

# Chapitre 3 : Encapsulation

**Enseignante : Fairouz Fakhfakh**

fairouz.fakhfakh@iit.ens.tn

fairouz.fakhfakh@gmail.com

# Le principe de l'encapsulation

## Définition

- Le principe d'encapsulation dit qu'un objet ne doit pas exposer sa représentation interne au monde extérieur.
- L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure (la classe).

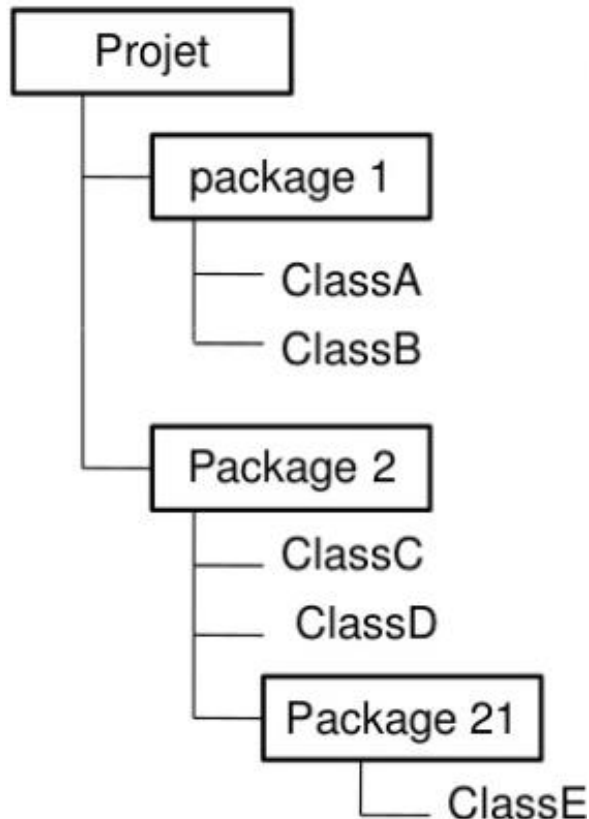
### ☐ Représentation interne d'un objet

- les attributs
- les méthodes

### ☐ Monde extérieur

- les autres classes dans le même package
- les classes des autres packages

# Notion de package



## □ Package = répertoire

- Les classes Java peuvent être regroupées dans des packages
- Un package (paquetage) est un **regroupement** de classes participant au même centre d'intérêt.

## □ Les objectifs sont :

- fournir une **organisation** cohérente des classes.
- définir un certain niveau de **protection** pour les classes

## □ Déclaration d'un package

***package nomPackage;***

Exemples :

- package Package2
- package Package2. Package21

# Notion de package

- Il y a **deux manières** d'utiliser une classe d'un package :
  - En utilisant le nom du package suivi du nom de la classe

```
java.util.Date v = new java.util.Date();  
System.out.println(v);
```

- En utilisant le mot clé **import**

```
import java.util.Date;  
....  
Date v = new Date();  
System.out.println(v);
```

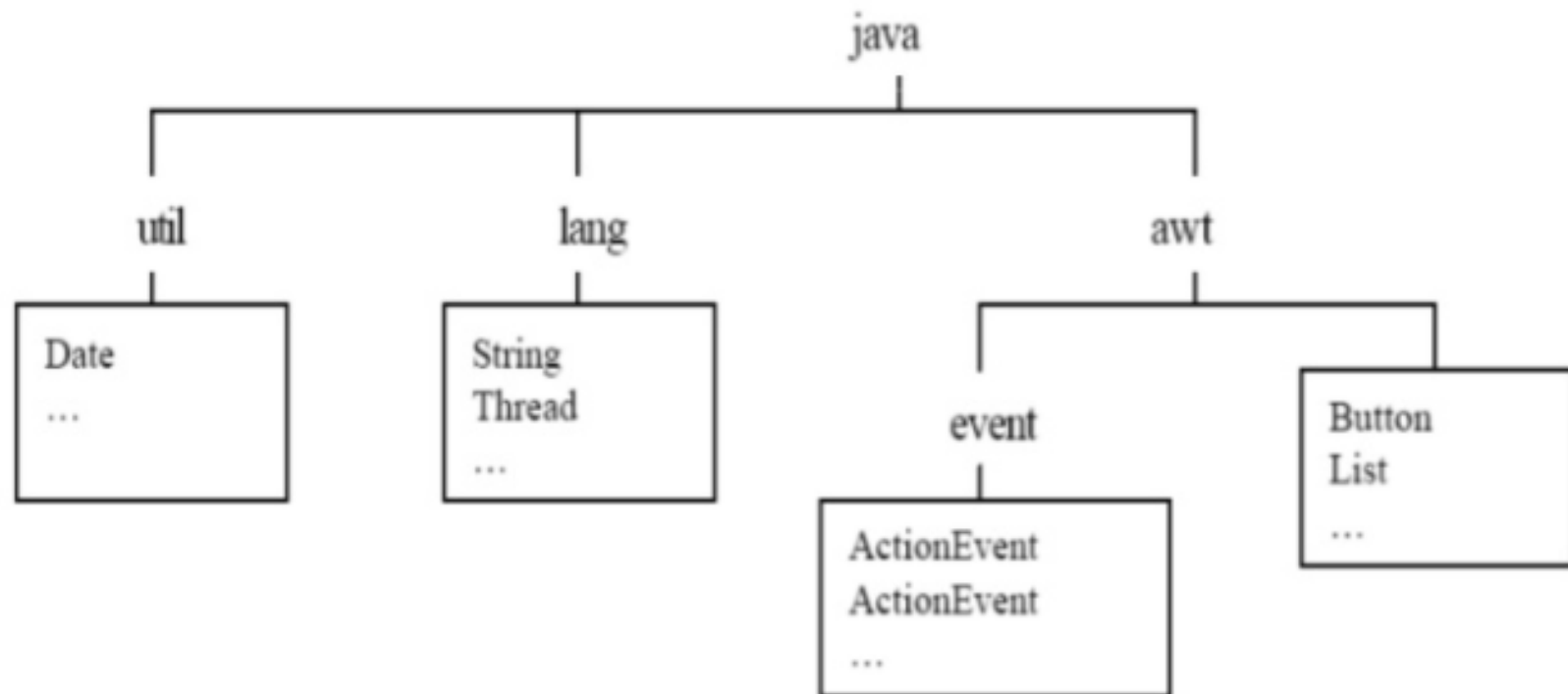
importer la  
classe du  
package

```
import java.util.*;  
....  
Date v = new Date();  
System.out.println(v);
```

importer tout  
le package

# Notion de package

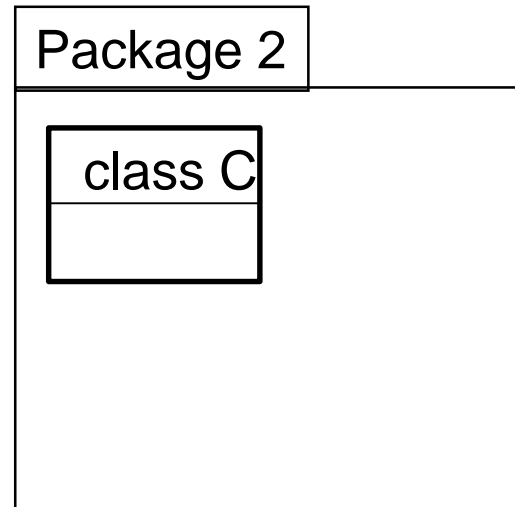
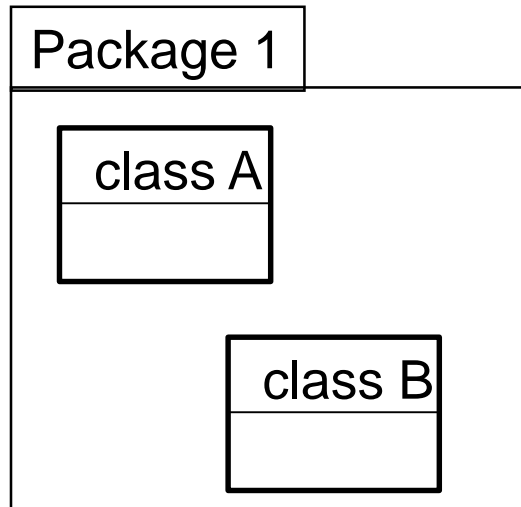
- Les packages sont eux mêmes organisés hiérarchiquement : on peut définir des sous-packages, des sous-sous packages,...



# Exemples de packages : Core API de java

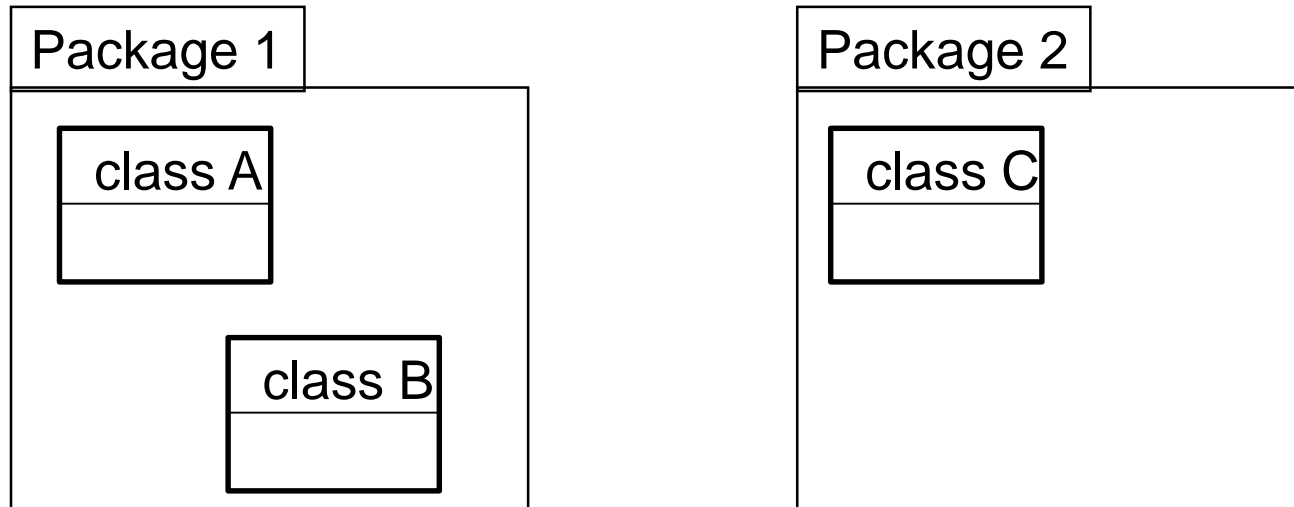
- **Core API de java** regroupe un ensemble de classes prédéfinies d'usage général que l'on retrouve dans le JDK ou le JRE sur toutes les plateformes java.
- Core API est découpée en plusieurs packages :
  - java.io : Gestion des entrées-sorties et des fichiers
  - java.awt : Interface graphique AWT
  - java.math : Traitements mathématiques
  - java.util : Structures de données et utilitaires
  - java.applet : Manipulation des applets

# Encapsulation des classes

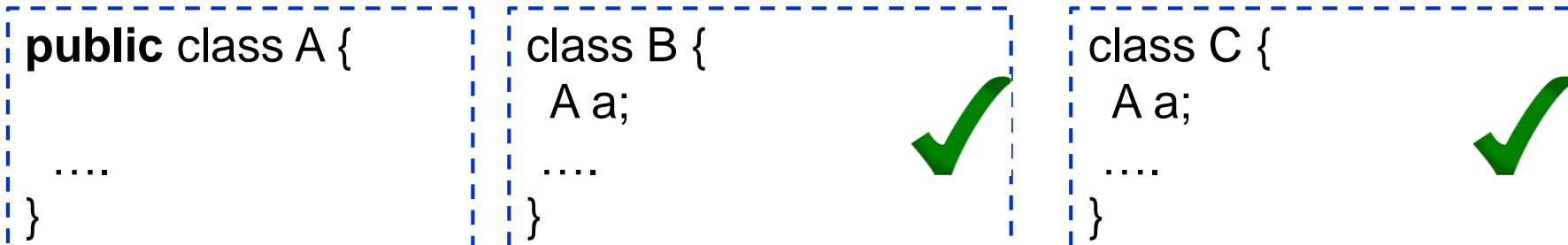


# Encapsulation des classes

## ❑ La classe public



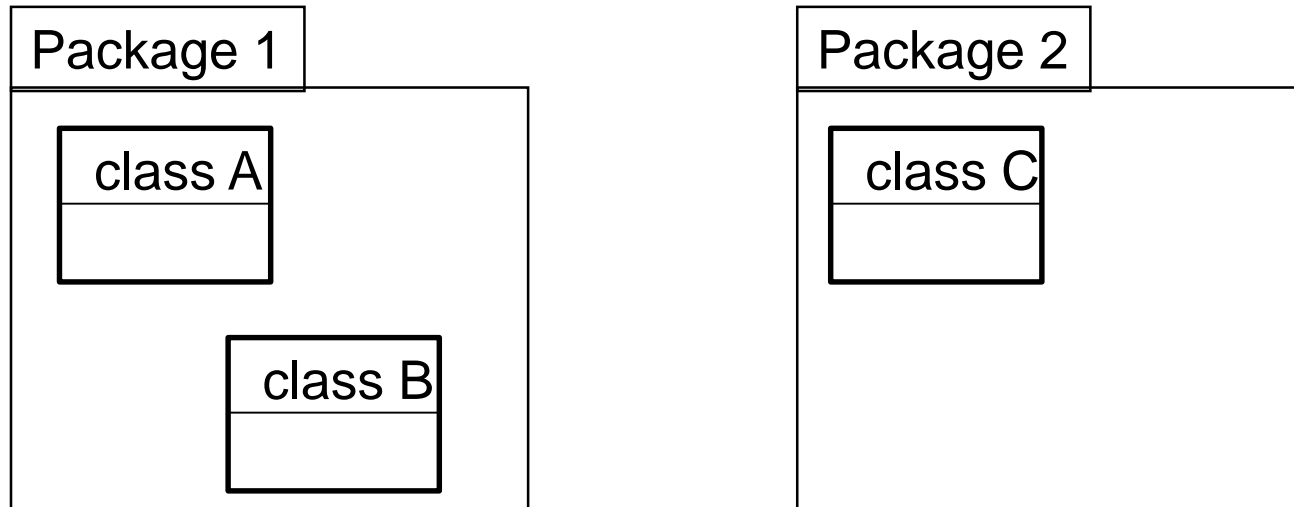
**public class A** ➡ La classe *public* est visible depuis n'importe quelle classe du projet.



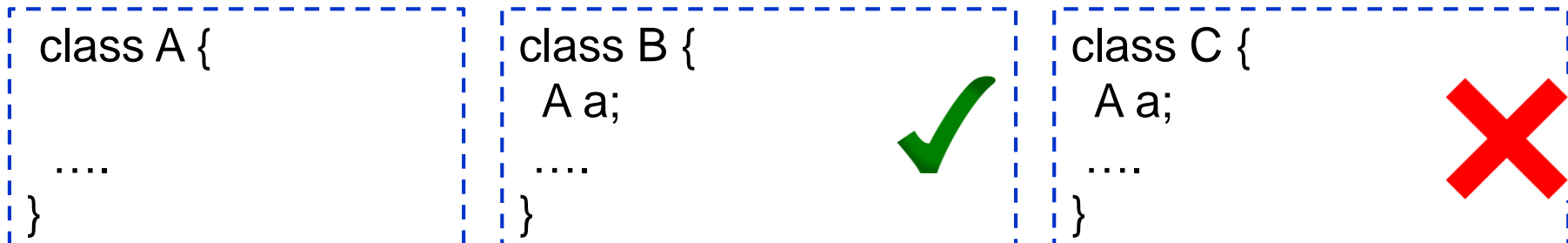


# Encapsulation des classes

## ❑ La classe default

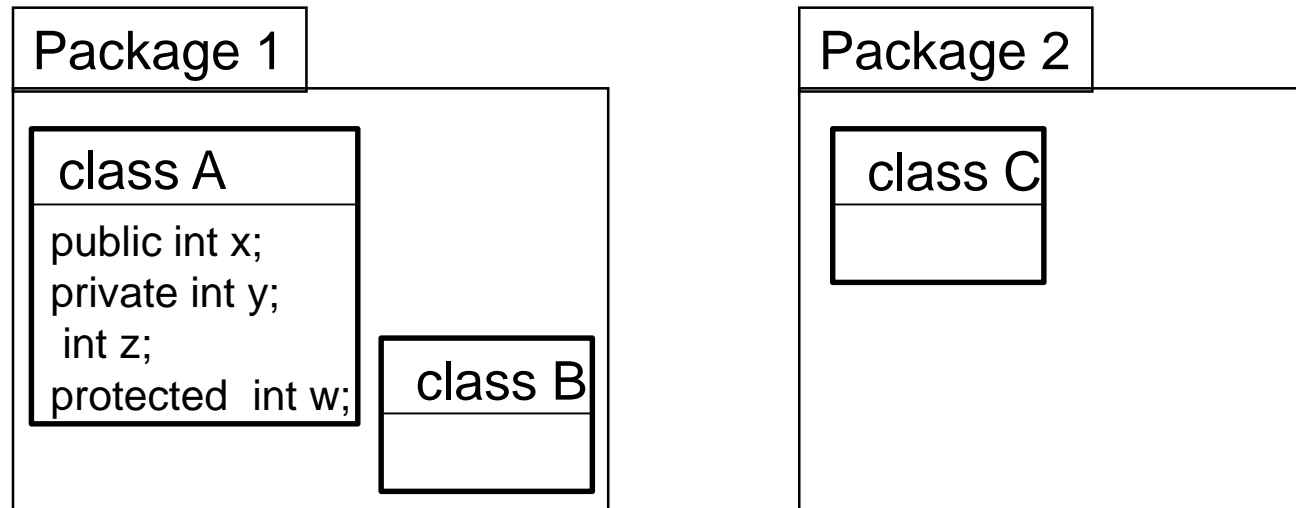


**class A** => La classe *default* est visible seulement par les classes de son package.

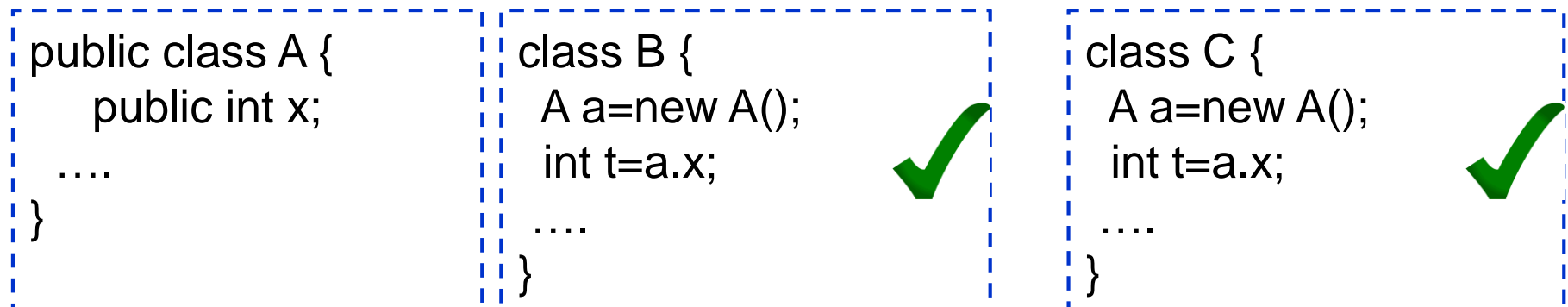


# Encapsulation des attributs

## ❑ L'attribut public

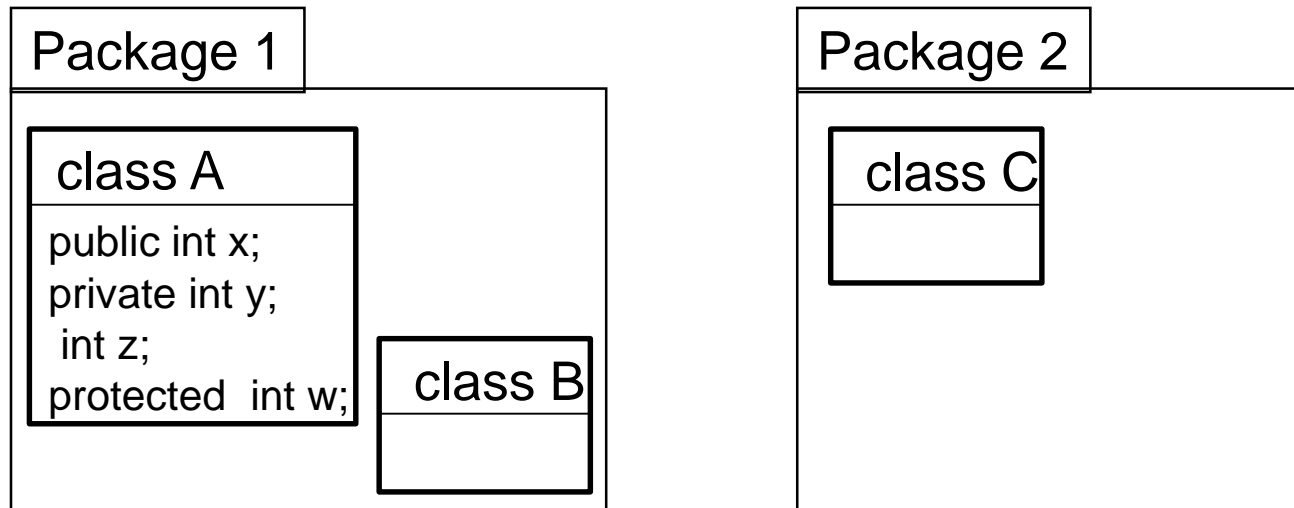


**public int x** => L'attribut public est visible par toutes les classes.



# Encapsulation des attributs


## ❑ L'attribut private




**private int x** => L'attribut private n'est accessible que depuis l'intérieur même de la classe.

```
public class A {  
    private int y;  
    ....  
}
```

```
class B {  
    A a=new A();  
    int t=a.y;  
    ....  
}
```

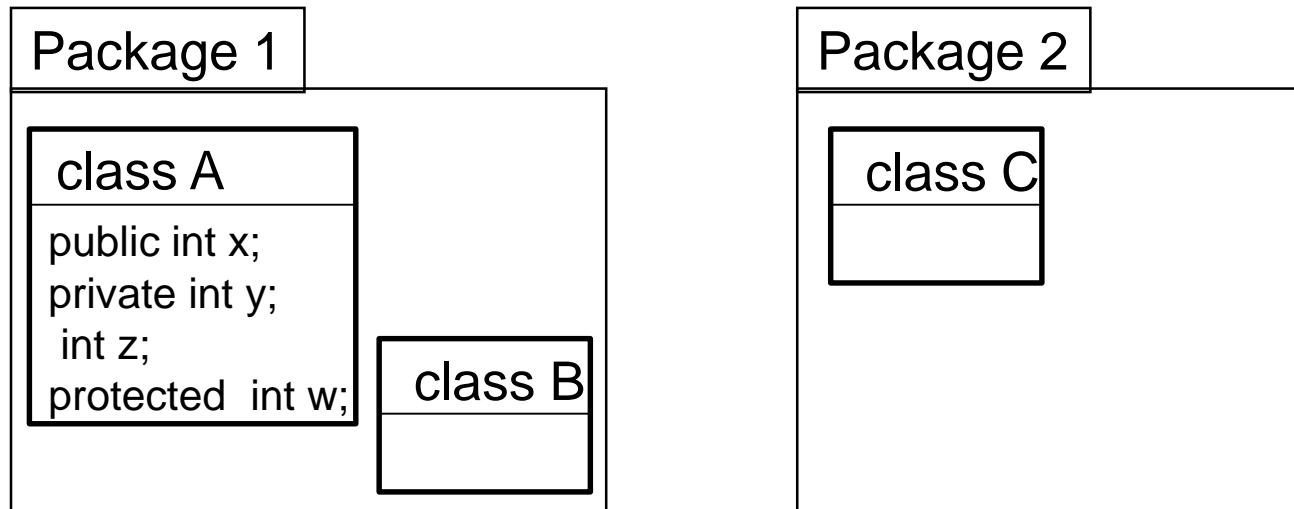


```
class C {  
    A a=new A();  
    int t=a.y;  
    ....  
}
```



# Encapsulation des attributs


## ❑ L'attribut default




**int z** => L'attribut *default* n'est accessible que depuis les classes faisant partie du même package.

```
public class A {  
    int z;  
    ....  
}
```

```
class B {  
    A a=new A();  
    int t=a.z;  
    ....  
}
```

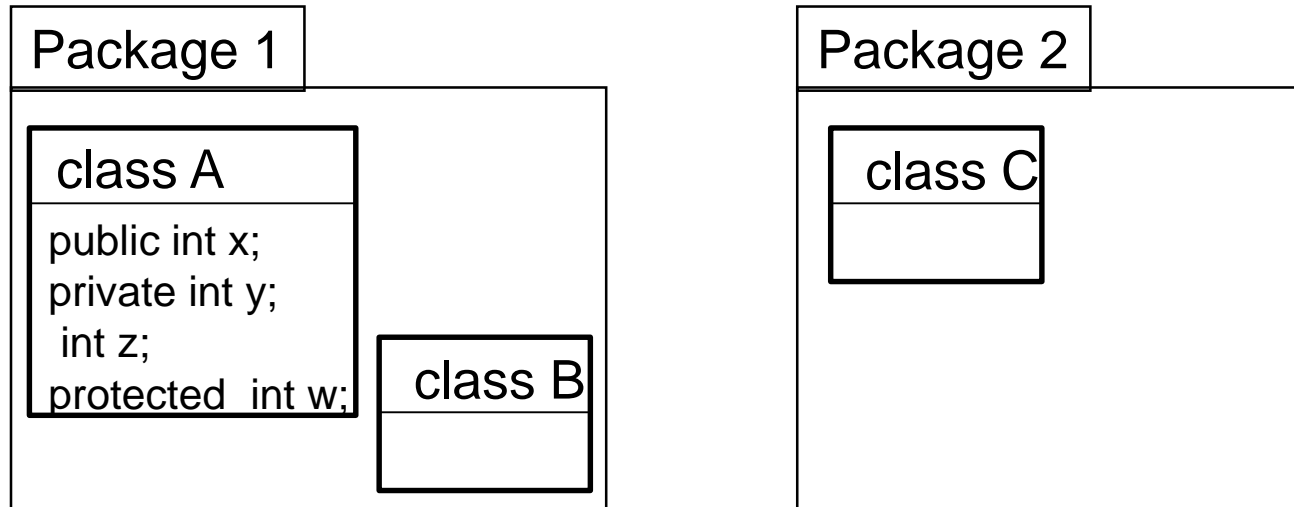


```
class C {  
    A a=new A();  
    int t=a.z;  
    ....  
}
```

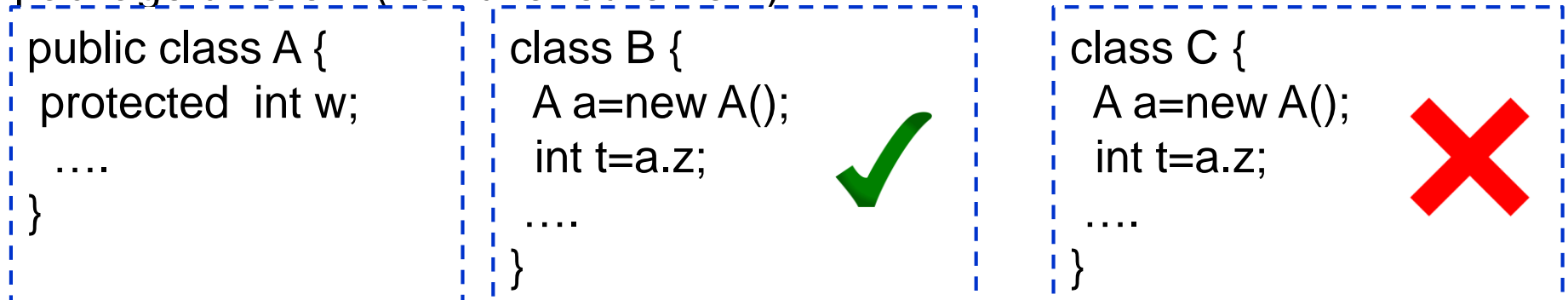


# Encapsulation des attributs

## ❑ L'attribut *protected*

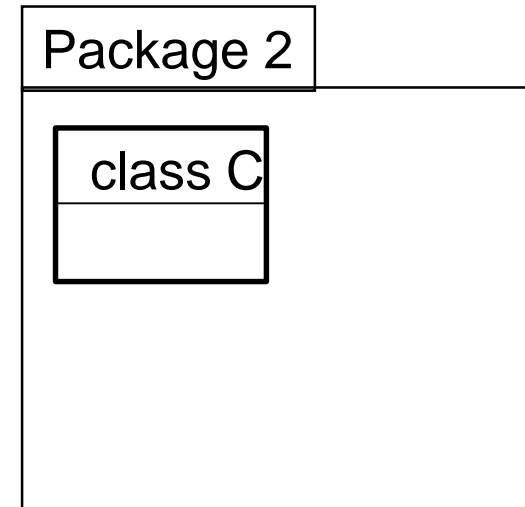
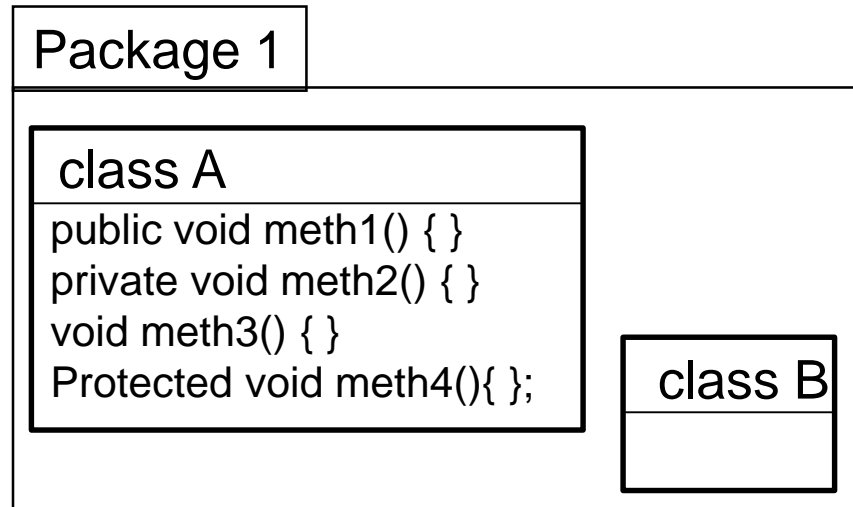


**protected int w** => L'attribut *protected* est accessible uniquement aux classes d'un même package et ses classes filles même si elles sont définies dans un package différent (voir ultérieurement).



# Encapsulation des méthodes

## ❑ La méthode public



**public void meth1()** => La méthode est accessible par toutes les classes du projet.

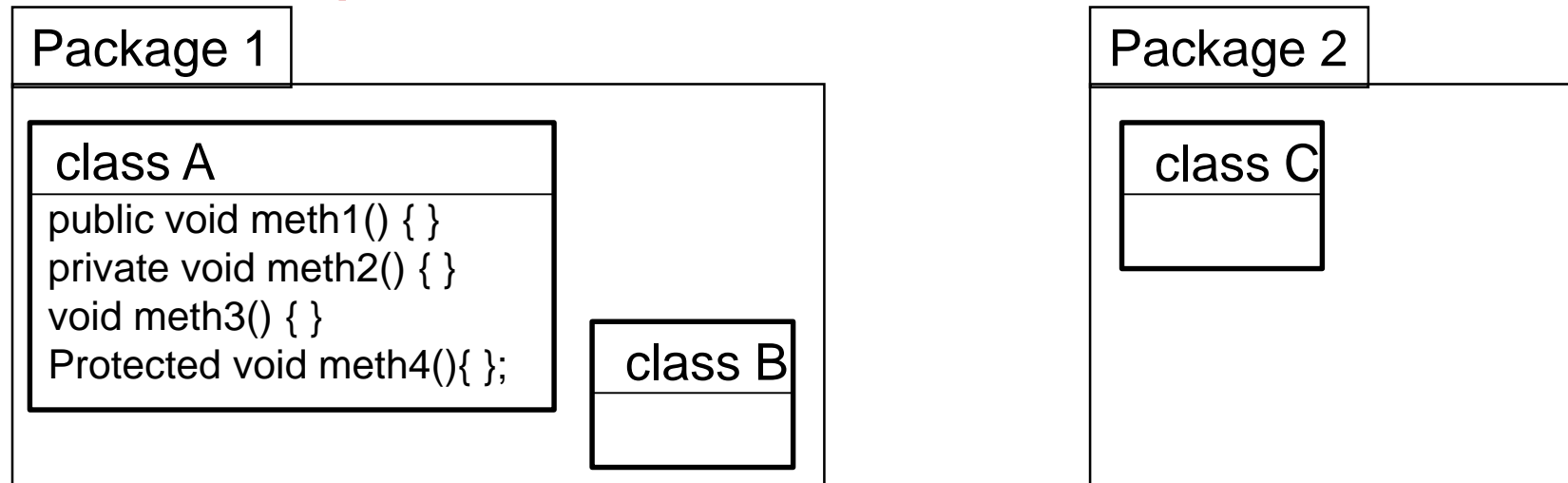
```
public class A {
    public void meth1() { }
    ....
}
```

```
class B {
    A a=new A();
    a.meth1();
    ....
}
```

```
class C {
    A a=new A();
    a.meth1();
    ....
}
```

# Encapsulation des méthodes

## ❑ La méthode *private*



**private void meth2()** => La méthode *private* n'est accessible que depuis l'intérieur de la même classe.

```
public class A {
private void meth2() { }
....
}
```

```
class B {
A a=new A();
a.meth2();
....
}
```

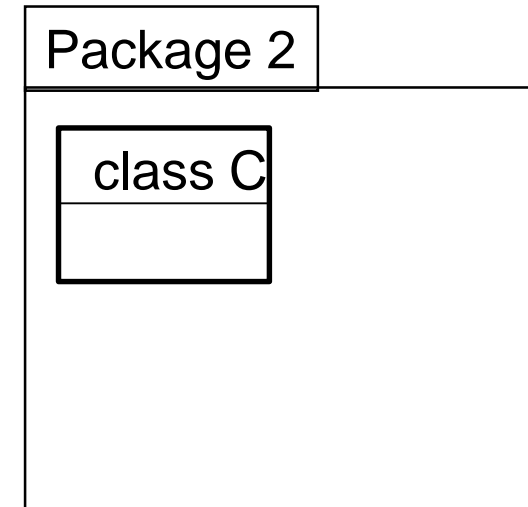
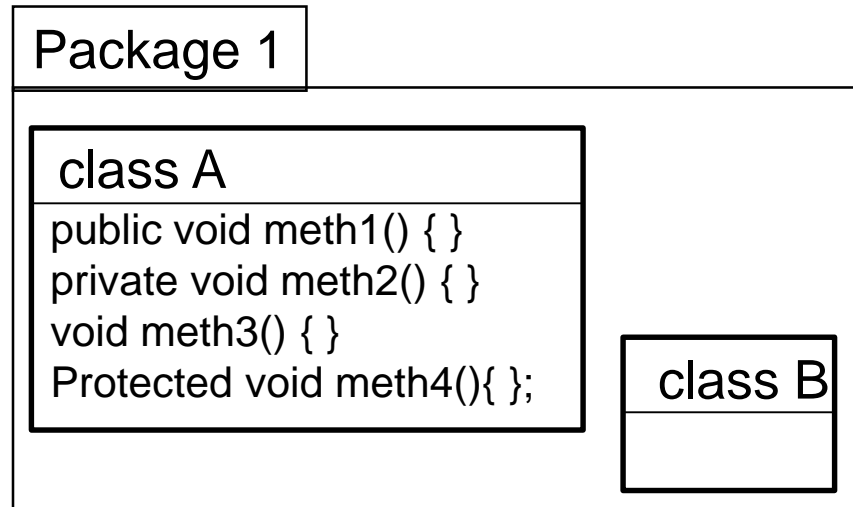


```
class C {
A a=new A();
a.meth2();
....
}
```



# Encapsulation des méthodes


## ❑ La méthode default




**void meth3()** => La méthode *default* n'est accessible que depuis les classes faisant partie du même package.

```
public class A {  
    void meth3() { }  
    ....  
}
```

```
class B {  
    A a=new A();  
    a.meth3();  
    ....  
}
```



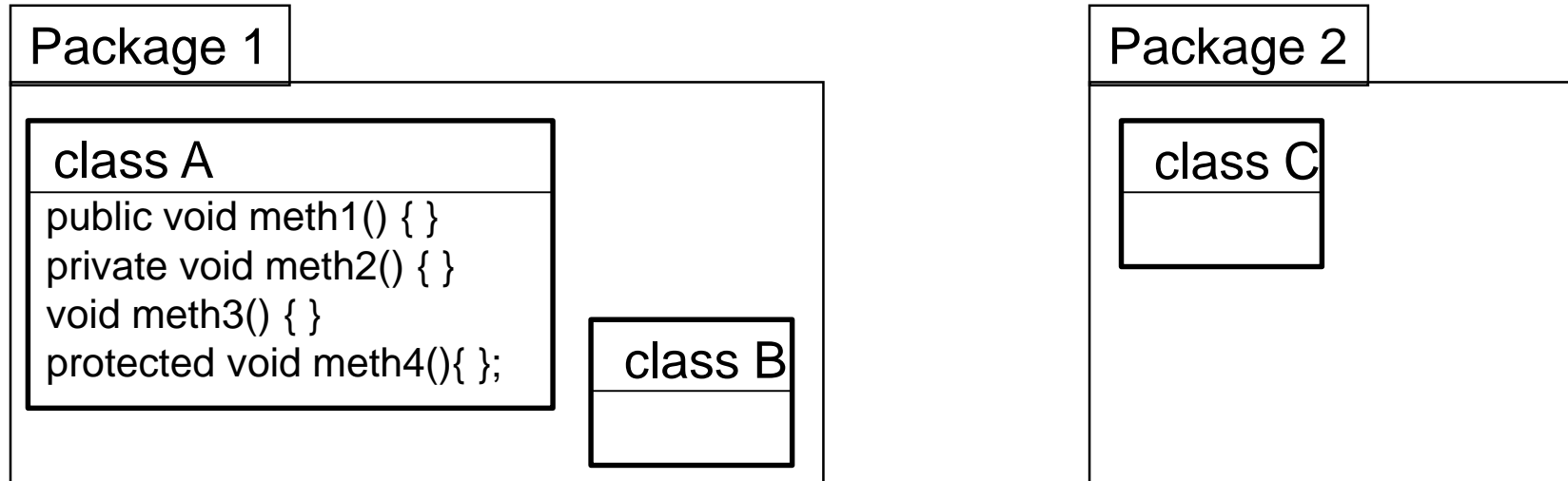
```
class C {  
    A a=new A();  
    a.meth3();  
    ....  
}
```





# Encapsulation des méthodes

## ❑ La méthode `protected`



**`protected void meth4()`** => La méthode *protected* est accessible uniquement aux classes du même package et ses sous classes même s'ils sont définies dans un package différents (voir ultérieurement).

```
public class A {
protected void meth4() { }
....
}
```

```
class B {
A a=new A();
a.meth4();
....
}
```



```
class C {
A a=new A();
a.meth4();
....
}
```



# Encapsulation des attributs/méthodes

- ❑ Dans le but de renforcer le contrôle de l'accès aux attributs d'une classe. Il est recommandé de les déclarer *private*.
- ❑ Pour la manipulation des attributs *private*, on utilise :
  - ❑ Un accesseur (**setter**) : une méthode qui permet de définir la valeur d'un attribut particulier.
  - ❑ Un accesseur (**getter**) : une méthode qui permet d'obtenir la valeur d'un attribut particulier.



Le setter et le getter doivent être déclarés *public*.

```
private int valeur;  
  
public int getValeur(){  
    return valeur;  
}  
  
public void setValeur(int val){  
    valeur = val;  
}
```

# Exemple

```
public class Parallelogramme
{
    private int longueur = 0; // déclaration + initialisation
    private int largeur = 0;  // déclaration + initialisation
    public int profondeur = 0; // déclaration + initialisation
    public void affiche ( )
    {System.out.println("Longueur= " + longueur + " Largeur = " + largeur +
                        " Profondeur = " + profondeur);
    }
}
```

```
public class ProgPpal
{
    public static void main(String args[])
    {
        Parallelogramme p1 = new Parallelogramme();
        p1.longueur = 5; // Invalide car l'attribut est privé
        p1.profondueur = 4; // OK car l'attribut profondeur est public
        p1.affiche( );      // OK car la méthode affiche est public
    }
}
```

# Manipulation des attributs

On distingue deux types d'attributs :

- ❑ **Attribut d'objet (ou attribut d'instance) :**

- déclaré sans le mot réservé *static*
- Une copie de cet attribut existe dans chaque objet

- ❑ **Attribut de classe (ou attribut statique) :**

- déclaré avec le mot réservé *static*
- S'il y a une seule instantiation, une seule copie de cet attribut est partagée par toutes les instances.

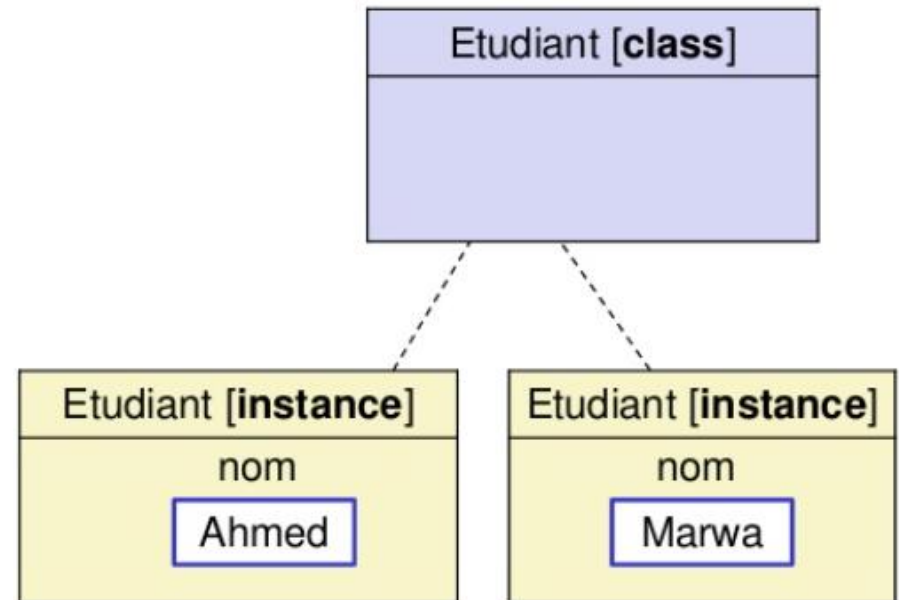
# Manipulation des attributs

## ❑ Attribut d'objet (ou attribut d'instance)

- Chaque instance de la classe possède ses propres valeurs d'attributs.

```
Class Etudiant{  
    String nom;  
  
    Etudiant(String nom){  
        this.nom=nom;  
    }  
}
```

```
Etudiant etud1 = new Etudiant ("Ahmed");  
Etudiant etud2 = new Etudiant ("Marwa");
```



# Manipulation des attributs

## ❑ Attribut d'objet (ou attribut d'instance)

- **Utilisation** : Les attributs sont appelés avec le nom de l'instance
- **Syntaxe** : *NomObjet.nomAttribut*

```
Class Etudiant{  
    String nom;  
  
    Etudiant(String nom){  
        this.nom=nom;  
    }  
}
```

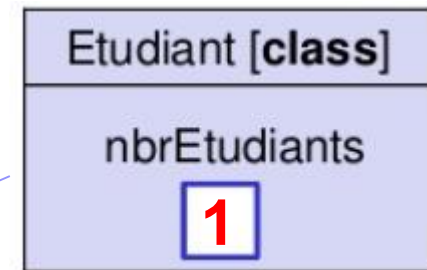
```
class Test{  
    public static void main(String[] args){  
  
        Etudiant etudiant=new Etudiant();  
  
        System.out.println(etudiant.nom);  
    }  
}
```

# Manipulation des attributs

## ❑ Attribut de classe (ou attribut statique) :

- N'appartient pas à une instance particulière, il appartient à la classe.
- Est **partagé** par toutes les instances de la classe.

```
Class Etudiant{  
    String nom;  
    static int nbrEtudiants;  
  
    Etudiant(String nom){  
        this.nom=nom;  
        nbrEtudiants++;  
    }  
}
```



```
Etudiant etud1 = new Etudiant ("Ahmed");  
Etudiant etud2 = new Etudiant ("Marwa");  
Etudiant etud3 = new Etudiant ("Fatma");
```

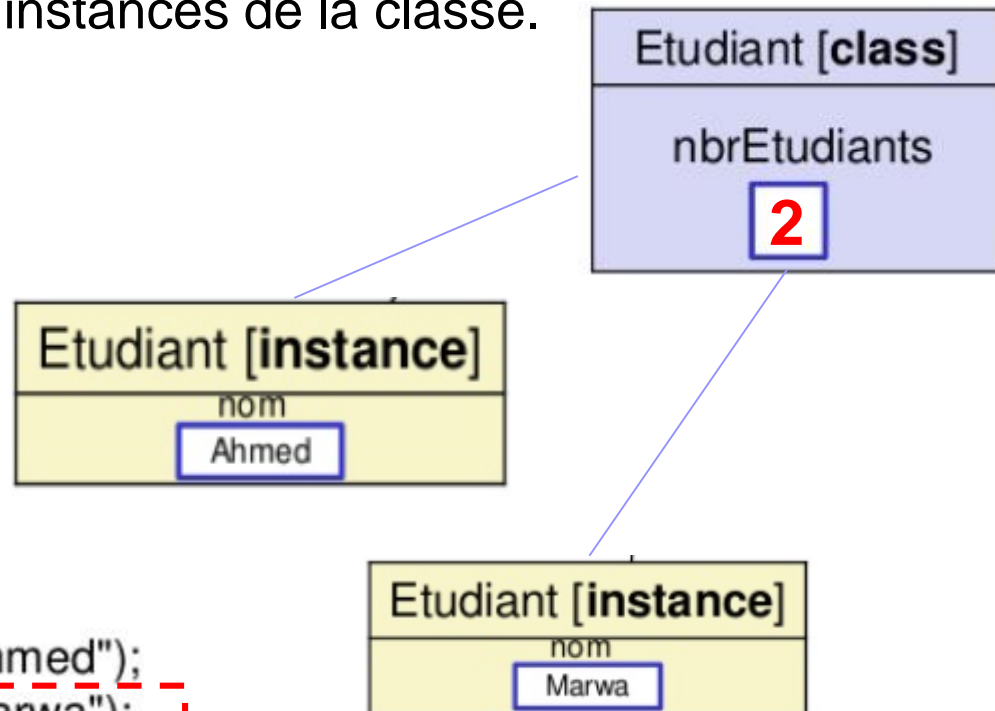
# Manipulation des attributs

## ❑ Attribut de classe (ou attribut statique) :

- N'appartient pas à une instance particulière, il appartient à la classe.
- Est **partagé** par toutes les instances de la classe.

```
Class Etudiant{  
    String nom;  
    static int nbrEtudiants;  
  
    Etudiant(String nom){  
        this.nom=nom;  
        nbrEtudiants++;  
    }  
}
```

```
Etudiant etud1 = new Etudiant ("Ahmed");  
Etudiant etud2 = new Etudiant ("Marwa");  
Etudiant etud3 = new Etudiant ("Fatma");
```





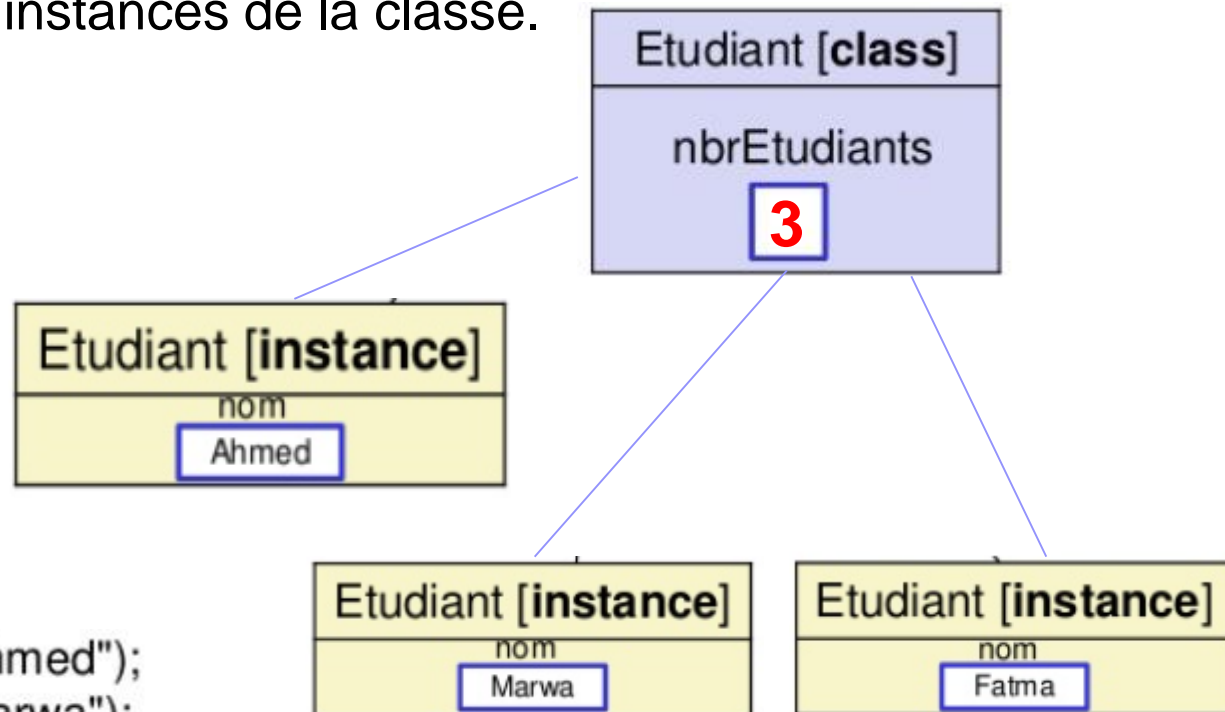
# Manipulation des attributs

## ❑ Attribut de classe (ou attribut statique) :

- N'appartient pas à une instance particulière, il appartient à la classe.
- Est **partagé** par toutes les instances de la classe.

```
Class Etudiant{  
    String nom;  
    static int nbrEtudiants;  
  
    Etudiant(String nom){  
        this.nom=nom;  
        nbrEtudiants++;  
    }  
}
```

```
Etudiant etud1 = new Etudiant ("Ahmed");  
Etudiant etud2 = new Etudiant ("Marwa");  
Etudiant etud3 = new Etudiant ("Fatma");
```



# Manipulation des attributs

## ❑ Attribut de classe (ou attribut statique) :

- **Utilisation** : Les attributs **static** sont appelés avec le nom de la classe (ou le nom de l'objet s'il y'a une instanciation).
- **Syntaxe** : *NomClasse.nomAttribut*  
*NomObjet.nomAttribut*

```
class Etudiant{  
    String nom;  
    static int nbrEtudiants;  
  
    Etudiant(String nom){  
        this.nom=nom;  
        nbrEtudiants++;  
    }  
}
```

```
class Test{  
    public static void main(String[] args){  
        System.out.println(Etudiant.nbrEtudiants);  
    }  
}
```

# Manipulation des méthodes

## ❑ Les méthodes d'objet :

- **Utilisation** : L'appel à une méthode d'objet se fait en utilisant le nom d'un objet.
- **Syntaxe** : *NomObjet.nomMethode()*;

```
public class A {  
  
    public void meth1() {  
    }  
    ....  
}
```

```
class Test {  
  
    public static void main (String [] args) {  
        A a=new A();  
        a.meth1();  
        ....  
    }  
}
```

# Manipulation des méthodes

## ❑ Les méthodes statiques :

- **Utilisation** : L'appel à une méthode statique se fait en utilisant le nom de la classe (ou le nom de l'objet s'il y'a une instantiation).
- **Syntaxe** : *NomClasse.nomMethode();*  
*NomObjet.nomMethode();*

```
class MaClassMath{  
    static int min(int a , int b){  
        if(a<b){  
            return a;  
        }else{  
            return b;  
        }  
    }  
}
```

```
class Test {  
    public static void main (String [] args ) {  
        int x = MaClassMath.min (21 ,4);  
    }  
}
```

# Manipulation des méthodes

## ❑ Les méthodes statiques :

- Les méthodes statiques ne peuvent accéder qu'aux attributs statiques.
- **Exemple :**

```
class A {  
    private float x ; // Attribut d'objet  
    private static int n ; // Attribut de classe  
    ...  
    public static void f() { // Méthode de la classe. Ici, on ne peut pas  
        ..... // accéder à x (attribut d'objet)  
        ..... // mais, on peut accéder au champ de classe n  
    }  
}
```

# Exemple

```
public class Circle {  
    public static int count = 0;  
    public static final double PI=3.14;  
    public double x,y,r;  
  
    public Circle(double r1) {r=r1; count++;}  
    public Circle bigger (Circle c) {  
        if(c.r>r) return c; else return this;  
    }  
    public static Circle bigger (Circle c1,Circle c2) {  
        if(c1.r>c2.r) return c1; else return c2;  
    }  
  
    public static void main (String args[])  
    {Circle c1 = new Circle(10);  
    Circle c2 = new Circle(20);  
    int n = Circle.count; // n= ?  
    Circle c3= c1.bigger (c2); // c3= ?  
    Circle c4= c1.bigger (c1,c2); // c4= ?  
    }  
}
```

Le rayon de l'objet courant

Objet courant