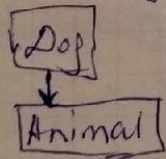


1. Haaregobanue 8 0017

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

```
2. { class Dog: public Animal {  
    public:  
        Dog(); → Dog(string new_name);  
        sound();  
}
```

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

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

// создаётся объект Bobik
класса Dog

При создании объектов можно вызывать конструктор определённого
Если есть намерение, надо создавать конструктор без аргументов

Если хотим вызвать конкретный конструктор определённого:

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

[2]

Ограничение доступа

public — доступен снаружи класса

protected — доступен внутри

private — доступен только внутри класса

Наследование

class Dog: public Animal {

public \Rightarrow public

protected \Rightarrow protected

private \nRightarrow

Доступ при наследовании
public

class Dog: private Animal {

public \Rightarrow private

protected \Rightarrow private

private \nRightarrow

— " —

private

class Dog: protected Animal {

public \Rightarrow protected

protected \Rightarrow protected

private \nRightarrow

— " —

protected

Можно в наследовании переписать уровень доступа
(private недоступен \rightarrow изменить уровень нельзя)

Инкапсуляция — размещение в одном компоненте данных
и методов работы с ними. Позволяет скрыть механизм реализации
от пользователя

Наследование → позволяет создать класс-потомок на основе
на основе уже существующего с переопределенными характеристиками родительского
класса-потомку

Полиморфизм → свойство классов, позволяющее использовать
объекты классов с одним интерфейсом без информации о типе и внутренней
структуре объектов

```
[ Dog Bobik ("Bobik");  
  Animal creature ("Хзос");  
  creature = Bobik; // в переменной creature копируется сам  
                     Bobik, кот. относится к Animal
```

Dog Bobik ("Bobik")

Animal *creature = &Bobik; // указатель

Animal &creature02 = Bobik;

Bobik.sound(); // вызывает sound() у Dog

creature01 → sound(); // вызовет у Animal

creature02.sound(); // вызовет у Animal

можно представить ука-ль класса Animal как указатель
на ^{объект} класса Dog

~~... ..~~


```
(dynamic_cast<Dog*>(creature)).sound();
```

// вызывает sound() у Dog

final запрещает в дальнейшем наследоваться классу, переопределять ф-ию

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

3 Виртуальные ф-ии

Виртуальные ф-ии - виртуальные во всех родственных классах

Виртуальная ф-я - позднее связывание (интерпретатор работает)

Деструктор должен быть виртуальным, если есть наследователи
важно избежать ошибки переопределения случайно при
имеи name ф-ии !

sound() override

так компилятор отловит переопределение

4 Абстрактные классы

Интерфейс заключается в отпавлении класса
и его без реализации самих методов

Данный класс реализуется в виртуальном классе


```

class Animal {
    private:
        string name;
        int age;
    public:
        string get_name() { }
        virtual void sound() = 0; // абстрактная ф-я
}

```

Абстрактный класс - класс, у кот. есть хотя бы одна абстр ф-я

Создание объекта абстрактного класса приводит к ошибке компиляции

5 Множественное наследование:

```

class Animal {
    ...
}

class Tiger: public Animal {
    int tail_length;
}

```

```

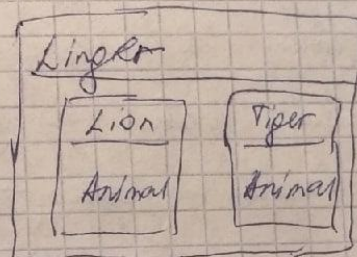
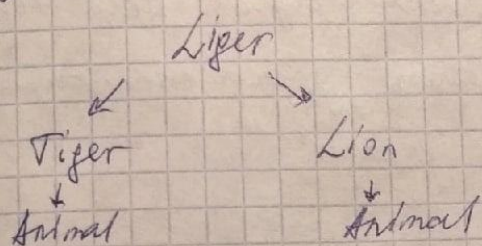
class Lion: public Animal {
    int tail_length;
}

```

```

class Liger: public Lion, public Tiger {
}

```



Проблема:

① 2 экземпляра Animal в одном классе

② 2 экземпляра one tail_length

Не нужно было делать копии, т.е. перегрузить

⑥ Другое решение - использовать класс

```

class Graph {

```

private:

~~Node~~ Node *root;

public

Node * search (Node * node) }


```
class Node
```

```
private:
```

```
void * data
```

```
std::list<Node* > neighbors;
```

```
friend class Graph; #
```

Примечание: в классе Graph есть выходы полей и методы класса Node, поэтому можно сделать представление полей Node friend → односторонняя "дружба"

Анонимные объекты

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

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

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

Анонимные объекты не хранятся в памяти