

# MOOC Intro POO C++

## Tutoriels semaine 6 : Héritage multiple

Les tutoriels sont des exercices qui reprennent des exemples similaires à ceux du cours et dont le corrigé est donné progressivement au fur et à mesure de la donnée de l'exercice lui-même.

Ils sont conseillés comme un premier exercice sur un sujet que l'étudiant ne pense pas encore assez maîtriser pour aborder par lui-même un exercice « classique ».

Les solutions sont fournies au fur et à mesure sur les pages paires.

Cet exercice correspond à l'exercice « *pas à pas* » page 155 de l'ouvrage *C++ par la pratique* (3<sup>e</sup> édition, PPUR).

### Introduction

Le but de cet exercice est de reprendre l'exemple du cours illustrant la notion d'héritage multiple et de classe virtuelle en approfondissant l'exemple des ovovivipares (vous auriez préféré les flots [iostream] ?;-)).

Dans le fichier zoo.cc, commencez par définir une classe Vivipare contenant un entier non signé représentant la durée de gestation (en jours) et un constructeur permettant d'initialiser cette valeur et pouvant servir de constructeur par défaut (avec une valeur de votre choix).

**[Essayez de le faire par vous-même avant de regarder la solution qui suit]**

(solution page suivante)

**Solution :**

```
// la classe :  
class Vivipare {  
};  
  
// l'attribut :  
class Vivipare {  
protected:  
    unsigned int gestation;  
};  
  
// le constructeur :  
class Vivipare {  
public:  
    Vivipare(unsigned int jours = 128) : gestation(jours) {}  
protected:  
    unsigned int gestation;  
};
```

---

Définissez maintenant une classe `Ovipare` ayant un attribut représentant le nombre d'œufs par ponte et un constructeur adapté.

(solution page suivante)

**Solution :**

```
class Ovipare {  
public:  
    Ovipare(unsigned int nombre = 12) : oeufs(nombre) {}  
protected:  
    unsigned int oeufs;  
};
```

---

Ajoutez à chacune des classes une méthode naissance affichant :

- "Après  $X$  jours de gestation, je viens de mettre au monde un nouveau bébé." dans le cas d'un vivipare,
- "Je viens de pondre environ  $X$  oeuf(s)." dans le cas d'un ovipare,

où  $X$  correspond à la valeur de l'attribut.

La solution est ici vraiment triviale, je continue donc...

Définissez maintenant la classe Ovovivipare comme héritant à la fois de Vivipare (en premier) et de Ovipare.

Ajoutez à cette classe un attribut de type booléen indiquant si l'espèce est rare ou non.

Ajoutez également un constructeur prenant une période de gestation, un nombre d'œufs et un booléen (false par défaut) indiquant la rareté de l'espèce.

(solution page suivante)

## Solution :

Pour l'héritage multiple, il suffit de mettre les différentes super-classes les unes après les autres, séparées par des virgules :

```
class Ovovivipare : public Vivipare, public Ovipare {  
protected:  
    bool espece_rare;  
};
```

On ajoute ensuite le constructeur :

```
class Ovovivipare : public Vivipare, public Ovipare {  
public:  
    Ovovivipare(unsigned int jours, unsigned int nb, bool rare = false)  
        : Vivipare(jours), Ovipare(nb), espece_rare(rare)  
        {}  
protected:  
    bool espece_rare;  
};
```

---

Pour terminer cette première exploration, faites afficher un message par les constructeurs des classes Vivipare et Ovipare afin d'observer l'ordre des appels, et créez un `main()` contenant une instance d'Ovovivipare.

Quel est l'ordre d'appel des constructeurs ?

Changez l'ordre des constructeurs dans le constructeur d'Ovovivipare. Recompilez et relancez votre programme. Cela change-t-il l'ordre d'appel ?

Appelez maintenant la méthode `naissance` de votre instance. Compilez et testez votre programme. Que se passe-t-il ?

Corrigez le programme pour que ce soit la méthode `naissance` de Vivipare qui soit appelée.

Recompilez et relancez votre programme.

Modifiez finalement le programme pour que la méthode `naissance` d'Ovovivipare affiche :

```
"Après X jours de gestation, je viens de mettre au monde Y nouveau(x) bébé(s)."
```

où X correspond à la période de gestation et Y au nombre d'œufs.

(solution page suivante)

## Solution :

- Au début de cette étape, on arrive donc au code suivant :

```
#include <iostream> // pour cout, endl
using namespace std; // pour ecrire "cout" au lieu de "std::cout"

// -----
class Vivipare {
public:
    Vivipare(unsigned int jours = 128) : gestation(jours) {
        cout << "je suis un vivipare" << endl;
    }
    void naissance() const {
        cout << "Apres " << gestation << " jours de gestation," << endl
            << "je viens de mettre au monde un nouveau bebe." << endl;
    }
protected:
    unsigned int gestation;
};

// -----
class Ovipare {
public:
    Ovipare(unsigned int nombre = 12) : oeufs(nombre) {
        cout << "je suis un ovipare" << endl;
    }
    void naissance() const {
        cout << "Je viens de pondre environ " << oeufs
            << " oeuf(s)." << endl;
    }
protected:
    unsigned int oeufs;
};

// -----
class Ovovivipare : public Vivipare, public Ovipare {
public:
    Ovovivipare(unsigned int jours, unsigned int nb, bool rare = false)
        : Vivipare(jours), Ovipare(nb), espece_rare(rare)
    {}
protected:
    bool espece_rare;
};

// =====
int main()
{
    Ovovivipare un_requin(220, 11); // de 6 a 12 mois et de 9 a 12 oeufs
    // un_requin.naissance();
    return 0;
}
```

- L'ordre d'appel des constructeurs est toujours celui de la déclaration des super-classes dans la déclaration de la classe, et ne dépend pas de l'ordre de leur appel dans le constructeur.

D'ailleurs, avec le compilateur utilisé ici, vous avez un message d'alerte (*warning*) indiquant que l'ordre d'appel sera changé si vous ne mettez pas les appels des constructeurs dans l'ordre de la déclaration des héritages.

- L'appel à la méthode naissance(), sans autre précision, provoque une erreur de compilation en raison de l'ambiguïté du nom due à l'héritage multiple :

```
zoo.cc:48: request for member `naissance' is ambiguous
zoo.cc:24: candidates are: void Ovipare::naissance() const
```

```
zoo.cc:10: void Vivipare::naissance() const
```

Il faut donc lever cette ambiguïté.

- Une première solution peut être d'utiliser celle de Vivipare en utilisant la directive using (dans la classe Ovovivipare):

```
class Ovovivipare : public Vivipare, public Ovipare {  
public:  
    Ovovivipare(unsigned int jours, unsigned int nb, bool rare = false)  
        : Vivipare(jours), Ovipare(nb), espece_rare(rare)  
    {}  
    using Vivipare::naissance; // attention, sans parenthèses !  
protected:  
    bool espece_rare;  
};
```

Attention, **sans** parenthèses!

- On peut aussi carrément redéfinir cette méthode dans la classe Ovovivipare (attention d'enlever le using précédent):

```
class Ovovivipare : public Vivipare, public Ovipare {  
public:  
    Ovovivipare(unsigned int jours, unsigned int nb, bool rare = false)  
        : Vivipare(jours), Ovipare(nb), espece_rare(rare)  
    {}  
    void naissance() const {  
        cout << "Après " << gestation << " jours de gestation, "  
            << "je viens de mettre au monde " << oeufs  
            << " nouveau(x) bebe(s)" << endl;  
    }  
protected:  
    bool espece_rare;  
};
```

---

Je voudrais finir par un exemple de classe virtuelle en introduisant la classe `Animal`.

Ajoutez une classe `Animal` au programme, ne comprenant qu'un constructeur par défaut et un destructeur affichant chacun un message.

Faites hériter `Vivipare` et `Ovipare` de `Animal`.

Recompilez et exécutez votre programme. Que se passe-t-il ?

Pour éviter cela, faites que la classe `Animal` soit virtuelle.

Recompilez et exécutez votre programme.

Notez bien à quel moment le constructeur de `Animal` est appelé.

(solution page suivante)

## Solution :

```
class Animal {
public:
    Animal() { cout << "coucou, un animal de plus" << endl; }
    virtual ~Animal() { cout << "adieu..." << endl; }
};
```

Avec cet héritage simple, on voit bien que DEUX instances de Animal sont créées pour un seul Ovovivipare, car aucune des deux relations d'héritage n'est virtuelle. Pour rendre l'héritage virtuel, il suffit de faire :

```
class Vivipare : public virtual Animal { ... };
class Ovipare : public virtual Animal { ... };
```

Le constructeur d'Animal est alors bien appelé en premier :

```
coucou, un animal de plus
je suis un vivipare
je suis un ovipare
...
```

Voici le code complet de l'exemple :

```
#include <iostream> // pour cout, endl
using namespace std; // pour ecrire "cout" au lieu de "std::cout"

// -----
class Animal {
public:
    Animal() { cout << "coucou, un animal de plus" << endl; }
    virtual ~Animal() { cout << "adieu..." << endl; }
};

// -----
class Vivipare : public virtual Animal {
public:
    Vivipare(unsigned int jours = 128) : gestation(jours) {
        cout << "je suis un vivipare" << endl;
    }
    void naissance() const {
        cout << "Après " << gestation << " jours de gestation," << endl
            << "je viens de mettre au monde un nouveau bebe." << endl;
    }
protected:
    unsigned int gestation;
};

// -----
class Ovipare : public virtual Animal {
public:
    Ovipare(unsigned int nombre = 12) : oeufs(nombre) {
        cout << "je suis un ovipare" << endl;
    }
    void naissance() const {
        cout << "Je viens de pondre environ " << oeufs
            << " oeuf(s)." << endl;
    }
protected:
    unsigned int oeufs;
};

// -----
class Ovovivipare : public Vivipare, public Ovipare {
public:
```

```
Ovovivipare(unsigned int jours, unsigned int nb, bool rare = false)
    : Vivipare(jours), Ovipare(nb), espece_rare(rare)
{}
// using Vivipare::naissance;
void naissance() const {
    cout << "Apres " << gestation << " jours de gestation, "
         << "je viens de mettre au monde " << oeufs
         << " nouveau(x) bebe(s)" << endl;
}
protected:
    bool espece_rare;
};

// =====
int main()
{
    Ovovivipare un_requin(220, 11); // de 6 a 12 mois et de 2 a 20+ oeufs
    un_requin.naissance();
    return 0;
}
```