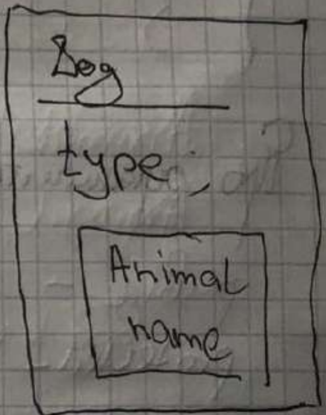
Наследование в ООП (Cpp)

```
class Animal {
public:
  string name;
  Animal (string newname = "") : name(new_name) {};
   sound();

}
```

```
class Dog : public Animal {
public
  string type
  Dog (string new_name = "");
   sound();

}
```



```
int main() {
 Dog Bobik("Bobik")
 Dog noname;
}
```

Конструкторы:

```
Animal();
Dog("Bobik");
```

! $\boxed{\text{При создании объектов неявно вызывается конструктор родителя (по умолчанию)}}$

=> Если есть наследование, то создавай кон-р по умолчанию (без аргументов)

Для вызова конкретного конструктора используем делигацию:

```
class Dog : public Animal {
  public:
    string type
    Dog(string new_name=""): Animal(new_name){}
      sound();
  }
```

Про ограничители доступа:

public - доступ снаружи класса
protected - доступ снаружи класса
private - достун изнутри класса

1) class Dog: public Animal

```
public    -> public
protected -> private
private   -> недоступны ∅
```

2) class Dog: private Animal

```
public -> private
protected -> private
private -> недоступны ∅
```

3) Class Dog : protected Animal
public -> protected
protected -> protected
private -> недоступны ∅

· В наследнике можно переписать ур-нь доступа (кроме private родителя)

```
Class Dog : public Animal {
    ...
    protected
        using Animal::sound(); // переписали на
                                  protected.
}
```

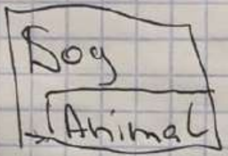
· Удалить ненужный функционал

```
Class Dog : public Animal {
    Sound() = delete;
}
```

```
} Dog Bobik ("Bobik");

Animal creature ("X305");

creature = Bobik; // в пер-ю creature скопируется
                     часть переменной bobik, кот-я
                     относится к Animal
```



```
Animal *creature01 = &Bobik;
Animal &creature = Bobik;          – равносильно
                                     (по смыслу)
Bobik.sound();            // sound() из Dog
creature01 -> sound();    // ... из Animal
creature02 sound();       // ... из Animal
```

Указатель Animal может указывать на объект Dog.

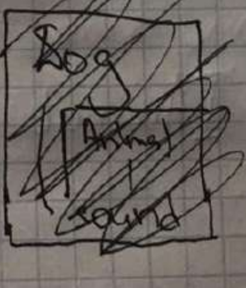
Указатель можно приводить к типу наследника

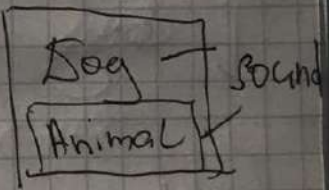
```
Animal* pA = new ~~Animal~~ Dog();
((Dog*) pA) -> sound();
```

· Виртуальные методы:

```
class Animal {
public:
  string name;
public:
  Animal (string new_name = "") : name (new_name) {};
  virtual sound ();
}
```

```
class Dog: public Animal {
public:
  string type;
  Dog (string new_name = "");
  virtual sound () override.
}
```





Если функция в предке — virtual
→ у потомков тоже — virtual

· Чисто виртуальная функция — пустая(= 0;)

· Вирт. функции — позднее связывание

Деструкторы должны быть виртуальными если есть наследование (для уничтожения объекта через указатель на родителя)

· final — запрещает в дальнейшем наследовать класс и переопределять функцию.

```
2) virtual sound () override final {..};
1) class Dog final : public Animal {...};
```

Абстрактные классы:

Идея интерфейса:

пишем то, что класс должен делать, но без реализации

```
class IAnimal {
public:
  string get_name() {};
  virtual void sount ()=0 ;
}
```

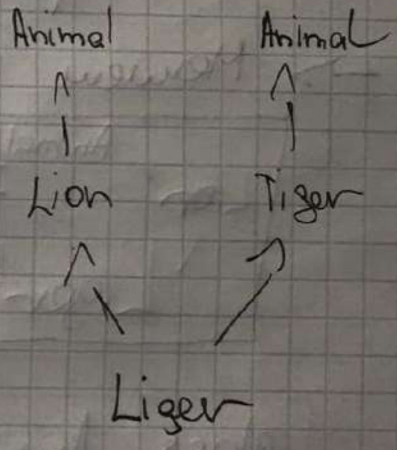
страктный класс - класс, у которого есть хотя бы один чисто виртуальный метод

Чисто абстрактный класс - все методы чисто виртуальные.

· Объекты абстрактного класса не могут быть созданы (ошибка компиляции)

→ class Animal : public IAnimal { ... }

Множественное наследование:



$$\begin{tikzcd} \text{Animal} & & \text{Animal} \\ \text{Lion} \arrow[u] & & \text{Tiger} \arrow[u] \\ & \text{Liger} \arrow[ul] \arrow[ur] & \end{tikzcd}$$

```
class Animal {
  string name;
}
```

```
class Tiger : public Animal {
  int tail_length;
}
```

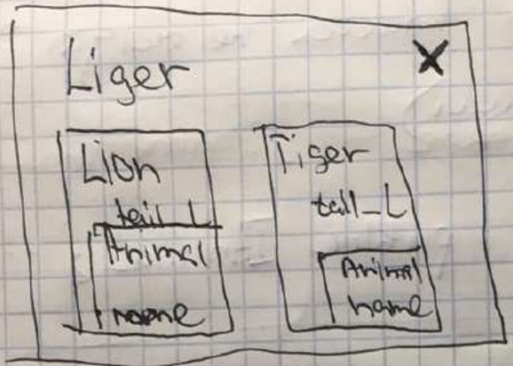
```
class Lion : public Animal {
  int tail_length;
}
```

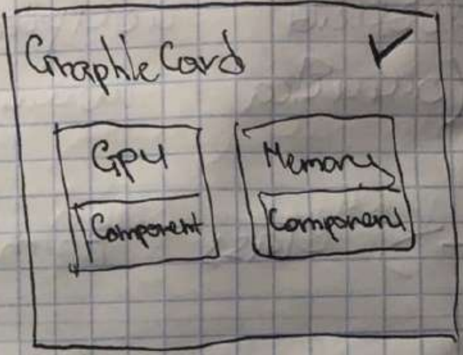
```
class Lige: public Lion, public Tiger {
   -
   }
```

→ Проблемы:

1) Два экземпляра Animal внутри Liger

2) Два одинаковых поля tail_length

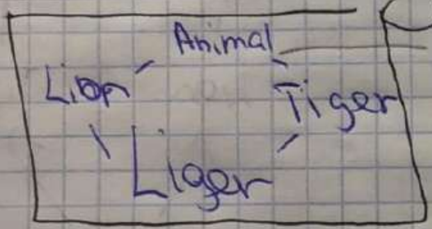




проблемы

все ok

→ Меняем схему наследования



Виртуальный базовый класс

- Виртуальное (ромбовидное) наследование

```
class Animal {
    String name;
    int tail_length;
}
```

```
class Tiger : virtual public Animal {
    ...
}

class Lion : virtual public Animal {...}

class Liger : ~~[illegible]~~ public Lion
    public Tiger {...}
```

· friend - односторонняя дружба

Ключевое слово позволяет получать доступ к private - секции извне для отдельных функций / методов или для классов

· Анонимные (Временные) объекты

```
Dog ("Bobik"); // создание анонимного объекта
Dog bobik = Dog ("Bobik");
Dog ("Bob"). sound(); // можно вызвать метод
                         анонимуса
```