Закрытое наследование

Закрытое наследование

Закрытое наследование

(private inheritance)

отличается от открытого (которое рассматривалось до сих пор) тем, что в строке объявления производного класса

как происходящего от базового класса используется ключевое слово p r i v a t e :

Закрытое наследование

```
class Base
{
    // ... переменные-члены и методы базового класса
};

class Derived: private Base // закрытое наследование
{
    // ... переменные-члены и методы производного класса
};
```

Закрытое наследование

Закрытое наследование базового класса означает, что все открытые члены и атрибуты базового класса являются закрытыми (т.е. недоступными) для всех, кроме экземпляра производного класса.

Другими словами, даже открытые члены и методы класса B a s e могут быть использованы только классом D e r i v e d , но ни кем-либо, еще владеющим экземпляром класса D e r i v e d .

Закрытое наследование

Это резко контрастирует с примерами класса Tuna и его базового класса Fish , которые мы рассматривали начиная с листинга 10.1. Функция m a i n ( ) в листинге 10.1 может вызвать функцию Fish : :Swim( ) у экземпляра класса Tuna, поскольку функция Fish : :Swim( ) является открытым методом и потому, что класс Tuna происходит от класса Fish с использованием открытого наследования.

Попробуйте переименовать ключевое слово p u b l i c на p r i v a t e в строке 17, и вы получите сбой компиляции.

Закрытое наследование

Таким образом, для мира вне иерархии наследования закрытое наследование по существу не означает отношение *есть* (is-a) (вообразите тунца, который не может плавать!).

Закрытое наследование

Поскольку закрытое наследование позволяет использовать атрибуты и методы базового класса только производным классам, которые происходят от него, создаются отношения, называемые *содержит* (has-a).

В окружающем мире есть множество примеров закрытого наследования (табл. 10.2).

Закрытое наследование

ТАБЛИЦА 10.2. Примеры закрытого наследования из повседневной жизни

Базовый класс	Примеры производных классов
`Motor` (Мотор)	`Car` (Автомобиль *содержит* мотор)
`Heart` (Сердце)	`Mammal` (Млекопитающее *содержит* сердце)
`Refill` (Стержень)	`Pen` (Ручка *содержит* стержень)
`Moon` (Луна)	`Sky` (Небо *содержит* луну)

Давайте рассмотрим закрытое наследование на примере отношений автомобиля с его мотором (листинг 10.8).

ЛИСТИНГ 10.8. Класс **Car,** связанный с классом **Motor** через закрытое наследование

```
 0: #include <iostream>
 1: using namespace std;
 2:
 3: class Motor
 4: {
 5: public:
 6:     void SwitchIgnition()
 7:     {
 8:         cout << "Ignition ON" << endl;
 9:     }
10:     void PumpFuel()
11:     {
12:         cout << "Fuel in cylinders" << endl;
13:     }
14:     void FireCylinders()
15:     {
16:         cout << "Vroooom" << endl;
17:     }
18: };
19:
```

ЛИСТИНГ 10.8. Класс **Car,** связанный с классом **Motor** через закрытое наследование

```
20: class Car:private Motor // private inheritance
21: {
22: public:
23:     void Move()
24:     {
25:         SwitchIgnition();
26:         PumpFuel();
27:         FireCylinders();
28:     }
29: };
30:
```

ЛИСТИНГ 10.8. Класс **Car,** связанный с классом **Motor** через закрытое наследование

```
31: int main()
32: {
33:     Car myDreamCar;
34:     myDreamCar.Move();
35:
36:     return 0;
37: }
```

ЛИСТИНГ 10.8. Класс **Car,** связанный с классом **Motor** через закрытое наследование

РЕЗУЛЬТАТ

```
Ignition ON
Fuel in cylinders
Vroooom
```

ЛИСТИНГ 10.8. Класс **Car,** связанный с классом **Motor** через закрытое наследование

Анализ

Класс Motor, **определенный в строках 3-18, очень прост, он содержит три защищенные функции-члена, включая зажигание** (SwitchIgnition()), **подачу топлива** (PumpFuel()) **и запуск** (FireCylinders()).

Класс Car **наследует класс** Motor **с использованием ключевого слова** private **(строка 20).**

ЛИСТИНГ 10.8. Класс **Car,** связанный с классом **Motor** через закрытое наследование

Анализ

Таким образом, открытая функция

C a r **::** Move **() обращается к членам базового класса** M o t o r. **Если вы попытаетесь вставить в функцию** main ( ) **строку**

```
myDreamCar.PumpFuel();
```

ЛИСТИНГ 10.8. Класс **Car,** связанный с классом **Motor** через закрытое наследование

Анализ

то **получите при компиляции ошибку с сообщением**

error C2247 **:** M o t o r**: :** P u m p F u e l not accessible because ’C a r ’

uses ‘ private ' to inherit from 'M o t o r '

(ошибка C2247: M o t o r**: :** P u m p F u e l **недоступен, поскольку** 'C a r ' **использует** ' private ' **при наследовании от** **'**M o t o r**‘).**

ЛИСТИНГ 10.8. Класс **Car,** связанный с классом **Motor** через закрытое наследование

ПРИМЕЧАНИЕ

Если от класса **Car** произойдет другой класс, например **SuperCar,** то, независимо от характера наследования, у класса **SuperCar** не будет доступа к открытым членам и методам базового класса **Motor.**

Дело в том, что отношения наследования между классами **Car** и **Motor** имеют закрытый характер, а значит, доступ для всех остальных, кроме класса **Car,** будет закрытым (т.е. доступа не будет), даже к открытым членам базового класса.

ЛИСТИНГ 10.8. Класс **Car,** связанный с классом **Motor** через закрытое наследование

ПРИМЕЧАНИЕ

Другими словами, наиболее ограничивающий модификатор доступа доминирует при принятии компилятором решения о том, должен ли у некого класса быть доступ к открытым или защищенным членам базового класса.

Защищенное наследование

Защищенное наследование

Защищенное наследование отличается от открытого наличием ключевого слова p r o t e c t e d **в строке объявления производного класс как происходящего от базового класса:**

Защищенное наследование

```
class Base
{
    // ... переменные-члены и методы базового класса
};

class Derived: protected Base // защищенное наследование
{
    // ... переменные-члены и методы производного класса
};
```

Защищенное наследование

Защищенное наследование подобно закрытому в следующем:

- **Реализует отношения**

содержит **(has-a).**

Защищенное наследование

- Позволяет производному классу обращаться ко всем открытым и защищенным членам базового.

Защищенное наследование

- Вне иерархии наследования нельзя при помощи экземпляра производного класса обратиться к открытым членам базового класса.

Защищенное наследование

Но защищенное наследование все же отличается от закрытого, когда дело доходит до следующего производного класса, унаследованного от него:

```
class Derived2: protected Derived
{
    // имеет доступ к членам Base
};
```

Защищенное наследование

Иерархия защищенного наследования позволяет производному классу производного класса (т.е. классу D e r i v e d 2) обращаться к открытым членам базового класса (листинг 10.9).

Это не было бы возможно, если бы при наследовании классом D e r i v e d класса B a s e использовалось ключевое слово p r i v a t e .

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,**

при защищенном наследовании

```
0: #include <iostream>
1: using namespace std;
2:
3: class Motor
4: {
5: public:
6:     void SwitchIgnition()
7:     {
8:         cout << "Ignition ON" << endl;
9:     }
10:    void PumpFuel()
11:    {
12:        cout << "Fuel in cylinders" << endl;
13:    }
14:    void FireCylinders()
15:    {
16:        cout << "Vroooom" << endl;
17:    }
18: };
19:
```

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,**

при защищенном наследовании

```
20: class Car:protected Motor
21: {
22: public:
23:     void Move()
24:     {
25:         SwitchIgnition();
26:         PumpFuel();
27:         FireCylinders();
28:     }
29: };
30:
```

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,**

при защищенном наследовании

```
31: class SuperCar:protected Car
32: {
33: public:
34:     void Move()

35:     {
36:         SwitchIgnition(); // имеет доступ к членам базового благодаря
37:         PumpFuel();       // защищенному наследованию между Car и Motor
38:         FireCylinders();
39:         FireCylinders();
40:         FireCylinders();
41:     }
42: };
43:
```

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,**

при защищенном наследовании

```
int main()
{
    SuperCar myDreamCar;
    myDreamCar.Move();

    return 0;
}
```

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,**

при защищенном наследовании

Результат

```
Ignition ON
Fuel in cylinders
Vroooom
Vroooom
Vroooom
```

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,** при защищенном наследовании

- **Анализ**

Класс C a r защищенно наследует класс M o t o r (строка 20). Класс S u p e r C a r защищенно наследует класс C a r (строка 31).

Как можно заметить, реализация метода S u p e r C a r: :Move () использует открытые методы, определенные в базовом классе M o t o r

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,** при защищенном наследовании

- **Анализ**

Этот доступ к самому первому базовому классу M o t o r через промежуточный базовый класс C a r обеспечивают отношения между классами C a r и M o t o r.

Если бы это было закрытое наследование, а не защищенное, то у производного класса не было бы доступа к открытым членам M o t o r, поскольку компилятор выберет самый ограничивающий из использованных модификаторов доступа.

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,** при защищенном наследовании

- **Анализ**

Обратите внимание, что характер отношений между классами C a r и S u p e r C a r не имеет значения при доступе к базовому классу. Так, даже если в строке 31 заменить ключевое слово p r o t e c t e d на p u b l i c или p r i v a t e , исход компиляции этой программы остается неизменным.

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,** при защищенном наследовании

- **ВНИМАНИЕ!**

Используйте закрытое или защищенное наследование только по мере необходимости.

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,** при защищенном наследовании

- **ВНИМАНИЕ!**

В большинстве случаев, когда используется закрытое наследование (как у классов **Car** и **Motor),** базовый класс также может быть атрибутом (членом) класса **Car,** а не суперклассом. При наследовании от класса **Motor** вы, по существу, ограничили свой класс **Car** наличием только одного мотора, без какого-либо существенного выигрыша от наличия экземпляра класса **Motor** как закрытого члена.

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,** при защищенном наследовании

- **ВНИМАНИЕ!**

Автомобили развиваются, и сейчас не редкость гибридные автомобили, например, в дополнение к обычному мотору может применяться газовый или электрический.

Наша иерархия наследования для класса **Car** оказалась бы узким местом, попытайся мы последовать за такими разработками.

ЛИСТИНГ 10.9. Класс **SuperCar,** производный от класса **Car,** происходящего от класса **Motor,** при защищенном наследовании

- **ВНИМАНИЕ!**

Автомобили развиваются, и сейчас не редкость гибридные автомобили, например, в дополнение к обычному мотору может применяться газовый или электрический.

Наша иерархия наследования для класса **Car** оказалась бы узким местом, попытайся мы последовать за такими разработками.

композиция (composition) или *объединение* (aggregation)

- **ПРИМЕЧАНИЕ**

Наличие экземпляра класса **Motor** как закрытого члена, вместо наследования от него, называется *композиция* (composition) или *объединение* (aggregation).

Такой класс **Car** выглядел бы следующим образом:

композиция (composition) или *объединение* (aggregation)

```
class Car
{
private:
    Motor heartOfCar;

public:
    void Move()
    {
        heartOfCar.SwitchIgnition();
        heartOfCar.PumpFuel();
        heartOfCar.FireCylinders();
    }
};
```

композиция (composition) или *объединение* (aggregation)

Это может быть хорошим ходом, поскольку позволяет легко добавлять к существующему классу **Car** больше моторов как атрибутов, не изменяя его иерархию наследования или предоставляемые клиентам возможности.

Проблема отсечения

Что будет, если программист сделает следующее?

```
Derived objectDerived;
Base objectBase = objectDerived;
```

Проблема отсечения

Или следующее?

```
void FuncUseBase(Base input);
...
Derived objectDerived;
FuncUseBase(objectDerived); // objectDerived будет отсечен при
                            // копировании во время вызова функции
```

Проблема отсечения

В обоих случаях объект производного класса копируется в другой объект, но уже базового класса явно, при присвоении, или косвенно, при передаче в качестве аргумента.
В этих случаях компилятор скопирует в объект o b j e c t D e r i v e d только часть, соответствующую классу Base, а не весь объект.

Проблема отсечения

Как правило, это вовсе не то, чего ожидает программист, и это нежелательное сокращение части данных, специализирующей производный класс относительно базового, называется ***отсечением*** **(slicing)**.

Проблема отсечения

ВНИМАНИЕ!
Чтобы избежать проблемы отсечения, не передавайте параметры по значению. Передавайте их как указатели на базовый класс или как ссылку (можно **const)** на него же.