Types en Java

- langage STATIQUEMENT typé
- et DYNAMIQUEMENT typé

$\Rightarrow$ Avantages ET inconvenients des 2

Types

Les types de Java sont construits de la manière suivante :

- types de base : si τ est un type de base alors τ est un type;
- tableaux : si τ est un type alors $\tau[]$ est un type;
- classes : si C est une classe alors C est un type;
- interfaces : si I est une interface alors I est un type.

Sous-types

En Java la relation d'héritage entraine la relation de sous-typage ($\leq$).

- si τ est un type alors $\tau \leq \tau$;
- si SC est sous-classe de C, alors $SC \leq C$;
- si SI est sous-interface de I, alors $SI \leq I$;
- si C implante I alors $C \leq I$;
- si $\tau_2 \leq \tau_1$, alors $\tau_2[] \leq \tau_1[]$

Si $\tau_2 \leq \tau_1$, alors toute valeur de type τ_2 peut être utilisée en place et lieu d'une valeur de type τ_1.

Contraintes de type (1)

implicite: : si $\tau_2 \leq \tau_1$

- affectation : τ_1 v = new τ_2 ();
- passage d'arguments :
  si void m1(τ_1 a, τ_1 b)
  m1 (new τ_2 (), new τ_2 ());

Contraintes de type (2)

explicite: : (τ) `expr`

τ doit être sous-type du type de `expr`

$\Rightarrow$ vérification DYNAMIQUE de types

Uniquement au niveau des types, ne modifie pas les instances!!!

```
SC sc = new SC();
C c = sc;
SC sc2 = (SC) c;
```

Exemple (implicite) : ev1.java

```
class p {
    int x,y;
    p(int x, int y) {this.x=x; this.y=y;}
    int gx() { return x;}
    void mv(int a, int b) {x= a; y = b;  }
    String ts() { return "("+x+","+y+")"; }
}

class c_p extends p {
    String c;
    c_p(int x, int y, String c) {super(x,y); this.c=c;}
    String gc() {return c;}
    String ts() { return (super.ts() + " : " + this.gc());}
}
```

```
class nc_p extends c_p {
    nc_p(int x, int y) {super(x,y,"NOTHING");}
    String gc() { return "NO COLOR";}
}


class ev1 {
    public static void main(String [] a) {
p p1 = new p (2,3);
c_p cp1 = new c_p(3,4,"bleu");
nc_p ncp1 = new nc_p(1,2);

System.out.println(p1.ts());  // (2,3)
System.out.println(cp1.ts()); // (3,4) : bleu
System.out.println(ncp1.ts());// (1,2) : NO COLOR
```

```
c_p cp2 = ncp1;
System.out.println(cp2.ts()); // (1,2) : NO COLOR
cp2.mv(10,10);
System.out.println(cp2.ts()); // (10,10) : NO COLOR
System.out.println(ncp1.ts()); // (10,10) : NO COLOR
p p2 = cp2;
System.out.println(p2.ts()); // (10,10) : NO COLOR
    }
}
```

Exemple (explicite) : ev2.java

```
class Empty extends Exception {}
class Full extends Exception {}

class queue {
    int tete, queue;
    Object [] q;
    int size;
    int len;

    queue (int n) { size = n; q = new Object[n];}
    void enq(Object  x) throws Full {
        if (len < size) {q[queue++ % size] = x;len++;}
        else throw new Full();
    }
```

```
    Object deq() throws Empty {
        if (len > 0) { len--; return q[tete++];}
        else throw new Empty();
    }
}


class ev2 {
    public static void main(String[] a) {
queue q = new queue(10);
p p1 = new p(2,3);
c_p cp1 = new c_p(2,4,"rouge");
p p2 = cp1;
try {
q.enq( (Object)p1); q.enq(cp1); q.enq(p2);
Object o1 = q.deq();
```

```
System.out.println(((p)o1).ts());   // (2,3)
Object o2 = q.deq();
System.out.println(((p)o2).ts());   // (2,3) : rouge
System.out.println(((c_p)o2).ts());  // (2,3) : rouge
System.out.println(((p)(q.deq())).ts()); // (2,3) : rouge
}
catch (Full f) {System.out.println("Plein");}
catch (Empty e) {System.out.println("Vide");}
    }
}
```

Exemple (explicite avec tests) : ev3.java

```
class ev3 {
    public static void main(String[] a) {
queue q = new queue(10);
p p1 = new p(2,3);
c_p cp1 = new c_p(2,4,"rouge");
p p2 = cp1;
try {
q.enq( (Object)p1); q.enq(cp1); q.enq(p2);
Object o1 = q.deq();
if (o1 instanceof c_p) {
    System.out.println("cas 1 : " + ((c_p)o1).ts());
}
else {System.out.println("cas 2 : " + ((p)o1).ts());} // (2,3)
Object o2 = q.deq();
```

```
if (o1 instanceof c_p) {
    System.out.println("cas 3 : " + ((c_p)o2).ts());
}
else {System.out.println("cas 4 : " + ((p)o2).ts());} // (2,4) rouge
}
catch (Full f) {System.out.println("Plein");}
catch (Empty e) {System.out.println("Vide");}
    }
}
```

Surcharge

- choix du type de la méthode à employer lors d'un appel de méthode
- résolue STATIQUEMENT selon une relation d'ordre sur la classe de définition et le type des arguments
- le type du résultat n'est pas pris en compte
- il y a des cas où la résolution échoue

Exemple

```
class A :  m2(A) m2(A,A)
class B :  m2(B) m2(A,B) m2(B,A)
class C :  m2(A) m2(B,C) m2(C,A)
```

C hérite de B qui hérite de A

```
A a1 = new A(); B b1 = new B(); C c1 = new C();
c1.m2(x,y)
```

Quel est le type de la méthode à utiliser?

La surchage en Java est résolue statiquement.

Une méthode est du point de vue des types un couple contenant le nom de la classe de définition et le produit cartésien des types des paramètres de la méthode.

Au niveau de C on a 8 méthodes m2 dont 5 à 2 arguments:

	classe de définition	type de la méthode
m2	A	(A)
	A	(A,A)
	B	(B)
	B	(A,B)
	B	(B,A)
	C	(C)
	C	(B,C)
	C	(C,A)

Sélection du type de la méthode (1)

Sur le nombre de paramètres (exemple sur 2) :

	classe de définition	type de la méthode
m2	A	(A,A)
	B	(A,B)
	B	(B,A)
	C	(B,C)
	C	(C,A)

Sélection du type de la méthode (2)

`c1.m2(x,y);`

Sur l'ensemble des méthodes `m2` du receveur (ici `c1`), on ne conserve que celles dont le type (τ_1,τ_2) vérifie :

- type de x $\leq \tau_1$
- type de y $\leq \tau_2$

Exemple : c1.m2(a1,b1)

	classe de définition	type de la méthode
m2	A	(A,A)
	B	(A,B)

Sélection du type de la méthode (3)

On note τ_{def} x (τ_1, τ_2) l'information sur la classe de définition et le type d'une méthode

On sélectionne alors les plus petites méthodes selon une des relations d'ordre suivantes :

1. $m_2^{'} \leq m_2^{''}$ ssi $\tau_{def}^{'} \leq \tau_{def}^{''}$ et $\tau_1^{'} \leq \tau_1^{''}$ et $\tau_2^{'} \leq \tau_2^{''}$
2. $m_2^{'} \leq m_2^{''}$ ssi $\tau_1^{'} \leq \tau_1^{''}$ et $\tau_2^{'} \leq \tau_2^{''}$

Dans les 2 cas, l'exemple donne la methode C x (B, C)

Cas d'ambiguité

Soit une classe B avec 2 méthodes m2(A,B) et m2(B,A) :

```
A a1 = new A();
B b1 = new B();
b1.m2(a1,a1);
```

Sur les types on obtient B x (A, B) et B x (A, B)
Aucune de ces deux méthodes ne possède un type tel que (A,A) soit plus petit.
il y a un clash à la compilation!!!

```
b1.m2(b1,b1)
```

Ici les deux méthodes ont un type telque (B,B) soit plus petit, mais aucune des deux méthodes est plus petite que l'autre
il y a un clash à la compilation!!!

Exemple : c1.m2(b1,b1)

	classe de définition	type de la méthode
m2	A	(A,A)
	B	(A,B)
	B	(B,A)

Exemple : abc.java

```
class A {
    void m2(A a) {System.out.println("A1");}
    void m2(A a1, A a2) {System.out.println("A2");}
}

class B extends A {
    void m2(B  b) {System.out.println("B1");}
    void m2(A a1, B b2) {System.out.println("B2");}
    void m2(B b1, A a2) {System.out.println("B2");}
}

class C extends B {
    void m2(C  c) {System.out.println("C1");}
    void m2(B b1, C c1) {System.out.println("C2");}
```

```
    void m2(C c1, A a1) {System.out.println("C3");}

}

class abc {
    public static void main(String [] a) {
A a1 = new A();
B b1 = new B();
C c1 = new C();
        c1.m2(b1,b1);;
    }
}
```

```
% javac abc.java
abc.java:25: reference to m2 is ambiguous, both method m2(A,B) in B
        c1.m2(b1,b1);;
          ^
```

```
1 error
```

Exemple : abc.java

```
class A {
    void m2(A a) {System.out.println("A1");}
    void m2(C c1, B b1) {System.out.println("A2");}
}

class B extends A {
    void m2(B  b) {System.out.println("B1");}
    void m2(C c1, C c2) {System.out.println("B2");}
}

class C extends B {
    void m2(C  c) {System.out.println("C1");}
    void m2(A a1, A a2) {System.out.println("C2");}
    void m2(B b1, B b2) {System.out.println("C3");}
```

```
}


class abcx {
    public static void main(String [] a) {
A a1 = new A();
B b1 = new B();
C c1 = new C();
A a2 = c1;
        a1.m2(c1,c1);  // A2
        b1.m2(c1,c1);  // B2
//    c1.m2(c1,c1);
// ambiguite entre Bx(C,C) et Cx(B,B)
c1.m2(b1,b1); //C3
//    c1.m2(c1,b1);
// ambiguite entre Ax(C,B) et Cx(B,B)
```

```
        }
    }
```

Exemples

Dans l'ensemble des méthodes compatibles du point de vue des types, le compilateur Java va choisir celle de plus petit type définie dans la classe de plus petit type. Néanmoins il peut exister des cas d'ambiguité.

Evolution en Java: : modification de l'algorithme de résolution de la surcharge (selon les JDK).

voir exemples suivants

Liaison retardée et surcharge

Pas de lien :

- surcharge : résolution à la compilation du choix du type de la méthode à employer, ce qui garantit qu'une telle méthode existe
- liaison retardée : par contre savoir quelle méthode répondant au bon type se joueara à l'exécution.

Exemple

```
class A {
    void m2(A a) {System.out.println("A1");}
    void m2(A a1, A a2) {System.out.println("A2");}
}

class B extends A {
    void m2(B  b) {System.out.println("B1");}
    void m2(B b1, A a2) {System.out.println("B2");}
}

class C extends B {
    void m2(C  c) {System.out.println("C1");}
    void m2(A a1, A a2) {System.out.println("C2");}
    void m2(B b1, B b2) {System.out.println("C3");}
```

```
}



class abcy {
    public static void main(String [] a) {
A a1 = new A();
B b1 = new B();
C c1 = new C();
B b2 = c1;
        a1.m2(c1,c1);  // A2
        b1.m2(c1,c1);  // B2
        b2.m2(c1,c1);  // B2   <----
c1.m2(c1,c1);  // C3

    }
}
```

Tests des environnements du GLA

Faites tourner les exemples sur les differents environnements du GLA.

1. premier exemple

```
class A {
    int m (A x) { System.out.println(1+" "); return (1);}
    boolean n (A x) {System.out.println(2+" "); return (true);}
    int m (A x, A y) {System.out.println(3+" "); return 3;}
    boolean n (A x, A y) {System.out.println(4+" "); return (false);}
}

class B extends A {
    int m (B x) { System.out.println(5+" "); return (5);}
    boolean n (B x) {System.out.println(6+" "); return (true);}
    int m (B x, B y) {System.out.println(7+" "); return 7;}
    boolean n (B x, B y) {System.out.println(8+ " "); return (false);}
}
```

```
class ex1 {
    public static void main(String [] args) {
        A a1 = new A ();
        B b1 = new B();
        A a2 = b1;
        System.out.println("*1-------");
        a1.m(a1);
        a1.n(a1);
        a1.m(a1,a1);
        a1.n(a1,a1);
        System.out.println("*2-------");
        b1.m(b1);
        b1.n(b1);
        b1.m(b1,b1);
        b1.n(b1,b1);
        System.out.println("*3-------");
        b1.m(a1);
```

```
b1.n(a1);
b1.m(a1,a1);
b1.n(a1,a1);
System.out.println("*4-------");
a2.m(a1);
a2.n(a1);
a2.m(a1,a1);
a2.n(a1,a1);
System.out.println("*5-------");
a2.m(a2);
a2.n(a2);
a2.m(a2,a2);
a2.n(a2,a2);
System.out.println("*6-------");
b1.m(a1);
b1.n(b1);
b1.m(a1,b1);
```

```
        b1.n(a1,b1);

    }
}
```

2. sortie du premier exemple

```
*1-------
1
2
3
4
*2-------
5
6
7
8
```

```
*3-------
1
2
3
4
*4-------
1
2
3
4
*5-------
1
2
3
4
*6-------
```

```
1
6
3
4
```

3. deuxième exemple : dépendance croisée

```
class A {
    int m (A x) { System.out.println(1+" "); return (1);}
    boolean n (B x) {System.out.println(2+" "); return (true);}
    int m (A x, A y) {System.out.println(3+" "); return 3;}
    boolean n (B x, B y) {System.out.println(4+" "); return (fals
}

class B extends A {
    int m (B x) { System.out.println(5+" "); return (5);}
```

```
    boolean n (A x) {System.out.println(6+" "); return (true);}
    int m (B x, B y) {System.out.println(7+" "); return 7;}
    boolean n (A x, A y) {System.out.println(8+ " "); return (fal
}


class ex2 {
    public static void main(String [] args) {
        A a1 = new A ();
        B b1 = new B();
        A a2 = b1;
        System.out.println("*1-------");
        a1.m(a1);
        a1.n(b1);
        a1.m(a1,a1);
```

```
a1.n(b1,b1);
System.out.println("*2-------");
b1.m(b1);
b1.n(b1);
b1.m(b1,b1);
b1.n(b1,b1);
System.out.println("*3-------");
b1.m(a1);
b1.n(a1);
b1.m(a1,a1);
b1.n(a1,a1);
System.out.println("*4-------");
a2.m(a1);
a2.n(b1);
a2.m(a1,a1);
```

```
        a2.n(b1,b1);
        System.out.println("*5-------");
        a2.m(a2);
        a2.n((B)a2);
        a2.m(a2,a2);
        a2.n((B)a2,(B)a2);
        System.out.println("*6-------");
        b1.m(a1);
        b1.n(b1);
        b1.m(a1,b1);
        b1.n(a1,b1);

    }
}
```

4. sortie du deuxième exemple

```
*1-------
1
2
3
4
*2-------
5
6
7
8
*3-------
1
6
3
8
*4-------
```

```
1
2
3
4
*5-------
1
2
3
4
*6-------
1
6
3
8
```

5. troisième exemple

```
class A {
    int m (A x) { System.out.println(1+" "); return (1);}
    boolean n (B x) {System.out.println(2+" "); return (true);}
    int m (A x, A y) {System.out.println(3+" "); return 3;}
    boolean n (A x, B y) {System.out.println(4+" "); return (fals
}

class B extends A {
    int m (B x) { System.out.println(5+" "); return (5);}
    boolean n (A x) {System.out.println(6+" "); return (true);}
    int m (B x, B y) {System.out.println(7+" "); return 7;}
    boolean n (B x, A y) {System.out.println(8+ " "); return (fal
}
```

```
class ex3 {
    public static void main(String [] args) {
        A a1 = new A ();
        B b1 = new B();
        A a2 = b1;
        System.out.println("*1-------");
        a1.m(a1);
        a1.n(b1);
        a1.m(a1,a1);
        a1.n(b1,b1);
        System.out.println("*2-------");
        b1.m(b1);
        b1.n(b1);
        b1.m(b1,b1);
        b1.n(b1,b1);
```

```
System.out.println("*3-------");
b1.m(a1);
b1.n(a1);
b1.m(a1,a1);
b1.n(b1,b1);
System.out.println("*4-------");
a2.m(a1);
a2.n(b1);
a2.m(a1,a1);
a2.n(b1,b1);
System.out.println("*5-------");
a2.m(a2);
a2.n((B)a2);
a2.m(a2,a2);
a2.n((B)a2,(B)a2);
```

```
        System.out.println("*6-------");
        b1.m(a1);
        b1.n(b1);
        b1.m(b1,b1);
        b1.n(b1,b1);

    }
}
```

6. sortie du troisime exemple

```
*1-------
1
2
3
4
*2-------
```

```
5
6
7
8
*3-------
1
6
3
8
*4-------
1
2
3
4
*5-------
1
```

```
2
3
4
*6-------
1
6
7
8
```

7. cas d'ambiguité

```
class A {
    int m (A x) { System.out.println(1+" "); return (1);}
    int m (A x, A y) {System.out.println(3+" "); return 3;}
}
```

```
class B extends A {
    int m (B x) { System.out.println(5+" "); return (5);}
    int m (A x, B y) {System.out.println(7+"bis  "); return 7;}
    int m (B x, A y) {System.out.println(7+"ter  "); return 7;}
}


class ex4 {
    public static void main(String [] args) {
        A a1 = new A ();
        B b1 = new B();
        A a2 = b1;
        System.out.println("*1-------");
        a1.m(a1);
        a1.m(a1,a1);
```

```
System.out.println("*2-------");
b1.m(b1);
b1.m(b1,b1);
System.out.println("*3-------");
b1.m(a1);
b1.m(a1,a1);
System.out.println("*4-------");
a2.m(a1);
a2.m(a1,a1);
System.out.println("*5-------");
a2.m(a2);
a2.m(a2,a2);
System.out.println("*6-------");
b1.m(a1);
b1.m(b1,b1);
```

```
    }
}
```

8. sortie de la compilation

```
ex4.java:24: The call of method "m" is ambiguous 15.11
```