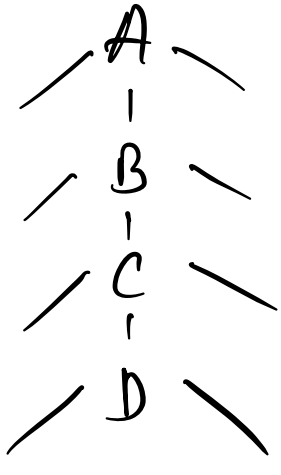
POO

Dans une classe, plusieurs méthodes peuvent porter le même nom, pour ce que leurs signatures soient distinctes signatures : la suite des types des arguments

```
void m(int x){..}
void m(int x, int y){...}
void m(B b){...}
void m(C c){...}
```

(int)
(int, int)

```
class P {
 void m(A a){...}
```




```
B b= new B();
P p= new P();
  p.m(b)
≡ A a =b, (1)
  pour(a); (2)
```

class A {}
class B extends A {...}
class C extends B {...}

1) Ca marche pas car il n'y a pas de constructeur, car new D(vide) dans y a pas

2) Class E extends C {...} marche
class F extends D {...} compile

Appel implicite / explicite de super constructeur

A
|
B

$$\frac{A(int\ x)\{...\}}{B(int\ x)\{ super(x);\ //appel\ explicite\ de\ A(x)}$$

D

D(int x)

pas de constructeur en D()

F{} vide

=> F(){} invoqué

=> E(){ super(); } invoqué

C

E()
implicite qui invoque super(vide)
≡ C()

D(int)

F()
implicite qui invoque super() ≡ D()

Ex2

```
A a = new A();
B b = new B();
A c = new B();
```

a.g() → A, f(A)

b.g() → implementation de g dans A héritée par B → f(new A())

this → Aref

f(new A()) → invocation de f(A a) → B, f(A) dans B (liaison dynamique)

+ choix de méthode en fonction de la signification des args

this → référence vers une instance de B

c.g() → même chose avec b.g()

a.f(a);
a.f(b);
a.f(c);
} f(A a), conversion en A ref de b

A a = new A();
B b = new B();
A c = new B();

b.f(a)
b.f(b)
c.f(c) A-ref

B, f(A a) (réimplémenté)
B, f(B b) (surcharge)
B, f(A a) (réimplémenté)

c.f(a) B, f(A a), par liaison dynamique

c.f(b) ——————

c.f(c) ——————

↳ c'est une A ref et on va voir que le nom f(A a) et pas f(B b)

a.f(a);
a.f(b);
a.f(c);

} f(A a), avec conversion en A ref de b

↳ B ref, convertie en A-ref

Ex 3

(1) Plante

Plante() { nbPlanteSoif ++ }

Plante Fleurie

(idem)

(2) dans Plante Fleurie

invoquer super.toString()

et compléter

(3) par retour de String

même idée, état statique

super.etat() dans PlanteFleurie

(4) static void arrosage (Plante p)

PlanteFleurie

---

static void arrosage (PlanteFleurie pf)

serre[i] ≡ Plante-réf

=> PlanteFleurie.arrosage (serre[i])

invoque l'implementation de

Plante (Plante) hérité

Dans Plante

static void arrosage (Plante p)

dans PlanteFleurie

static void arrosage (Plante p)

par héritage

+ (PlanteFleurie pf)

Plante.arrosage(Plante p)

≡ PlanteFleurie.arrosage(Plante p)

Ex 4

Batiment String adresse double surface_habitable	get surface_jardin ↪ 0
	get surface_habitable
Maison int nbPieces double surface_jardin	get surface_jardin (redef → this.surface_jardin)

dans Main

10 × null

```
Batiment[] tab = new Batiment[10]

Maison m1 = new Maison(...);

Maison m2 = new Maison(...);

tab[3] = m1;
tab[5] = m2;
for(Batiment b: tab){
        System.out.println(b);
}
```

// null saut en 3, 5 avec appel de toString()

Personne

Personne getLocataire(){
return locataire
}

Personne proprietaire

boolean louable(){return false}

Maison

boolean louable;

Maison(... boolean louable){
...this.louable = louable
}

boolean getLouable(){
return louable;
}