Ограничители доступа:

public - доступен снаружи класса

protected - доступен снаружи класса

private - доступен только внутри класса

Наследование

Class Dog : public Animal {

public => public

protected => protected

private - недоступен

} изменения доступа при наследовании

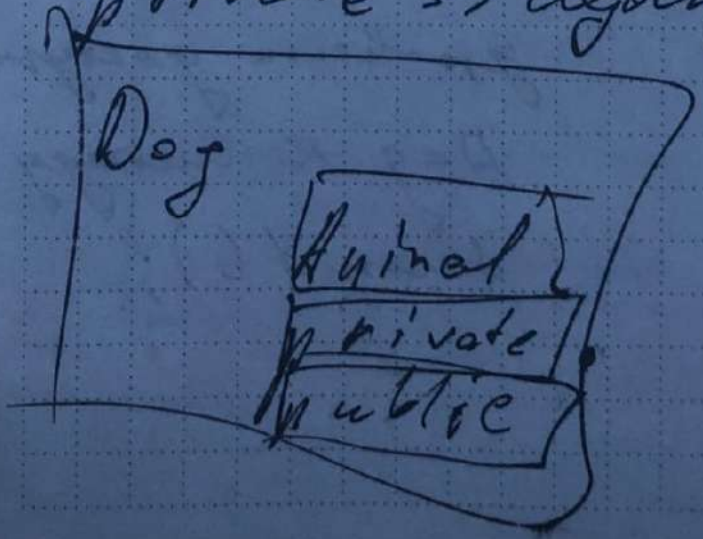
Class Dog : private Animal {

public => private

protected => private

private => недоступен

} изменение доступа при наследовании

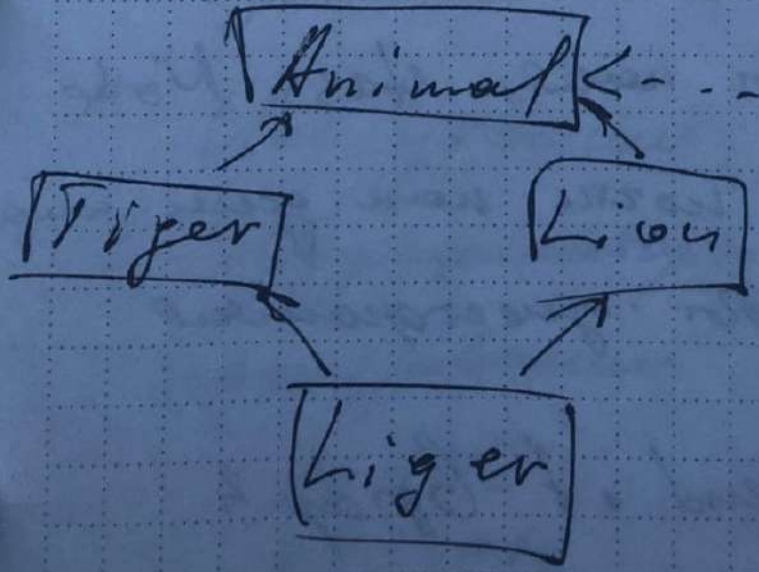


Проблемы:

1) Два экземпляра Animal внутри класса

2) Два одинаковых поля tail_length

Нужно делать так, чтобы не было неоднозначностей, т.е. не должно быть одинаковых полей и методов у непосредственных родителей. (и вообще у предков)

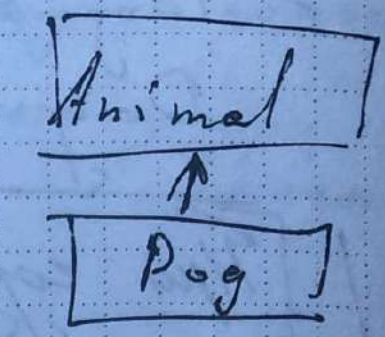


виртуальной базовый класс

Наслідування в ООП

```
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; // Dog();

При создании объектов неявно вызывается конструктор умолчания (который без аргументов)

↓

Если есть наследование, то создавайте конструктор без аргументов

Но если мы хотим вызвать конкретный конструктор то:

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

Но без фактической реализации.

```
class I Animal{
public:
string get_name() = {};
virtual void sound() = 0;
// функция чист. абстрактная
```

Абстрактный класс - класс у кот. есть хотя бы одна абстрактная функция. Объекты абстрактного класса не создаются.

final - запрещает в дальнейшем наследовать класс и переопределять функцию:

```
virtual sound() override final;
```

↑ Больше переопределять ее нельзя

```
class Dog final : public Animal{
....
}
```

↑ Нельзя создавать наследников класса Dog.

<u>Абстрактные классы:</u>

Идея интерфейса: вы пишете что класс должен делать (методы, переменные), но в виде кода - код рабочий (компилируется),

② Порядок вызова конструкторов при наследовании

③ Порядок вызова деструкторов при наследовании

④ Повышающее приведение типов (к базе)

⑤ Понижающее приведение типов (к типу)

⑥ Перегрузка объектов (animal = dog);

⑦ virtual методы (virtual деструктор)

⑧ абстрактный класс (интерфейсы)

⑨ Множественное наследование (виртуальные классы)

⑩ Дружественные методы и дружественные классы.

⑪ Анонимные объекты

У Mod - private методи и поля
Graph недоступны

A - friend of B
B - friend of C
A - не друг C

Анонимные объекты

Dog ("Bobik"); // создание анонимного объекта

Dog bobik = Dog ("Bobik");

Dog ("Bobik").Sound();

Наследование:

① Модификаторы доступа при наследовании

public
private
protected

[Можно в наследнике переписать уровень доступа если нужно то, и переписать методы, т.е private не переписать]

```
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();

Множественное наследование:

```
class Animal {
    string name;
}

class Tiger: public Animal{
  int tail_length
}

class Lion: public Animal {
    int tail_length;
}

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

Можно не представлять учащегося на классах Animal или учащегося на классе Dog Сеттер он действительно указывает на объект класса Dog)

Если вы определили функцию как виртуальную, то она должна быть виртуальной у всех потомков.

Виртуальная функция это подход Связывание

Деструктор должен быть виртуальным если есть наследование

Дружественное функции и класс

```
class Graph{
private:
Node* root;
public:
Node* Search (Node* Node);
}
class Node
private:
void * data;
std::list<Node*> neighbors;
friend class Graph.
```

и теперь у class Graph будут видны поля class Node

т.е. это могли как один класс friend - другого односторонний

Node is Friend of Graph

но не наоборот.