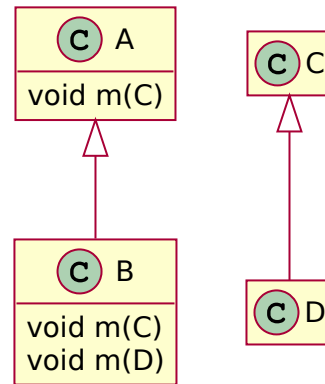


## Rappels et compléments

### V. Héritage

1. Mécanisme d'héritage
  - i. Définition générale
  - ii. Relation entre classes
  - iii. Formes d'héritage en Java
2. Sous-typage Java
  - i. Sous-type Java direct
  - ii. Sous-type Java
  - iii. Sous-types tableaux
3. Expressions
  - i. Définitions
  - ii. Valeur d'une expression
  - iii. Types d'une expression
  - iv. Transtypage
    - a. Définition
    - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
  - i. Redéf° et liaison dynamique
  - ii. Modification d'entête
  - iii. @Override
  - iv. Redéfinition et surcharge
7. Invocation de méthode
  - i. Principe
    - a. Méthode virtuelle
    - b. Méthode de classe
    - c. Méthode privée
    - d. Méthode avec super
  - ii. Invocation et surcharge
    - a. Résolution d'appel
    - b. Surcharge et héritage
    - c. Piège de la surcharge
    - d. Ambiguïté
8. Accessibilité
  - i. Paquetage
  - ii. Accessibilité des types
  - iii. Accessibilité des caract.



```

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

compilation

invokevirtual <?;m;?>

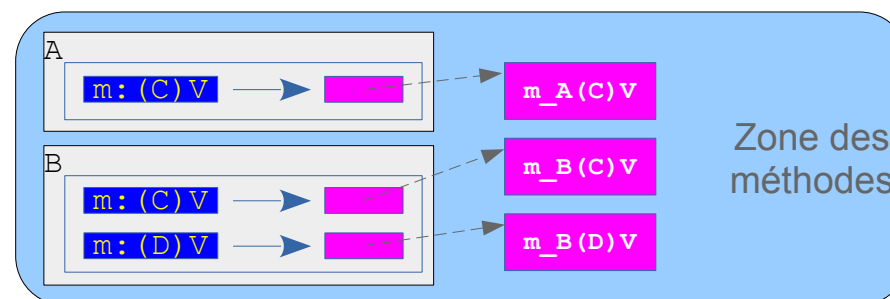
invokevirtual <?;m;?>

<B;m;(D) V> exécution

m\_B(D) V

<A;m;(C) V> exécution

m\_B(C) V



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

#### Compilation

données :

$T = CT(c), T1 = CT(u1), \dots, Tn = CT(un)$

calcul :

$(S1, \dots, Sn) = \min\{(X1, \dots, Xn) \mid T1 \leq X1, \dots, Tn \leq Xn, m(X1, \dots, Xn) \in T\}$

résultat : **SIG** =  $m(S1, \dots, Sn)$

#### 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)

## Rappels et compléments

### V. Héritage

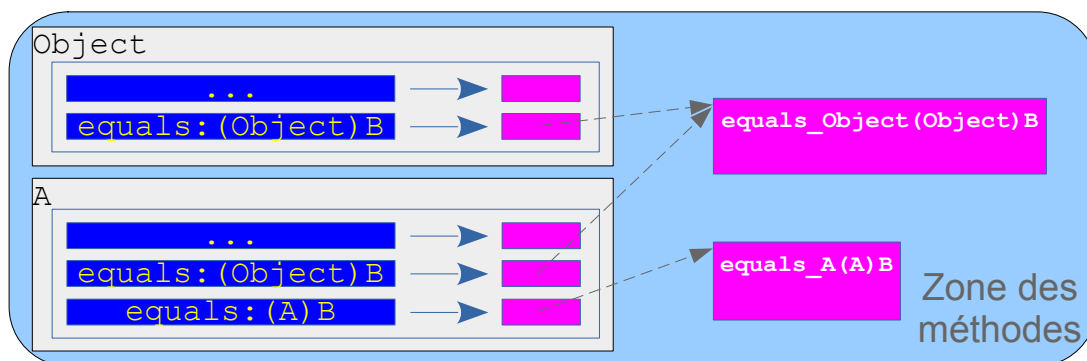
1. Mécanisme d'héritage
  - i. Définition générale
  - ii. Relation entre classes
  - iii. Formes d'héritage en Java
2. Sous-typage Java
  - i. Sous-type Java direct
  - ii. Sous-type Java
  - iii. Sous-types tableaux
3. Expressions
  - i. Définitions
  - ii. Valeur d'une expression
  - iii. Types d'une expression
  - iv. Transtypage
    - a. Définition
    - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
  - i. Redéf° et liaison dynamique
  - ii. Modification d'entête
  - iii. @Override
  - iv. Redéfinition et surcharge
7. Invocation de méthode
  - i. Principe
    - a. Méthode virtuelle
    - b. Méthode de classe
    - c. Méthode privée
    - d. Méthode avec super
  - ii. Invocation et surcharge
    - a. Résolution d'appel
    - b. Surcharge et héritage
    - c. Piège de la surcharge
    - d. Ambiguïté
8. Accessibilité
  - i. Paquetage
  - ii. Accessibilité des types
  - iii. Accessibilité des caract.

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

je veux redéfinir equals dans A

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

surcharge  
ou  
redéfinition ?



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

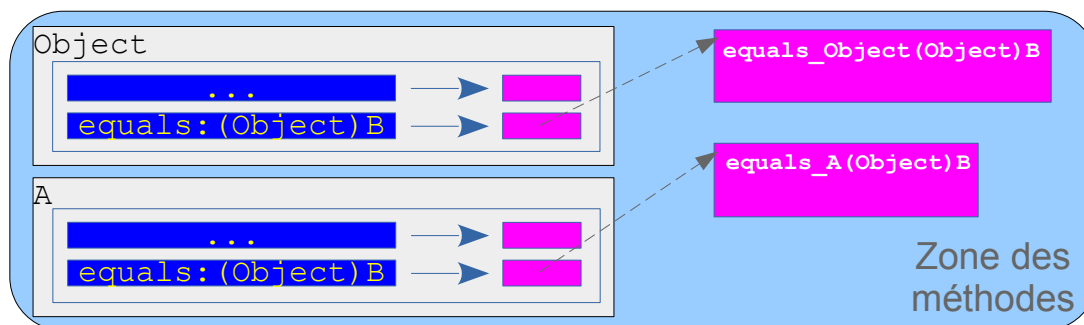
compilation

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

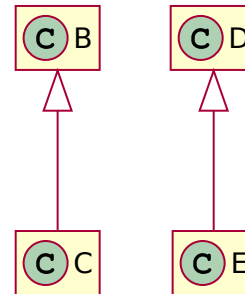
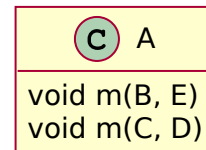
```
class A {
    @Override
    ... equals(Object obj) {...}
}
```

## Rappels et compléments

### V. Héritage

1. Mécanisme d'héritage
  - i. Définition générale
  - ii. Relation entre classes
  - iii. Formes d'héritage en Java
2. Sous-typage Java
  - i. Sous-type Java direct
  - ii. Sous-type Java
  - iii. Sous-types tableaux
3. Expressions
  - i. Définitions
  - ii. Valeur d'une expression
  - iii. Types d'une expression
  - iv. Transtypage
    - a. Définition
    - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
  - i. Redéf° et liaison dynamique
  - ii. Modification d'entête
  - iii. @Override
  - iv. Redéfinition et surcharge
7. Invocation de méthode
  - i. Principe
    - a. Méthode virtuelle
    - b. Méthode de classe
    - c. Méthode privée
    - d. Méthode avec super
  - ii. Invocation et surcharge
    - a. Résolution d'appel
    - b. Surcharge et héritage
    - c. Piège de la surcharge
    - d. Ambiguïté
8. Accessibilité
  - i. Paquetage
  - ii. Accessibilité des types
  - iii. Accessibilité des caract.

**Ambiguïté** :  $\min\{(X1, \dots, Xn)\}$  n'existe pas toujours !



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

#### Compilation

données :

$T = CT(c), T1 = CT(u1), \dots, Tn = CT(un)$

calcul :

$(S1, \dots, Sn) = \min\{(X1, \dots, Xn) \mid T1 <: X1, \dots, Tn <: Xn, m(X1, \dots, Xn) \in T\}$

résultat : **SIG** =  $m(S1, \dots, Sn)$

#### 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  $RT(c)$

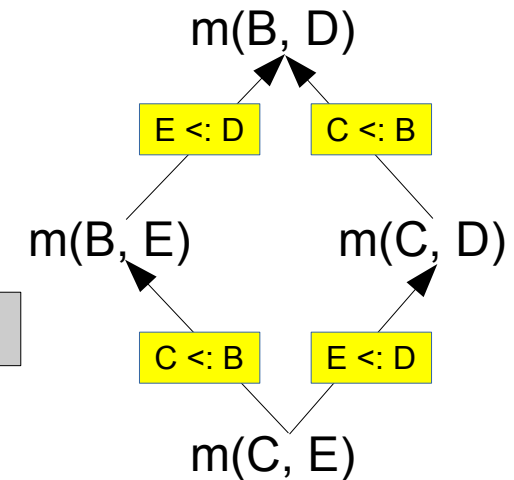
```

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



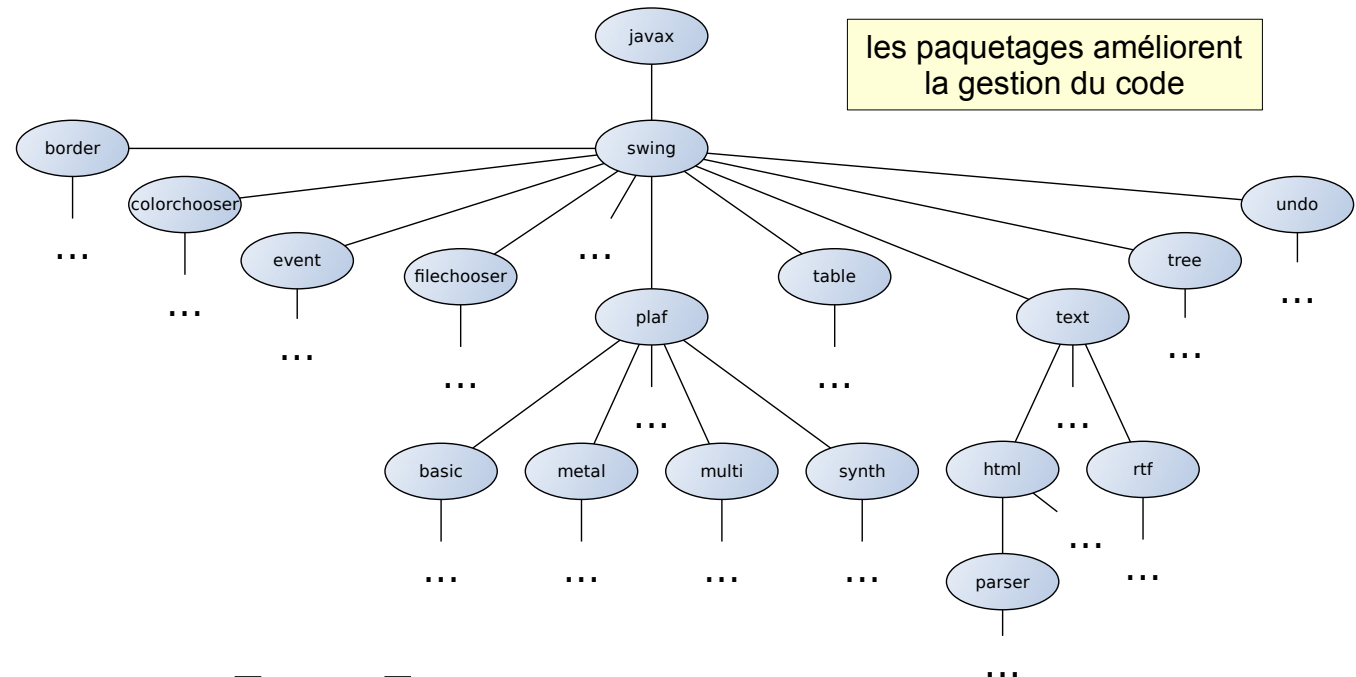
solutions : `x.m( (B) u, v )` ou `x.m(u, (D) v)`

## Rappels et compléments

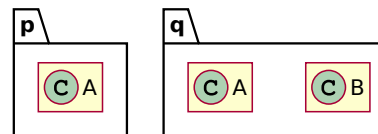
### V. Héritage

1. Mécanisme d'héritage
  - i. Définition générale
  - ii. Relation entre classes
  - iii. Formes d'héritage en Java
2. Sous-typage Java
  - i. Sous-type Java direct
  - ii. Sous-type Java
  - iii. Sous-types tableaux
3. Expressions
  - i. Définitions
  - ii. Valeur d'une expression
  - iii. Types d'une expression
  - iv. Transtypage
    - a. Définition
    - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
  - i. Redéf° et liaison dynamique
  - ii. Modification d'entête
  - iii. @Override
  - iv. Redéfinition et surcharge
7. Invocation de méthode
  - i. Principe
    - a. Méthode virtuelle
    - b. Méthode de classe
    - c. Méthode privée
    - d. Méthode avec super
  - ii. Invocation et surcharge
    - a. Résolution d'appel
    - b. Surcharge et héritage
    - c. Piège de la surcharge
    - d. Ambiguïté
8. Accessibilité
  - i. Paquetage
  - ii. Accessibilité des types
  - iii. Accessibilité des caract.

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



les paquets 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

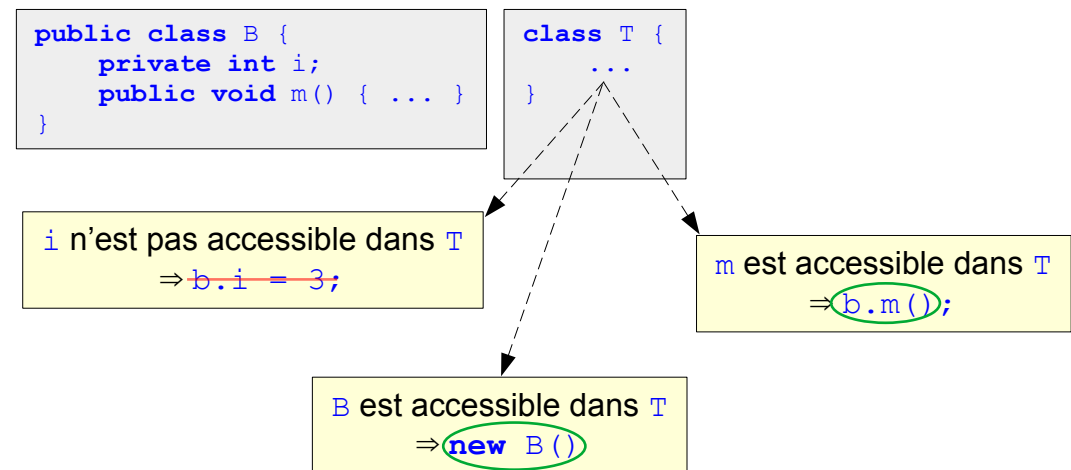
un paquetage permet de rendre certains types inaccessibles

## Rappels et compléments

### V. Héritage

1. Mécanisme d'héritage
  - i. Définition générale
  - ii. Relation entre classes
  - iii. Formes d'héritage en Java
2. Sous-typage Java
  - i. Sous-type Java direct
  - ii. Sous-type Java
  - iii. Sous-types tableaux
3. Expressions
  - i. Définitions
  - ii. Valeur d'une expression
  - iii. Types d'une expression
  - iv. Transtypage
    - a. Définition
    - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
  - i. Redéf° et liaison dynamique
  - ii. Modification d'entête
  - iii. @Override
  - iv. Redéfinition et surcharge
7. Invocation de méthode
  - i. Principe
    - a. Méthode virtuelle
    - b. Méthode de classe
    - c. Méthode privée
    - d. Méthode avec super
  - ii. Invocation et surcharge
    - a. Résolution d'appel
    - b. Surcharge et héritage
    - c. Piège de la surcharge
    - d. Ambiguïté
8. Accessibilité
  - i. Paquetage
  - ii. Accessibilité des types
  - iii. Accessibilité des caract.

## Accessibilité d'un élément dans le code



### Accessibilité d'un paquetage p

- p est accessible dans tout T

### Accessibilité d'un type $B \in p$

- **public** : B est accessible dans tout T
- *<rien>* : B n'est accessible que dans  $T \in p$

## Rappels et compléments

### V. Héritage

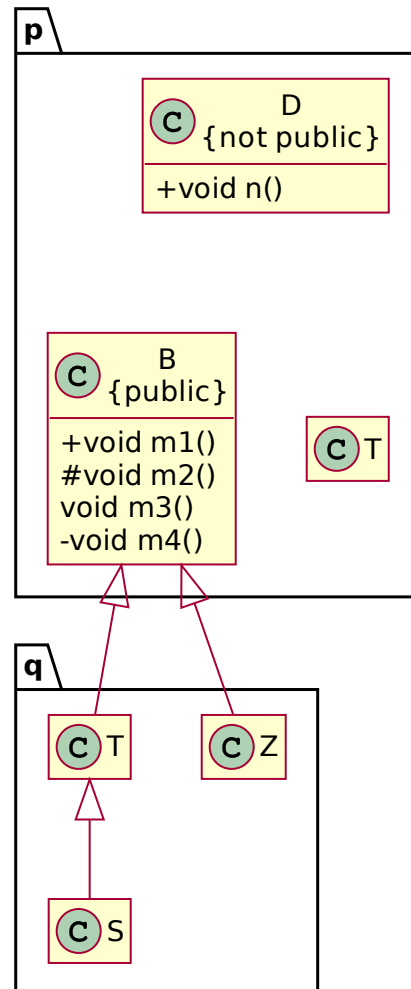
1. Mécanisme d'héritage
  - i. Définition générale
  - ii. Relation entre classes
  - iii. Formes d'héritage en Java
2. Sous-typage Java
  - i. Sous-type Java direct
  - ii. Sous-type Java
  - iii. Sous-types tableaux
3. Expressions
  - i. Définitions
  - ii. Valeur d'une expression
  - iii. Types d'une expression
  - iv. Transtypage
    - a. Définition
    - b. Extensibilité potentielle
4. Masquage d'attribut
5. Chaînage des constructeurs
6. Redéfinition de méthode
  - i. Redéf° et liaison dynamique
  - ii. Modification d'entête
  - iii. @Override
  - iv. Redéfinition et surcharge
7. Invocation de méthode
  - i. Principe
    - a. Méthode virtuelle
    - b. Méthode de classe
    - c. Méthode privée
    - d. Méthode avec super
  - ii. Invocation et surcharge
    - a. Résolution d'appel
    - b. Surcharge et héritage
    - c. Piège de la surcharge
    - d. Ambiguïté
8. Accessibilité
  - i. Paquetage
  - ii. Accessibilité des types
  - iii. Accessibilité des caract.

### Accessibilité d'une caractéristique $c$ , déclarée dans $B \in p$

- **public** :  $c$  n'est accessible que dans les  $T$  pour lesquels  $B$  est accessible
- **protected** :  $c$  n'est accessible que dans les  $T \in p$ , 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  $T \in p$
- **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**