

# СЕМИНАР 1.7 << НАСЛЕДОВАНИЕ И ИЕРАРХИИ КЛАССОВ >>

C++ унаследовал концепцию иерархий из языка Simula 67.

Каждый хорошо спроектированный класс предназначен для одной конкретной задачи. Хороший пример: `std::vector`; плохой пример: `std::string` (класс БОГА, слишком монолитный).

Класс отражает концепт, но он не находится в изоляции.

Нужно сформулировать варианты отношений класса: (связи)

- Композиция - А часть В, под управлением В, не знает о В;
- Агрегация - А часть В<sub>1</sub>, В<sub>2</sub>, ..., не управляет ими, не знает их;
- Ассоциация - независимые объекты, используют друг друга;
- Зависимость - использование без хранения связи с объектом;

КАК реализуются эти отношения в C++?

Примеры иерархий: живая природа, геометрические фигуры ...

наследование { интерфейса (для чего?) - «является разновидностью»  
реализации (для чего?) - «реализован посредством»

`class D : keyword B`

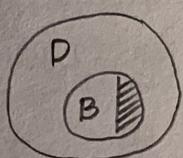
↓

<code>public</code> - открытое <code>private</code> - закрытое <code>protected</code> - защищ.-ое	<b>влияние на секции доступа</b> + особенности работы
---	--

Пример: Employee → Manager, Developer, ...

**Пример** (live) РАБОТА С СЕКЦИЯМИ И ТИПАМИ НАСЛЕДОВАНИЯ

Особенности работы к-ров и д-ров



ПРИ создании экземпляра D создаётся подобъект B

Конструктор: сначала вызывается к-р базового класса

Деструктор: в конце вызывается д-р базового класса

Вспомнить трёхэтапную схему работы к-ра и д-ра.

**Пример** (live) ПОРЯДОК ВЫЗОВА К-РА И Д-РА

**Пример** (live) демонстрация наследования интерфейса

**Пример** (live) демонстрация переопределение интерфейса

**Пример** (live) демонстрация изменения режима доступа

## Использование указателей в иерархиях классов

D\* p-d = &d; // СТАНДАРТИО (аналогично - ссылки)

B - ГАСТ D, => B\* p-B = &d; - это допустимо и полезно

Здесь (за счёт особого размещения в памяти частей базового и производного классов - не углубляемся) p-B знает про то, что есть часть от D, а при B B=d; - обрезка.  
плохо!

P-B { динамический тип D\* } проявление дин-кой типизации  
статический тип B\* }

Теперь можно сделать только указатели на базовый класс, но как работать с частями производных классов? -

## Виртуальные функции - (полиморфизм)

- Ключевые слова virtual, override, final
  - Полное совпадение сигнатур функций
  - Работа с экземплярами через \* или &
  - Допускает изменение реализации
- } демонстрация

**Пример** (live) работа с в. ф. типа print в Employee

Замечания:

↓  
дополнительно operator <<

- нельзя использовать в. ф. в к-рах и д-рах
- деструктор в базовом классе должен быть virtual

Виртуальные функции работают через таблицу виртуальных функций (vtbl). Она создаётся при компиляции и содержит информацию о ветках в. ф. В классе есть указатель на эту таблицу (vptr). Выбор в. ф. в run-time.

## АБСТРАКЦИЯ (абстрактные базовые классы) ~ интерфейсы Java

- при наименование виртуальной функции, которая обязана быть переопределена в производных классах, но нельзя создать экземпляр абстрактного базового класса.

**Пример** (live) демонстрация абстрактного базового класса

Путешествия по иерархии классов вниз: (не лучшая идея)

static-cast - без проверки (есть уверенность)

dynamic-cast - с проверкой (нет уверенности)

## Множественное наследование

```
class B
{
public:
    int m;
    static int sm; - с этим проблем нет
};
```

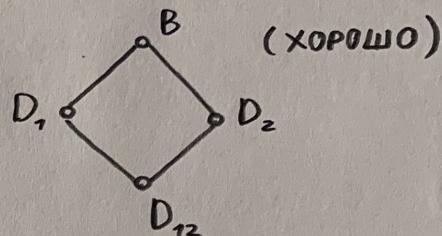
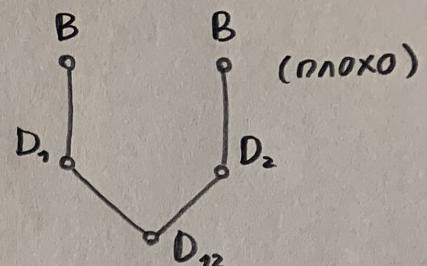
```
class D1 : public virtual B {};
class D2 : public virtual B {};
class D12 : public D1, private D2 {};
```

D12\* pd = new D12;

B\* pB = pd;

pD → m - есть проблема

pD → sm - нет проблем почему?



## ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

Пример factory.cpp

Пример template-method.cpp + NVI

Пример adapter-public.cpp

Пример adapter-private.cpp

## КЛАССИФИКАЦИЯ ПАТТЕРНОВ:

- ГОРОЖДАЮЩИЕ (factory)
- СТРУКТУРНЫЕ (adapter)
- ПОВЕДЕНИЯ (template)