Chapitre 4: L’encapsulation

plan

- L’encapsulation
- L’encapsulation des classes
- L’encapsulation des attributs
- L’encapsulation des méthodes
- Les accesseurs et les mutateurs

plan

- L’encapsulation
- L’encapsulation de classes
- L’encapsulation des attributs
- L’encapsulation des méthodes
- Les accesseurs et les mutateurs

L’encapsulation (1/5)

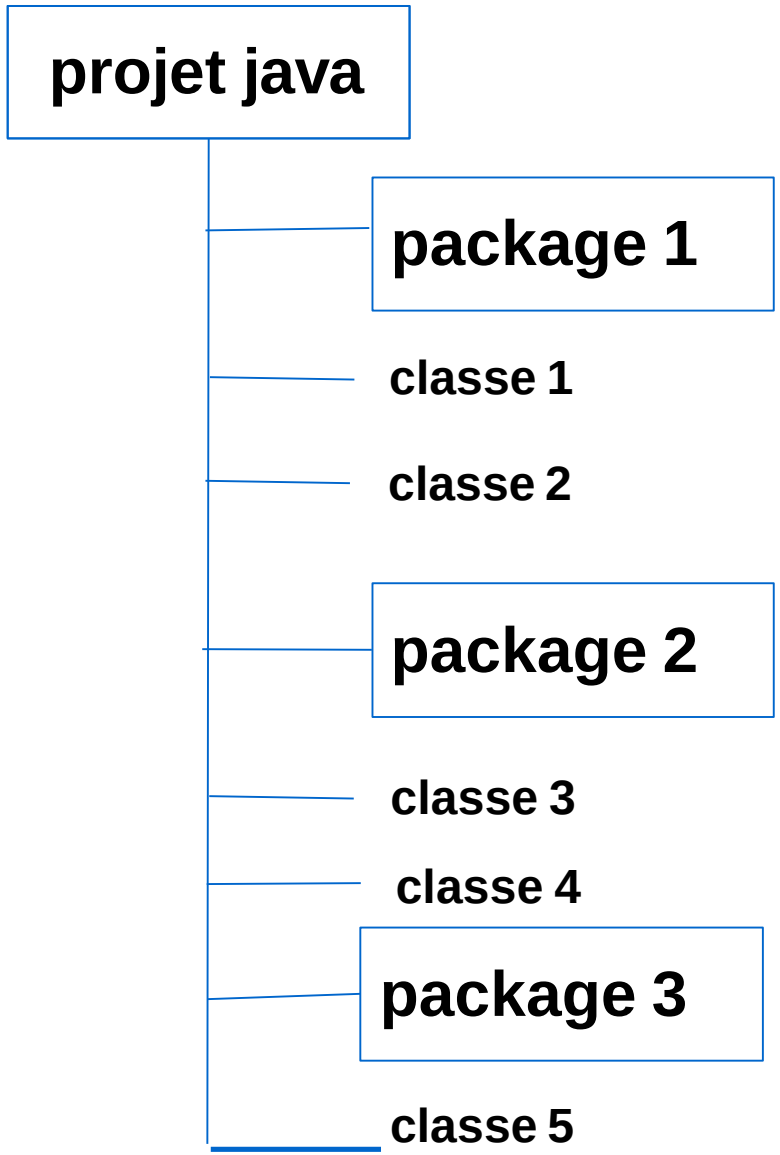
L’encapsulation c’est quoi ?

- L’encapsulation est l’un des concepts de base de la programmation orientée objet
- Le principe de l’encapsulation dit qu’un objet ne doit pas exposé sa représentation interne au monde extérieur
- Représentation interne :
  - Attributs
  - Méthodes
- Monde extérieur :
  - Les autre classes dans le même package
  - Les classes des autres packages

L’encapsulation (2/5)

- L'encapsulation permet de définir le niveau de visibilité des éléments de la classe à travers des modificateurs d’accès. Ils définissent les droits d'accès via des mots-clés:
  - **publique (public)**
  - **privée (private)**
  - **protégée (protected)**
  - **Sans modificateur**
- Ce contrôle des accès s’exerce hiérarchiquement sur :
  - les paquetages(packages)
  - les classes
  - les membres d’une classe (attributs et méthodes)
  - Ce contrôle s’exerce également sur la relation d’héritage.

L’encapsulation (3/5)



☐Déclaration d’un package

```
package pack1;
```

☐Import de la classe du package

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

☐Import de tout le package

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

L’encapsulation (4/5)

- L’encapsulation est un mécanisme syntaxique qui consiste à déclarer comme *private* une large partie des caractéristiques de la classe, tous les attributs et de nombreuses méthodes, tout en rendant l’accès à ces champs via des méthodes publiques (getters and setters)

Pourquoi encapsuler ?

- L’encapsulation permet de :
- Masquer les détails de la mise en œuvre
- Protéger la cohérence des données (dépendance)
- Faciliter la mise au point des programmes (le code modifant les données se situe uniquement dans les méthodes)
- Simplifier la compréhension du fonctionnement de l’objet (moins de membres visibles)

L’encapsulation (5/5)

Principe de l’encapsulation :

- Consiste à regrouper des données (attributs) et un comportement (méthodes) dans une même classe et à réglementer l’accès aux données de l’extérieur (par d’autres objets).
- Le principe de l’encapsulation est que, vu de l’extérieur, un objet se caractérise uniquement par ses méthodes visibles appelées interface. Les données, elles, restent invisibles et inaccessibles.

- Afin de respecter le principe d’encapsulation, les attributs non statiques d’une classe sont déclarés “private”.

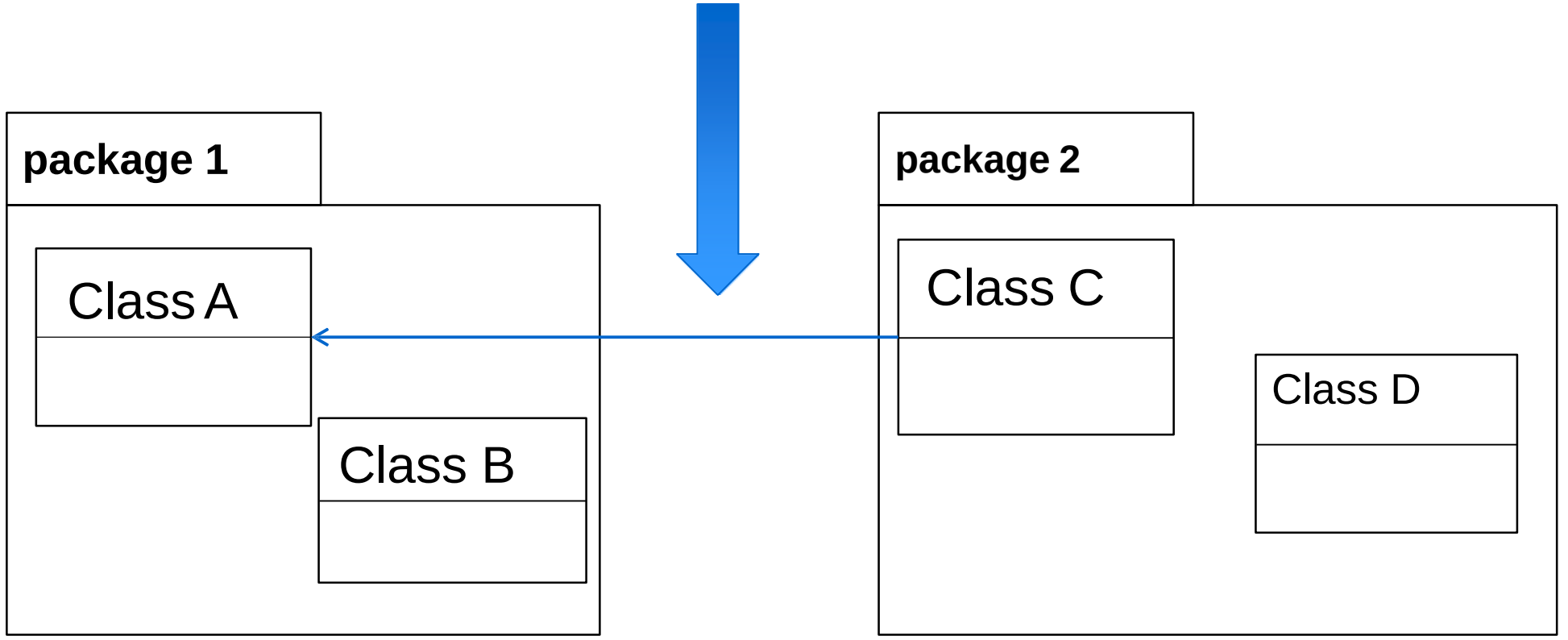
- Cela signifie qu’il n’est pas possible d’agir directement sur les données d’un objet ; il est nécessaire de passer par ses méthodes.

plan

- L’encapsulation
- L’encapsulation de classes
- L’encapsulation des attributs
- L’encapsulation des méthodes
- Les accesseurs et les mutateurs

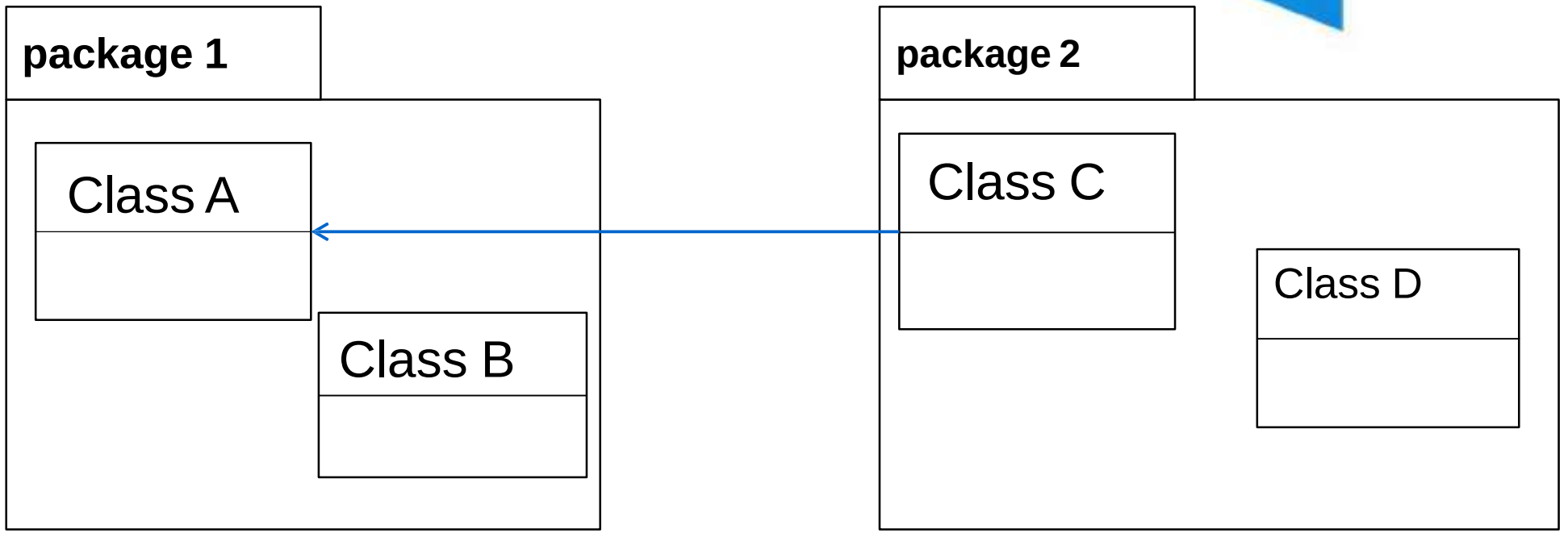
Encapsulation des classes(1/3)

La classe C est la classe fille de class A



Encapsulation des classes(2/3)

La classe public



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

```
public class A {

....
}
```

```
class B {
A a;
....
}
```

✓

```
class C extends A
{
A a;
....
}
```

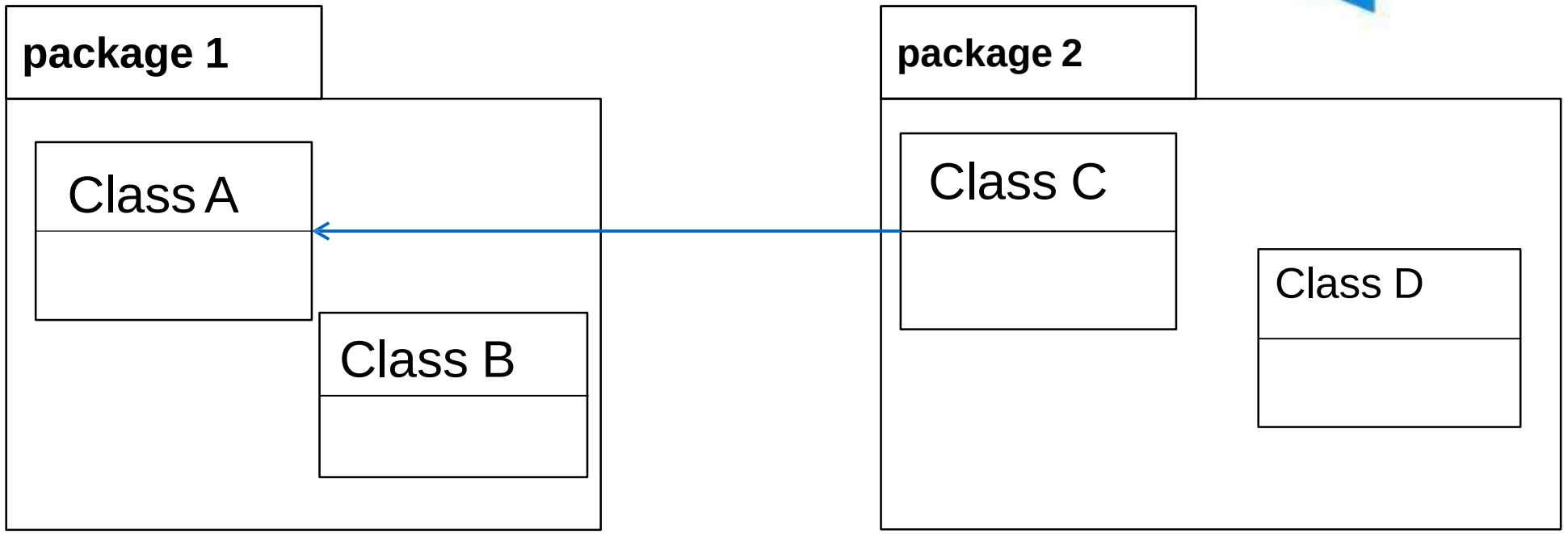
✓

```
class D {
A a;
....
}
```

✓

Encapsulation des classes(3/3)

La classe default



default class A ➡ La classe default est visible seulement par les classes de son package

```
class A {

....
}
```

```
class B {
A a;
....
}
```

✓

```
class C extends A
{
A a;
....
}
```

✗

```
class D {
A a;
....
}
```

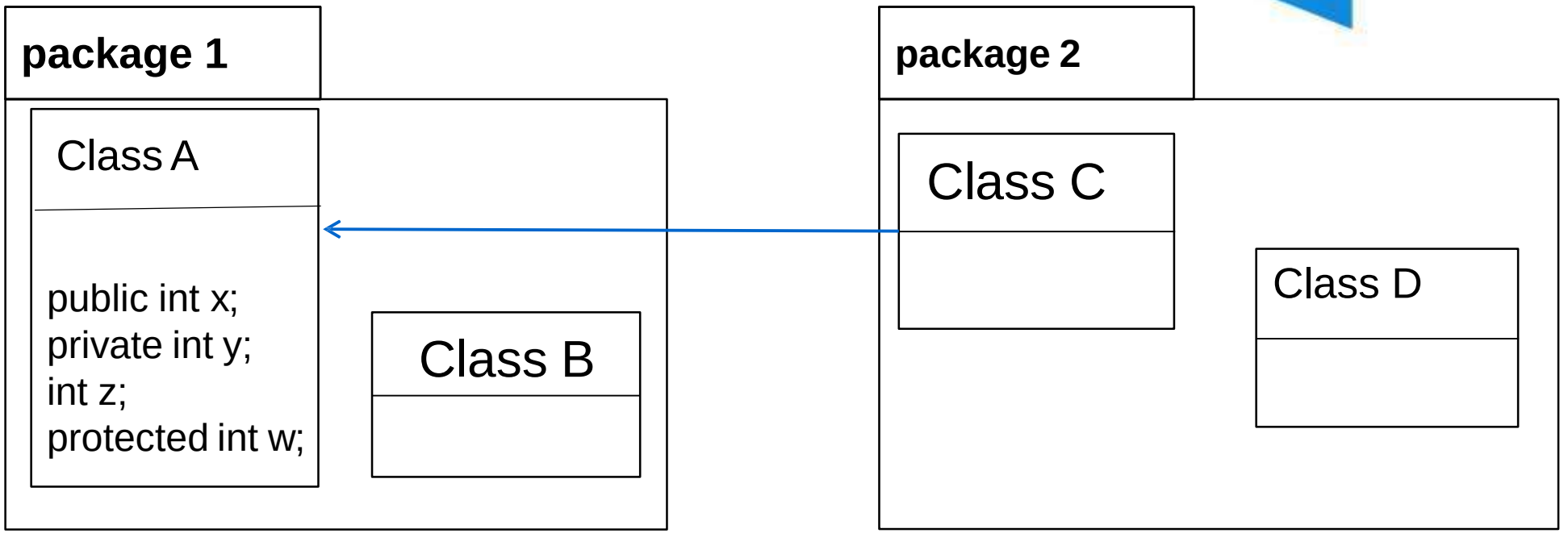
✗

plan

- L’encapsulation
- L’encapsulation de classes
- L’encapsulation des attributs
- L’encapsulation des méthodes
- Les accesseurs et les mutateurs

Encapsulation des attributs(1/4)

L'attribut public



public int x ➡ l'attribut public x est visible par toutes les classes

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

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

✓

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

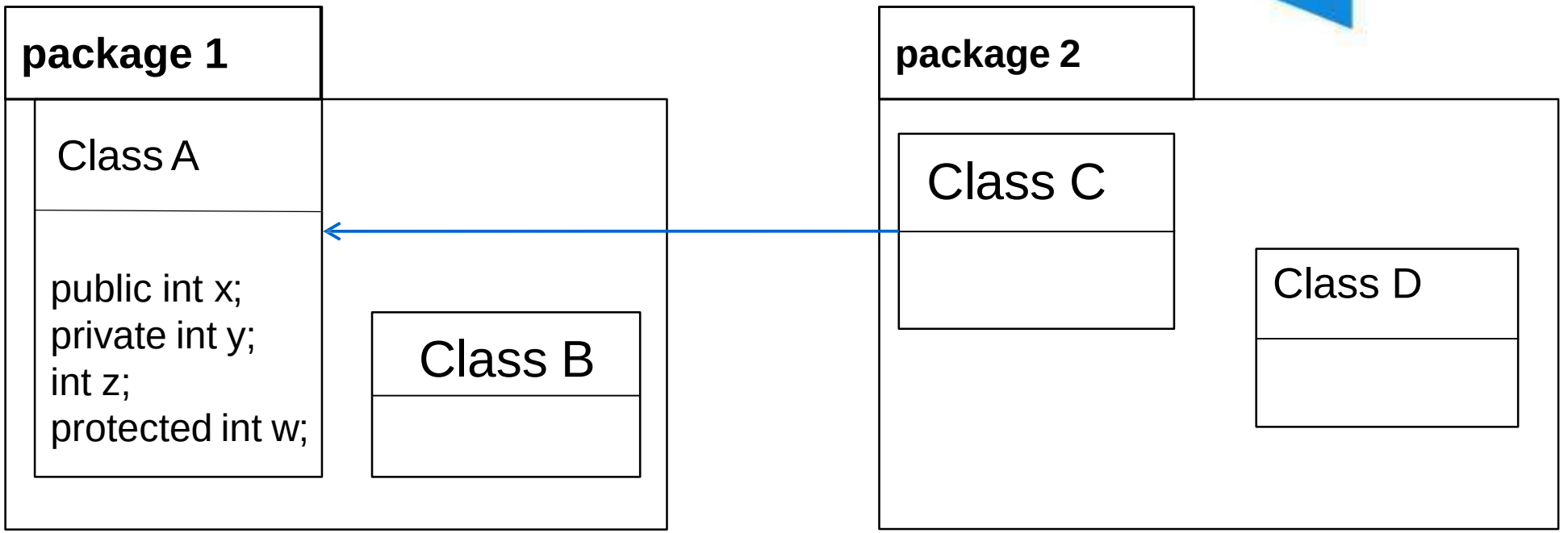
✓

```
class D {
A a=new A();
int t=a.x;
}
```

✓

Encapsulation des attributs (2/4)

L'attribut private



private int y ➡ l'attribut private y est visible seulement par la classe A

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

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

✗

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

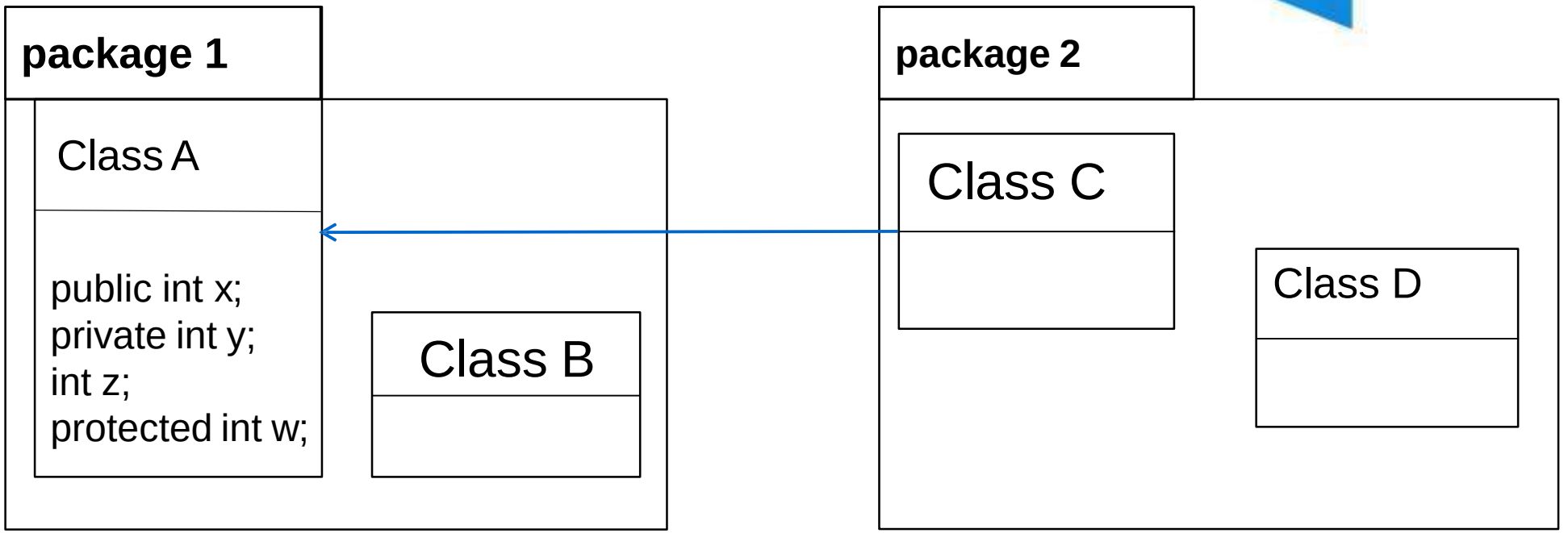
✗

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

✗

Encapsulation des attributs (3/4)

L'attribut default



int z ➡ l'attribut default z n'est accessible que depuis les classes du même package

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

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

✓

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

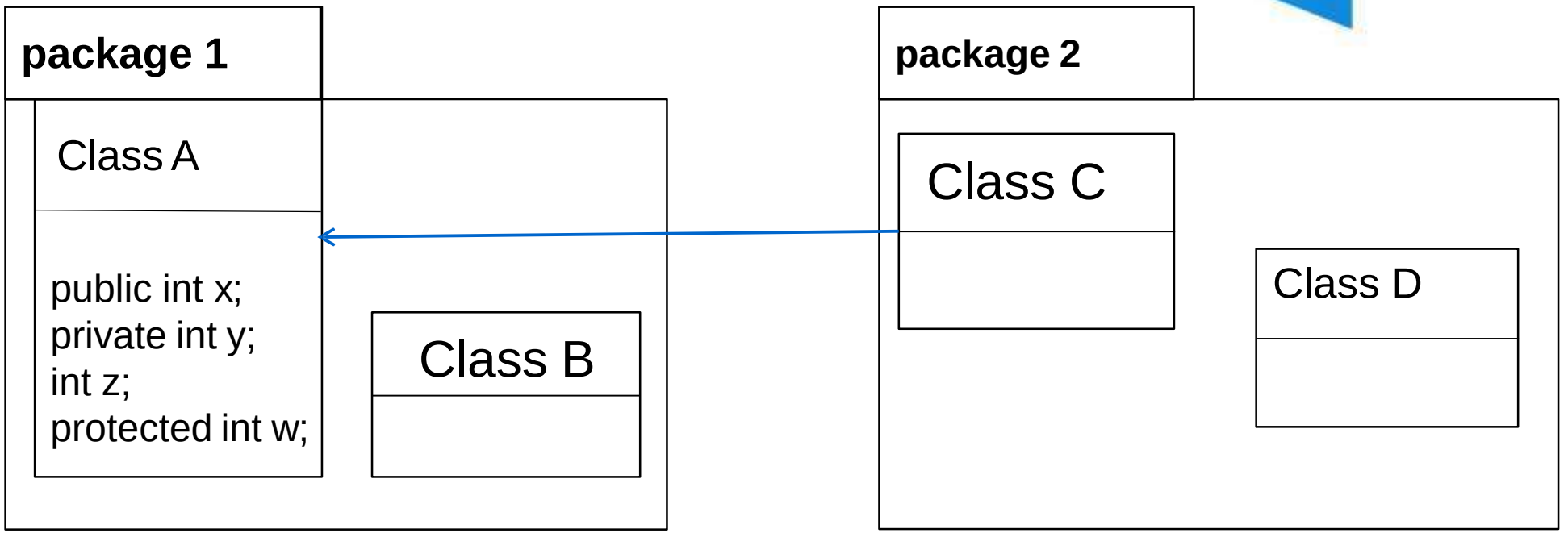
✗

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

✗

Encapsulation des attributs(4/4)

L'attribut protected



int w ➡ l'attribut protected w accessible uniquement aux classes du même package et à ces classes filles même si elles sont définies dans un package différent

```
public class A {
protected int w;
....
}
```

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

✓

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

✓

```
class D {
A a=new A();
int t=a.w;
}
```

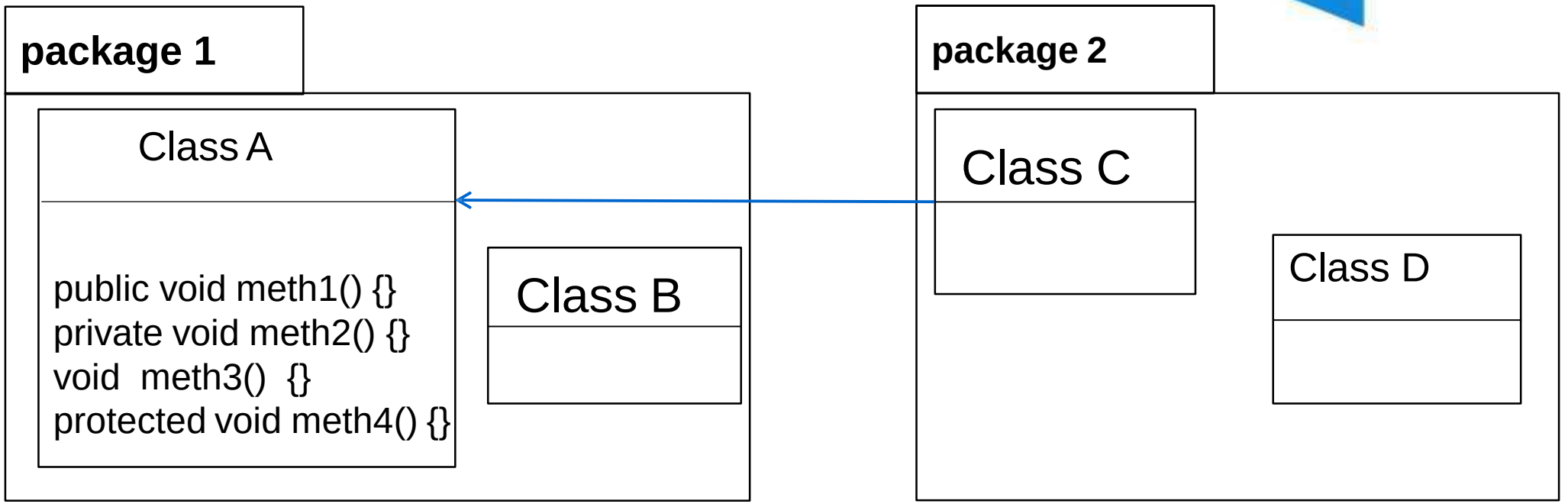
✗

plan

- L’encapsulation
- L’encapsulation de classes
- L’encapsulation des attributs
- L’encapsulation des méthodes
- Les accesseurs et les mutateurs

Encapsulation des méthodes(1/4)

Méthode public



public void meth1() ➡ la méthode public est accessible par toutes les classes du projet

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

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

✓

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

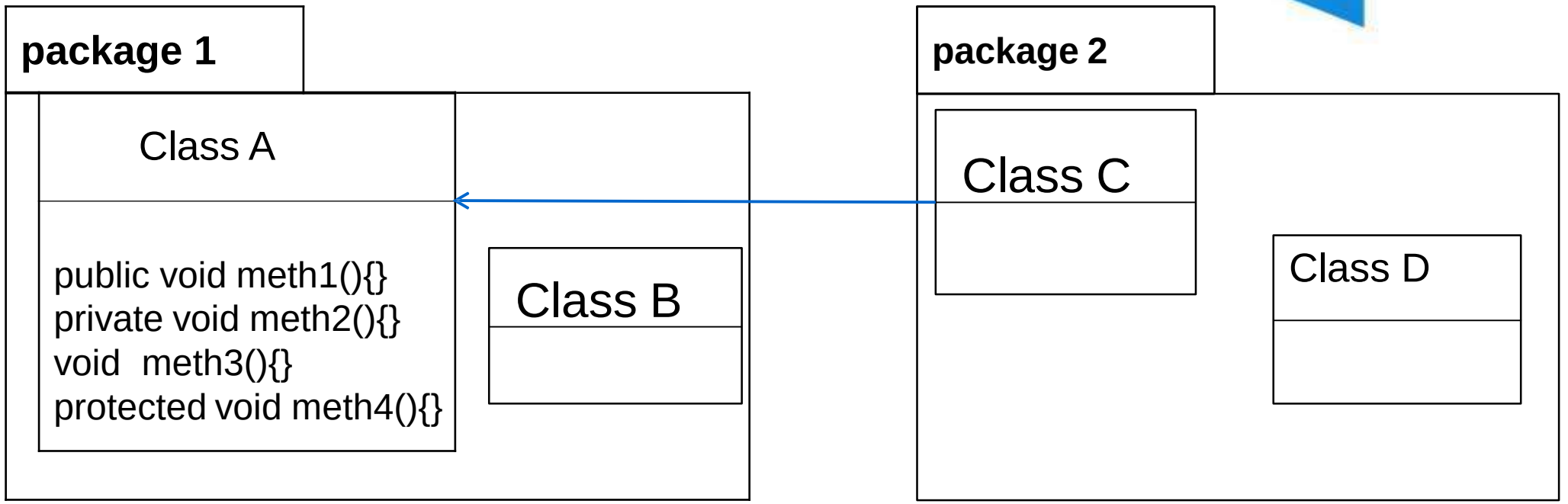
✓

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

✓

Encapsulation des méthodes(2/4)

Méthode private



private void meth2() ➡ la méthode private 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 extends A
{
A a=new A();
a.meth2();
}
```

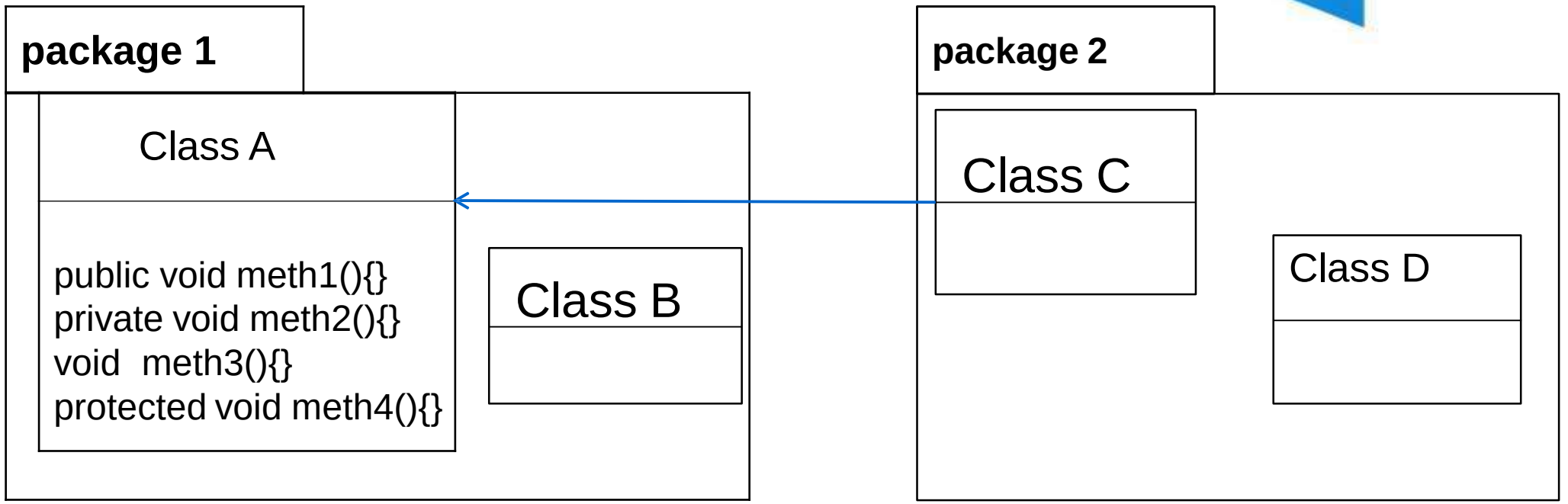
✗

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

✗

Encapsulation des méthodes(3/4)

Méthode default



void meth3() ➡ la méthode default 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 extends A
{
A a=new A();
a.meth3();
}
```

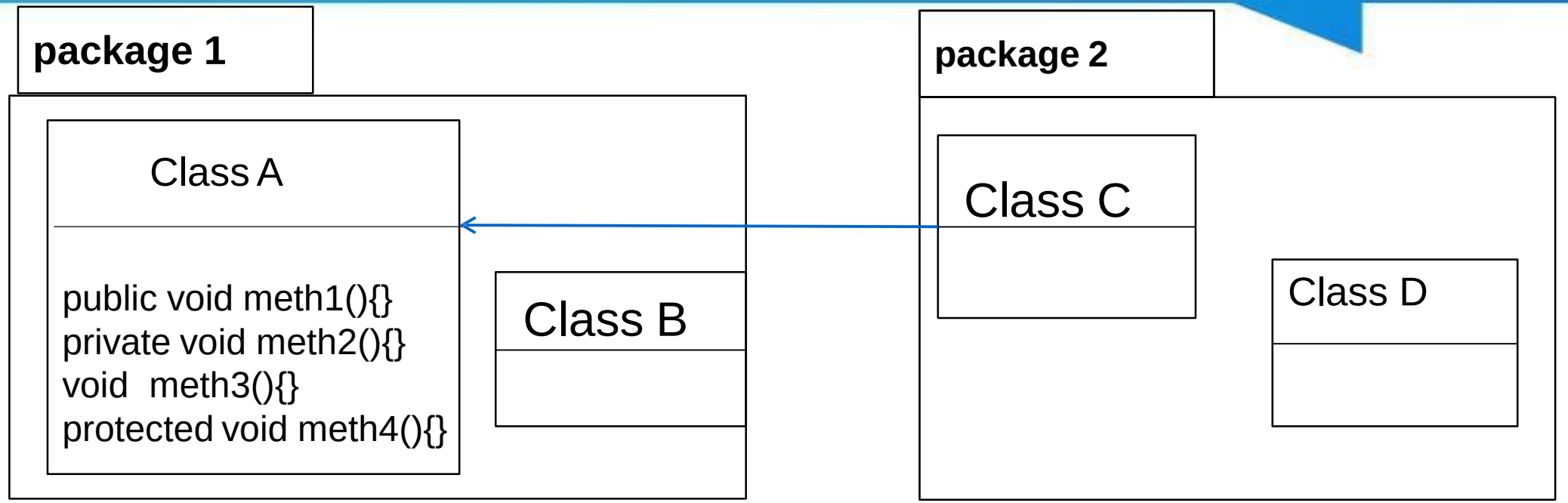
✗

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

✗

Encapsulation des méthodes(4/4)

Méthode protected



protected void meth4() ➡ la méthode protected est accessible que depuis les classes faisant partie du même package et aux sous classes même si elles sont définies dans un package différent

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

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

✔

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

✔

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

✘

plan

- L’encapsulation
- L’encapsulation de classes
- L’encapsulation des attributs
- L’encapsulation des méthodes
- Les accesseurs et les mutateurs

Les accesseurs et les mutateurs (getters & setters)(1/2)

☐ Pour la manipulation des attributs private on utilise :

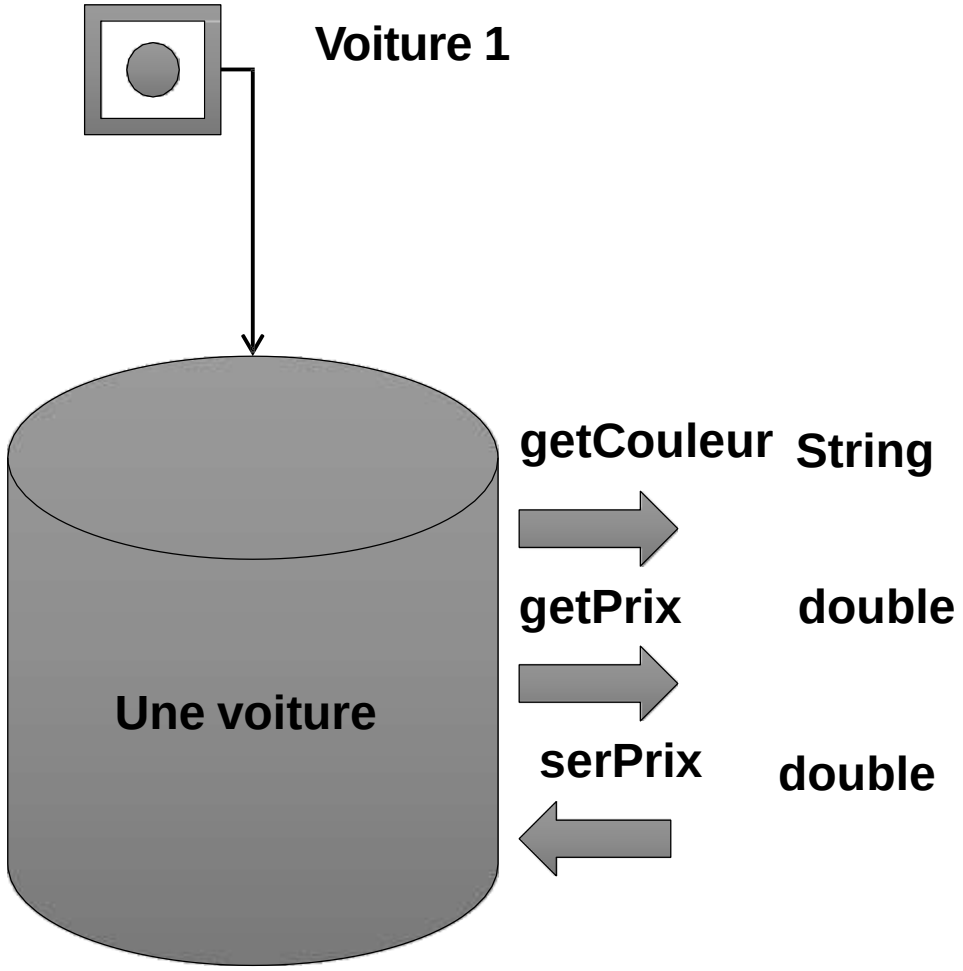
- Un mutateur(setter): une méthode qui permet de définir la valeur d'un attributs particulier
- Un accesseur(getter): une méthode qui permet d'obtenir la valeur d'un attribut particulier
- Les setters et les getters doivent être **public**

Méthodes	**Rôle**
Méthodes d'accès (getter)	Méthodes qui retourne l'état d'un objet (certains champs) qui sont généralement privés . La notation utilisée est getX ; X le champs retourné
Méthodes de modification (setter)	Méthodes qui modifie l'état d'un objet (certains champs). La notation utilisée est setX ; X le champs modifié

Les accesseurs et les mutateurs (getters & setters)(2/2)

- Exemple :

```
public class Voiture
{
  private String couleur;
  private double prix;
      public String getCouleur() {
            return couleur;
      }
      public void setPrix(double
      prix) {
            this.prix = prix;
      }
      public double getPrix() {
            return prix;
      }
      }
}
```



Exemple

Package 1

```
public class Point
{
private double x; // abscisse
private double y; // ordonnée
public double getX (){
      return x;
      }
public double getY(){
      return y;
      }
public void setX(double dx){
      x=dx;
      }
public void setY(double dy){
      y=dy;
      }
public void affiche(){
System.out.println(“abscisse  est:“+ x + ordonnée est :“+y);      }
}
```

Package 1

```
public class TestPoint
    {
       public static void main(String args[])
       {
    Point p1 = new Point();
  p1.x = 5;          // Invalide car l'attribut est privé
          p1.setX()= 4;// OK
         p1.affiche( );// OK
        }
  }
```

Résumé encapsulation de classe

Modificateur	Rôle
abstract	Une classe déclarée abstract ne peut pas être instanciée : il faut définir une classe qui hérite de cette classe et qui implémente les méthodes nécessaires pour ne plus être abstraite.
final	la classe ne peut pas être modifiée, sa redéfinition grâce à l'héritage est interdite. Les classes déclarées final ne peuvent donc pas avoir de classes filles.
public	La classe est accessible partout
par défaut : package friendly	Il n'existe pas de mot clé pour définir ce niveau, qui est le niveau par défaut lorsqu'aucun modificateur n'est précisé. Cette déclaration permet à une classe d'être visible par toutes les classes se trouvant dans le même package.

Résumé encapsulation des attributs et méthodes

Modificateur	Rôle
public	Une variable, méthode déclarée public est visible par tous les autres objets.
par défaut : friendly	Il n'existe pas de mot clé pour définir ce niveau, qui est le niveau par défaut lorsqu'aucun modificateur n'est précisé. Cette déclaration permet à une entité (méthode ou variable) d'être visible par toutes les classes se trouvant dans le même package.
Protected	Si une méthode ou une variable est déclarée protected , seules les méthodes présentes dans le même package que cette classe ou ses sous classes pourront y accéder.
private	C'est le niveau de protection le plus fort. Les composants ne sont visibles qu'à l'intérieur de la classe : ils ne peuvent être modifiés que par des méthodes définies dans la classe prévues à cet effet.

Fin chapitre 4