Université de Biskra
Département d'Informatique

Relations entre classes (exposé 3)

Héritage

Vidéos 20-21-22

L2-POO

Pr. Laid Kahloul

2019-2020

Plan

- Héritage dans la nature et dans le monde réel
- Pourquoi l'héritage
- C'est quoi l'héritage
- Types d'héritages
- Héritage dans la POO
- Héritage Simple en C++: modificateurs d'héritage, modificateurs de visibilité, …
- Héritage Multiple en C++

Héritage dans la nature

héritage

- **Être vivant**:

Propriété: **âge, type**, …

Actions: **se nourrir, respirer**, …

héritage

- **Un Humain**: **est un Être vivant**:

Propriété: **âge, type**, …, **Taille, Couleur, Poids**, …

Actions: **se nourrir, respirer**, …, **dormir, marcher**, …

- **Un étudiant**: **est un Humain**:

Propriété: **âge, type, …, Taille, Couleur, Poids**, **…, Numéro d'insc, Année, Groupe, Spécialité**, …

Actions: **se nourrir, respirer, …, dormir, marcher**, … **, étudier, s'examiner, préparer cours**, …

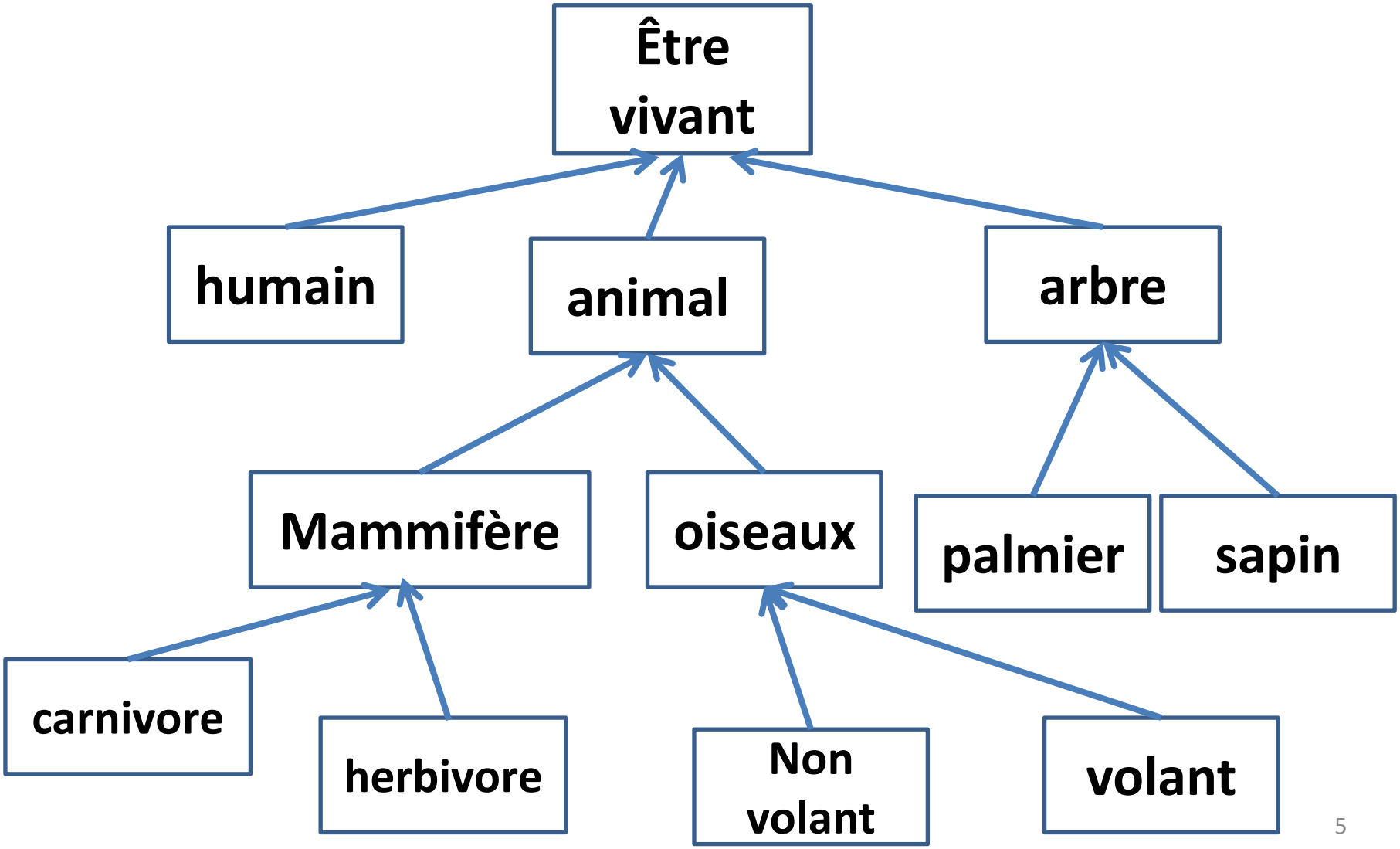
Pourquoi l'héritage? (1)

- Une relation qui **existe dans le monde réel**, et la nature

- Une relation qui **permet la richesse** du monde réel et de la nature

- Une richesse par **la dérivation de nouvelle** races (types) d'autres races (types)

Pourquoi l'héritage? (2)



C’est quoi l’héritage? (1)

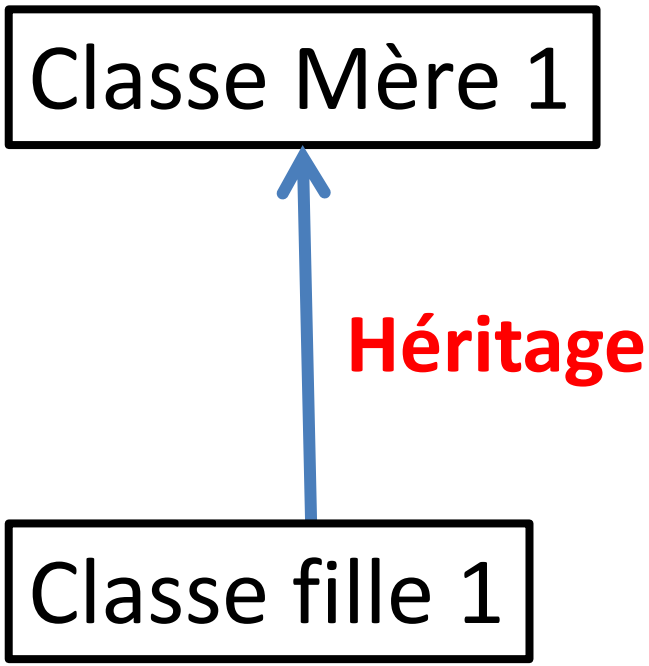
- La relation **la plus importante** entre classes dans la POO.
- Elle est la **source de la puissance** de la POO.
- Elle est **la base de l’idée de la réutilisation** représentant le point fort dans la POO

C’est quoi l’héritage? (2)

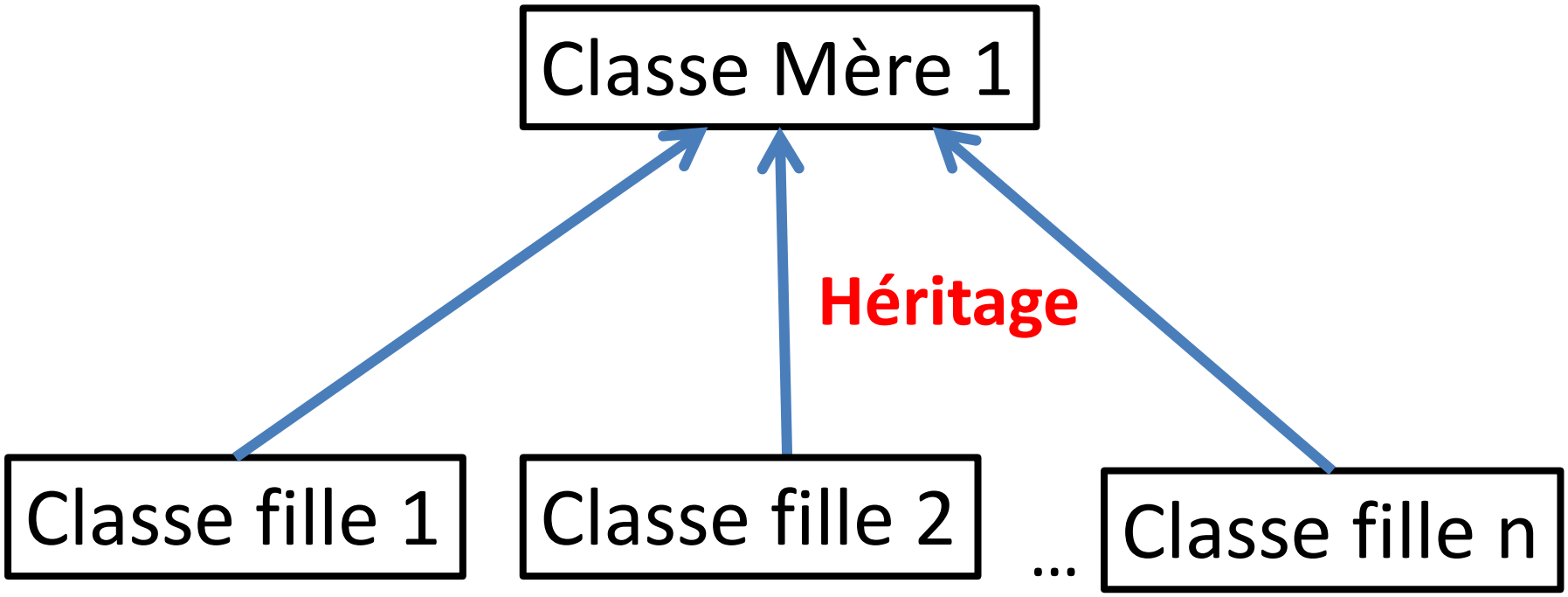
- Une relation de **dérivation**.

- Elle permet dériver **des classes descendantes** d’autre classes qu’on nomme **classes mères ou classes ancêtres**.

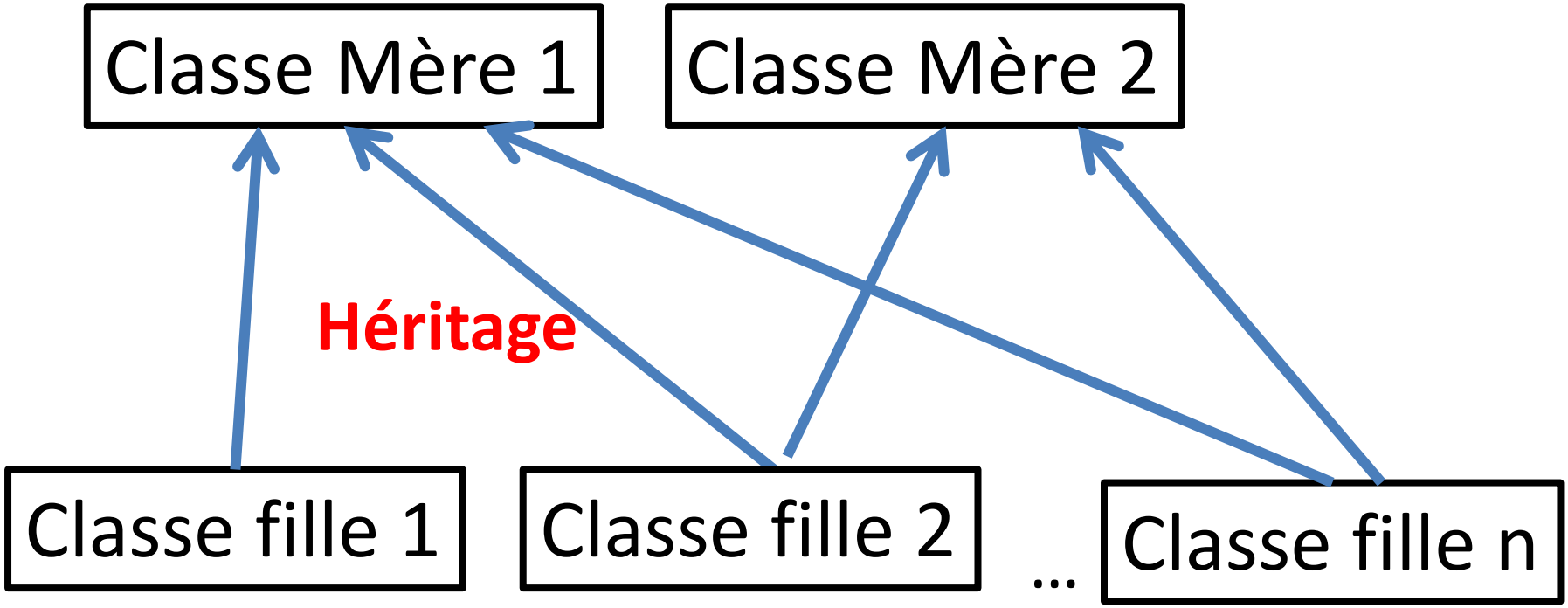
C’est quoi l’héritage? (3)



C’est quoi l’héritage? (4)



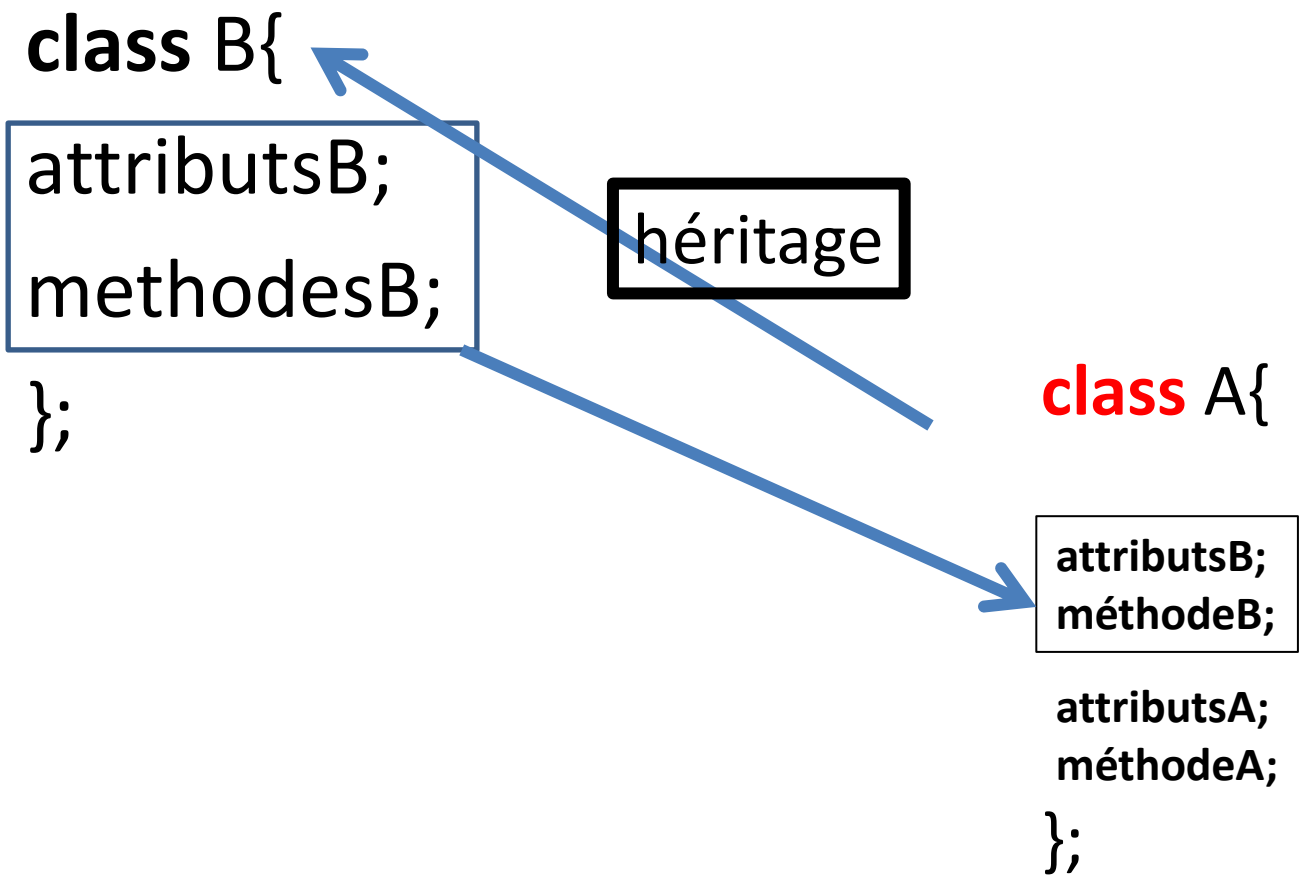
C’est quoi l’héritage? (5)



C’est quoi l’héritage? (6)

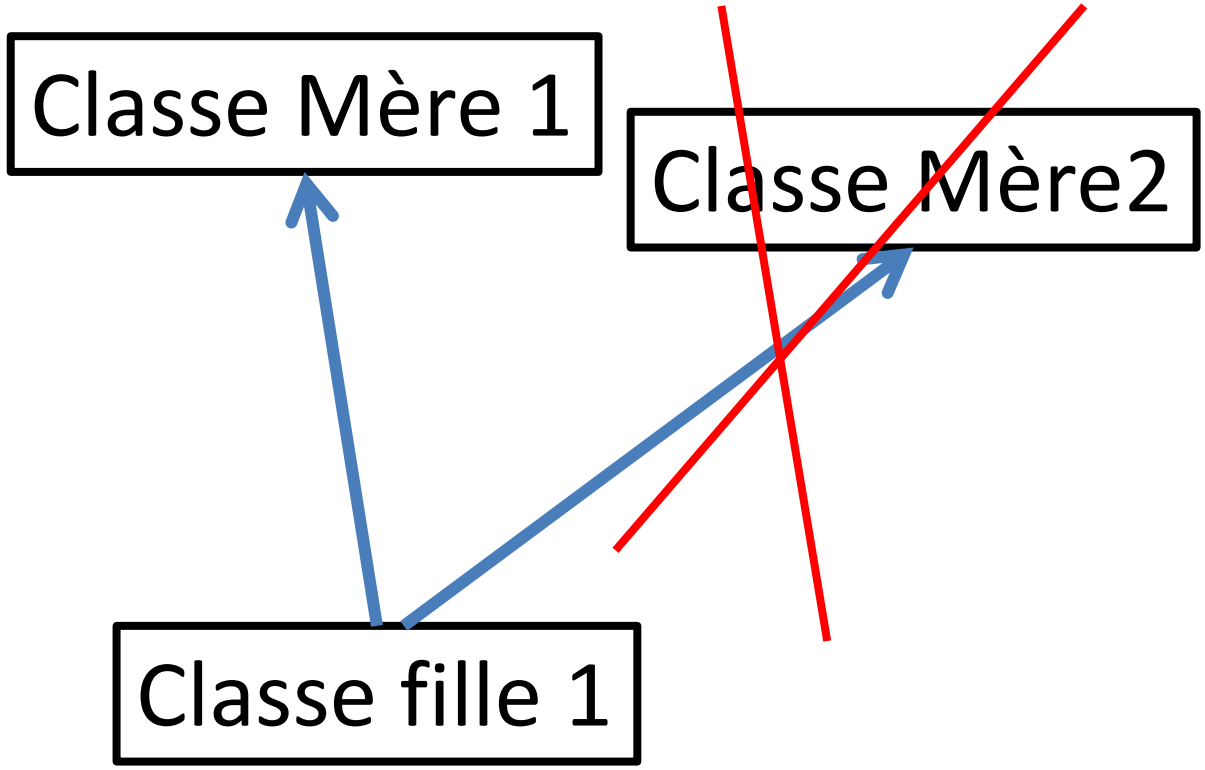
- Une relation selon la quelle la classe fille **A** va hériter tous les attributs, les méthodes ainsi que leurs visibilités depuis sa classe mère **B**
- Les attributs de **B** seront des attributs de **A**
- Les méthodes de **B** seront des méthodes de **A**
- Les visibilités des attributs et des méthodes de **B** sera aussi hérité dans **A**
- **A** peut avoir d’autres attributs et méthodes (enrichissement) que **B** n’a pas

Exemple abstrait



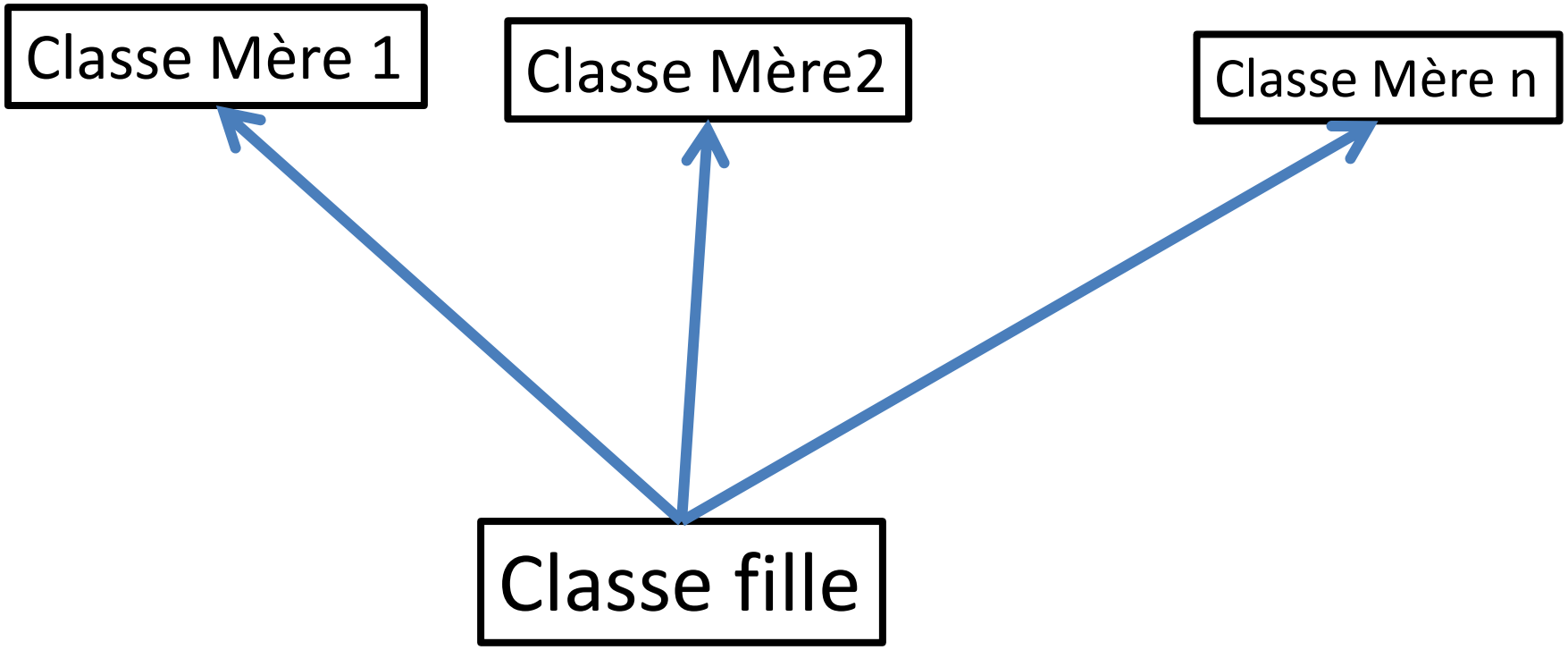
Types d'héritages (1)
Simple

- Une classe fille peut hériter une seule classe mère.



Types d'héritages (2)
Multiple

- Une classe fille peut hériter de plusieurs classes mères



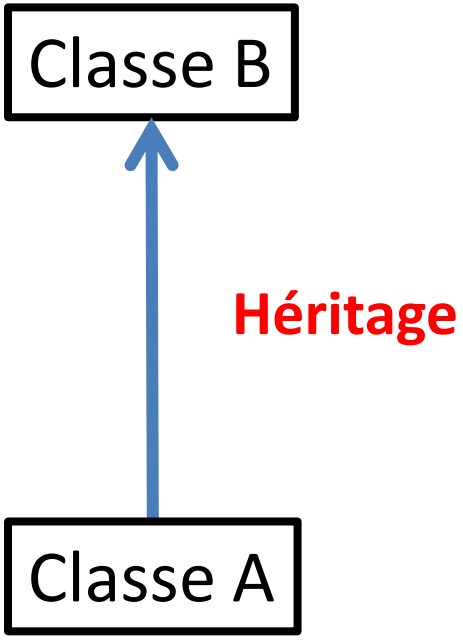
Héritage: Programmation

- Tout langages de POO doit avoir ce mécanisme

- Il y'a des langages qui permet **les deux types d'héritages**: comme le **C++**

- Il y'a des langages qui permet seulement **l'héritage simples**: **Java**

Héritage en C++ (1): simple



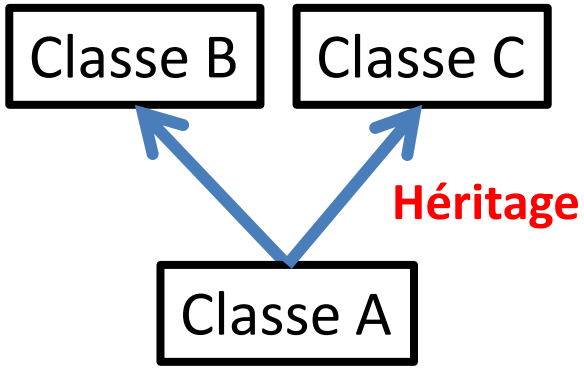

```
class B{ // B est la classe mère
};
class A: public B{ // A est la classe fille
};
```

Héritage en C++ (2):
modificateur d'héritage

- Le mot clé **public** avant le nom de **la classe mère B** permet à la **classe fille A** d'**hériter les mêmes visibilité** de la classe B

- Ce mot clé est dit: **modificateur d'héritage** (comme le cas des **modificateur de visibilité** déjà vu dans le cours de visibilité)

Héritage en C++ (3): multiple




```
class B{ // B est la classe mère 1
};
class C{ // C est la classe mère 2
};

class A: public B, public C{ // A est la classe fille
};
```

Héritage Simple en C++

Exemple complet en C++ (1)

```
class B{
        int x;
public:
        B();
        ~B();
        void setX(int x);
        int getX();
};

B::B(){
cout<<"objet B créé"<<endl;
}
B::~B(){
        cout<<"objet B
détruit"<<endl;
}
void B::setX(int x){
        this->x=x;
}
int B::getX(){
        return x;
}
```

```
class A:public B{
        int y;
public:
        A();
        ~A();
        void setY(int y);
        int getY();
};
A::A(){
y=0;
cout<<"objet A créé"<<endl;
}
A::~A(){
        cout<<"objet A
détruit"<<endl;
}
void A::setY(int y){
        this->y=y;
}
int A::getY(){
        return y;
}
```

Exemple complet en C++ (2)

```
int main() {
        A a;
        a.setX(5);
        a.setY(4);
        cout<<"a.y="<<a.getY()<<endl;
        cout<<"a.x="<<a.getX()<<endl;
        return 0;
}
```

Question: Quel résultat on va avoir??????

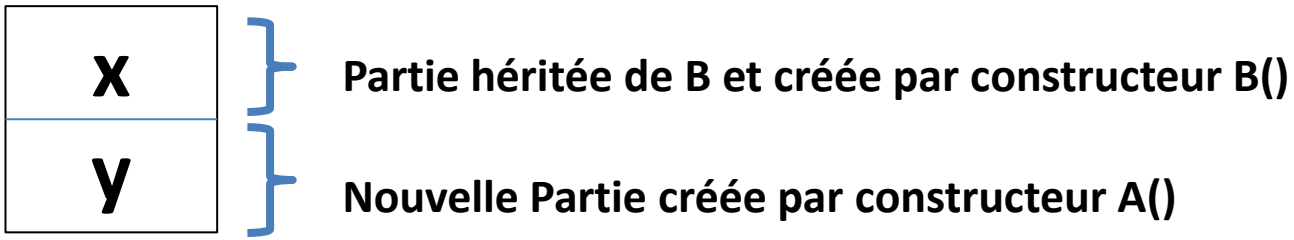
Exemple complet en C++ (3)

```
objet B créé
objet A créé
a.y=4
a.x=5
objet A détruit
objet B détruit
```

Héritage en C++ (4)

- Lorsqu'une **instance de la classe fille est créé**, automatiquement les **deux constructeurs des deux classes mère et fille sont exécutés**.
- Exemple:

Objet a de A



Héritage en C++ (5)

Constructeur non implicite dans la classe mère

```
class B{
        int x;
public:
        B(int x);
        ~B();
        void setX(int x);
        int getX();
};

B::B(){
this->x=x;
cout<<"objet B créé"<<endl;
}
B::~B(){
cout<<"objet B détruit"<<endl;
}
void B::setX(int x){
        this->x=x;
}
int B::getX(){
        return x;
}
```

```
class A:public B{
        int y;
public:
        A();
        ~A();
        void setY(int y);
        int getY();
};
A::A(){
y=0;
cout<<"objet A créé"<<endl;
}
A::~A(){
        cout<<"objet A
détruit"<<endl;
}
void A::setY(int y){
        this->y=y;
}
int A::getY(){
        return y;
}
```

Héritage en C++ (5)

Constructeur non implicite dans la classe mère

- Dans l'exemple précédent, lors de la création d'un objet a de A, **l'appel du constructeur de la classe B échoue** car le constructeur implicite B() n'est plus accessible => **il faut appeler le constructeur B(int x).**

- Ceci doit être fait dans la définition du constructeur A(), comme suit:

```
A::A():B(Valeur à affecter à x){
y=0;
cout<<"objet A créé"<<endl;
}
```

Héritage en C++ (5)

Constructeur non implicite dans la classe mère

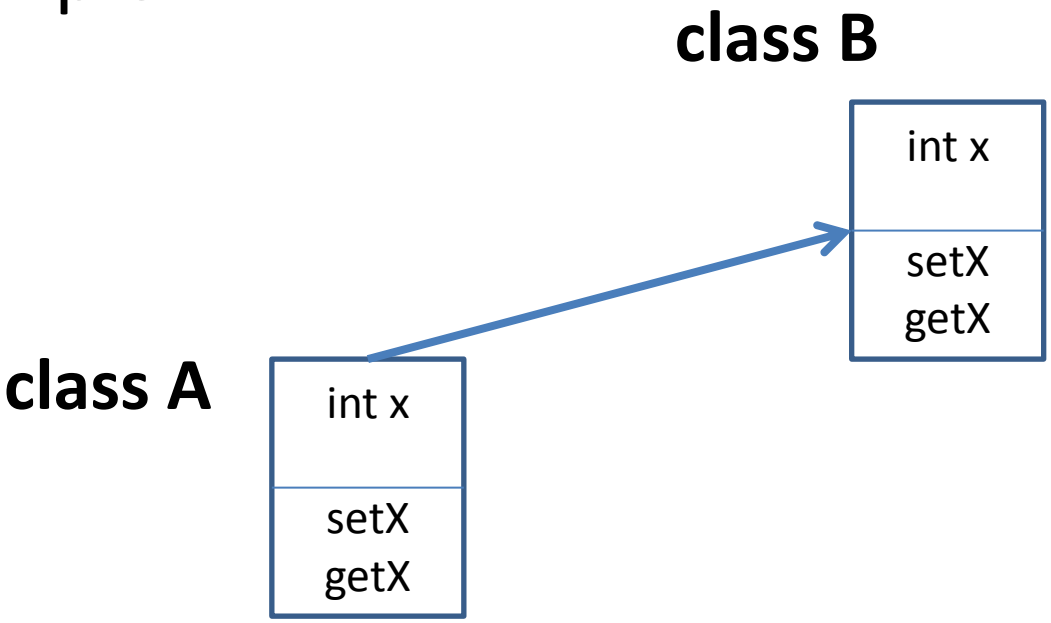
- Exemple:

```
A::A():B(3){
y=0;
cout<<"objet A créé"<<endl;
}
```

- Comme ça, quand a est créé le constructeur A fait appel au constructeur explicite et la valeur de x sera initialisé à 3

Masquage d'attributs et des méthodes (1)

- Les **noms des attributs et méthodes** de A **masquent** les attributs et méthodes hérités depuis B, **ayant les mêmes noms**
- Exemple:



Masquage d'attributs et des méthodes (2)

```
//**********************
class A: public B{
         int x;
public:
         A();
         ~A();
         void setX(int x);
         int getX();
};
A::A():B(3){
//x=0;
y=0;
cout<<"objet A créé"<<endl;
}
A::~A(){
cout<<"objet A détruit"<<endl;
}
void A::setX(int x){
         this->x=x;
}
int A::getX(){
         return x;
}
```

```
int main() {
  A a;
  a.setX(4);

  cout<<"a.x="<<a.getX()<<endl;
return 0;
}
```

Et pour le x hérité de B????

Masquage d'attributs et des méthodes (3)

- L'attributs et méthodes hérité depuis B sont toujours accessibles, mais on doit spécifier le nom de la classe B
- Exemple:

```
int main() {
  A a;
  a.B::setX(3);
  cout<<"a.xB="<<a.B::getXB()<<endl;
return 0;
}
```

Retour sur la visibilité (1)

- Les attributs et méthodes **privés** de la classe mère sont **accessibles seulement dans la classe mère**. Même ses classes filles ne peuvent pas les voir.
- Exemple:

```
class B{
int x;
};
class A: public B{
int y; // normalement x est aussi attribut de A par héritage
int getX(); // donc normalement, on peut consulter x
};
int A::getX(){
return x; // normalement on peut avoir accès à x ici
}
```

Mais ça marche pas, car x est privée dans la classe mère

Retour sur la visibilité (2):
modificateur ***protected***

- Affin de rendre les attributs et méthodes privées dans la classe mère **accessibles** dans ses classes filles=> on remplace le modificateur ***private*** par le modificateur ***protected***.

```
class B{
protected:
        int x;
};
```

Retour sur la visibilité (3)

	Attribut de la classe mère Public	Attribut de la classe mère Private	Attribut de la classe mère Protected
Visible dans la classe mère	**OUI**	**OUI**	**OUI**
Visible dans la classe fille	**OUI**	**NON**	**OUI**
Visible dans les autres classes	**OUI**	**NON**	**NON**

Modificateur d'héritage (1)

- Que se passe-t-il si on utilise le modificateur d'héritage **private** ou **protected** à la place du modificateur d'héritage public?

```
class A: private B{
//…
};

class A: ptotected B{
//…
};
```

Modificateur d'héritage (2)

	Attribut de la classe mère Public	Attribut de la classe mère Private	Attribut de la classe mère Protected
Modificateur d'héritage public	public dans la classe fille	Non accessible dans la classe fille	protected dans la classe fille
Modificateur d'héritage private	private dans la classe fille	Non accessible dans la classe fille	private dans la classe fille

Problème avec l'héritage simple

- L'héritage simple ne permet pas d'hériter de plus d'une classe, donc c'est **un mécanisme restreint pour la réutilisation**

- Une classe fille **ne peut pas hériter les attributs et méthodes qui parviennent de différentes classes mère**

À suivre …

Héritage multiple en C++

C’est quoi?

- Une classe fille A peut hériter de plusieurs classes mères: B, C, D, …

- De la même manière que dans l’héritage simple, cette fois la classe A va avoir les attributs et méthodes des classes B, C, D, …

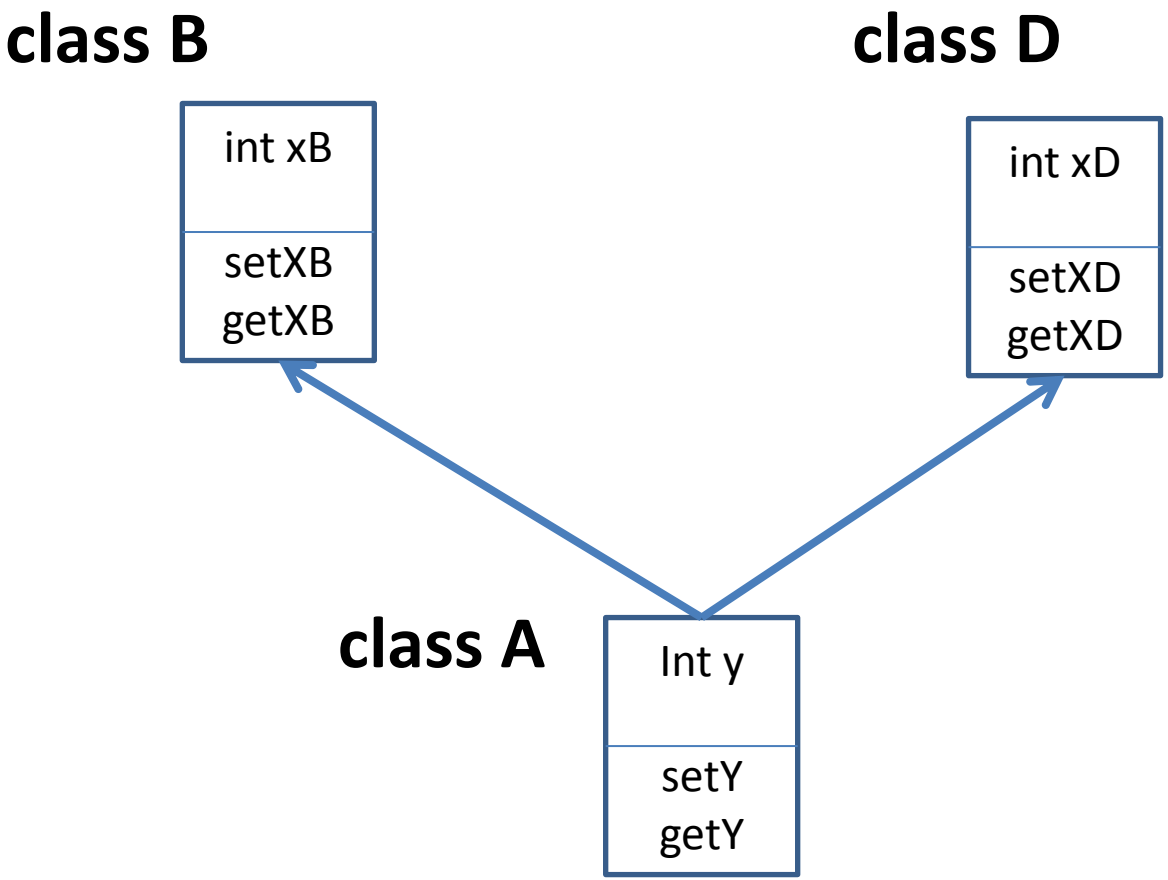
Syntaxe

```
class B{
        //…
};

class D{
//…
};

class A:  public B, public D  {
        //…
};
```

Exemple complet (1)



Exemple complet (2)

```
class B{
        int xB;
public:
        B(int xB);
        ~B();
        void setXB(int xB);
        int getXB();
};
B::B(int xB){
this->xB=xB;
cout<<"objet B créé"<<endl;
}
B::~B(){
cout<<"objet B détruit"<<endl;
}
void B::setXB(int xB){
        this->xB=xB;
}
int B::getXB(){
        return xB;
}
//*********************
```

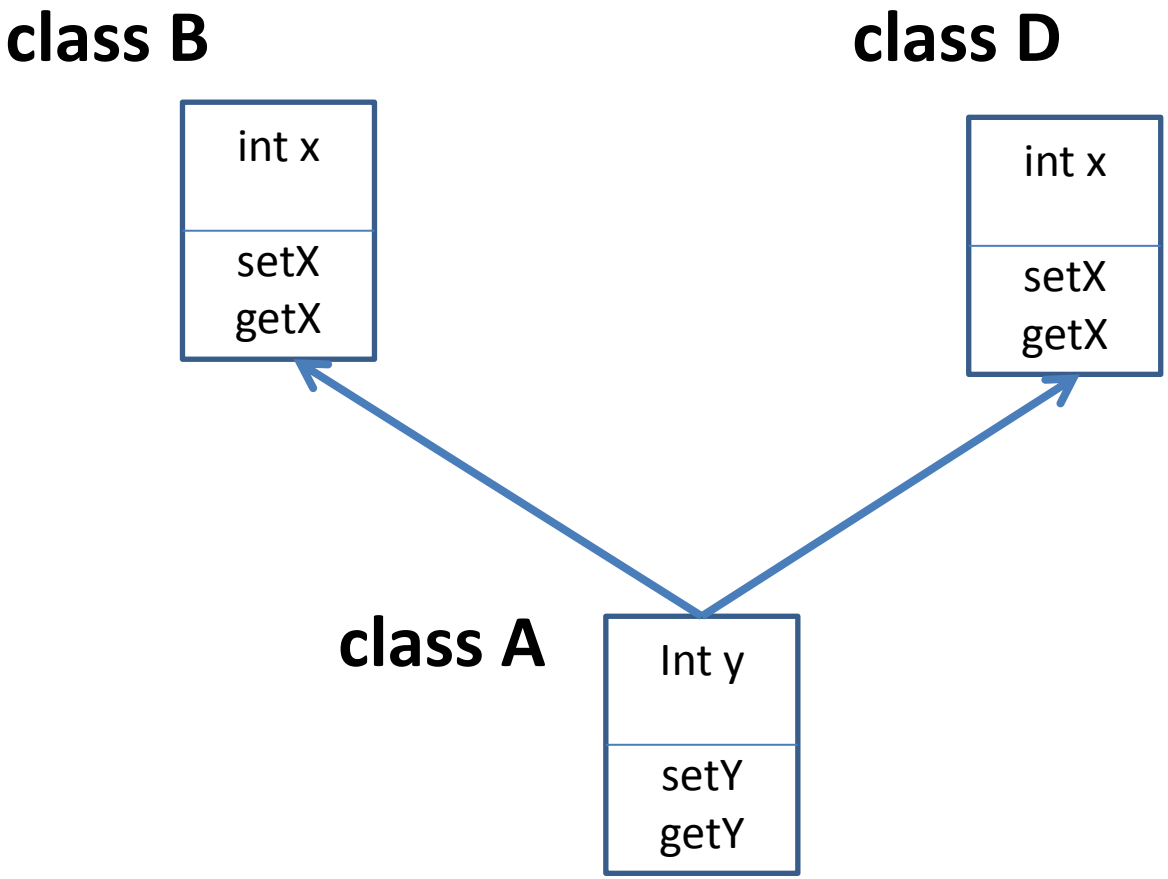
```
//*********************
class D{
        int xD;
public:
        D(int xD);
        ~D();
        void setXD(int xD);
        int getXD();
};
D::D(int xD){
this->xD=xD;
cout<<"objet D créé"<<endl;
}
D::~D(){
cout<<"objet D détruit"<<endl;
}
void D::setXD(int xD){
        this->xD=xD;
}
int D::getXD(){
        return xD;
}
```

Exemple complet (3)

```
//*********************
class A: public B, public D {
        int y;
public:
        A();
        ~A();
        void setY(int y);
        int getY();
};
A::A():B(3), D(2){
//x=0;
y=0;
cout<<"objet A créé"<<endl;
}
A::~A(){
        cout<<"objet A détruit"<<endl;
}
void A::setY(int y){
        this->y=y;
}
int A::getY(){
        return y;
}
```

```
int main() {
  A a;
  a.setY(4);
  cout<<"a.y="<<a.getY()<<endl;
  cout<<"a.xB="<<a.getXB()<<endl;
  cout<<"a.xD="<<a.getXD()<<endl;
  cout << "!!!Hello World!!!" << endl;
// prints !!!Hello World!!!
  return 0;
}
```

Problème des noms **homonymes** en héritage multiple (1)



Problème des noms **homonymes** en héritage multiple (2)

- Comment accéder à l'attribut x hérité de B dans la classe A? et Comment accéder à l'attribut x hérité de D dans la classe A?

Problème d'ambigüité

- Comment accéder à l'attribut setX héritée de B dans la classe A? et Comment accéder à l'attribut setX hérité de D dans la classe A?

Problème d'ambigüité

Problème des noms **homonymes** en héritage multiple (3)

Confusion???

```
int main() {
  A a;
  a.setY(4);
  cout<<"a.y="<<a.getY()<<endl;
  cout<<"a.x de B="<<a.getX()<<endl;
  cout<<"a.x de D="<<a.getX()<<endl;
  cout << "!!!Hello World!!!" << endl;
// prints !!!Hello World!!!
  return 0;
}
```

Problème des noms **homonymes** en héritage multiple (4): **solution**

- Désigner le nom de la classe mère lors de l'utilisation de l'attribut ou de la méthode.

```
int main() {
  A a;
  a.setY(4);
  cout<<"a.y="<<a.getY()<<endl;
  cout<<"a.x de B="<<a.B::getX()<<endl;
  cout<<"a.x de D="<<a.D::getX()<<endl;
  cout << "!!!Hello World!!!" << endl;
// prints !!!Hello World!!!
  return 0;
}
```