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??????

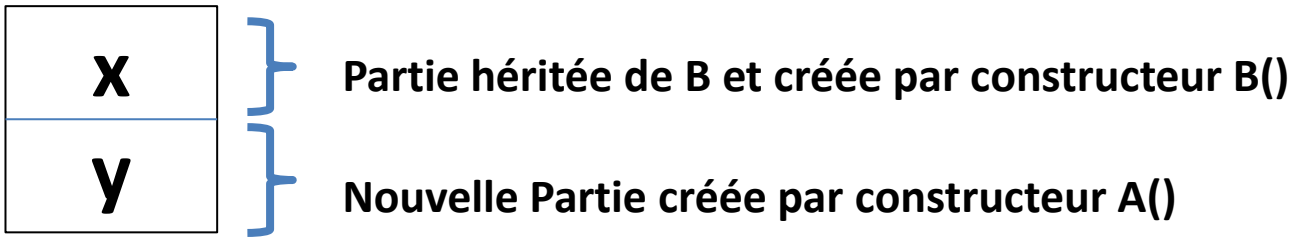
Exemple complet en C++ (3)

```
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

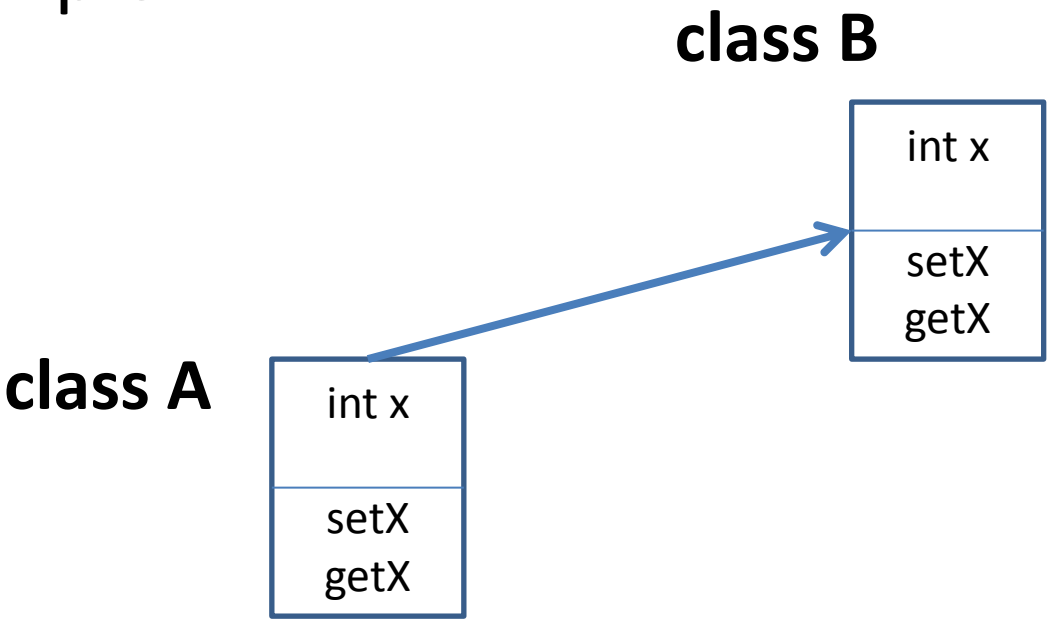
- Exemple:

```
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**