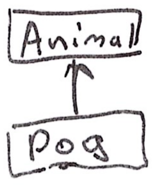
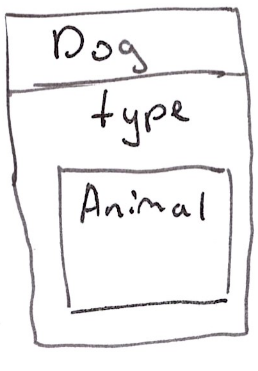
Наследование в ООП

```
class Animal {
  public:
    string name;
  public:
  → Animal(string new_name=" "): name(new_name) {};
    sound();
}
class Dog: ~~p~~ public Animal {
   public:
      string type;
   public:
      Dog(string new_name=" ");
      sound();
}
int main(){
 → Dog bobik ("Bobik"); // Dog ("Bobik")
   Dog noname;          // Dog();
}
```





Конструкторы:
  Animal();
  Dog ("Bobik");

При создании объектов неявно вызывается конструктор родителя (без аргументов).

⇒ Если есть наследование, то следует создать констр без аргументов.

Если необходимо вызвать конкретный конструктор, то требуется:

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

<u>Ограничители класса</u>:

public — доступны снаружи класса

protected — доступны снаружи класса.

private — доступны <u>только</u> внутри класса.

<u>Наследование:</u>

```
Class Dog: public Animal {
    public => public
    protected => protected
    private => недоступен.
```

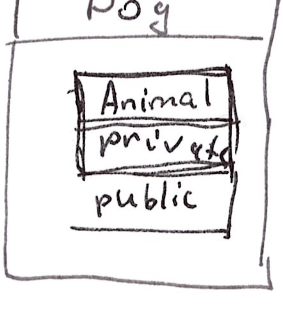
} изменение доступа при наследовании

```
Class Dog: private Animal {
    public
    protected } => private
    private => недоступен
```

} изменение доступа при наслед.

```
class Dog: protected Animal {
    public
    protected } => protected
    private => недоступ.
```

} измен. доступ. при насл.



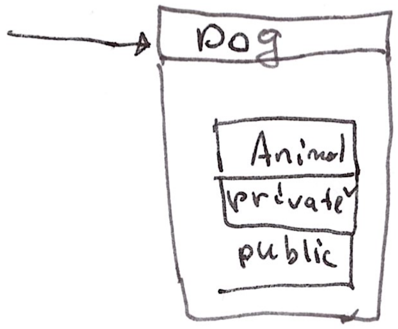
Можно в наслед. переписать уровень доступа. Но private переписать нельзя

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

}
```

//Переписали на protected уровень доступа в Dog к методу sound();

Сокрытие или удаление функционала:

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

}
```



```
Dog bobik ("Bobik");
Animal creature ("X305");
    creature = bobik;
```

В пер. creature скопир. часть перем. bobik, которая относ. к Animal

```
Dog bobik ("Bobik");
Animal *creature01 = & bobik;
Animal & creature02 = bobik;

    bobik.sound(); // sound из Dog
    creature01 -> sound(); // sound() из Animal
    creature02 . sound(); // sound() из Animal
```

Можно представить указатель из класса Animal как указатель на класс Dog (если он действ. указывает на объект класса Dog)

! Если одна из функций опред. как виртуальная, то она д.б. виртуальной ~~во~~ у всех потомков.

Виртуальные ф-ии это позднее связывание

! Деструкторы д.б. виртуальными, если есть наследов.

final - запрещает в дальнейшем наследовать класс.

```
    virtual sound() override final;
```
больше пер. ее нельзя.

```
  class Dog final: public Animal {
  ...
  }
```
нельзя создавать насл. класса Dog.

Абстрактные классы.

Идея интерфейса: мы пишем, что класс должен делать (методы, переменные), в ~~ф~~ виде кода - код рабочий (Компилиру но без фактической реализации.

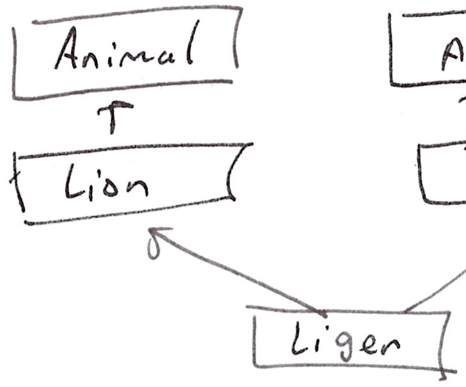
```
class Animal{
  public:
    string get_name() {};
    virtual void sound () = 0; // ф-ия явл. абстрактной
}
```

Абстрактный класс - класс, у которого есть хотя бы одна абстрактная ф-ия

Объекты абстрактного класса не создаются
(ошибка компиляции.)

Множественное наследование:

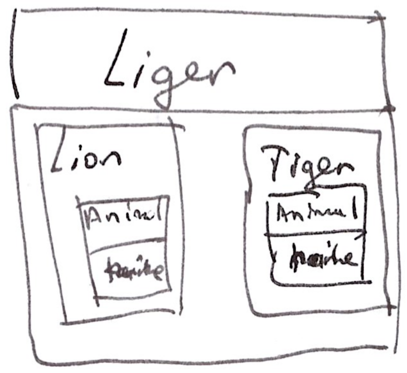
```
class Animal{
   string name;
}
class Tiger: public Animal{
    int tail_lenght;
}
class Lion: public Animal{
    int tail_lenght;
}
class Liger: public Lion, public Tiger{
}
```



Проблемы:

① Два экземпляра Animal внутри класса.

② Два одинак. поле tail_lenght.



Нужно делать так, чтобы не было неоднозн., т.е. не должно быть одинаковых полей и методов у непосредств. родителей.

Дружественные ф-ии и классы.

```
class Graph {
  private:
     Node* Root
  public
     Node * search (Node * node);
}
class Node {
private:
     void *data;
     std::list<Node*> neighbords;
     friend class Graph;  // Теперь из class Graph будут
}                                видны поля интерф. class Node
```

Наследование:

① Модификаторы доступа при насл.

public
private
protected

② Порядок вызова конструкторов при наследовании:

③ Порядок вызова деструкторов при наслед.

④ Повышающее приведение типов (к наслед)

⑤ Понижающее приведение типов (к родит.)

⑥ Обрезка объектов (animal = dog);

⑦ Virtual методы. (virtual деструктор)

⑧ Абстрактный класс (интерф.)

⑨ Множеств. наследование. (вирт. классы)

⑩ Друж. методы и друж. классы

⑪ Анонимные объекты;

```
Dog ("Bobik"); //созд. аноним. объекта
Dog bobik = Dog("Bobik");
Dog ("Bobik").sound;
```