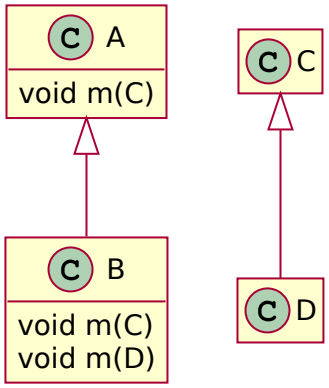
Rappels et compléments



Appel `c.m(u1,...,un)`

Compilation
données :
T = CT(c), T1 = CT(u1), ..., Tn = CT(un)
calcul :
(S1, ..., Sn) = min{(X1, ..., Xn) | T1<:X1, ..., Tn<:Xn, m(X1,...,Xn) ∈ T}
résultat : SIG = m(S1, ..., Sn)

Exécution
si `m` est **`static`** ou **`private`**, ou si `c` est **`super`**
liaison statique : exécution de SIG de CT(c)
sinon
liaison dynamique : exécution de SIG de RT(c)

	T	T1	SIG
(α)	A	D	m(C)
(β)	B	D	m(D)

```
void test() {
    B y = new B();
    A x = y;
    D u = new D();
    x.m(u); // α
    y.m(u); // β
}
```

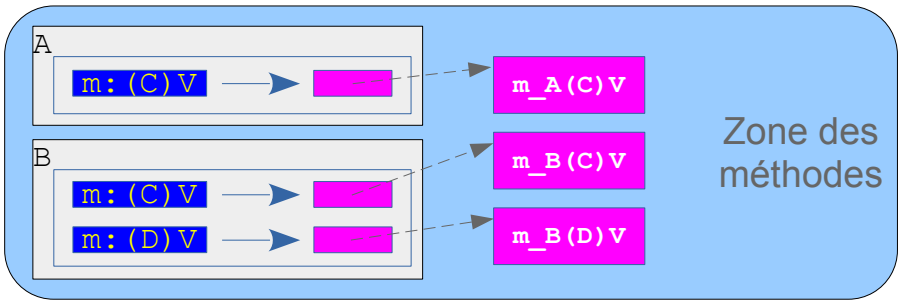
compilation

`invokevirtual <?;m;?>`

`invokevirtual <?;m;?>`

`<A;m;(C)V>` exécution **`m_B(C)V`**

`<B;m;(D)V>` exécution **`m_B(D)V`**

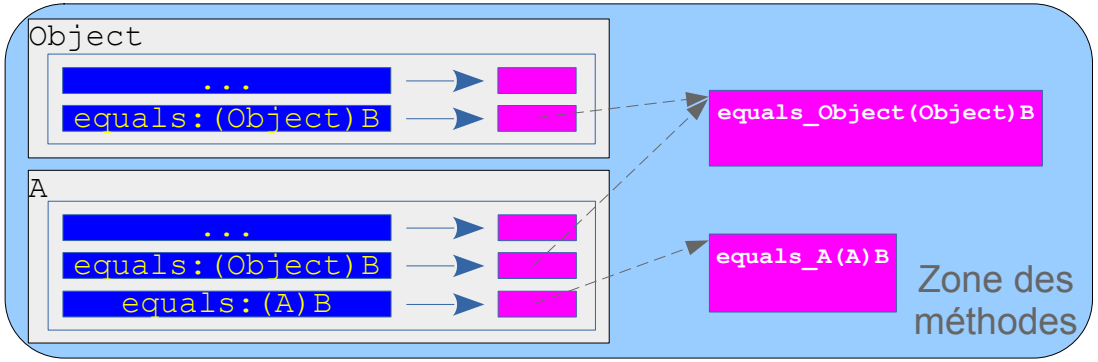


```
public class Object {
    public boolean equals(Object obj) {
        return this == obj;
    }
}
```

je veux redéfinir `equals` dans `A`

surcharge ou redéfinition ?

```
class A {
    public boolean equals(A obj) {...}
}
```

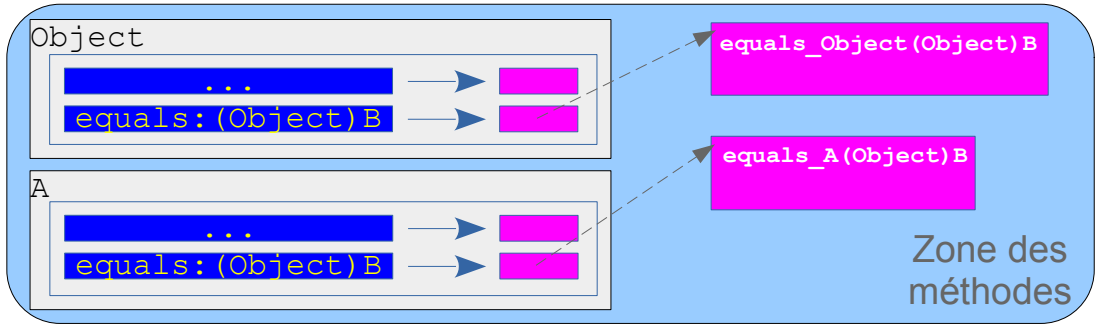



```
void test(Object u, A v) {
    S.o.p(u.equals(v));
}
```

compilation → `invokevirtual <Object;equals;(Object)B>`

```
Object u = new A();
A v = new A();
obj.test(u, v);
```

exécution → `equals_Object(Object)B`




```
void test(Object u, A v) {
    S.o.p(u.equals(v));
}
```

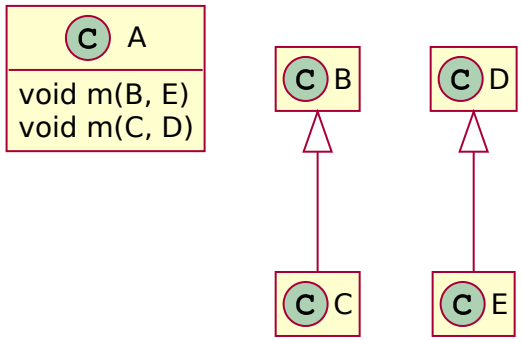
compilation → `invokevirtual <Object;equals;(Object)B>`

```
Object u = new A();
A v = new A();
obj.test(u, v);
```

exécution → `equals_A(Object)B`

```
class A {
    @Override
    ... equals(Object obj) {...}
}
```

Ambiguïté : min{(X1, ..., Xn)} n'existe pas toujours !



Appel `c.m(u1,...,un)`

Compilation
données :
T = CT(c), T1 = CT(u1), ..., Tn = CT(un)
calcul :
(S1, ..., Sn) = min{(X1, ..., Xn) | T1<:X1, ..., Tn<:Xn, m(X1,...,Xn) ∈ T}
résultat : SIG = m(S1, ..., Sn)

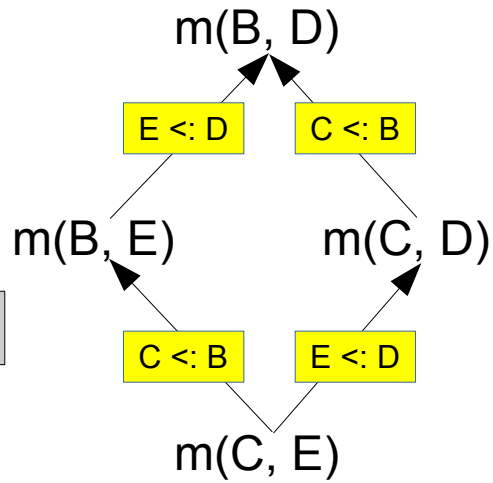
Exécution
si m est **static** ou **private**, ou si c est **super**
liaison statique : exécution de SIG de CT(c)
sinon
liaison dynamique : exécution de SIG de RT(c)

```
void test(C u, E v) {
    A x = new A();
    x.m(u, v);
}
```

compilation

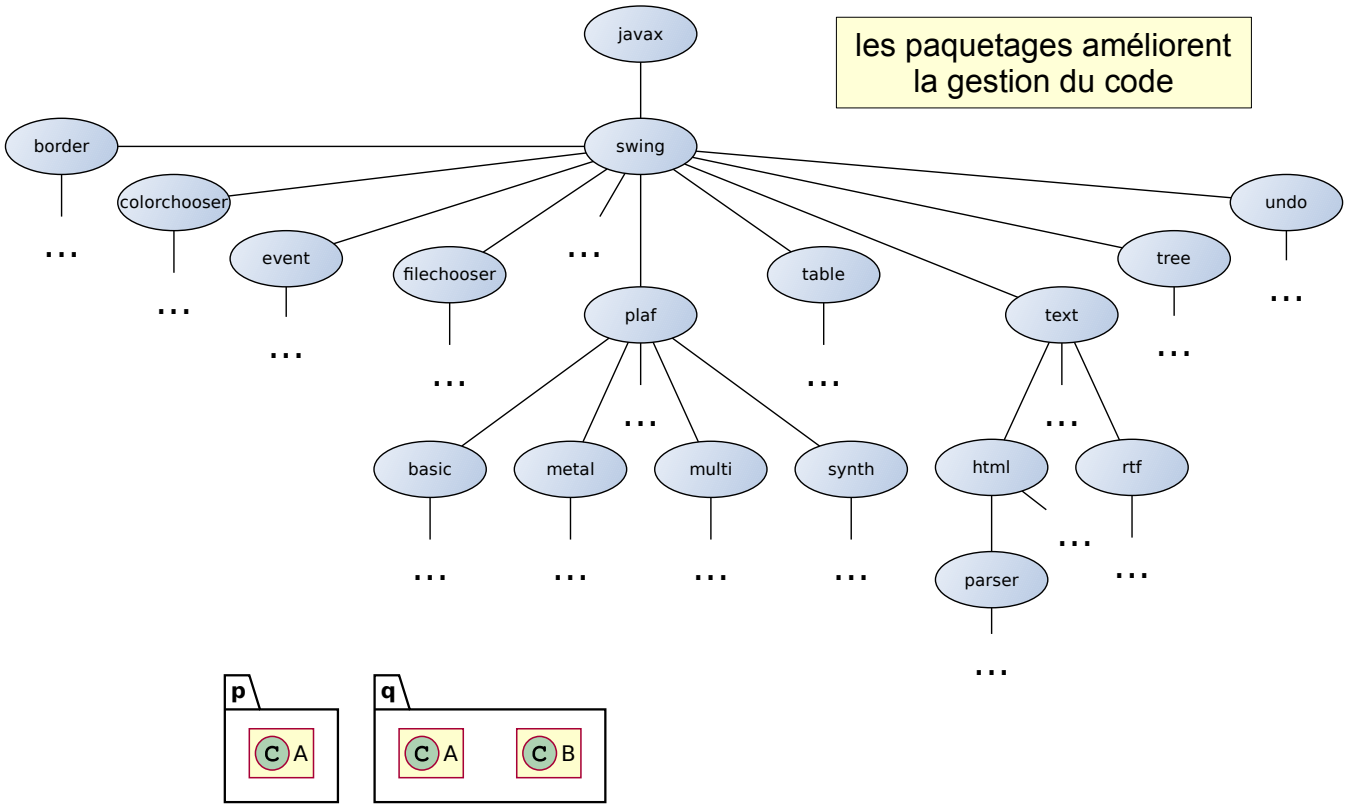
```
invokevirtual <?;m;?>
```

```
erreur : m(B, E) est ambiguë pour le type A
```



solutions : `x.m((B) u, v)` ou `x.m(u, (D) v)`

Paquetage Java : unité de regroupement de types et de sous-paquetages en un tout logiquement cohérent.



les paquetages améliorent la gestion du code

```
package p;

import q.B;

class A {
    private q.A a1;
    private [p.]A a2;
    private B b;
}
```

```
package q;

// p.A est inaccessible
// dans ce fichier

public class A { ... }

public class B { ... }
```

un paquetage est un espace de nommage

un paquetage permet de rendre certains types inaccessibles

Accessibilité d'un élément dans le code

```
public class B {
    private int i;
    public void m() { ... }
}
```

```
class T {
    ...
}
```

`i` n'est pas accessible dans `T`
⇒ ~~`b.i = 3;`~~

`m` est accessible dans `T`
⇒ `b.m();`

`B` est accessible dans `T`
⇒ `new B()`

Accessibilité d'un paquetage p
- `p` est accessible dans tout `T`

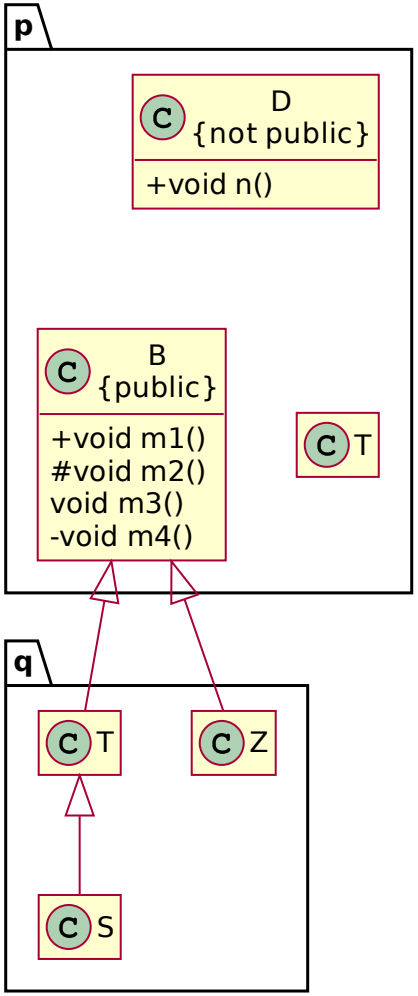
Accessibilité d'un type B ∈ p
- **`public`** : `B` est accessible dans tout `T`
- *`<rien>`* : `B` n'est accessible que dans `T` ∈ `p`

Rappels et compléments

Accessibilité d'une caractéristique c, déclarée dans B ∈ p

- **public** : c n'est accessible que dans les T pour lesquels B est accessible
- **protected** : c n'est accessible que dans les T ∈ p, et hors de p, dans les T <: B et « *responsables de l'implémentation de la cible d'appel* »
- *<rien>* : c n'est accessible que dans les T ∈ p
- **private** : c n'est accessible que dans B

la cible x est
- **this**
- ou **super**
- ou tq CT(x) <: T




```
package p;
class T {
    ...
    b.m4();
    b.m3();
    d.n();
    b.m1();
    b.m2();
    ...
}
```

```
package q;
class T extends B {
    ...
    d.n();
    this.m4();
    super.m4();
    b.m4();
    this.m3();
    super.m3();
    b.m3();
    this.m1();
    super.m1();
    b.m1();
    this.m2();
    super.m2();
    b.m2();
    t.m2();
    s.m2();
    z.m2();
    ...
}
```

q.T **est** responsable de l'implémentation de **this** et de **super**

q.T **n'est pas** responsable de l'implémentation de b

q.T **est** responsable de l'implémentation de t et de s

q.T **n'est pas** responsable de l'implémentation de z