

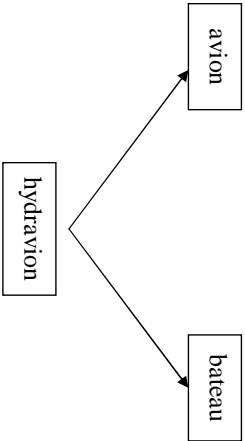
Chapitre 14

Héritage multiple et classes virtuelles

1. Généralités

Héritage simple: classe dérivée n'a qu'une classe de base.

Héritage multiple: classe dérivée a plus d'une classe de base.



Peu utilisé en pratique à cause de problèmes d'ambiguïté.

2. Syntaxe

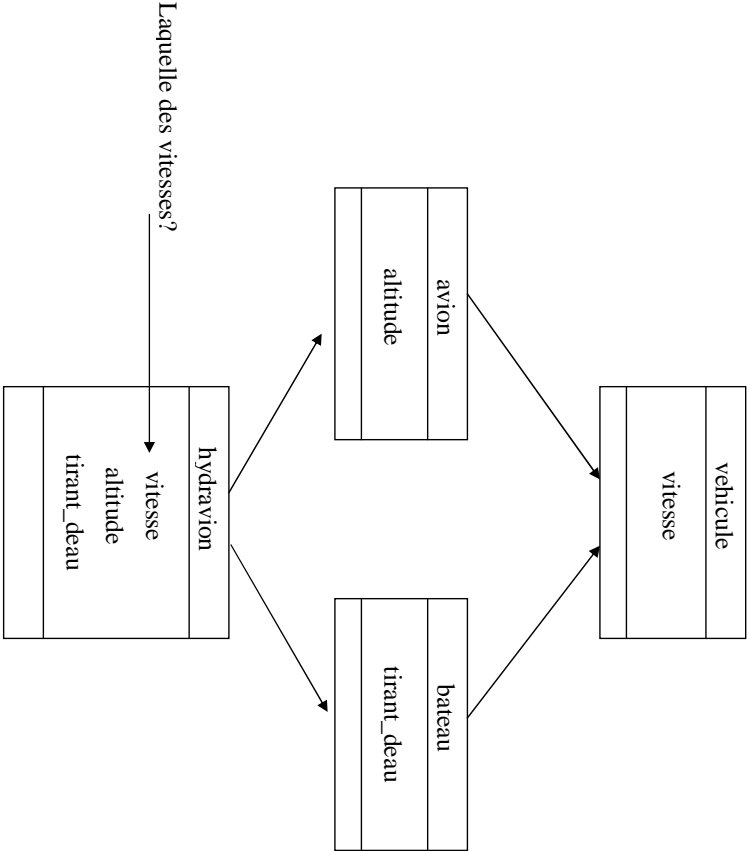
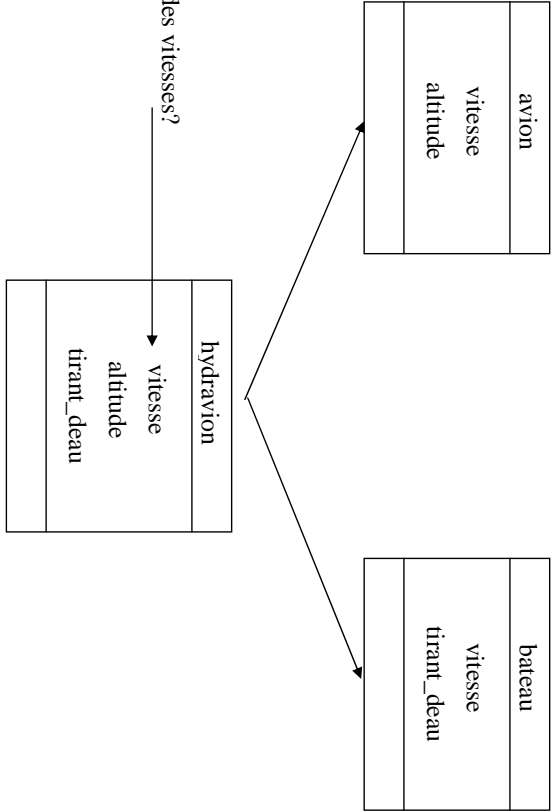
```
class classe_dérivée:protection classe_base1, protection classe_base2 {  
    /* etc. */  
};
```

Protection peut être: `public`, `protected` ou `private`.

```
#include <iostream>  
using namespace std;  
class avion {  
    double altitude;  
public:  
    avion(double a):altitude(a){}  
};  
class bateau {  
    double tirant_deau; // Calaison  
public:  
    bateau(double pm):tirant_deau(pm) {}  
};
```

```
// hydravion hérite des membres de deux classes: avion, et bateau.  
class hydravion:public avion,public bateau {  
public:  
    // Appel de deux constructeurs, pour éviter d'avoir un warning, l'ordre des constructeurs doit être le même  
    // que lors de la déclaration d'héritage.  
    hydravion(double a,double b):avion(a),bateau(b){}  
};  
int main() {  
    hydravion h(100,200);  
    return 0;  
}
```

3. Ambiguïtés possibles



```
#include <iostream>
using namespace std;
class vehicule {
public:
    double vitesse;
    vehicule(double v):vitesse(v){}
    void affiche() { cout << "vehicule/vitesse: " << vitesse << endl;}
};
class avion:public vehicule {
public:
    double altitude;
    avion(double a,double v):vehicule(v),altitude(a){}
};
class bateau:public vehicule {
public:
    double tirant_deau;
    bateau(double pm,double v):vehicule(v),tirant_deau(pm) {}
};
class hydravion:public avion,public bateau {
public:
    hydravion(double a,double b,double v):avion(a,v),bateau(b,v){}
};
int main() {
    hydravion h(100,200,300);

    cout << h.vitesse << endl; // Erreur! Laquelle des vitesses? via avion ou bien bateau?
    cout << h.affiche() << endl; // Erreur! Laquelle des fonctions affiche?

    return 0;
}
```

Pour lever l'ambiguïté,

On remplace

```
cout << h.vitesse << endl;
h.affiche();

Par
```

```
cout << "vitesse/avion: " << h.avion::vitesse << endl;
cout << "vitesse/bateau: " << h.bateau::vitesse << endl;

h.avion::affiche();
h.bateau::affiche();
```

Pas commode comme écriture!

Que faire pour avoir une seule version d'un champ?

4. Classes virtuelles

Si on ne veut qu'une seule version d'un champ,

1^{er} cas (figure de la page 255):

Le champ en double vient de deux classes différentes →
Rien à faire, on doit utiliser l'opérateur de résolution de portée :: pour éliminer l'ambiguïté.

2^e cas (figure de la page 256):

Le champ en double vient de la même classe →
Déclarer la classe de base (dans cet exemple, la classe véhicule) comme virtual.

```
#include <iostream>

class vehicule {
public:
    double vitesse;
    vehicule(double v):vitesse(v){}
    void affiche() { cout << "vehicule/vitesse: " << vitesse << endl;}
};

class avion:public virtual vehicule {
public:
    double altitude;
    avion(double a,double v):vehicule(v),altitude(a){}
};
```

Chapitre 14 : Héritage multiple et classes virtuelles

```
class bateau:public virtual vehicule {
public:
    double tirant_deau;
    bateau(double pm,double v):vehicule(v),tirant_deau(pm) {}
};

class hydravion:public avion,public bateau {
public:
    // Constructeur véhicule doit apparaître sinon c'est le constructeur par défaut qui sera appelé.
    hydravion(double a,double b,double v):vehicule(v),avion(a,v),bateau(b,v){}
};

int main() {
    hydravion h(100,200,300);

    cout << h.vitesse << endl; // cette fois ça marche! affiche: 300

    h.affiche(); // la aussi ça marche.

    return 0;
}
```

Dans les descendants de avion et bateau, les membres ne seront créés qu'une seule fois.

5. Appel des constructeurs

5.1. Classes non virtuelles

Les appels se font dans l'ordre suivant:

1- constructeur de la classe de base dans l'ordre d'appel,

2- constructeur classe dérivée

```
class hydravion; public avion, public bateau {} ;
```

1- avion

2- bateau

3- hydravion

5.2. Classes virtuelles

Les appels se font dans l'ordre suivant:

1- constructeur de la classe virtuelle avant tout le monde y compris ses propres ascendants,

2- ordre précédent (classes non virtuelles).