SCIENCES SORBONNE UNIVERSITÉ

2I002
CAS COMPLEXES DE REDÉFINITION DE MÉTHODES

Vincent Guigue

LIP6

MLIA Machine Learning & Deep Learning for Information Access

Compilation:

1. Le **compilateur** vérifie si les méthodes existent en fonction des **types de variables**

```
1 Figure f = new Carre(...);
2 // je peux appeler f.fonctionDefinieDansFigure()
3 // je peux appeler f.fonctionReDefinieDansCarre()
4 // je NE peux PAS appeler f.fonctionDefinieDansCarre()
```

2. Le **compilateur** présélectionne des méthodes

```
1
2 // Dans figure:
3 public void maMethode(C c)
4 // Dans carre (extends Figure)
5 public void maMethode(D d) // D extends C
6
7 D d = new D();
8 Figure f = new Carre(...);
9 f.maMethode(d); // Le compilateur retient la signature:
10                 //   void maMethode(C)
```

Execution:

3. La **JVM** regarde l'instance de l'**objet qui invoque la méthode**

```
1  // Dans A:
2  public void maMethode(C c)
3  // Dans B extends A
4  public void maMethode(C c)
5  // main
6  A ab = new B(); // subsomption
7  B b = new B();
8
9  ab.maMethode(c);
10 b.maMethode(c); // => meme resultat
```

4. La **JVM** recherche la fonction la plus appropriée dans toutes celles disponibles sur l'objet (avec héritage)

```
1  // Dans A:
2  public void maMethode(C c) //1
3  // Dans B extends A
4  public void maMethode(D d) //2
5  // main
6  C c = new C()
7  A ab = new B(); // subsomption
8  B b = new B();
9
10 ab.maMethode(c); // selection de 1
11 b.maMethode(c); // selection de 1
```

5. La **JVM** ne regarde pas les instances des **arguments**

```
12 C cd = new D(); // avec D extends C
13 ab.maMethode(cd); // selection de 1
14 b.maMethode(cd); // selection de 1
```

$\Rightarrow$ **Toutes les difficultés viennent des combinaisons de règles**

- Classe A:
  - `public void maMethode(C obj)`
- Classe B extends A:
  - **CAS 1 :** `public void maMethode(C obj)`
  - **CAS 2 :** `public void maMethode(D obj)`

(avec D extends C)

CAS 1: pas d'ambiguité, la JVM décidera en fonction des instances

CAS 2: pas simple du tout...

Exemple de redéfinition

La redéfinition a été vue dans le cours précédent... Mais il existe des cas particulièrement complexe!

Au début, un cas d'école:

- Classe Mère A:

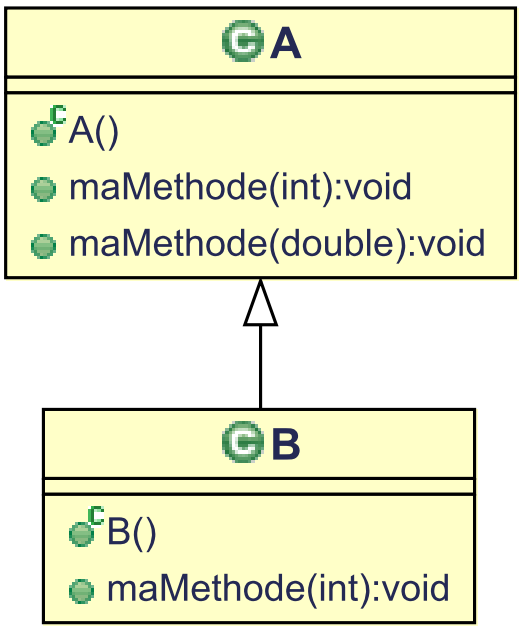
```
1 void maMethode(int i) // 1
```

- Classe Fille B (extends A):

```
1 void maMethode(int i) // 2
```

```
1 A var1 = new A();
2 A var2 = new B(); // subsomption
3 var1.maMethode(i); // 1
4 var2.maMethode(i); // 2 -> la JVM regarde les instances
```

Sélection des méthodes en contexte ambigu

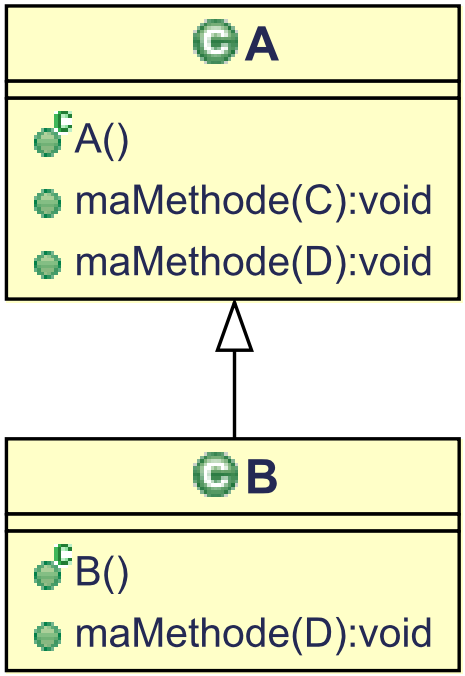


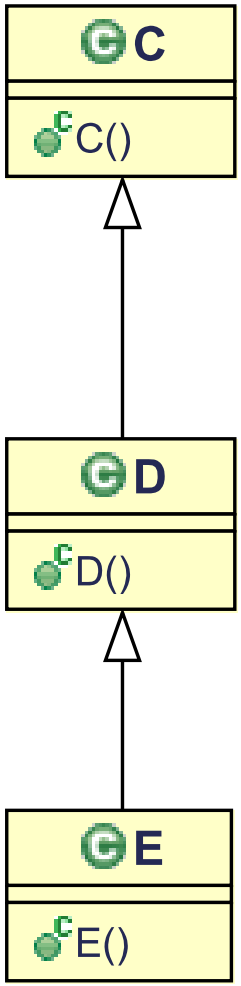
De haut en bas, les méthodes sont numérotées 1, 2 et 3.

```
1  int i = 7;
2  double d = 7.0;
3  A a = new A();
4  B b = new B();
5  A b2 = new B();
6
7  a.maMethode(d);
8  b.maMethode(d);
9  b2.maMethode(d);
10
11 a.maMethode(i);
12 b.maMethode(i);
13 b2.maMethode(i);
```

Quels sont les méthodes utilisées?

Suite... Avec des arguments objets

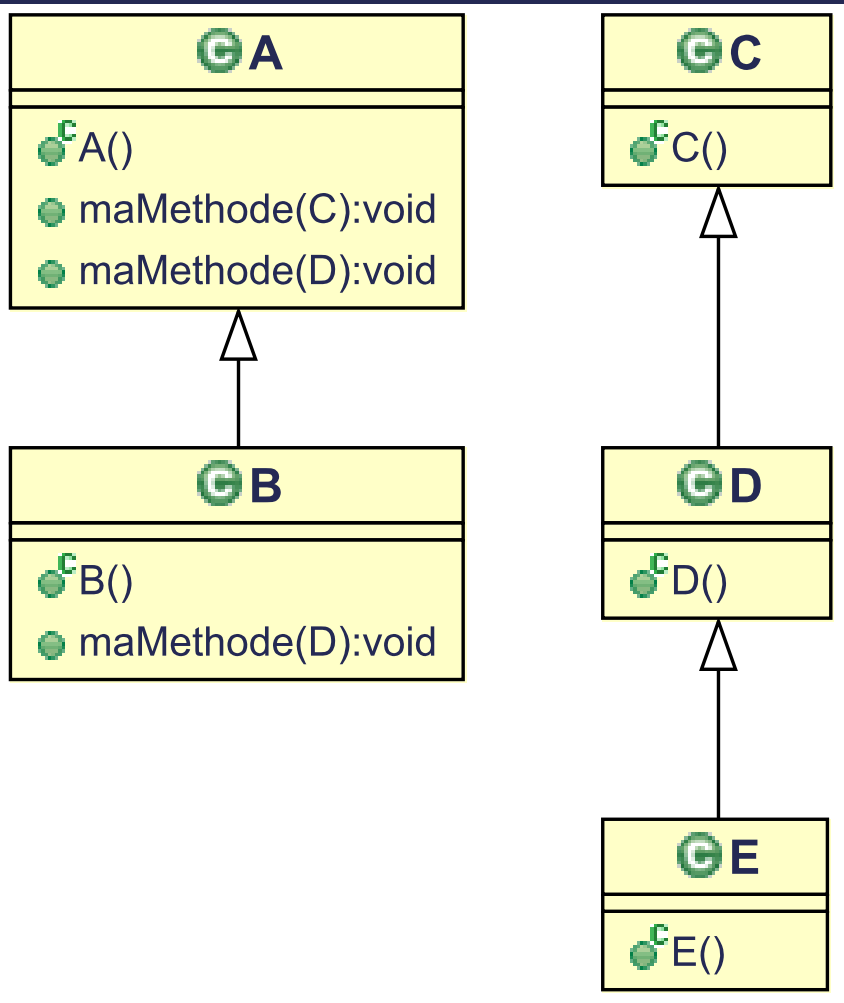





```
1  C c = new C();
2  D d = new D();
3  E e = new E();
4
5  A a = new A();
6  B b = new B();
7  A b2 = new B();
8
9  a.maMethode(e);
10 b.maMethode(e);
11 b2.maMethode(e);
```

- Compilation?
- Méthodes utilisées ?
- Si on change e → c, d

Suite... Avec des arguments objets (2)



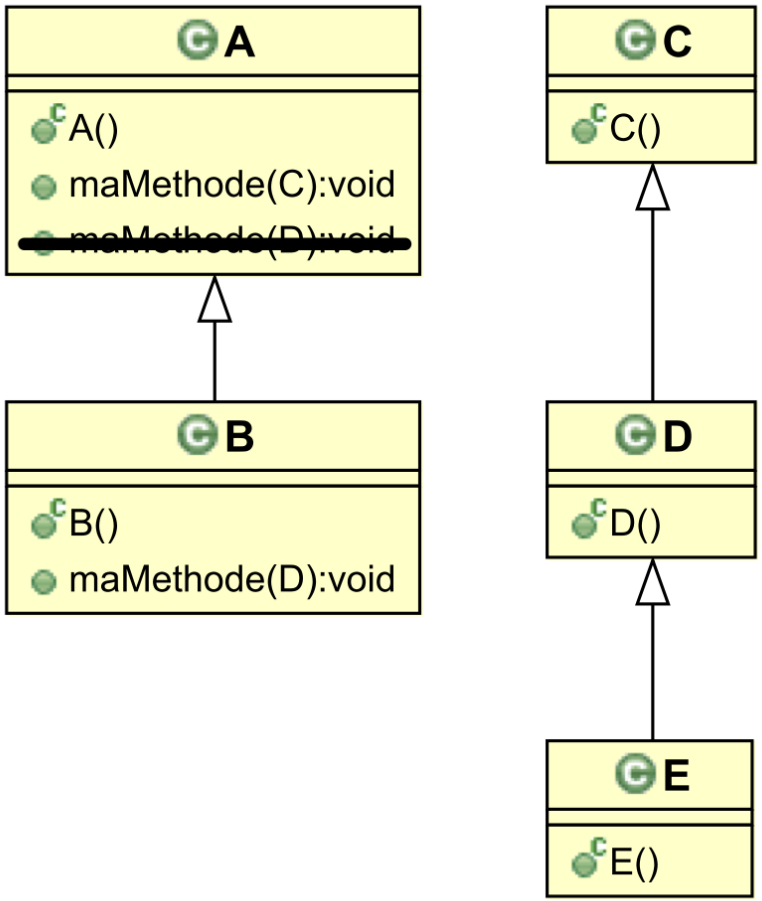

```
1 C cd = new D(); // subsomption
2
3 A a = new A();
4 B b = new B();
5 A b2 = new B();
6
7 a.maMethode(cd);
8 b.maMethode(cd);
9 b2.maMethode(cd);
```

- Compilation?
- Méthodes utilisées ?

ATTENTION: la JVM traite différemment

(i) l'instance qui invoque
(ii) les arguments

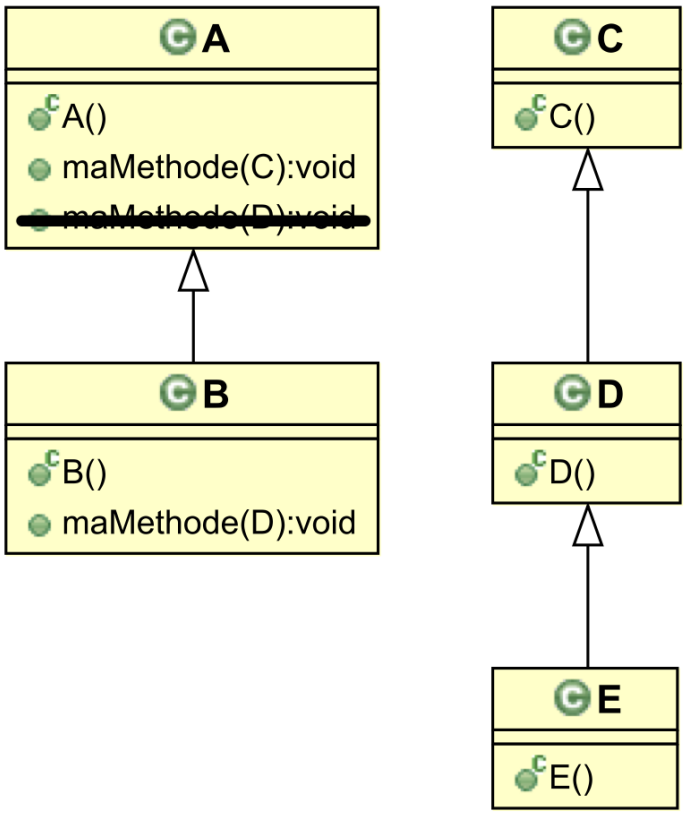
Le cas très complexe




```
1 D d = new D();
2
3 A a = new A();
4 B b = new B();
5 A b2 = new B();
6
7 a.maMethode(d);
8 b.maMethode(d);
9 b2.maMethode(d);
```

- Compilation?
- Méthodes utilisées ?

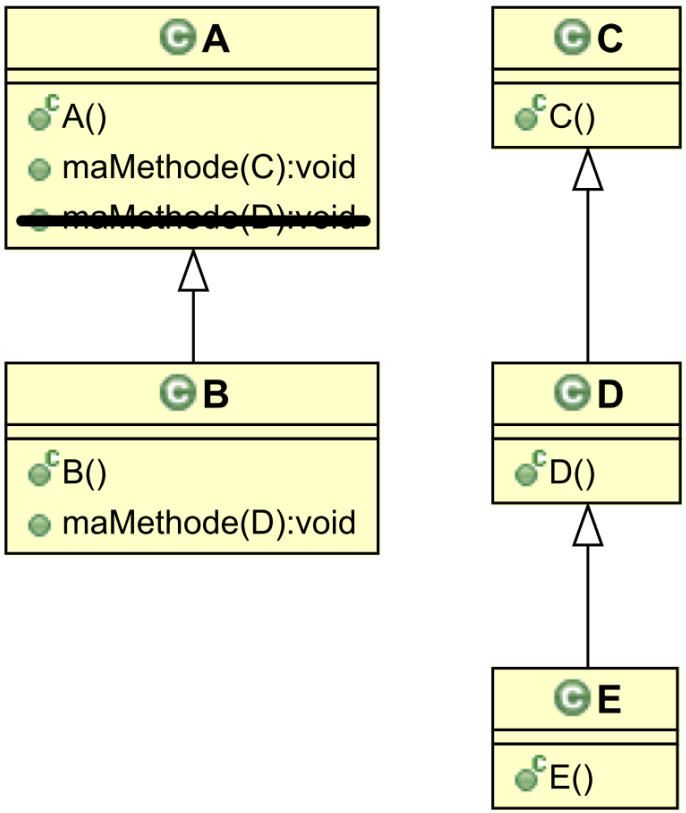
... Si on ne fait pas attention, oui.



- Evolution d'un programme = ajout de classes
- Nouvel item = `B`
- Nouvel outil périphérique = `D`

Sur un exemple...

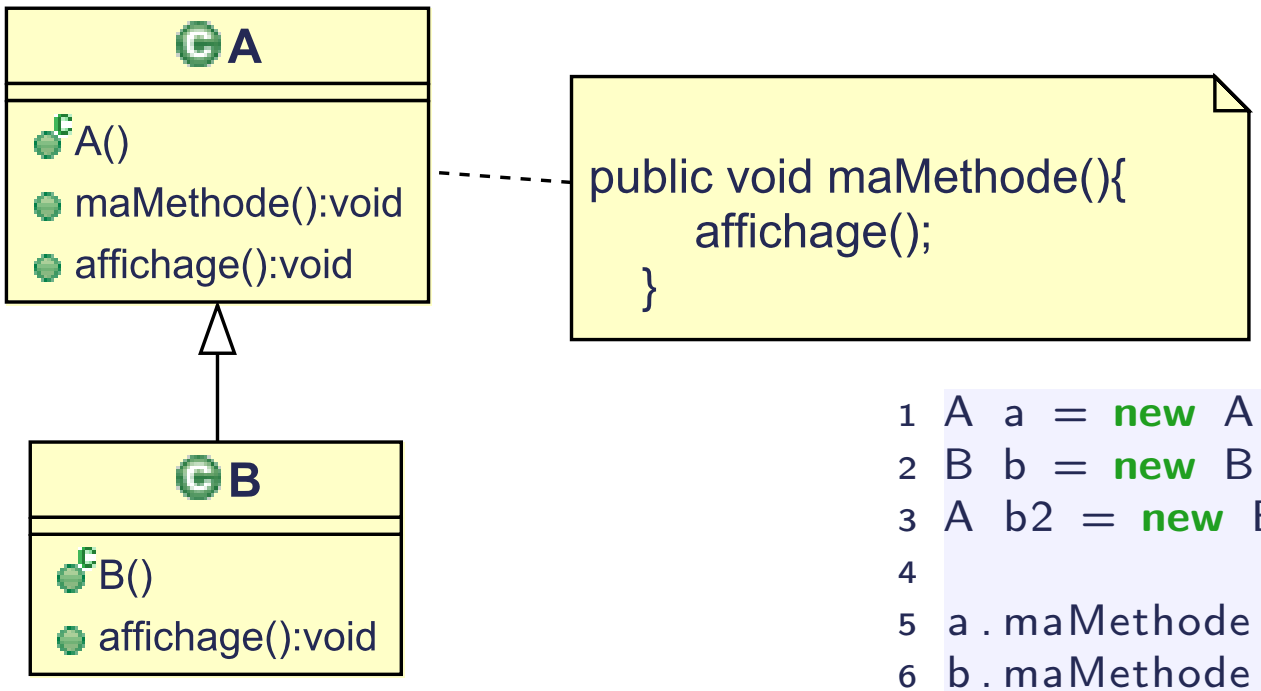
... Si on ne fait pas attention, oui.



Les *Sims*

- Classe `Personnage` existante (déplacements, fcts de base...)
- Nouvelles classes `AnimalDomestique`, `Chien`...
- $\Rightarrow$ pour la nourriture, définition de nouvelle classe `Os`, `Croquette`

Dynamicité dans la sélection des méthodes




```
1 A a = new A();
2 B b = new B();
3 A b2 = new B();
4
5 a.maMethode();
6 b.maMethode();
7 b2.maMethode();
```

- Affichage ?

Le cas de `equals` avec le Polymorphisme

```
1  public class Point {
2      private double x,y;
3  ...
4      public boolean equals(Point p){ // Version simplifiee
5          return x == p.x && y==p.y;
6      }
7  }
```

Rappel: `boolean equals(Object)` existe dans `Object`

Premier test:

```
1      public static void main(String[] args) {
2          Point p = new Point(1,2);
3          Point p2 = new Point(1,2);
4          Point p3 = p;
5
6          System.out.println(p == p2);
7          System.out.println(p == p3);
8          System.out.println();
9          System.out.println(p.equals(p2));
10         System.out.println(p.equals(p3));
11     }
```

equals, test 2

```
1  // dans Object : public boolean equals(Object p), egal. ref.
2  // dans Point : public boolean equals(Point p)
3      public static void main(String[] args) {
4          Point p = new Point(1,2);
5          Point p2 = new Point(1,2);
6
7          Object p4 = p; // subsomption
8
9          System.out.println(p.equals(p4));
10         System.out.println(p2.equals(p4));
11         System.out.println();
12         System.out.println(p4.equals(p));
13         System.out.println(p4.equals(p2));
14
15     }
```

Forme standard de equals

Quand on peut éviter les cas difficile, on en profite!!

```java
// Dans Point:

public boolean equals(Object obj) {
        if (this == obj)
          return true;
        if (obj == null)
          return false;
        if (getClass() !=
                obj.getClass())
          return false;
        Point other = (Point) obj;
        if (x!=other.x)
          return false;
        if (y!=other.y)
          return false;
        return true;
}
```

```java

// main
Point p = new Point(1,2);
Point p2 = new Point(1,2);

Object p4 = p;

p.equals(p4);
p2.equals(p4);

p4.equals(p);
p4.equals(p2);

}
```