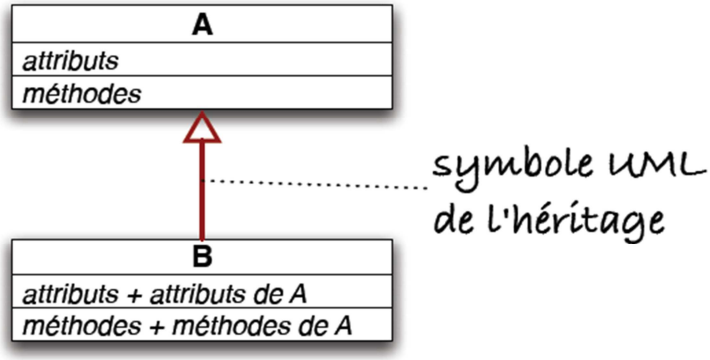
L’héritage



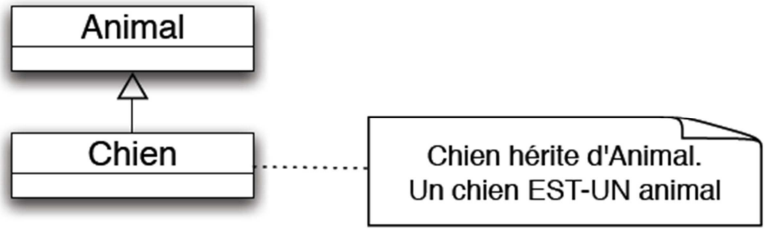
A est la super-classe ou classe mère

B est une sous-classe ou classe dérivée

Tout objet issu de la classe B dispose des attributs et des méthodes de A

Si un membre (attribut, constructeur, méthode) de A est déclaré « private », le code de la classe B n'y a pas accès.

La relation d'héritage est une relation **EST-UN : B est un A**



Chien chien = new Chien(); Ok

Animal chien = new Chien(); Ok

~~Chien chien = new Animal();~~ Non : ne compile pas

Syntaxe Java pour l'héritage

public class B extends A {

. . .

}

public class A {

. . .

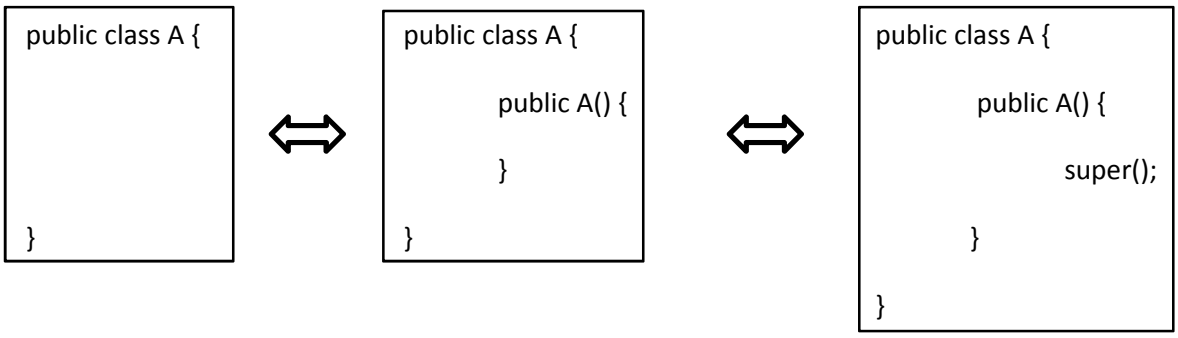
}

⟺

public class A extends Object {

. . .

}



L'appel explicite au constructeur de la super-classe :

La syntaxe générale est « **super(arguments);** »

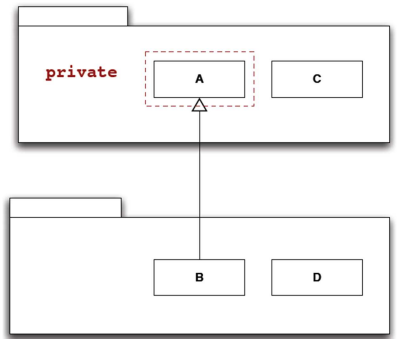
super() s'il est utilisé doit être placé au début du code (première ligne).

Modificateurs de visibilité

private

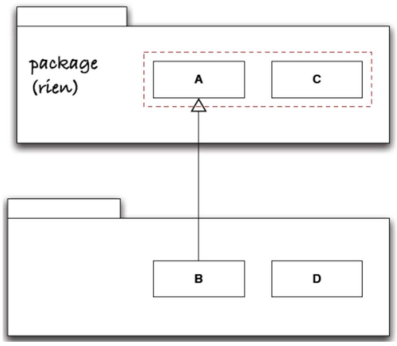
visible seulement des membres de la classe.

À privilégier pour les attributs.



package (rien)

visible des membres de la classe et dans les classes du même package. Possible pour les attributs, c'est la visibilité par défaut en Java pour les attributs.

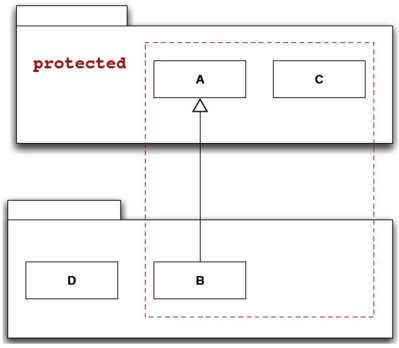


protected

visible des membres de la classe, des classes du même package et des sous-classes de n'importe quel package.

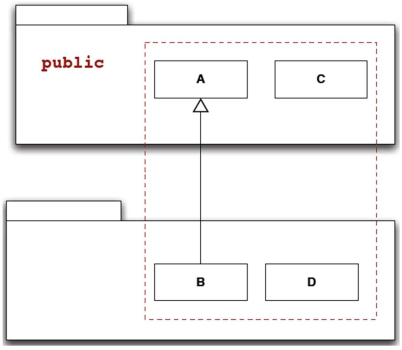
Presque équivalent à public !

À éviter pour les attributs.



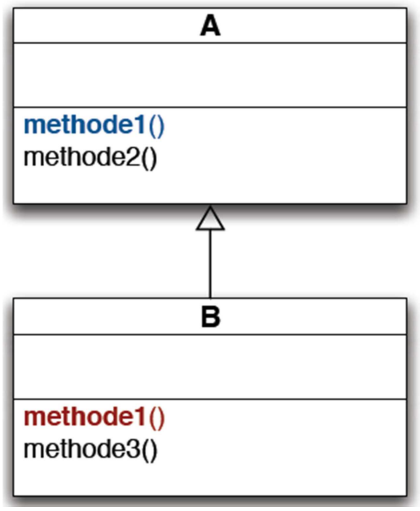
public

visible partout



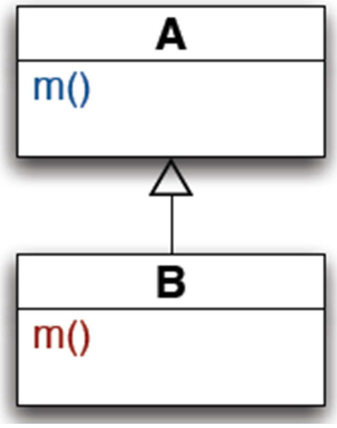
Redéfinition & polymorphisme

redéfinir, c'est déclarer dans une classe une méthode qui a la même signature (identifiant + paramètres) qu'une méthode déclarée dans une classe ancêtre.



Typage statique, liaison dynamique

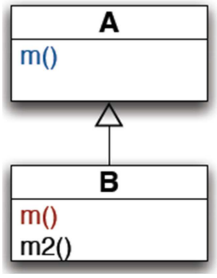
```
A a = new A();     //a est de type A
B b = new B();     //b est de type B et de type A
a.m();
b.m();
A aa = new B();    //aa est de type A
aa.m();
```



En Java la liaison se fait à l'exécution (liaison dynamique)

Certes aa est de type (statique) A mais c'est en réalité une instance de la classe B, c'est donc la méthode rouge qui est appelée.

```
A a = new B();   //a est de type A
a.m2();       // NE SE COMPILE PAS
```

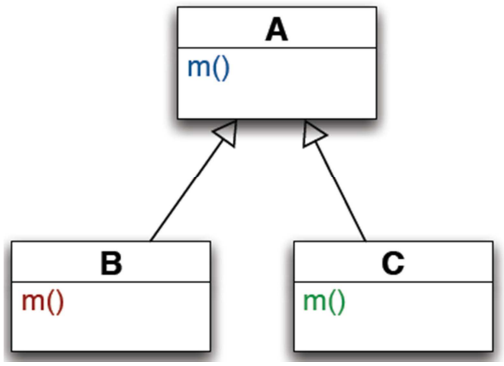


Certes à l'exécution a sera bien une instance de B,

mais à la compilation a est de type A et non B.

Polymorphisme

```
A b = new B();
A c = new C();
b.m();  // méthode m() de la classe B
c.m();  // méthode m() de la classe C
```



Utilisation de super

```
public class A {
        public String f() {
                return "A";
        }
}
public class B extends A {
        public String f() {
                return "B";
        }
        public String fA() {
                return super.f();
        }
        public String fB() {
                return f();
        }
}
```

```
public class Main {
  public static void main(String[] args) {
    A a = new A();
    B b = new B();
    System.out.println(a.f());        // A
    System.out.println(b.f());        // B
    System.out.println(b.fA());       // A
    System.out.println(b.fB());       // B
  }
}
```