici a :



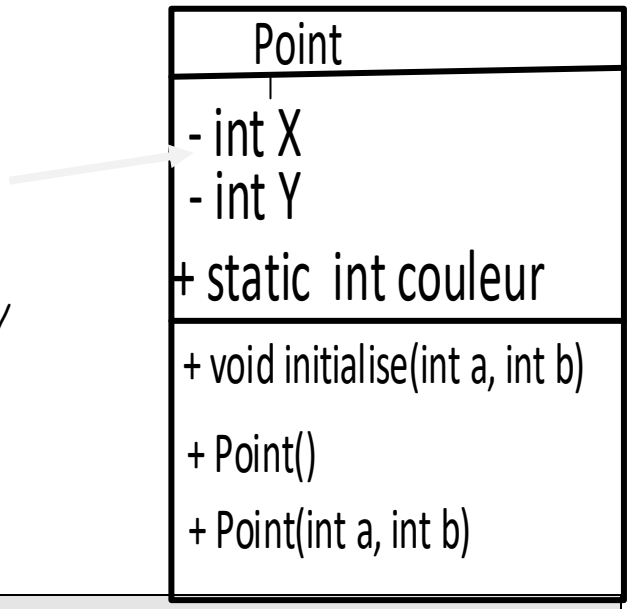
Deux cas peuvent déroger à cette règle la déclaration **d'un membre ou d'une méthode statique**.

Prenant le cas d'un membre statique :

On veut que la couleur du point en échelle de gris soit la même pour tous les objets créés on va donc déclarer un membre public statique color du type int :

```
class Point
{
/* déclaration des membres privés ou publics*/
private  int x ;
private int y ;
public static int  color= 2 ;

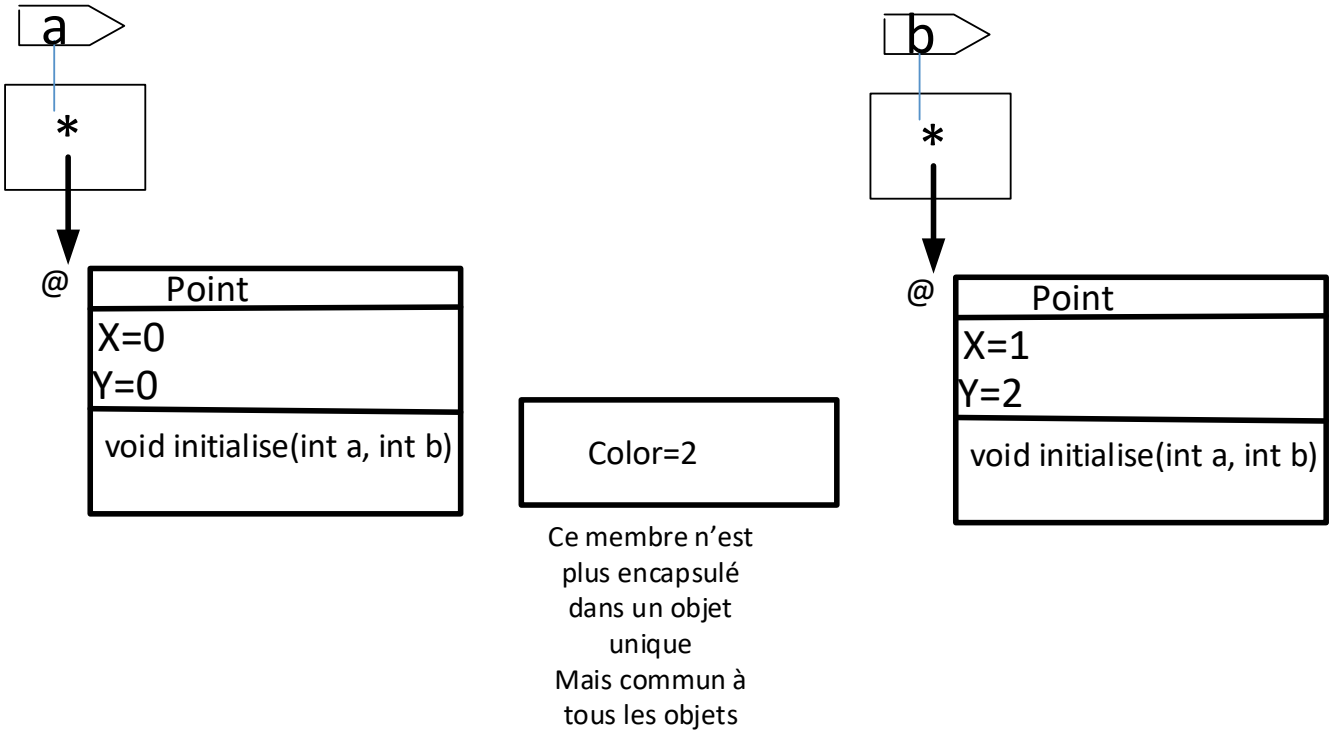
/* déclaration  et définition des méthodes publics */
---------------------------------- ;
---------------------------------- ;
---------------------------------- ;
---------------------------------- ;
} ;
```



Point a = new Point( ) ; // appel du constructeur par défaut
Point b= new Point(1,2) ;
Lors d'un appel d'une méthode même une méthode sans paramètre d'entrée n'oubliez pas les parenthèses :

Point a = new Point ;

Un membre donné déclaré avec l'attribut **static** est partagé par tous les objets de la même classe. Il existe même lorsqu' aucun objet de cette classe n'a pas été déclaré. En général, son initialisation se fait dans la déclaration de la classe.



a.color=2 ;
b.color=3 ;

On pourra appeler ce membre dans n'importe quels objets de n'importe quelles classes en utilisant la syntaxe suivante :
Point.color=4 ;// syntaxe Nomdeclasse.membre

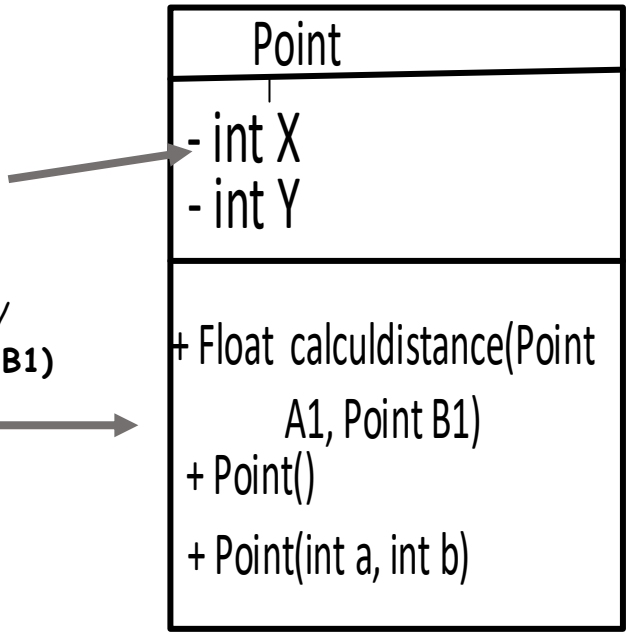
Prenant le cas d'une méthode statique :

On veut calculer la distance entre deux points :

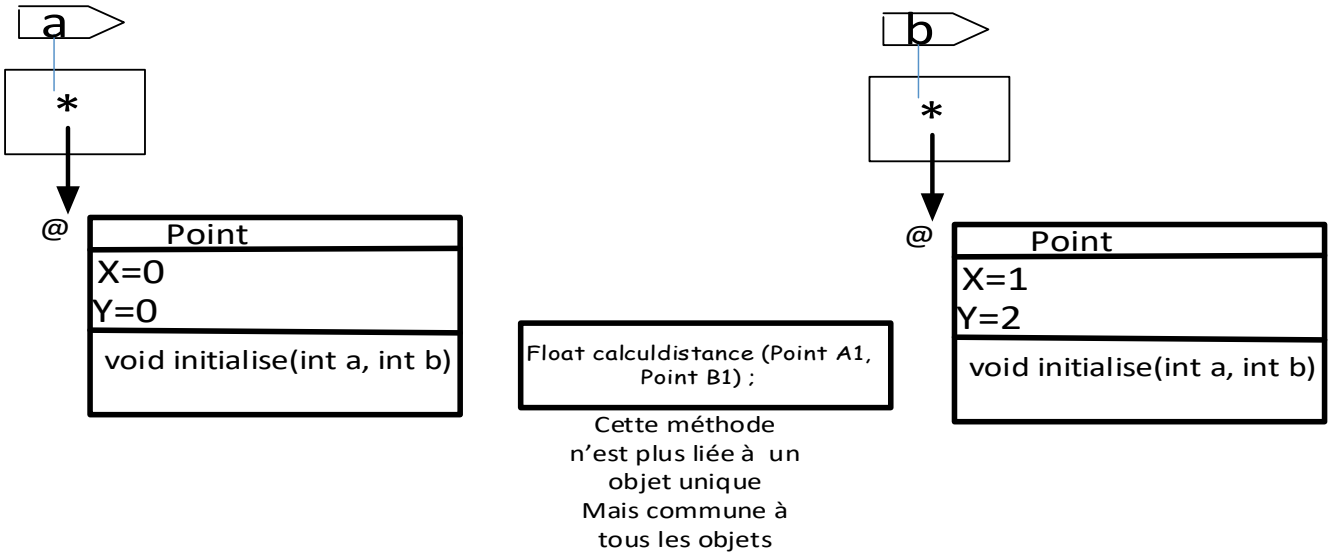
```
class Point
{
/* déclaration des membres privés ou publics*/
private  int x ;
private int y ;


/* déclaration  et définition des méthodes publics */
Public static float calculdistance (Point A1, Point B1)
------------------------------------ ;
------------------------------------ ;
------------------------------------ ;


} ;
```



Point a = new Point( ) ; // appel du constructeur par défaut
Point b= new Point(1,2) ;



On pourra appeler cette méthode dans n'importe quels objets de n'importe quelles classes en utilisant la syntaxe suivante :

```
float c=2 ;
Point a = new Point( ) ; // appel du constructeur par défaut
Point b= new Point(1,2 ) ;
c= Point.calculdistance ( a, b) ;// affecté=nomdelaclasse.appelméthode(…..)
```

Un objet peut être initialisé sans faire appel à l'opérateur new, prenons l'exemple suivant :
Soit la méthode symétrique qui permet d'obtenir un point symétrique à un autre par rapport à une droite affine. Cette méthode dispose comme paramètre d'entrée le coefficient directeur et le point d'interception entre la droite et l'axe des Y pour X=0. Cette méthode retourne le point symétrique.

```
Public point symétrique (double a, double b)
{


}
Point pt = new Point(4,1) ;
Point C= pt. symétrique (2.0, 3.0) ;// C est le point symétrique de pt par rapport à la droite
// C n' a pas été initialisé par l'opérateur new
```

Les objets et classes anonymes

On peut créer des objets sans être obligé d'avoir une référence à un objet. Prenons le cas suivant :

Point a = new Point( ) ; // appel du constructeur par défaut
Point b= new Point(1,2) ;
c= Point.calculdistance (new Point(3,4) , b) ;//

Le premier paramètre passé à la méthode **calculdistance** est un objet de la classe anonyme Point

On peut même définir ou redéfinir dans la classe anonyme lors de la création de l'objet une méthode. Situation que l'on a déjà rencontrée :

```
btncon.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent arg0)  {
                        try   {
                              ---------------------------------- ;
                              ---------------------------------- ;

                              }
                              catch (Exception e) {
                              JOptionPane.showMessageDialog(null, e.getMessage());
                              e.printStackTrace();}

                                                          }
                                                });
```

On créé un objet écouteur d'événement implémentant l'interface ou la classe anonyme ActionListener
(`new ActionListener()` et on redéfinit directement par la suite la méthode actionPerformed
Cette méthode actionPerformed sera appelé à chaque événement intercepté par l'objet écouteur d'événement.

4) Les méthodes et les paramètres d'entrée et de sortie

Déclarations et appels

a) Méthode qui reçoit en paramètre d'entrée un type primitif et qui ne renvoie rien

déclaration

```
public class A
{
// Déclaration des membres
Public void methode1 (int k)
{
------------------------------------------------ ;
}
}
```

Appel de la méthode

```
int b =3 ;
A obja = new A( ) ; // Appel du constructeur par défaut par défaut
Obja. methode1 (2) ; //passage d'une valeur directement
Obja. methode1 (b) ;// passage de la valeur de la variable b
```

Dans cet appel on a un passage par copie. La valeur passée en paramètre d'entrée sera copiée dans une variable locale K qui sera détruite à la fin de l'appel.

b) Méthode qui ne reçoit aucun paramètre d'entrée et qui renvoie un type primitif

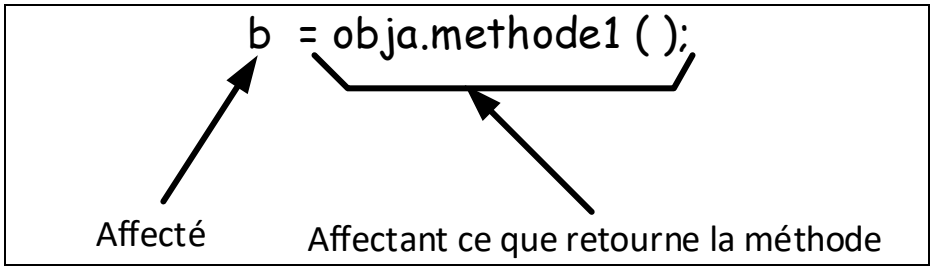
déclaration

```
public class A
{
// Déclaration des membres
Public int methode1 ( )
{
int w=2 ;
------------------------------------------------ ;
return w ;
}
}
```

Appel de la méthode

```
int b =3 ;
A obja = new A( ) ; // Appel du constructeur par défaut par défaut
```

Lorsqu'une méthode renvoie une variable on doit absolument affectée une variable de même type



c) ***Méthode qui reçoit un tableau en paramètre d'entrée et qui ne renvoie rien***

déclaration

```
public class A
{
// Déclaration des membres
Public void methode1 (int tab[])
{
---------------------------------------------- ;
tab[2]=0 ; // impact le tableau passé en paramètre d'entrée
}
}
```

Appel de la méthode

```
int b =3 ;
int tab1 = new int[5] ;
A obja = new A( ) ; // Appel du constructeur par défaut par défaut
Obja. methode1 (tab1) ;
```

Obja. methode1 (tab1 [ ]) ; // Syntaxe fausse et sanctionnée par une erreur

Dans cet appel on a un passage par référence. L'adresse du tableau tab1 passée en paramètre d'entrée va servir de référence au tableau tab.
Toute modification du tableau tab impactera le tableau tab1
Lorsqu'une méthode renvoie une variable on doit absolument affectée une variable de même type

d) Méthode qui ne reçoit rien et qui renvoie un tableau

déclaration

```
public class A
{
Public byte[] methode1 ()
{
byte []tab = new byte [5];
-----------------------------------------------;
Return tab ;
}
}
```

Appel de la méthode

```
A obja = new A( ) ; // Appel du constructeur par défaut par défaut
byte []data;  //
data=Obja. methode1( ) ;
```

La référence du tableau renvoyé par la méthode affecte la référence du tableau data

e) Méthode qui reçoit un objet en paramètre d'entrée et qui ne renvoie rien

déclaration

```
public class A
{
// Déclaration des membres
Public void methode1 (B objb)
{
-----------------------------------------------;
}
}
```

Appel de la méthode

```
A obja = new A( ) ; // Appel du constructeur par défaut par défaut
B objb1 = new B( ) ; //
Obja. methode1 (objb1) ;
```

Dans cet appel on a un passage par référence. L'adresse de l'objet objb1 passée en paramètre d'entrée va servir de référence à l'objet local objb.
Toute modification des membres de objb1 impactera les membres de l'objet objb.

f) ***Méthode qui reçoit rien et qui renvoie un objet en paramètre de sortie***

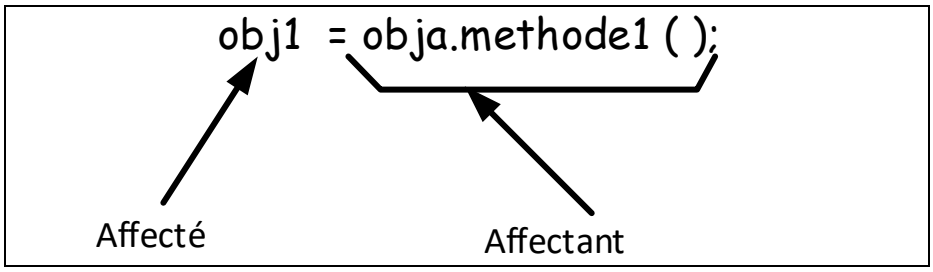
déclaration

```
public class A
{
// B objb = new B( ) ;
 Déclaration des membres
Public B methode1 ( )
{
---------------------------------------------- ;
Return objb
}
}
```

Appel de la méthode

```
A obja = new A( ) ; // Appel du constructeur par défaut par défaut
B objb1 = new B( ) ; //
Obj1=Obja. methode1 ( ) ;
```

Lorsqu'une méthode renvoie un objet on doit absolument affectée un objet de même type



La référence de l'objet renvoyé par la méthode affecte la référence de l'objet obj1