

# Конспект лекции от 29.11.2021

Фиронов Алексей

## Наследование в Объектно-Оrientированном Программировании

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

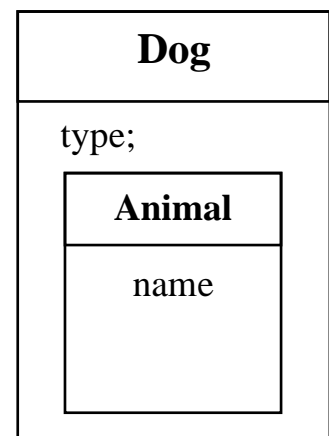
Одним из вариантов инициализации конструктора может быть инициализирован аргументом *new\_name*:

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

В данном случае класс *Dog* наследует параметры класса *Animal*.

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

При создании объекта класса *Dog* неявно вызывается конструктор класса *Animal* (класса родителя) и это происходит по умолчанию.



При необходимости вызова конкретного конструктора родителя – используется следующий синтаксис:

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

## Уровни доступа.

Различают 3 модификатора доступа к параметрам класса:

- `public` – разрешён доступ из внешней программы;
- `protected` – разрешён доступ дружественным или наследующим классам;
- `private` – доступ разрешён только внутри класса.

```
class Dog: public Animal{  
    public -> public  
    protected -> protected  
    private -> not available  
};
```

```
class Dog: protected Animal{  
    public -> protected  
    protected -> protected  
    private -> not available  
};
```

```
class Dog: private Animal{  
    public -> public  
    protected -> private  
    private -> not available  
};
```

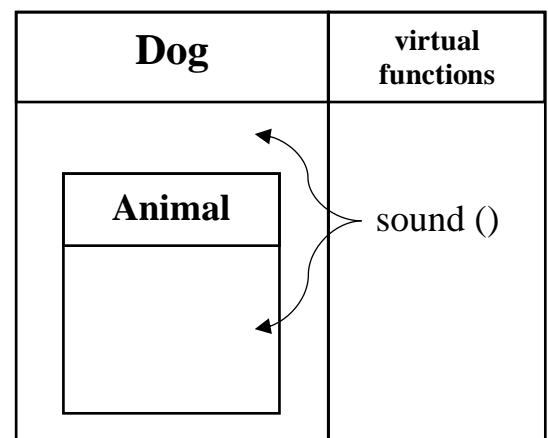
Указатель класса *Animal* может быть представлен как указатель на класс *Dog*.

## Виртуальные функции.

Виртуальные функции, определённые в классе-родителе, должны виртуально присутствовать в дочерних классах. При этом, виртуальные функции являются общими для всех классов, в то время как неvirtуальные могут дублироваться.

**Деструктор должен быть виртуальным для класса-родителя.**

Модификатор `final` запрещает дальнейшее наследование класса и изменение заданных функций.



## Абстрактные классы. Интерфейсы.

Основной функцией интерфейса является описание действий участка кода без непосредственной реализации функций.

Для этого используются абстрактные классы:

```
class Animal {  
private:  
    string name;  
    int age;  
public:  
    string get_name() {}  
    virtual void sound() = Ø  
}
```

В данном случае функция *sound* – абстрактная функция.

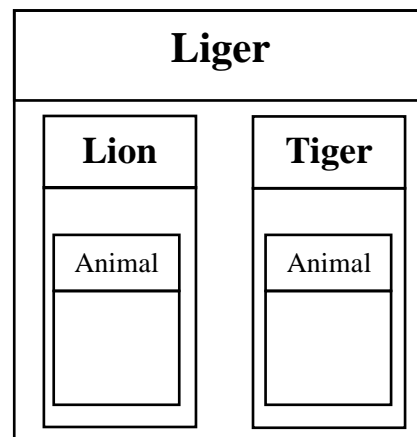
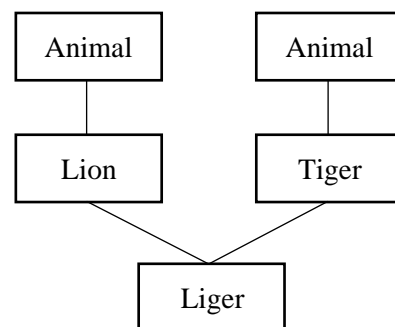
Класс называется абстрактным, если содержит в себе хотя бы 1 абстрактную функцию.

Создание объекта абстрактного класса приводит к ошибке компиляции.

## Множественное наследование.

Множественное наследование – это параллельное наследование аргументов из 2 или более классов-родителей.

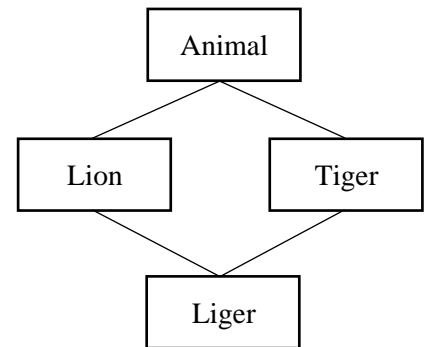
```
class Animal{  
...  
}  
  
class Tiger: virtual public Animal{  
...  
    int tail_length;  
...  
}  
  
class Lion: virtual public Animal{  
...  
    int tail_length;  
...  
}  
  
class Liger: public Lion, public Tiger{  
...  
}
```



Мы получили 2 проблемы:

- Два раза мы используем класс `Animal`;
- Два поля с именем `tail_length`.

В связи с этим появляется необходимость делать так, чтобы не появлялось неопределённости, вызванной этими двумя проблемами.



Для этого базовый класс необходимо сделать виртуальным (прописать для дочерних классов *virtual*).

### Дружественные классы и функции.

Дружественные функции и классы – это структуры, имеющие доступ к защищённым структурам другого класса.

```
class Graph {
private:
    Node *root;
public:
    Node * search (Node *node);
};

class Node{
private:
    void *data;
    std::list <Node*> neighbors;
friend class Graph;
};
```

В данном случае из класса `Graph` видны структуры класса `Node`.

### Анонимные объекты.

Отличительной особенностью анонимных объектов является то, что они не хранятся в памяти.

```
Dog ("Bobik");// Создание анонимного объекта
Dog ("Bobik").sound(); // Обращение к объекту
```