Отношения между классами. Вложенные классы. Агрегация и композиция

5 июля 2017 г.

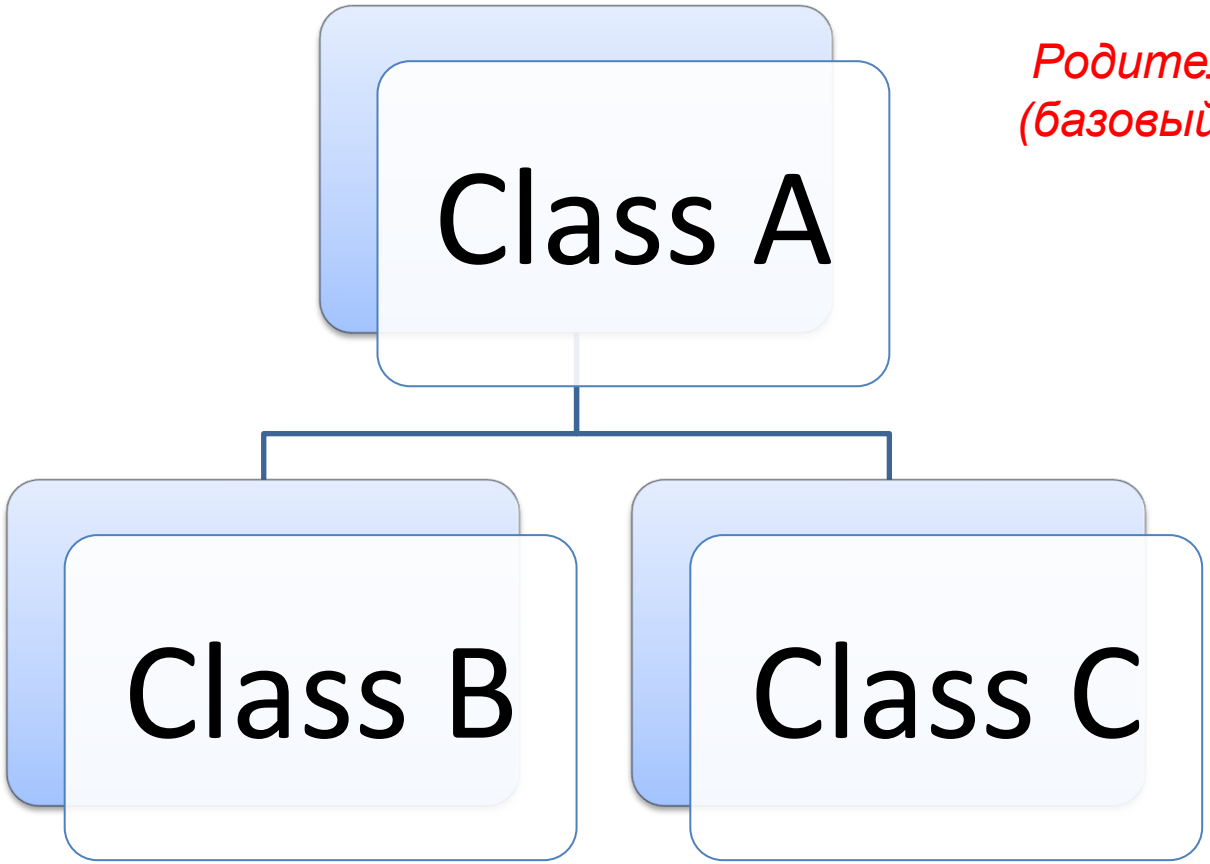
Повторение

1. Что такое наследование?
2. Зачем нужно наследование?
3. Какие примеры наследования вы можете привести?
5. Что такое множественное наследование?
4. В чем недостатки наследования?

Отношения между классами

- наследование
- вложение
- агрегация и композиция

Наследование



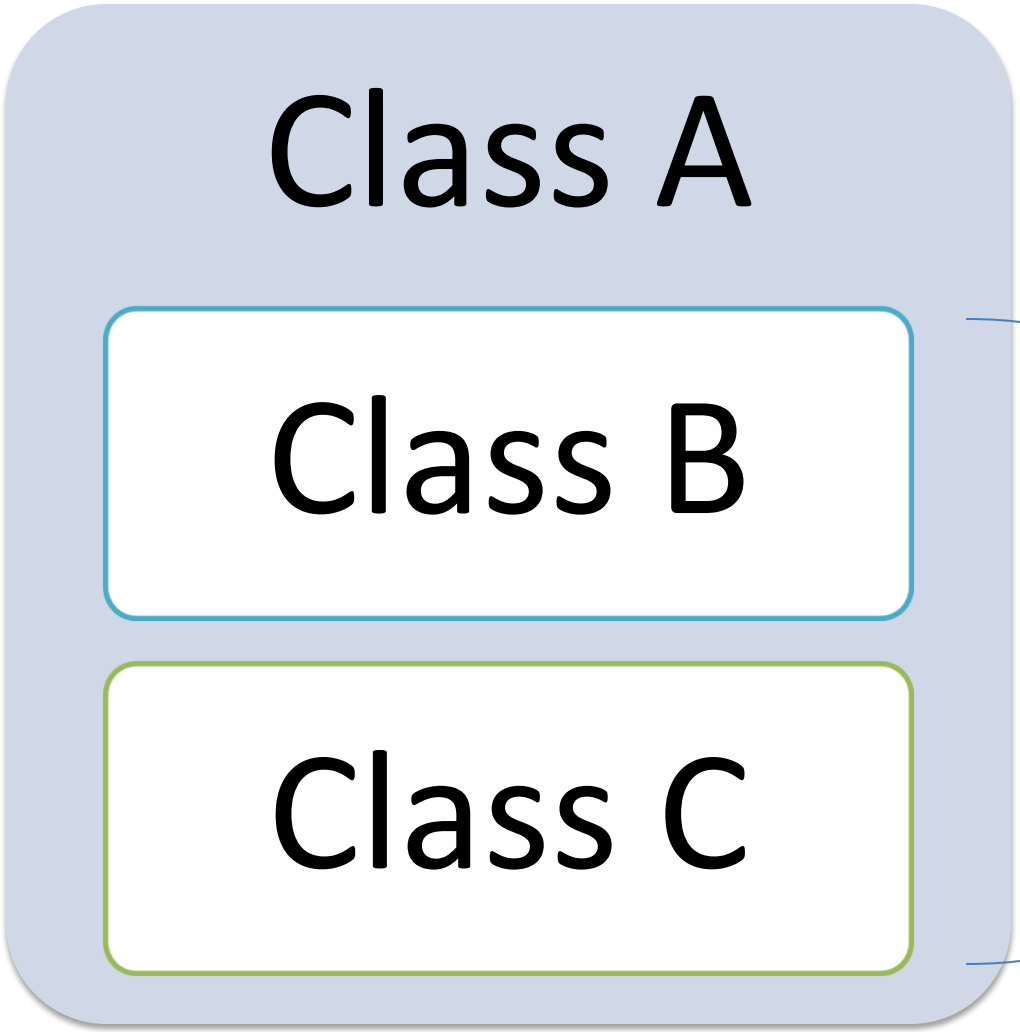
Родительский (базовый) класс

Дочерние (производные) классы

Иерархия классов

Вложение

Включающий (внешний, объемлющий) класс



Вложенные (внутренние) классы

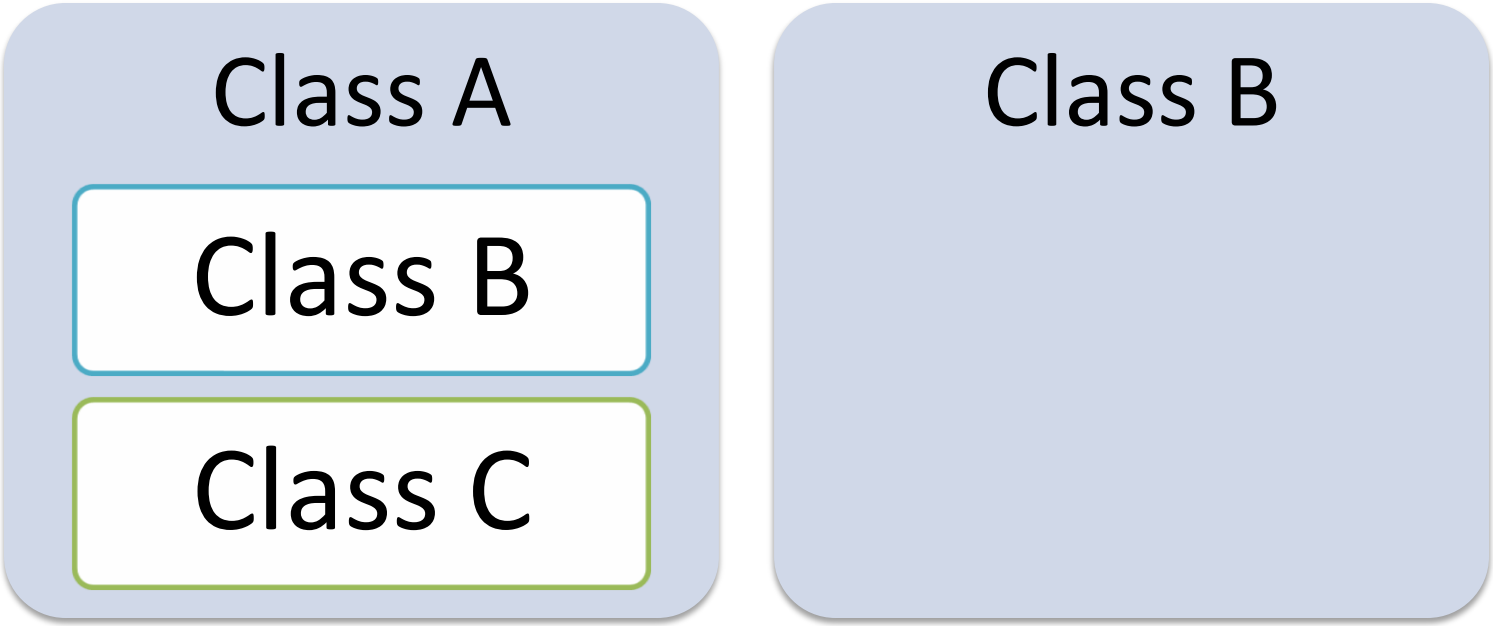
Вложенный класс

Класс, полностью определенный внутри другого класса

- член включающего класса
- объекты вложенных классов обычно не существуют без объекта включающего класса
- нужны в том случае, если вложенный класс поставляет некую вспомогательную структуру для включающего класса, которая нигде больше не будет использоваться

Вложенный класс

Имя вложенного класса находится в области видимости включающего класса
=> не конфликтует с именами не вложенных классов



Определение вложенного класса

1) inline

```
class Surround {
    class Inner {
    public:
        int value;
        int getValue( ) const { return value; }
    };
    int data;
    Inner* obj;
};
```

Определение вложенного класса

2) в глобальной области видимости

```
class Surround {
    class Inner;   // нужно предварительное объявление
    Inner* obj;   // можно объявлять указатели или ссылки
    int data;
};

class Surround::Inner {    // нужен квалификатор внешнего класса
public:
        int value;
        int getValue( ) const { return value; }
};
```

Определение членов

1) inline

```
class Surround {
    class Inner {
    public:
        int value;
        int getValue( ) const { return value; }
    };
};
```

Определение членов

2) в глобальной области видимости

```
class Surround {
    class Inner {
    public:
        int value;
        int getValue( ) const;
    };
};

int Surround::Inner::getValue( ) const {
        return value;
}
```

Спецификаторы доступа

Вложенный класс можно объявить в области private, public или protected

public	• вложенный класс виден **вне** включающего класса
protected	• вложенный класс виден в классах, **унаследованных** от включающего
private	• вложенный класс виден **только членам** включающего класса

Public (пример)

```
class Surround {
    int value;
public:
    Surround( ) : value(10) { }
    class Inner {
        int innerValue;
    public:
        Inner( ) : innerValue(20) { }
        int getInnerValue( ) const { return innerValue; }
    };
    int getValue( ) const { return value; }
};
void f( ){
     Surround::Inner d;        // OK
     std::cout << d.getInnerValue( ) << std::endl;
}
```

Private (пример)

```
class Surround {
    int value;
    class Inner {
        int innerValue;
    public:
        Inner( ) : innerValue(20) { }
        int getInnerValue( ) const { return innerValue; }
    };
public:
    Surround( ) : value(10) { }
    int getValue( ) const { return value; }
};
void f( ){
     Surround::Inner d;          // Error
     std::cout << d.getInnerValue( ) << std::endl; // Error
}
```

Доступ к членам вложенного класса

- члены вложенного класса не являются членами включающего класса
- чтобы объект включающего класса имел доступ к членам вложенного класса, он должен содержать объект вложенного класса
- объявление вложенного класса должно идти до объявления поля этого типа

```
…
class Inner { };
Inner inObject;    // OK
```

```
…
Inner inObject;    // Error
class Inner { };
```

Доступ к открытым членам (пример)

```
class Surround {
    class Inner {
        int innerValue;
    public:
        Inner( ) : innerValue(20) { }
        int getInnerValue( ) const { return innerValue; }
    };
    Inner inObject;
public:
    Surround( ) { }
    int getValue( ) const { return inObject.getInnerValue( ); }
};
void f( ){
    Surround s;
    std::cout << s.getValue( ) << std::endl;
}
```

Доступ к закрытым членам (пример)

```
class Surround {
    class Inner {
        friend class Surround;  // Surround видит все поля Inner
        int innerValue;
        int getInnerValue( ) const { return innerValue; }
    public:
        Inner( ) : innerValue(20) { }
    };
    Inner inObject;
public:
    int getValue( ) const { return inObject.getInnerValue( ); }
};
void f( ){
    Surround s;
    std::cout << s.getValue( ) << std::endl;
}
```

Традиционный подход к доступу

- вложенный класс находится в private
- все данные вложенного класса – public

  (не нужен дружественный включающий класс)

```
class Surround {
    class Inner {
    public:
        int value;
    };
};
```

Доступ к членам включающего класса

- члены включающего класса не являются членами вложенного класса
- для доступа требуется указатель или ссылка на объект включающего класса или сам объект включающего класса
- доступ к статическим членам – через класс-друг
- доступ к нестатическим членам – через указатель, ссылку или сам объект

Доступ к статическим членам включающего класса (пример)

```
class Surround {
   const static int value = 100;
   class Inner {
   public:
      int getValue( ) const { return value + 11; }
   };
   Inner* inObject;
   friend class Inner;    // Вложенный класс объявляется другом,
                                          но может работать и так
public:
   int getValue( ) const { return inObject->getValue();}
};

void f( ){
    Surround s;
    std::cout << s.getValue( ) << std::endl;
}
```

Доступ к нестатическим членам включающего класса (пример)

```
class Surround {
   int value = 100;
   class Inner {
   public:    // Далее доступ идет через указатель или ссылку на объект
      int getValue(const Surround* s) const { return s->value + 11; }
   };
   Inner* inObject;
public:
   int getValue( ) const {
         const Surround* ss = this;
         return inObject->getValue(ss); // return inObject->getValue(this);
   }
};
void f( ){
    Surround s;
    std::cout << s.getValue( ) << std::endl;
}
```

Агрегация

Включение объектов одного класса в объект другого класса

- выражает отношение «быть частью»
- класс содержит член – объект другого класса
- класс получает объект-«часть» извне (например, как параметр)

Композиция

Включение объектов одного класса в объект другого класса

- выражает отношение «быть частью»
- класс содержит член – объект другого класса
- класс полностью управляет временем жизни объекта-«части»

Агрегация и композиция: пример

агрегация

```
class A{
public:
   void f( ) { std::cout << 1; }
};

class B{
private:
   A* a;
public:
   B(A* obj) : a(obj) { }
   void g( ) { a->f(); }
};

void func( ){
   A a;
   B b(&a);
   b.g();
}
```

композиция

```
class A{
public:
   void f( ) { std::cout << 1; }
};

class B{
private:
   A* a;
public:
   B( ) : a(new A) { }
   void g( ) { a->f(); }
};

void func( ){
   B b;
   b.g();
}
```

Выводы

- для разных целей подходят разные виды отношений между классами
- связь между классами:
  - наследование, вложение – сильная связь
  - композиция – связь слабее
  - агрегация – самая слабая связь
- считается, что чем слабее связаны классы, тем лучше

Вопросы?