Cours 3 : L'héritage

Rabii EL GHORFI

Module : Technique de programmation avancées

Département : Mathématiques, informatique et géomatique (MIG)

EHTP 2017-2018

Principaux axes du cours

- La notion d'héritage
- Les types d'héritage
- Surcharge de méthodes
- Constructeurs dans l'héritage
- Héritage multiple
- Héritage virtuel

Principe

Définition : **L'héritage**

- Aussi appelé dérivation
- L'héritage permet de définir une nouvelle classe en se basant sur une classe déjà existante qu'on appelle la « classe mère »

Avantages :

- Réutilisation totale ou partielle des classes déjà développées
- Gain de temps et d'énergie pour le développement et la maintenance des codes sources
- Les méthodes identiques ne sont codées qu'une fois dans la classe mère

Implémentation (1)

Syntaxe :

- On utilise l'opérateur : et on spécifie le type d'héritage
- La classe B hérite publiquement de la classe A

```
class B : public A
```

Exemple : La classe Carre

- Les attributs : longueur et largeur et la méthode : calculer_Perimetre() ne sont déclarés qu'une fois dans la classe mère Rectangle
- La classe Carre hérite de la classe Rectangle

```
class Carre : public Rectangle
```

Implémentation (2)

```
class Rectangle {
public:
        Rectangle(int lon, int lar);
        int longueur;
        int largeur;
        int calculer_Perimetre();
};

class Carre {
public:
        Carre(int cote);
        int longueur;
        int largeur;
        int calculer_Perimetre();
};

int main() {
        Rectangle R1(10, 10);
        Carre R2(10);
        cout << R1.calculer_Perimetre();
        cout << R2.calculer_Perimetre();
        return 0;
}
```

Implémentation (3)

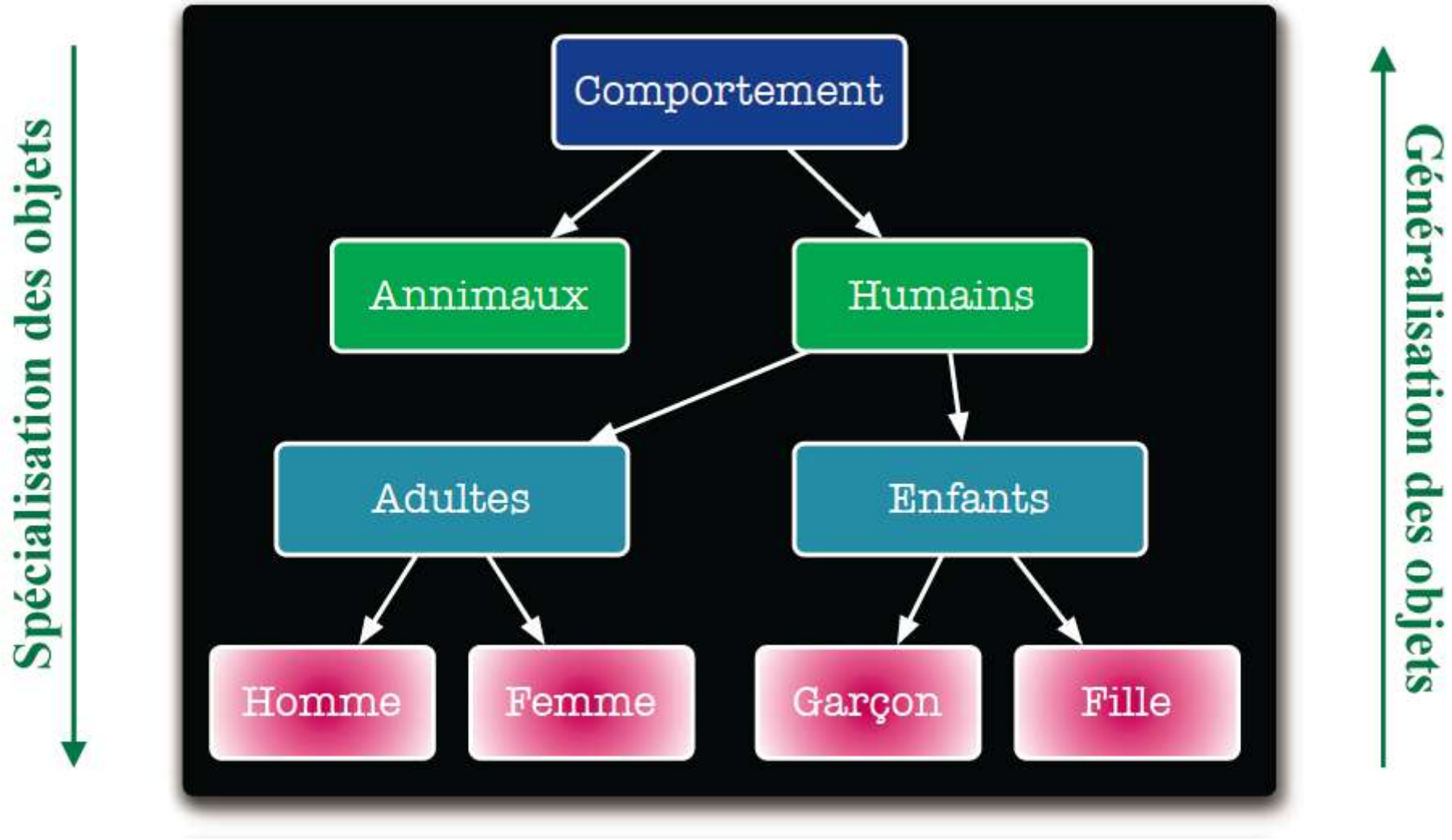
```
class Rectangle {
public:
        Rectangle(int lon, int lar);
        int longueur;
        int largeur;
        int calculer_Perimetre();
};

class Carre : public Rectangle {
public:
        Carre(int cote);
};
```

```
int main() {
        Rectangle R1(10, 10);
        Carre R2(10);
        cout << R1.calculer_Perimetre();
        cout << R2.calculer_Perimetre();
        return 0;
}
```

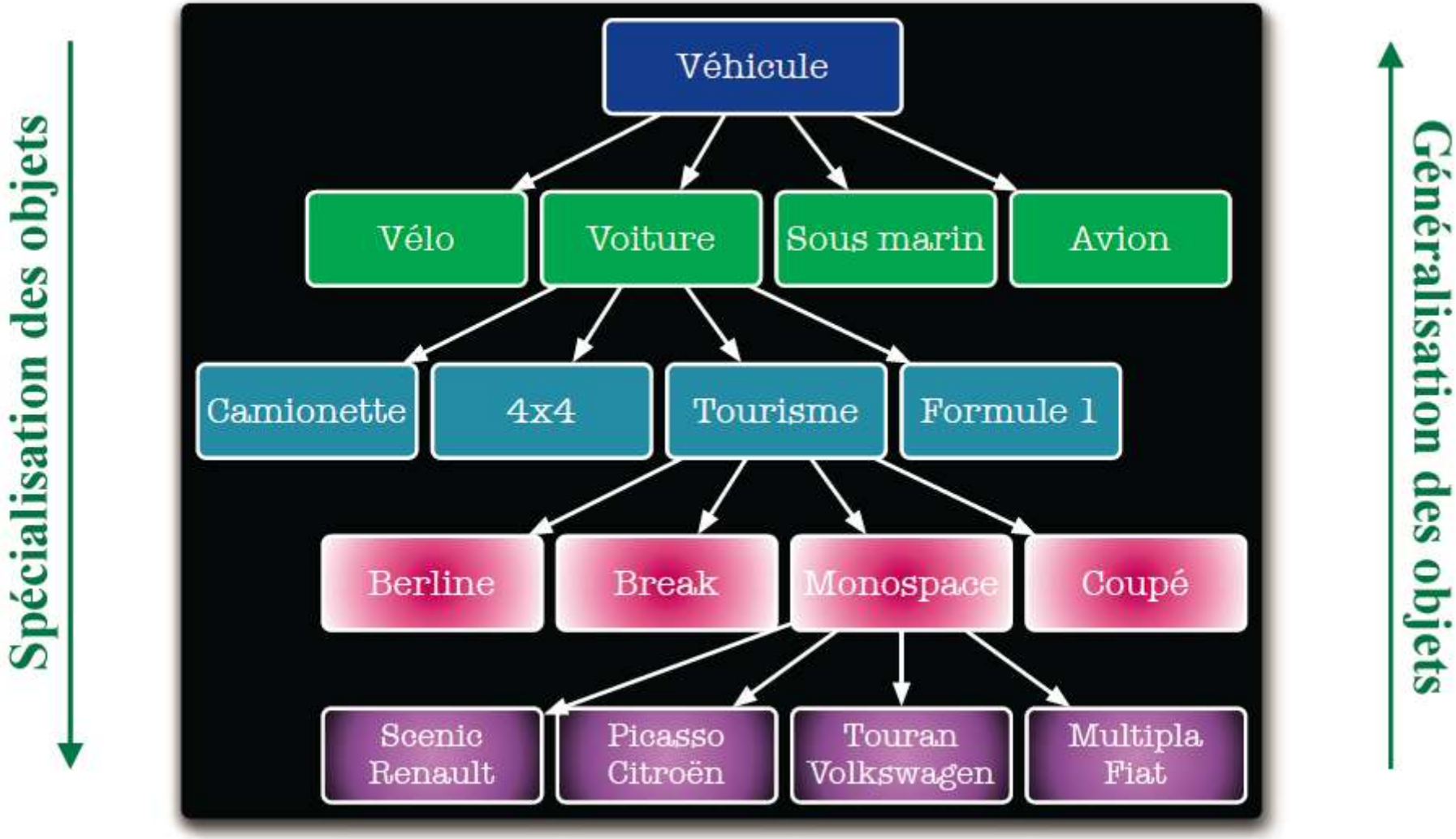
Propriétés

- L'héritage permet de donner à une classe toutes les caractéristiques d'une autre classe ou de plusieurs autres classes (héritage multiple)
- La classe qui hérite est appelée classe fille, classe dérivée ou classe descendante
- Les classes dont elle hérite sont appelées classes mères, classes de base ou classes antécédentes
- Les propriétés héritées par la classe fille sont : les attributs et les méthodes
- Après héritage, les classes filles peuvent définir de nouveaux membres ou redéfinir des méthodes de la classe mère

Exemple (1)


Spécialisation des objets
Comportement
Annimaux
Humains
Adultes
Enfants
Homme
Femme
Garçon
Fille
Généralisation des objets

Exemple (2)


Spécialisation des objets
Véhicule
Vélo
Voiture
Sous marin
Avion
Camionette
4x4
Tourisme
Formule 1
Berline
Break
Monospace
Coupé
Scenic Renault
Picasso Citroën
Touran Volkswagen
Multipla Fiat
Généralisation des objets

Exemple (3)

- Exemple 1 : La classe Garçon

```
class Garçon : public Enfants { // … // };
class Enfants : public Humains { // … // };
class Humains : public Comportements { // … // };
```

- Exemple 2 : La classe BMW320D

```
class BMW320D : public Berline { // … // };
class Berline : public Tourisme { // … // };
class Tourisme : public Voiture { // … // };
class Voiture : public Véhicule { // … // };
```

- Exemple 3 : La classe Tab_op

```
class Tab_op : public Tab_etud { // … // };
```

Types d'héritage

- Il existe 3 types d'héritage :
  - Public
  - Protected
  - Private

- Le mode d'héritage détermine quels sont les méthodes et les attributs de la classe de base qui seront accessibles dans la classe dérivée

- Si aucun mode d'héritage n'est spécifié, C++ prend par défaut : private

- Les membres privés de la classe de base ne sont jamais accessibles par les membres des classes dérivées

L'héritage publique

Définition : **L'héritage publique**

- Donne aux membres publics et protégés de la classe de base le même statut dans la classe dérivée
- C'est la forme la plus courante d'héritage, car il permet de modéliser : "B est une sorte de A" ou "B est une spécialisation de A"

Syntaxe :

- La classe B hérite de façon publique de la classe A

```
class B : public A
```

L'héritage privé

Définition : **L'héritage privé**

- Donne aux membres publics et protégés de la classe de base le statut de membres privés dans la classe dérivée
- Très peu utilisé (car pas utile), il permet de modéliser :
"B est composé de A"

Syntaxe :

- La classe B hérite de façon privée de la classe A

```
class B : private A
```

L'héritage protégé

Définition : **L'héritage protégé**

- Donne aux membres publics et protégés de la classe de base le statut de membres protégés dans la classe dérivée
- Il est utilisé lorsque l'on souhaite que des méthodes soient accessibles aux futures dérivées de la classe

Syntaxe :

- La classe B hérite de façon protégée de la classe A

```
class B : protected A
```

Surcharge de méthodes (1)

Définition : **Surcharge de méthodes**

- Correspond à redéfinir une fonction dans une classe dérivée
- Il est nécessaire que la méthode possède le même nom et les mêmes attributs que dans la classe de base

Principe :

- Il est possible de choisir quelle méthode on désire exécuter
  - La méthode surchargée
  - La méthode de la classe de base (avec l'opérateur ::)

Surcharge de méthodes (2)

```
class A {
public:
	int getValue() { return 0; }
};

class B : public A {
public:
	int getValue() { return 1; }
};
```

```
int main() {
	B b;
	cout << b.getValue(); // -> 1
	cout << b.A::getValue(); // -> 0
	return 0;
}
```

Constructeurs dans l'héritage (1)

- Les constructeurs, constructeur de copie, destructeurs et opérateurs d'affectation ne sont jamais hérités
- Les constructeurs par défaut des classes de bases sont automatiquement appelés avant le constructeur de la classe dérivée
- L'appel des destructeurs se fait dans l'ordre inverse des constructeurs
- Remarque : Pour ne pas appeler les constructeurs par défaut, mais des constructeurs avec des paramètres, on utilise une liste d'initialisation

Constructeurs dans l'héritage (2)

- Constructeurs par défaut et constructeurs avec une liste d'initialisation

```
class A {
public: A() {}      // Constructeur par default
        A(int n) {} // Autre constructeur
};

class B : public A {
public: B() {}      // Constructeur par default
B(int i) : A(i) {}  // Liste d'initialisation
};

int main() {
        B b1;       // A() -> B()
        B b2(7);    // A(7) -> B(7)
        return 0;
} // Ordre d'appel des constructeurs : A -> B
```

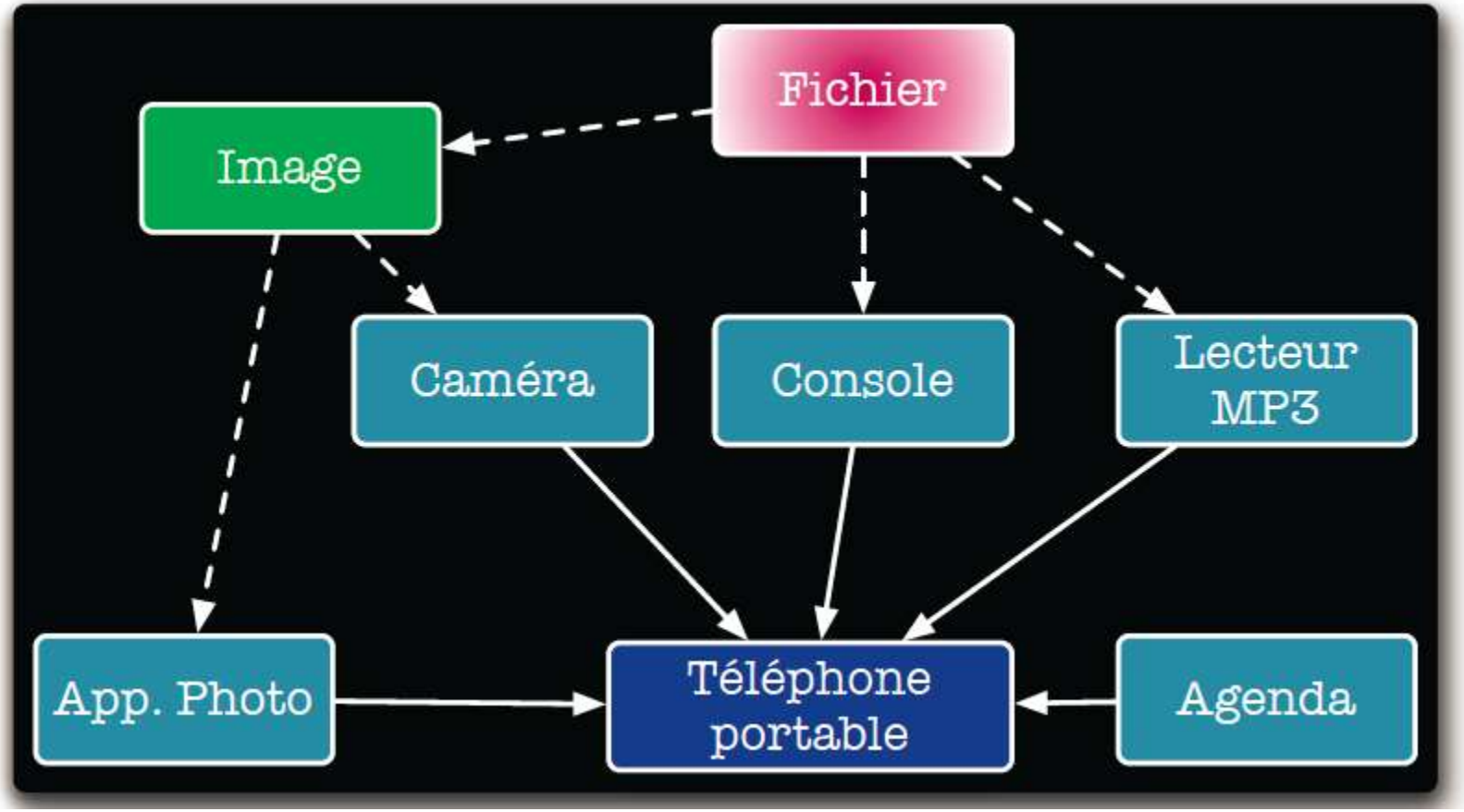
```
class Tab_op : public Tab_etud {
public:
Tab_op(int n) : Tab_etud(n) {}
//Liste d'initialisation
void supprimer();
friend Tab_op operator + (Tab_op,
Tab_op);
//Tab_op operator + (Tab_op) ;
void operator = (Tab_op);
};
```

Constructeurs dans l'héritage (3)

```
class Animal {
public:
	Animal() {cout << "Animal" << endl;}
	~Animal() {cout << "~Animal" << endl;}
};

class Chien : public Animal {
public:
	Chien() {cout << "Chien" << endl;}
	~Chien() {cout << "~Chien" << endl;}
};
```

```
int main() {
	Chien *york = new Chien();
	delete york;
	return 0;
}
	/* Le programme affiche :
	Animal
	Chien
	~Chien
	~Animal
	*/
```

Héritage multiple (1)



Héritage multiple (2)

- Le langage C++ permet de réaliser des héritages multiples
- Pour chaque classe de base, on peut définir le mode d'héritage : public, private ou protected
- L'héritage multiple peut mener à une complexification du code source et de sa compréhension

Syntaxe :

```
class C : public B, public A
```

Héritage multiple (3)

```
class A {
public:
      int dataA; // l'attribut dataA de la classe A
};
class B {
public:
      int dataB; // l'attribut dataB de la classe B
};
class C : public B, public A {
public:                                          // Dans la classe C, nous utilisons
      int somme() { return dataA + dataB; }      // les attributs des classes A et B
};                                               // dont nous héritons
```

Constructeurs dans l'héritage multiple

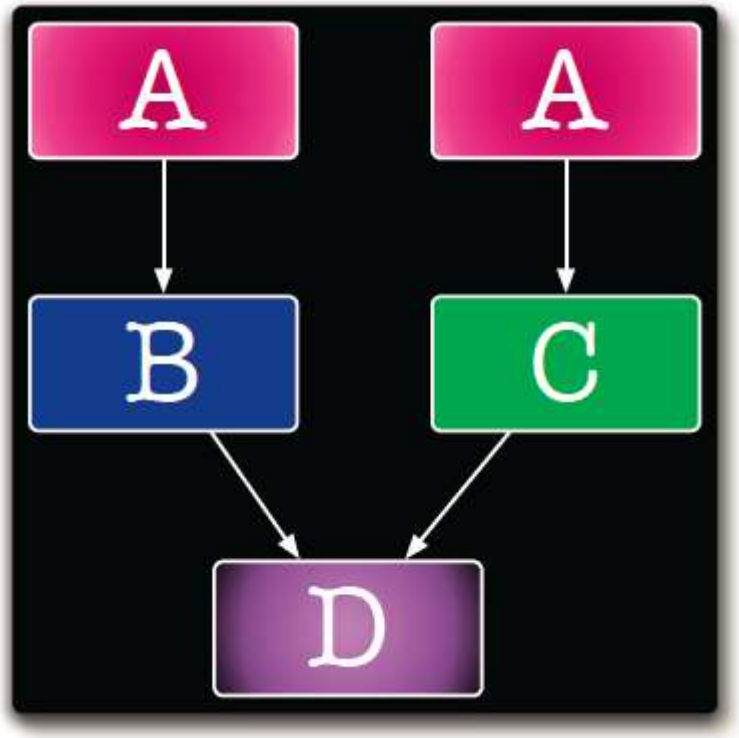
- Les constructeurs sont appelés dans l'ordre de déclaration de l'héritage
- Les destructeurs sont appelés dans l'ordre inverse de celui des constructeurs

```
class A {
public: A(int n = 0) { /* ... */ }
};
class B {
public: B(int n = 0) { /* ... */ }
};
class C : public B, public A {
public: C(int i, int j) : A(i), B(j)
{ /*...*/ }
};
```

```
int main() {
        C c1; // B() -> A() -> C()
        C c2(5,4); // B(4) -> A(5) -> C(5,4)
        return 0;
}

        // ordre d'appel des constructeurs
        // B -> A -> C
```

Héritage virtuel (1)

- Problème : (héritage multiple) Un objet de la classe D contiendra 2 fois les données héritées de A
  - Une fois par héritage de la classe B
  - Une autre fois par héritage de C



Supposons que A contienne une variable var :

```
int main() {
        D obj;
        obj.var = 0; // Ambiguité (erreur)
        obj.B::var = 1; // OK
        obj.C::var = 2; // OK
        return 0;
}
```

Héritage virtuel (2)

- Solution : On souhaite une unique occurrence des membres de la classe mère

Pour que la classe D n'hérite qu'une seule fois de la classe A, il faut que les classes B et C héritent virtuellement de A

```
class A { int var; }
class B : virtual public A { /* … */ };
class C : virtual public A { /* … */ };
class D : public B : public C { /* … */ };
```