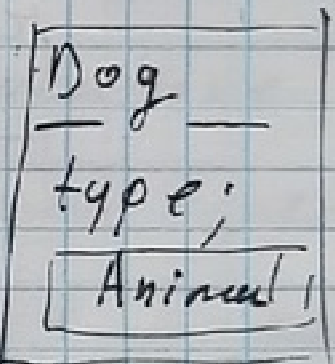
Наследования в ОПП

```
Class Animal {
  public
    string name;                     конструктор аним.
                                                 name
  publ                                           ↓
    Animal (string new_name = " "): name(new_name)
                                                 {}
    sound();
}
Class Dog: public Animal {
  public          ←———— наслед от Animal
    string type;
    Dog (string new_name = " ");
                            ↑
    sound();            зн. констр по умолч
}
```



! При созд. объектов неявно выз. констр. родителя (в нашем случае Animal)

Если есть наследование то обяз. созд. <u>констр. без аргументов</u>

Но можно вызвать и конкретный констр

```
Dog ( string new name ....) : Animal(new_name) {};
```

Огранич. доступа.

public – д. снаружи класса

protected – о —//—

private – только изнутри класса

При наследовании

Class Dog: public Animal

недоступен только privet Animal

—//—— : private Anima

public → private

protected → private

privet → недост.

Можно в наследовании перемешать уровни доступа

```
class Dog: public Animal {
    —//—
    —//—
   protected
    usins Animal :: sound
```

public

sound() = delete; ← ~~скры~~ удаление метода из класса Dog

Dog bobik("bobik")
Animal creature("ХВОЗ")
-//-
creature = bobik; в переменную creature скопируется часть перем. bobik кот. отн. к Animal

Animal *craature01 = &bobik; - указ.
Animal &creature02 = bobik; - превдоним
bobik.sound(); - из Dog
creation01 → sound(); - из Anima
creation02.~~→~~ sound() - из Animal

Можно преобр. указ. Animal как указ. на Dog
Констр.
dynamic_cast < Dog* > (craction)).sound()};

Можно ~~возд~~ созд. virtual ф-цию
и она будет общей для всех
но она должна быть виртуальной
у всех потомков

Вирт. ф-ция - позднее связывание
(медленнее работает)

Деструктор — виртуальным если есть
наследование

[virtual sound(~~[illegible]~~); override final
переопределяет

class ... final : public ... {}
запр наслед. класса Dog

Абстрактный класс

Идея: созд. методы и переменные но без факт. реализации.

```
Class IAnimal {
    —//—
    virtual void soun() = 0;   - ф-ция абстр.
```

Множественное наследие

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

```
Liger : class Liger: public Lion, public Tiger{
}
```

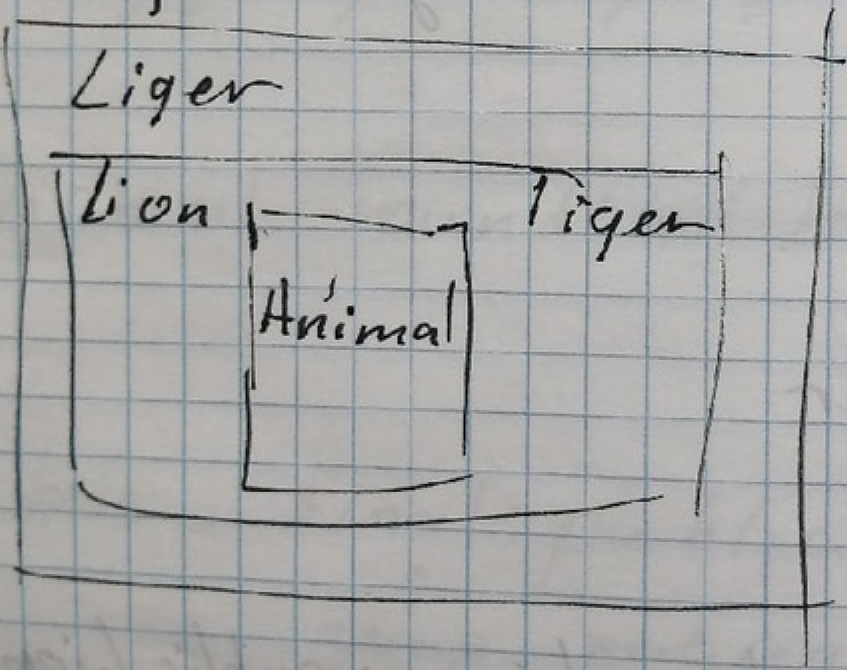
Проблемы

Дублируются поля и экземпляры

Решение

Или не допускать повторы или делать Animal- virtual

```
class Animal {
  ...
}
class Tiger: virtual public Animal {
 ...
}
class Lion: virtual public Animal {
 .. }
class Liger: public Lion, public Tiger {
 -- }
```



Дружественные ф-ции и классы

```
class G {
 private
   Node *root;
 public
   Node * Search(Node * node);
}

class Node {
 private
   void *data
   list <Node*> neighbous;
   friend class G; // теперь из G видны
```

односторонняя поля Node
(практ. один class)

Node is friend of G

но из Node private методы и поля G недоступны

$A \rightarrow B$

$B \rightarrow C$

$A \not\rightarrow C$

```
class G {
 —//—
   Node*  search(Node* note)
}
class Node {
 —//—
friend Node* G::search(Node* note
}
```

друзья т.е. ~~из~~ дост private поля Node

Анонимные объекты

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

Обращение

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