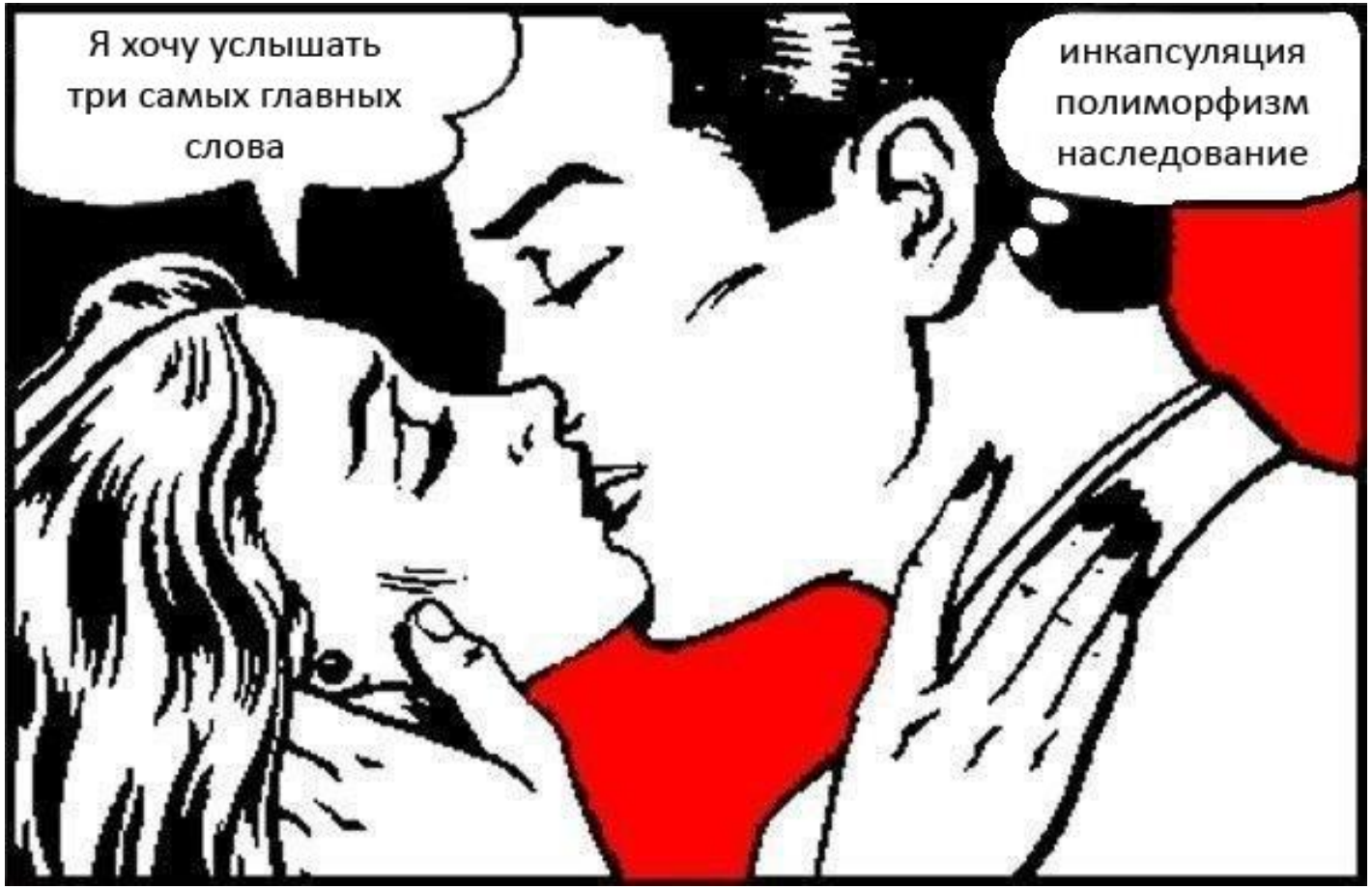
Алгоритмизация и программирование

Лекция 7

Принципы ООП



Инкапсуляция

Инкапсуляция

Без инкапсуляции

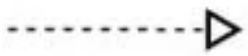
С инкапсуляцией

Инкапсуляция

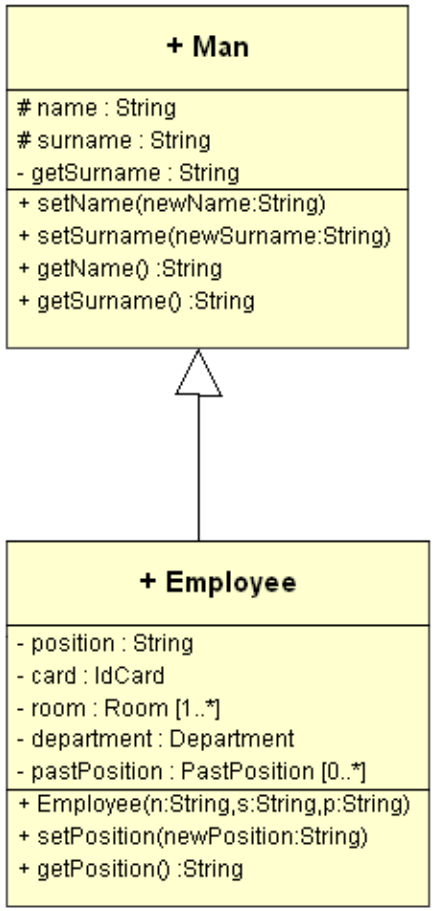
Без инкапсуляции

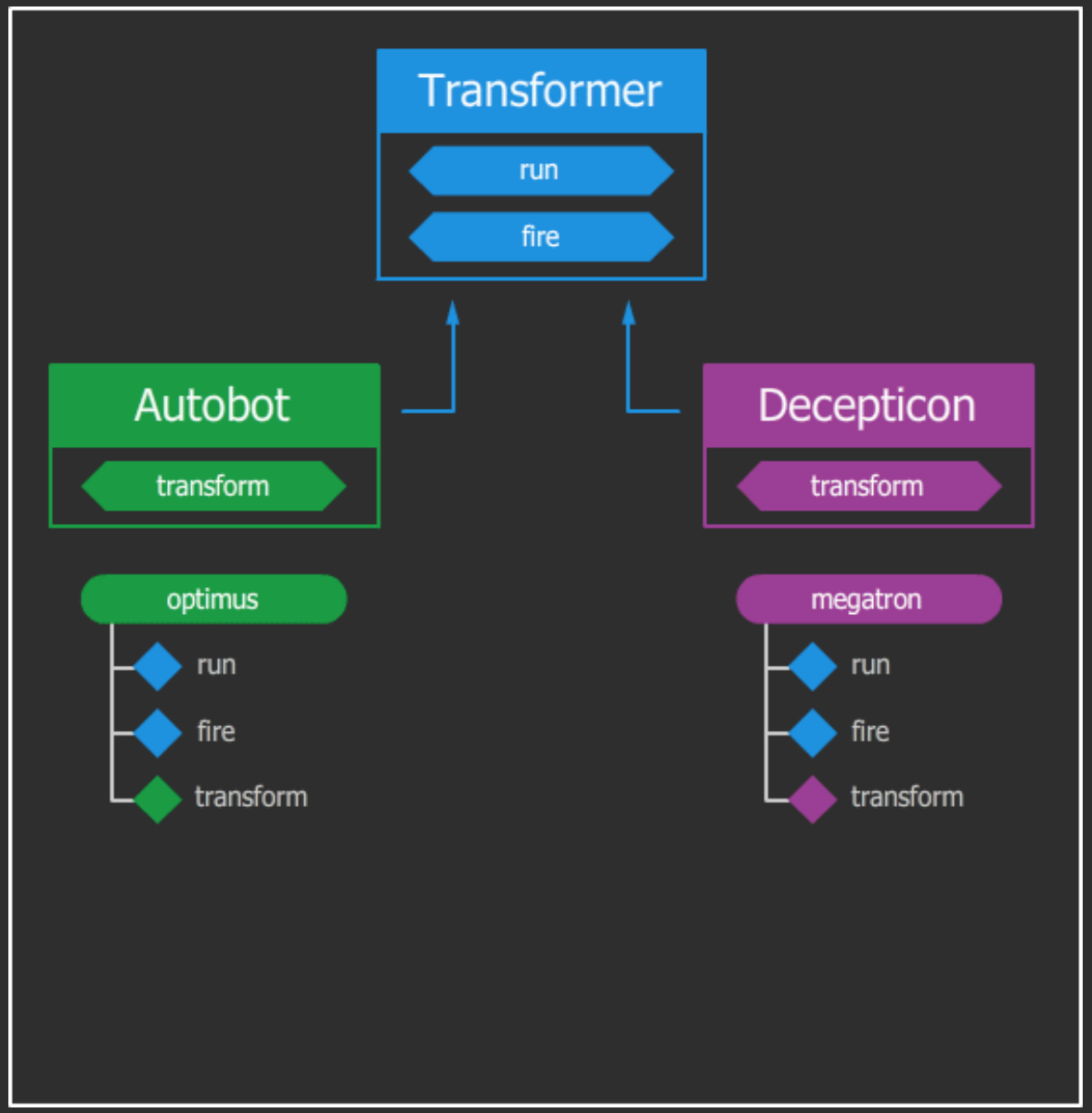
С инкапсуляцией

Отношение между классами

Тип отношения	UML-синтаксис		Краткая семантика
	источник	цель	
Зависимость			Исходный элемент зависит от целевого элемента и изменение последнего может повлиять на первый.
Ассоциация			Описание набора связей между объектами.
Агрегация			Целевой элемент является частью исходного элемента.
Композиция			Строгая (более ограниченная) форма агрегирования.
Включение			Исходный элемент содержит целевой элемент.
Обобщение (наследование)			Исходный элемент является специализацией более обобщенного целевого элемента и может замещать его.
Реализация			Исходный элемент гарантированно выполняет контракт, определенный целевым элементом.

Наследование (обобщение)

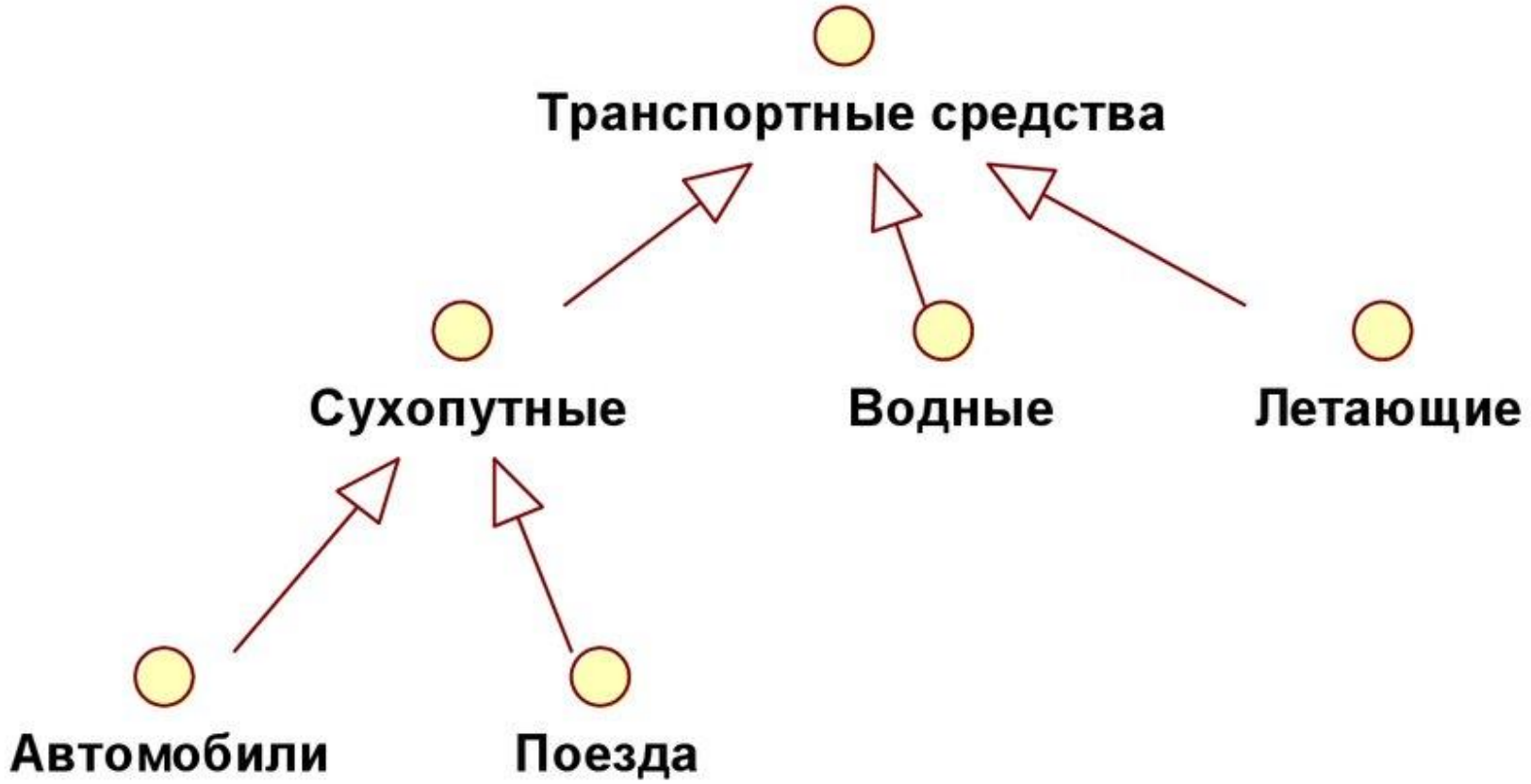




https://habr.com/ru/post/463125/

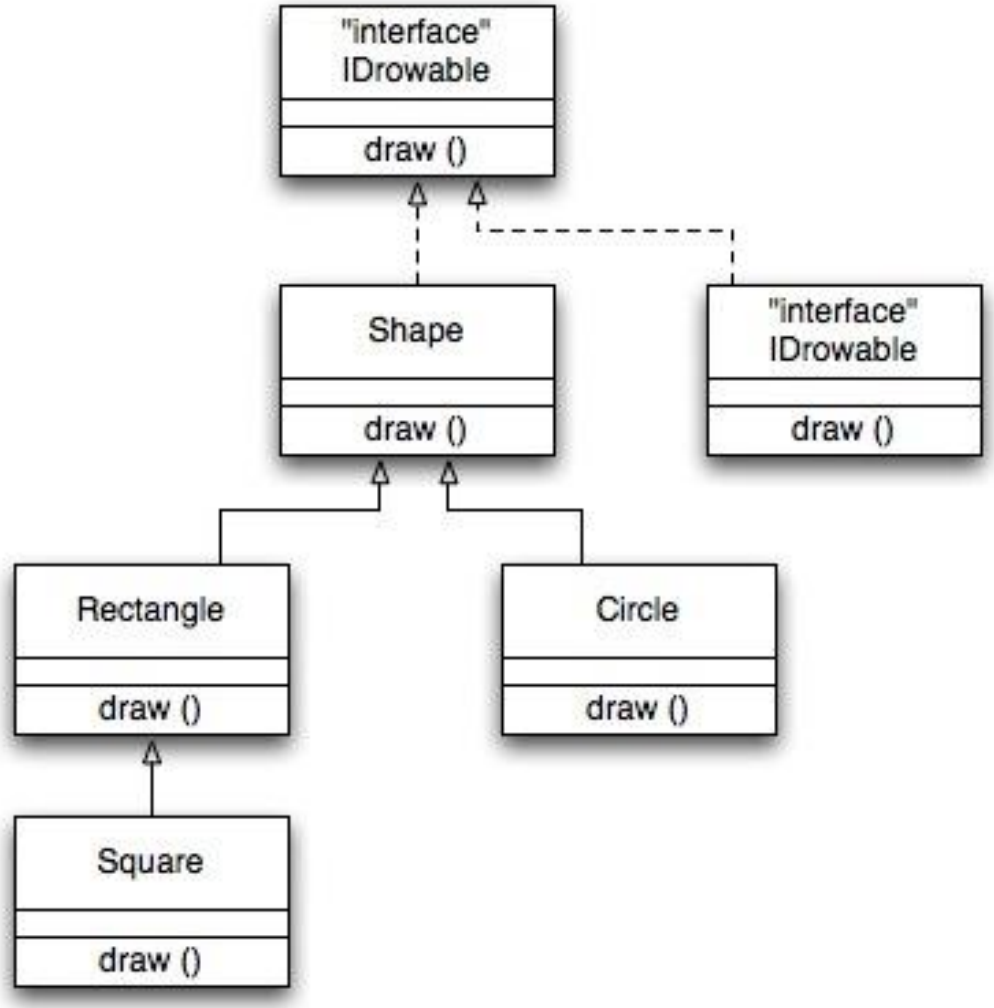
Примеры наследования

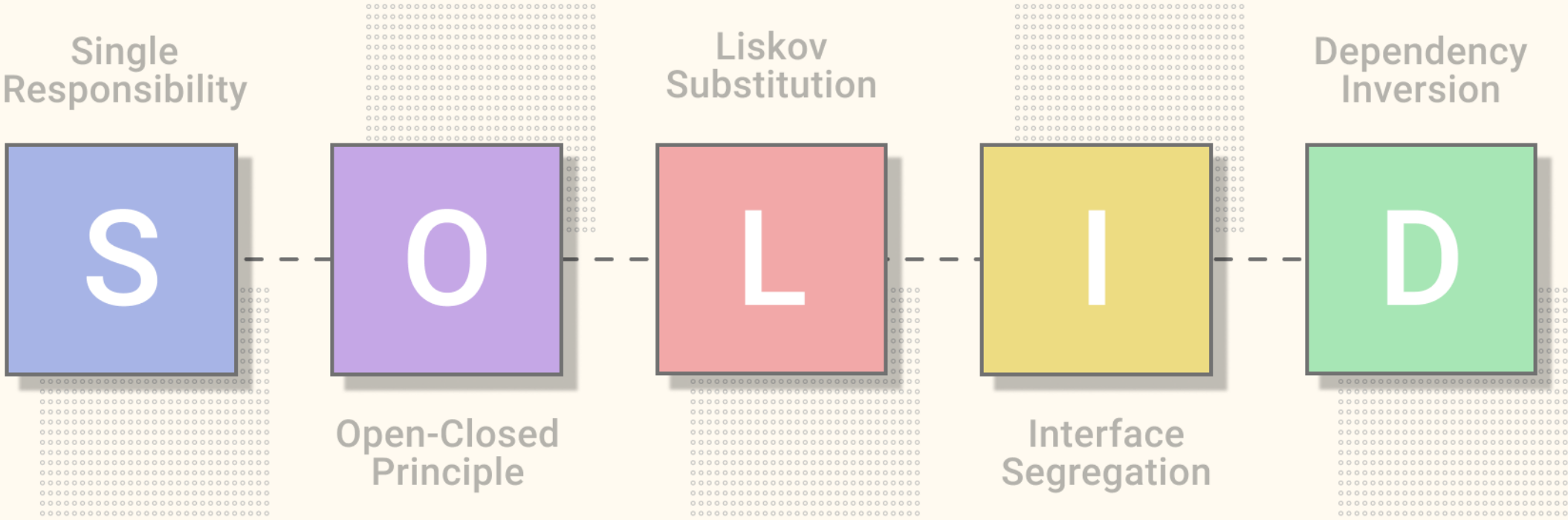
Примеры наследования



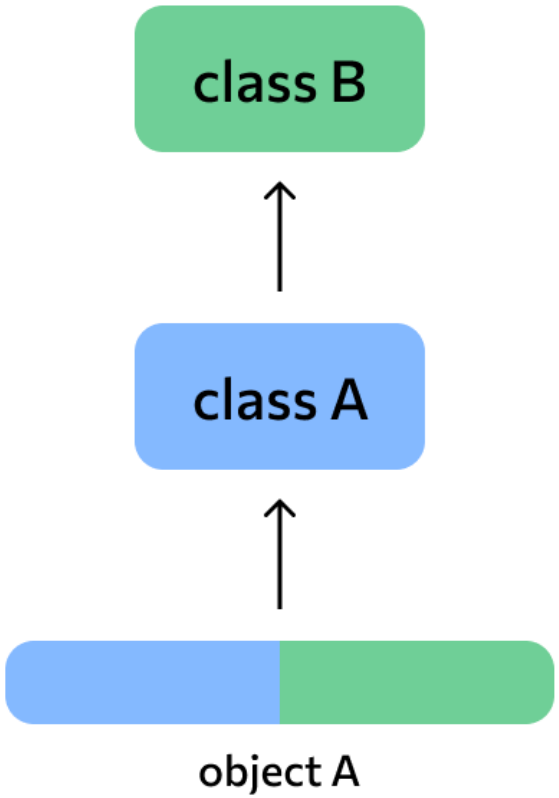
Кто кому предок?

Как вариант

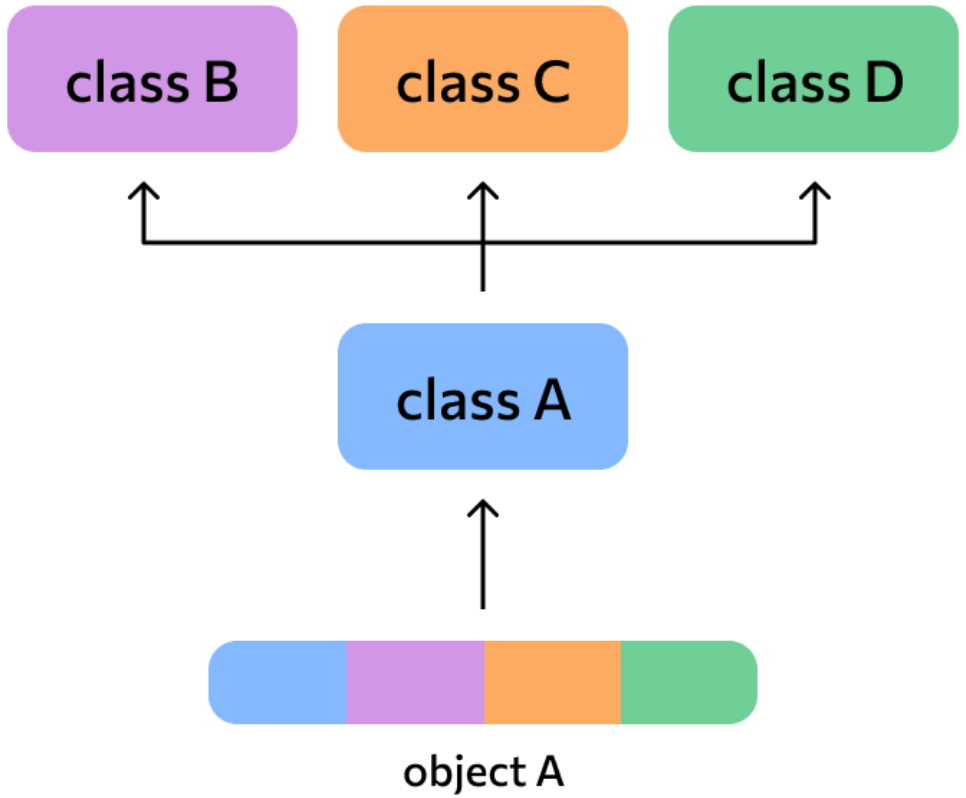

"interface"
IDrowable
draw ()
Shape
draw ()
"interface"
IDrowable
draw ()
Rectangle
draw ()
Circle
draw ()
Square
draw ()



Виды наследования в C++



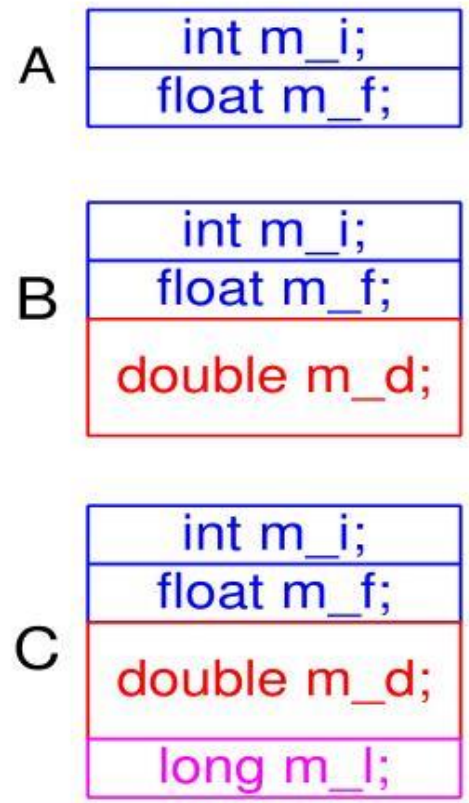
Одиночное наследование



Множественное наследование

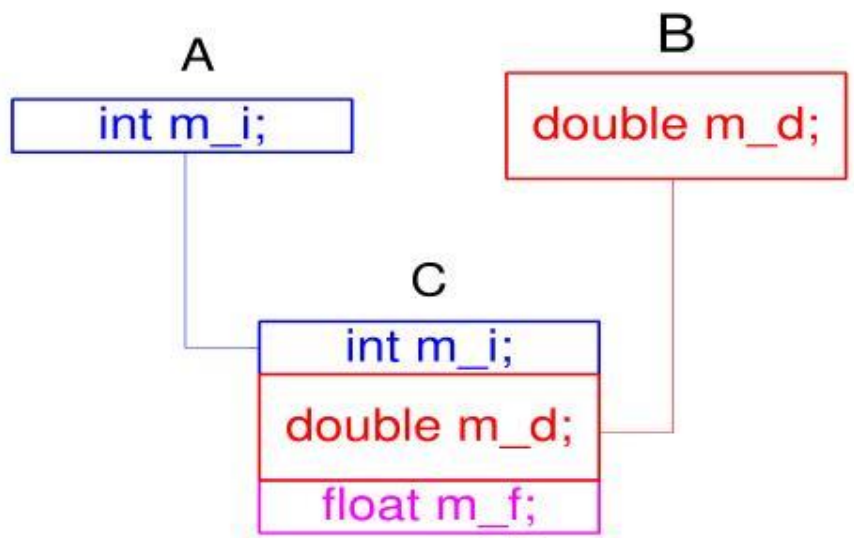
Наследование. Пример.

```
class A{
    int   m_i;
    float m_f;
};

class B: A{
    double m_d;
};


class C: B{
    long m_l;
};
```



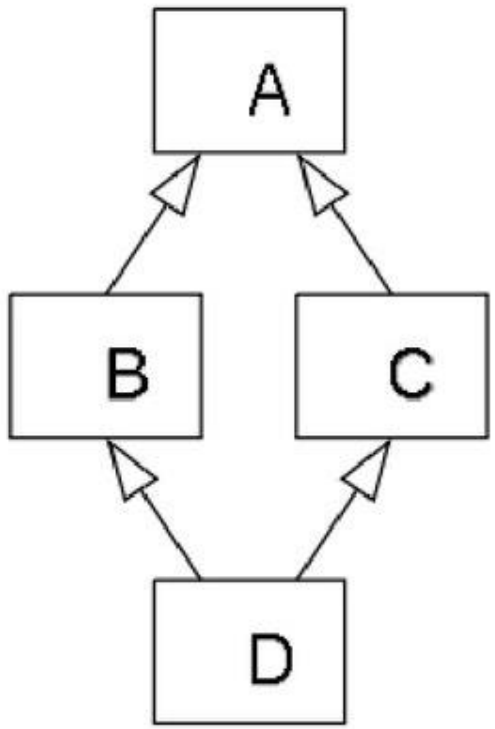
Множественное наследование.

```
class A{
    int m_i;
};

class B {
    double m_d;
};

class C: A, B{
    float m_f;
};
```



Ромбовидное наследование

Проблема ромба
("diamond problem")

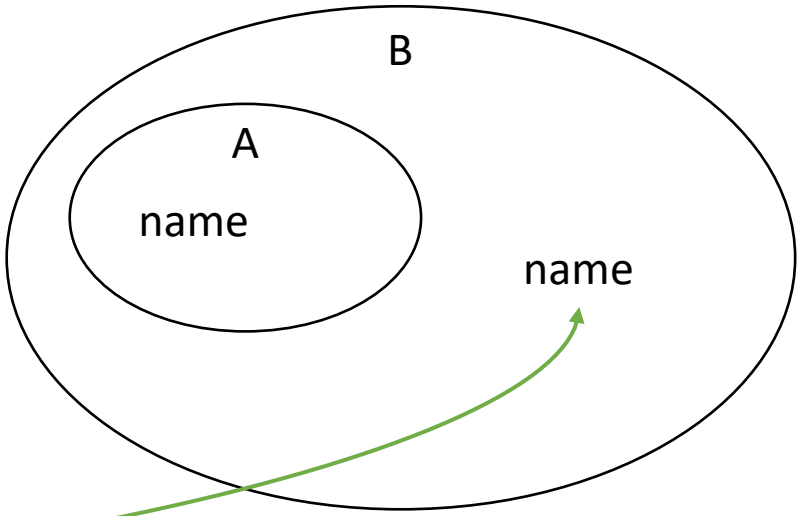


- Члены-данные класса A в объектах класса D в двух экземплярах
- Обращение к членам класса A в класе D невозможно без уточнения непосредственного наследника (B или C)
- Порядок наследования может привести к изменению семантики
- Решение

class B : virtual public A

Правила поиска имён

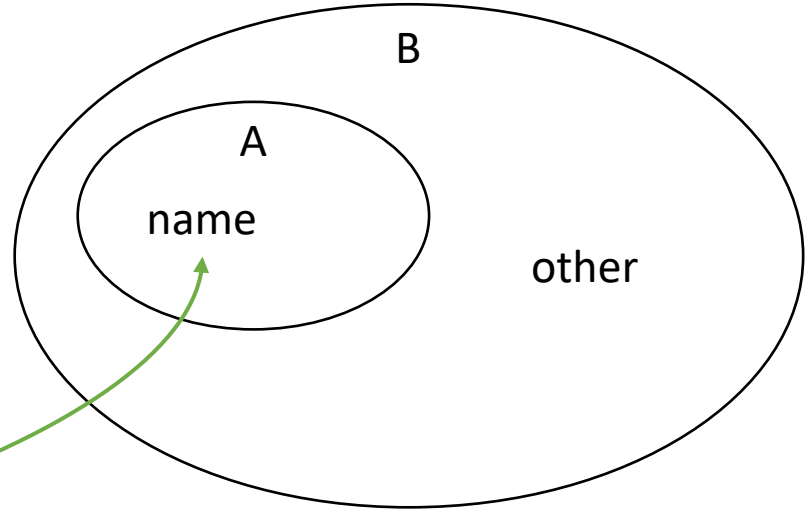
```
class A{
public:
    int name = 10;
};

class B: public A{
public:
    int name = -10;
};

B object;
object.name = 5;
```



Поиск имени выполняется начиная с класса B. Т.к. имя name в нём есть, то дальнейший поиск не происходит и `object.name = 5;` присвоит 5 собственной переменной name класса B.

Правила поиска имён

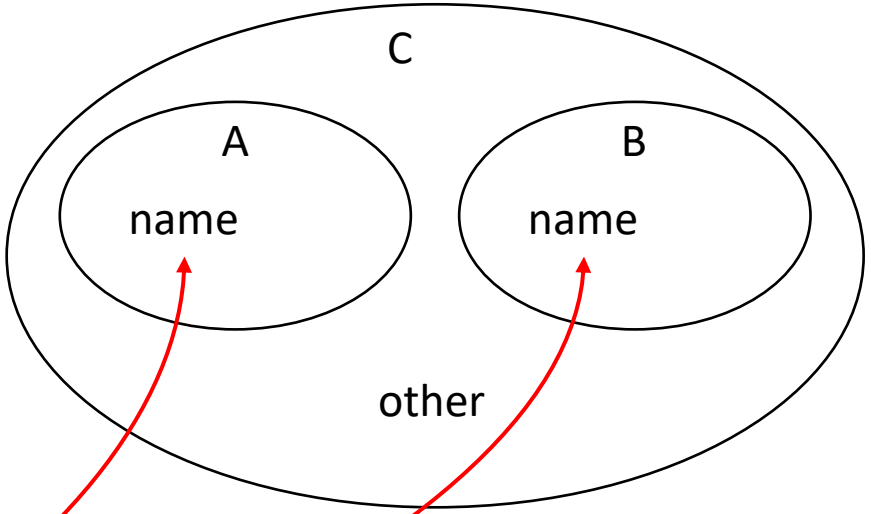
```
class A{
public:
    int name = 10;
};

class B: public A{
public:
    int other = -10;
};

B object;
object.name = 5;
```



Поиск имени выполняется начиная с класса B. Т.к. имени `name` в нём нет, то поиск продолжается в родительских классах, т.е. в классе `A`. Поэтому `object.name = 5;` присвоит 5 переменной `name` доставшейся классу B от класса `A`.

Правила поиска имён

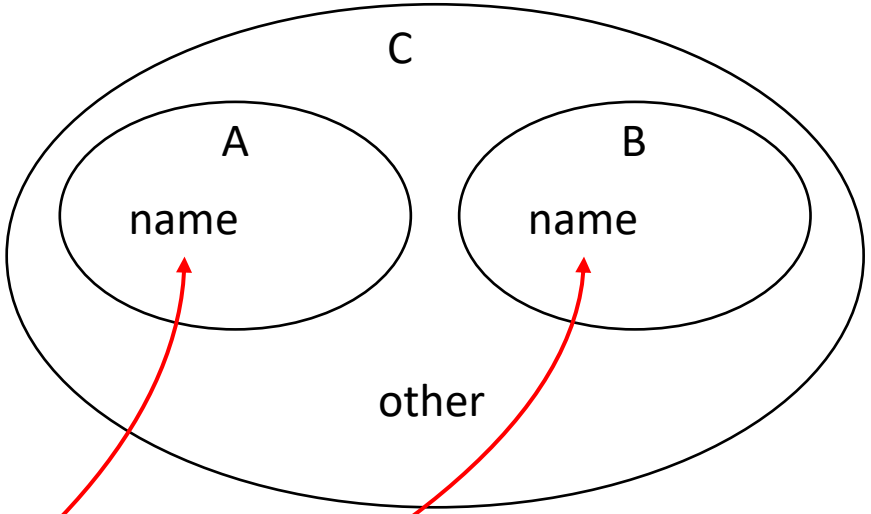
```
class A{
public:
    int name = 10;
};

class B{
public:
    int name = -10;
};

class C: public A, public B{
public:
    int other;
};

C object;
object.name = 5;
```



Поиск имени выполняется начиная с класса C. Т.к. имени name в нём нет, то поиск продолжается в родительских классах, т.е. в классе A и классе B. Имя name присутствует в обоих классах, поэтому компилятор не может выбрать какое-то одно. Т.е. такой код приведёт к ошибке.

Правила поиска имён

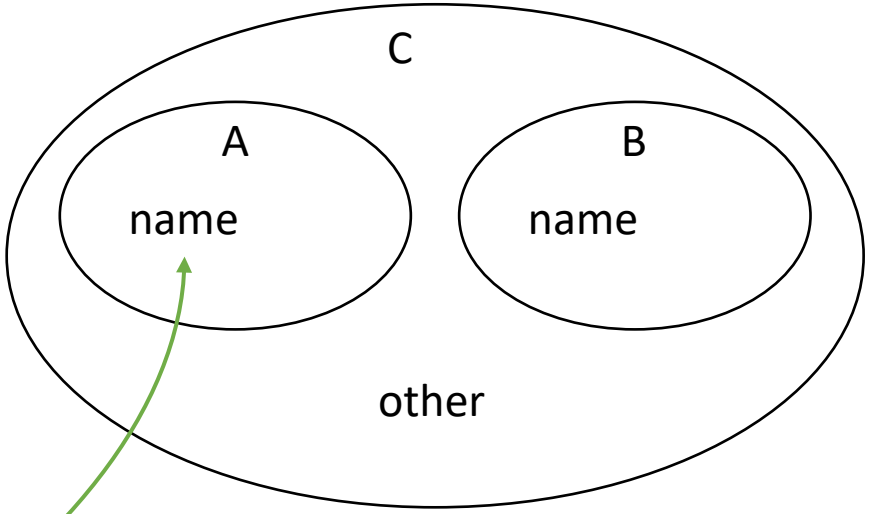
```
class A{
public:
    int name = 10;
};

class B{
public:
    int name = -10;
};

class C: public A, public B{
public:
    int other;
};

C object;
object.name = 5;
```



На процесс поиска имён квалификаторы доступа не влияют. Несмотря на то, что у класса **A** имя **name** находится в приватной секции и 100% не может быть доступно в точке вызова, а у класса **B** есть доступное имя **name**, мы получаем ошибку, по той же причине, что и на предыдущем слайде.

Правила поиска имён

```
class A{
public:
  int name = 10;
};

class B{
public:
  int name = -10;
};

class C: public A, public B{
public:
  int other;
};

C object;
object.A::name = 5;
```



Квалификатор A:: однозначно определяет какое имя name нужно использовать.

		Модификатор члена класса		
		public	protected	private
Место использования	Класс	+	+	+
	Наследник	+	+	-
	Внешний код: • функции • другие классы	+	-	-

```
class Der: public Base{};
class Der: protected Base{};
class Der: private Base{};
```

Если Base это **class**: `class Der: /* private */ Base{};`
Если Base это **struct**: `class Der: /* public */ Base{};`

		Модификатор члена базового класса		
		public	protected	private
Тип наследования	public	public	protected	-
	protected	protected	protected	-
	private	private	private	-