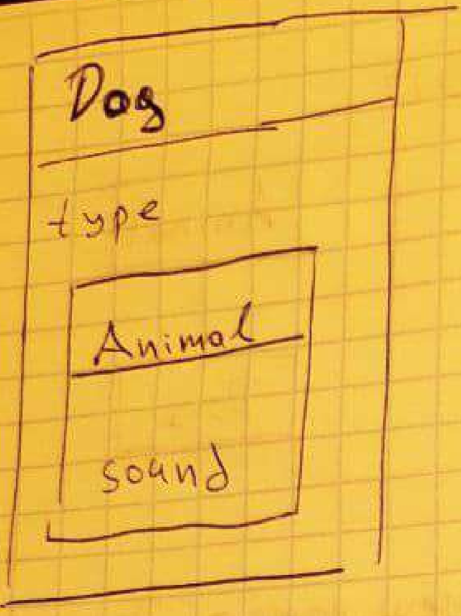
Наследование в ООП

$$\begin{tikzcd}\boxed{\text{Animal}} \\ \boxed{\text{Dog}} \arrow[u]\end{tikzcd}$$

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

```
class Dog: public Animal {
  public:    string type;
  public:
    Dog (string new-name = " ");
    Sound ();
}
```

```
int main() {
   Dog bobik ("Bobik"); // Dog("Bobik)
   Dog noname
```



! при создании объектов неявно вызывается конструктор родителя (без аргументов)

констр-ры:

```
Animal();
Dog("bobik");
```

Если есть наследование, то созд. констр-р без аргументов

Но если хотите вызвать конкретный констр-р, то:

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

```
    Sound();
 }
```

Ограничители доступа:

public — доступны снаружи класса

protected — доступны снаружи класса

private — доступны только изнутри

Наследование

```
class Dog: public Animal {
      public => ~~protected~~ public
      protected => protected
      private => ~~private~~ недоступен
```

private

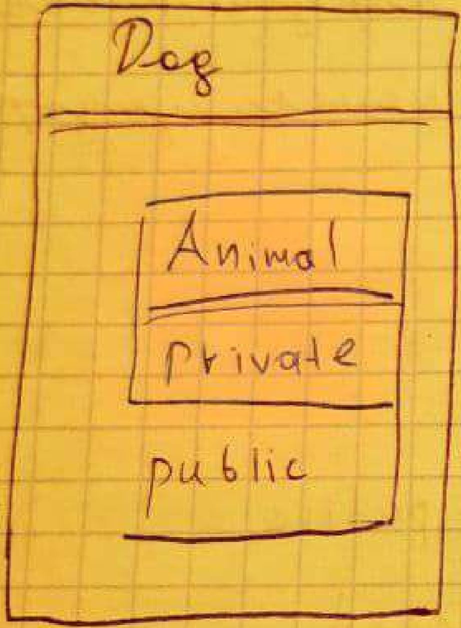
```
public => private
protected => private
private => недоступен
```

protected

public => protected

protected => protected

private => недоступен



Можно в наследнике переписать уровень доступа (кроме private)

```
class Dog : public Animal {
  —||—
using Animal::sound;
}
```

Сокрытие или удаление

—||—

```
sound() = delete; // скрыли метод
                     из Dog
```

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

  creature = bobik; // в creature
                       скопируется
                       часть bobik,
                       д отн. к Animal
```

```
Dog bobik("Bobik");
Animal *creature01 = &bobik;
Animal &creature02 = bobik;
  bobik.sound(); // sound из Dog
  creature01 → sound(); // из Animal
  creature02.sound(); // из Animal
```

! можно представить указатель
на класс Animal как указатель на
класс Dog (если он указывает на
Dog)

dynamic_cast <Dog*> (creature)
// из Dog

```
class Animal {

}

class Dog : Animal {
}

void sound_3_times (Animal &creature) {
}
```

/ Dog => sound из
Animal

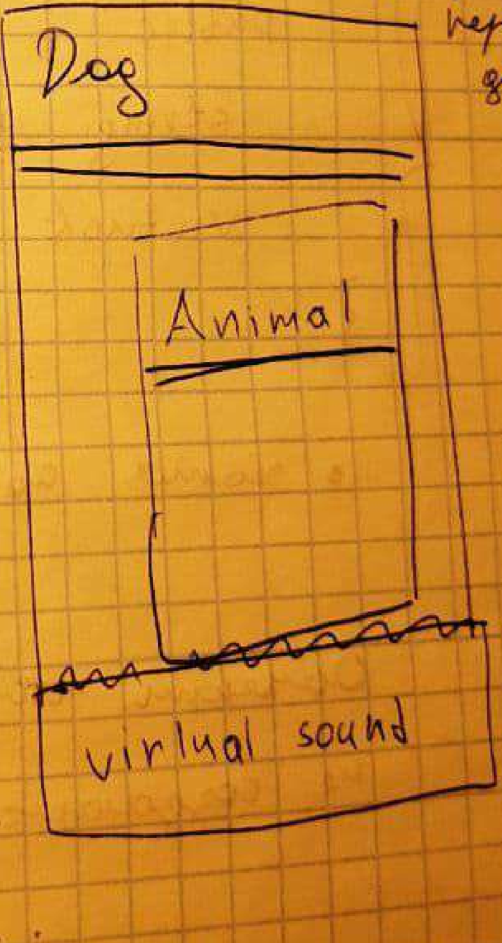
виртуальная функция ↓

```
class Animal {
    :
    virtual sound();
}
```

```
class Dog
    :
    virtual sound(); | override | | final |
}
```

✓ переопределение

✗ запрет переопределения далее



! если ф-ция вирт., то она должна быть вирт. у всех потомков

! деструкторы должны быть вирт. если есть насл-ие.

Абстрактные классы

идея: вы пишете что класс должен делать (методы, переменные) в виде кода - код компилируется но без фактической реализации

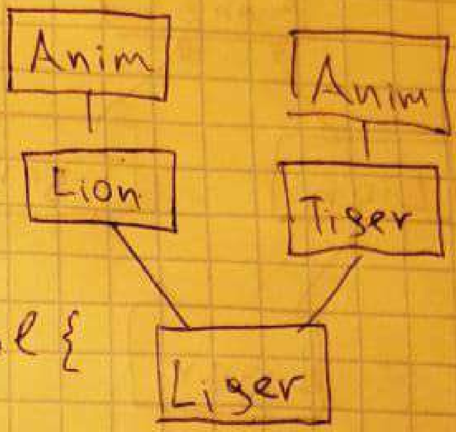
```
class IAnimal {
    public
      string get_name() {};
      virtual void sound() = 0 ;  - абстр.
                                     ф-ия
}
```

- хотя бы одна абстр. ф-ия

Объекты абстрактного класса не создаются

Множественное наследование:

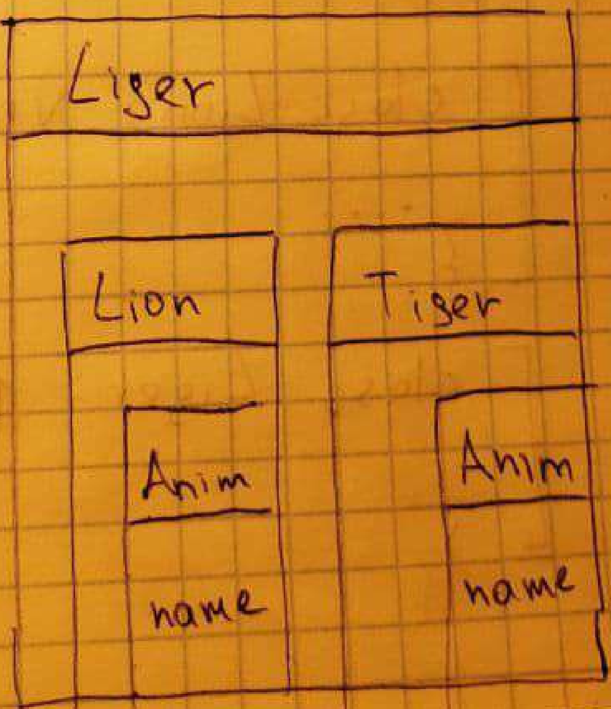
```
class Animal {
    string name;
}

class Tiger: public Animal {
    int tail_length;
}

Class Lion: public Animal {
    int tail_length;
}

Class Liger: public Lion, public Tiger {
  —
}
```

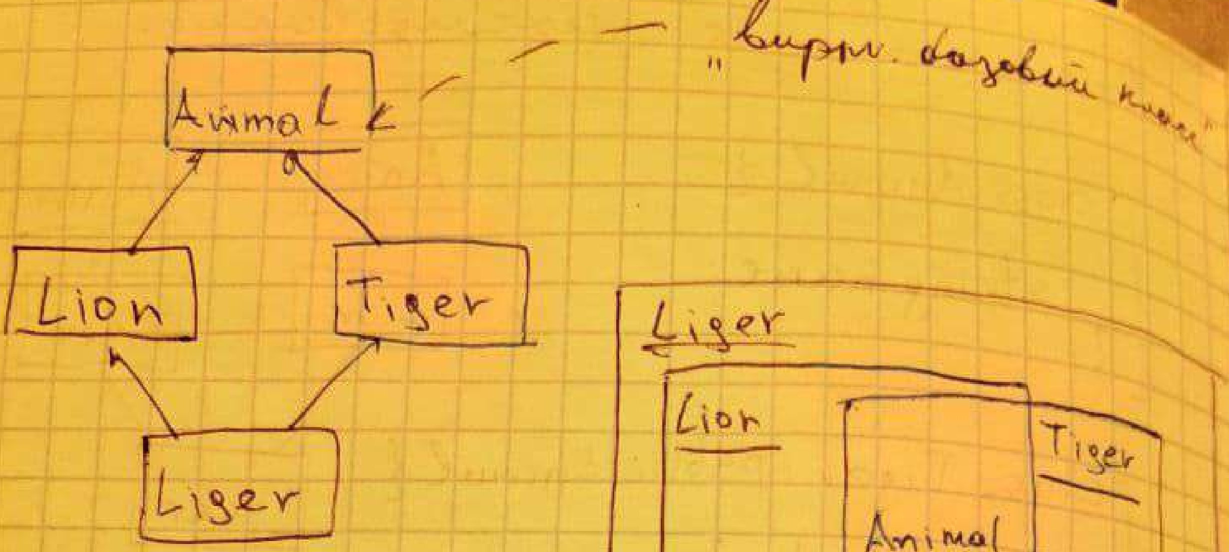


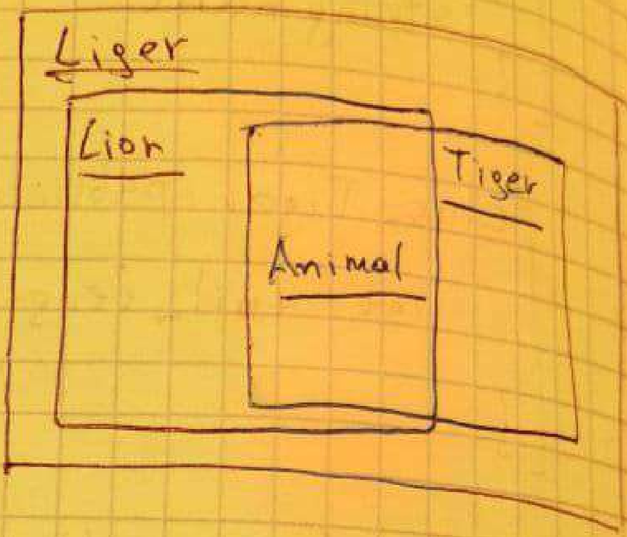
Проблемы:

① 2 экземпляра Animal

② 2 одинаковых поля tail_length

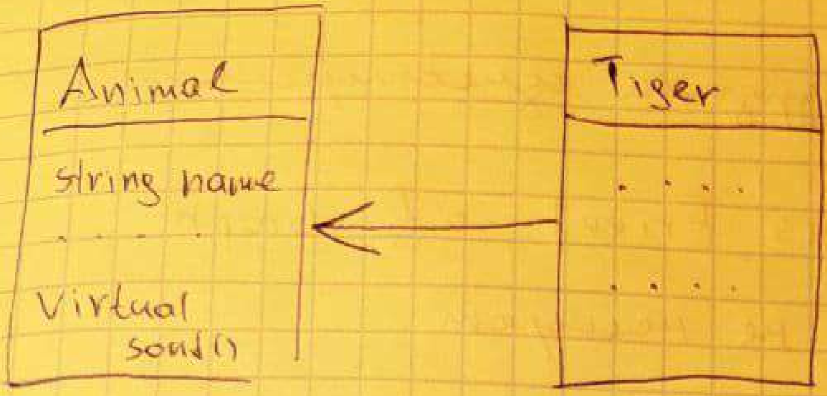






```
class Animal {
    string name;
    int tail_length;
}

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

class Lion : virtual public Animal {
    ....
}

class Liger : public Lion, public Tiger {

}
```

UML



Дружественные функции и классы

```
class Graph {
  private:
    Node* Root
 public
   Node * Search (Node *node);
}
```

```
class Node {
  private:
     void* data;
     std::list<Node*>neighbors;
     friend class  Graph; // теперь из class Graph
                             будут видны поля
                                    class Node
}
```

friend – дружба односторонняя

Node is Friend of Graph

но не наоборот

A – friend of B

B – friend of C

A – не друг C

```
class Graph {

  ...

}

class Node {
  ...

friend Node* Graph::Search(Node* node);
}
```

Анонимные объекты:

```
Dog("Bobik"); // создание анон. объекта

Dog bobik = Dog("Bobik");
Dog("Bobik").sound();
```