Множественное наследование

Множественное наследование

Ранее упоминалось о том, что иногда могло бы пригодиться ***множественное наследование*** **(multiple inheritance),** как в случае с утконосом.

Утконос — частично млекопитающее, частично птица, частично рептилия.

Множественное наследование

Для таких случаев язык C++ позволяет унаследовать класс от двух и более базовых классов:

```
class Производный: модификаторДоступа Базовый1, модификаторДоступа Базовый2
{
    // члены класса
};
```

Множественное наследование

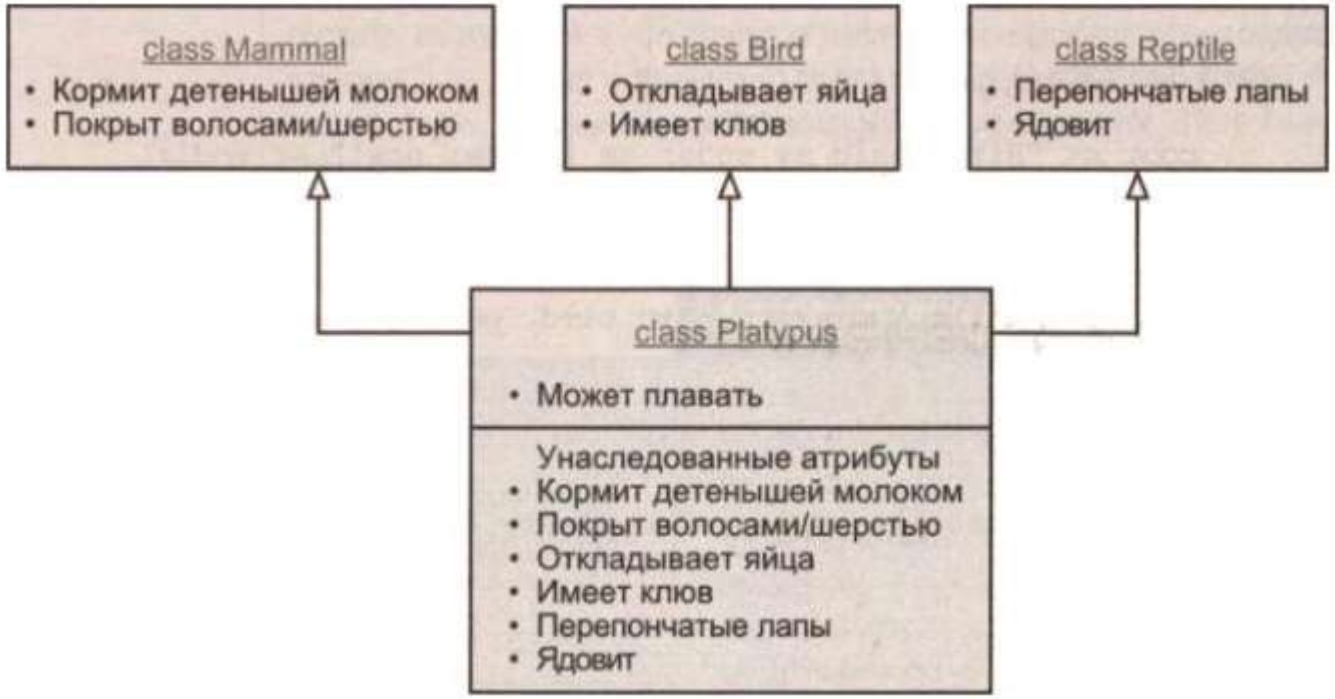

РИС. 10.3. Отношения между классом `Platypus` и классами `Mammal`, `Reptile` и `Bird`

Множественное наследование

Таким образом, синтаксическое представление C++ класса P l a t y p u s будет следующим:

```
class Platypus: public Mammal, public Reptile, public Bird
{
    // ... члены класса Platypus
};
```

ЛИСТИНГ 10.10. Использование множественного наследования для моделирования утконоса, являющегося млекопитающим, птицей и рептилией

```
 0: #include <iostream>
 1: using namespace std;
 2:
 3: class Mammal
 4: {
 5: public:
 6:    void FeedBabyMilk()
 7:    {
 8:       cout << "Mammal: Baby says glug!" << endl;
 9:    }
10: };
11:
```

ЛИСТИНГ 10.10. Использование множественного наследования для моделирования утконоса, являющегося млекопитающим, птицей и рептилией

```
12: class Reptile
13: {
14: public:
15:     void SpitVenom()
16:     {
17:         cout << "Reptile: Shoo enemy! Spits venom!" << endl;
18:     }
19: };
20:
21: class Bird
22: {
23: public:
24:     void LayEggs()
25:     {
26:         cout << "Bird: Laid my eggs, am lighter now!" << endl;
27:     }
28: };
29:
```

ЛИСТИНГ 10.10. Использование множественного наследования для моделирования утконоса, являющегося млекопитающим, птицей и рептилией

```
30: class Platypus: public Mammal, public Bird, public Reptile
31: {
32: public:
33:     void Swim()
34:     {
35:         cout << "Platypus: Voila, I can swim!" << endl;
36:     }
37: };
38:
```

ЛИСТИНГ 10.10. Использование множественного наследования для моделирования утконоса, являющегося млекопитающим, птицей и рептилией

```
39: int main()
40: {
41:     Platypus realFreak;
42:     realFreak.LayEggs();
43:     realFreak.FeedBabyMilk();
44:     realFreak.SpitVenom();
45:     realFreak.Swim();
46:
47:     return 0;
48: }
```

ЛИСТИНГ 10.10. Использование множественного наследования для моделирования утконоса, являющегося млекопитающим, птицей и рептилией

РЕЗУЛЬТАТ

```
Bird: Laid my eggs, am lighter now!
Mammal: Baby says glug!
Reptile: Shoo enemy! Spits venom!
Platypus: Voila, I can swim!
```

ЛИСТИНГ 10.10. Использование множественного наследования для моделирования утконоса, являющегося млекопитающим, птицей и рептилией

Анализ

Определение средств класса P l a t y p u s действительно компактно (строки 30-37).

По существу, он просто наследует их от трех классов: M a m m a l, R e p t i l e и B i r d .

Функция main () в строках 41-44 способна обратиться к трем индивидуальным характеристикам базовых классов, используя объект r e a l F r e a k производного класса P l a t y p u s .

ЛИСТИНГ 10.10. Использование множественного наследования для моделирования утконоса, являющегося млекопитающим, птицей и рептилией

Анализ

Кроме вызова функций, унаследованных от классов Mammal, B i r d и R e p t i l e , функция m a i n () в строке 45 вызывает метод P l a t y p u s :: Swim ( ) .

Эта программа демонстрирует синтаксис множественного наследования, а также то, как производный класс предоставляет все открытые члены

(в данном случае методы) многих своих базовых классов.

ЛИСТИНГ 10.10. Использование множественного наследования для моделирования утконоса, являющегося млекопитающим, птицей и рептилией

ПРИМЕЧАНИЕ

Утконос может плавать, но это не рыба. Следовательно, в листинге 10.10 мы не наследовали класс **Platypus** также от класса **Fish** только для удобства использования существующего метода **Fish: :Swim().**

Принимая решения впроекте, не забывайте, что открытое наследование должно также отражать отношения и не должно использоваться без разбора, лишь для решения определенных задач, связанных с многократным использованием. Этого можно достичь и по-другому.

Наследование:итоги

РЕКОМЕНДУЕТСЯ

Создавайте открытую иерархию наследования для установки отношений *есть*

Создавайте закрытую или защищенную иерархию наследования для установки отношений *содержит*

Помните, открытое наследование означает, что у классов, производных от производного класса, есть доступ к открытым и защищенным членам базового класса

Помните, закрытое наследование означает, что даже классы, производные от производного класса, не имеют доступа к базовому классу

Помните, защищенное наследование означает, что у классов, производных от производного класса, есть доступ к защищенным и открытым методам базового класса

Помните, что, независимо от характера наследственных отношений, закрытые члены в базовом классе недоступны никаким производным классам

НЕ РЕКОМЕНДУЕТСЯ

Не создавайте иерархию наследования только для многократного использования обычных функций

Не используйте закрытое и открытое наследование без разбора, поскольку впоследствии это может стать узким местом архитектуры вашего приложения

Не создавайте функции производного класса (с тем же именем, но другим набором входных параметров), которые непреднамеренно скрывают таковые в базовом классе