# Лекция 7

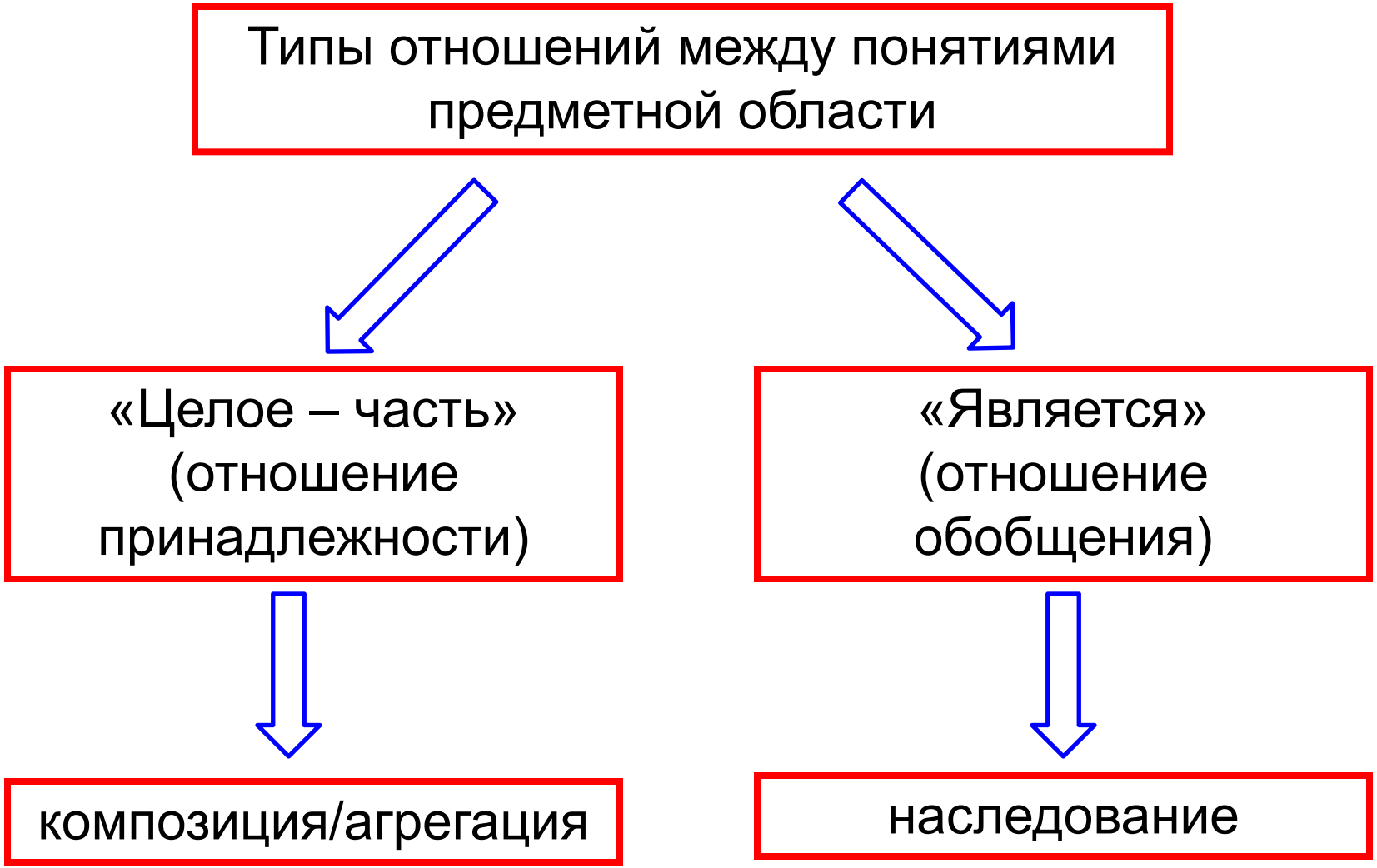
## Отношения между классами: наследование

# Роль наследования в ООП

Композиция и агрегация – не единственный механизм взаимодействия классов. Другим часто используемым в ООП методом является **наследование**.

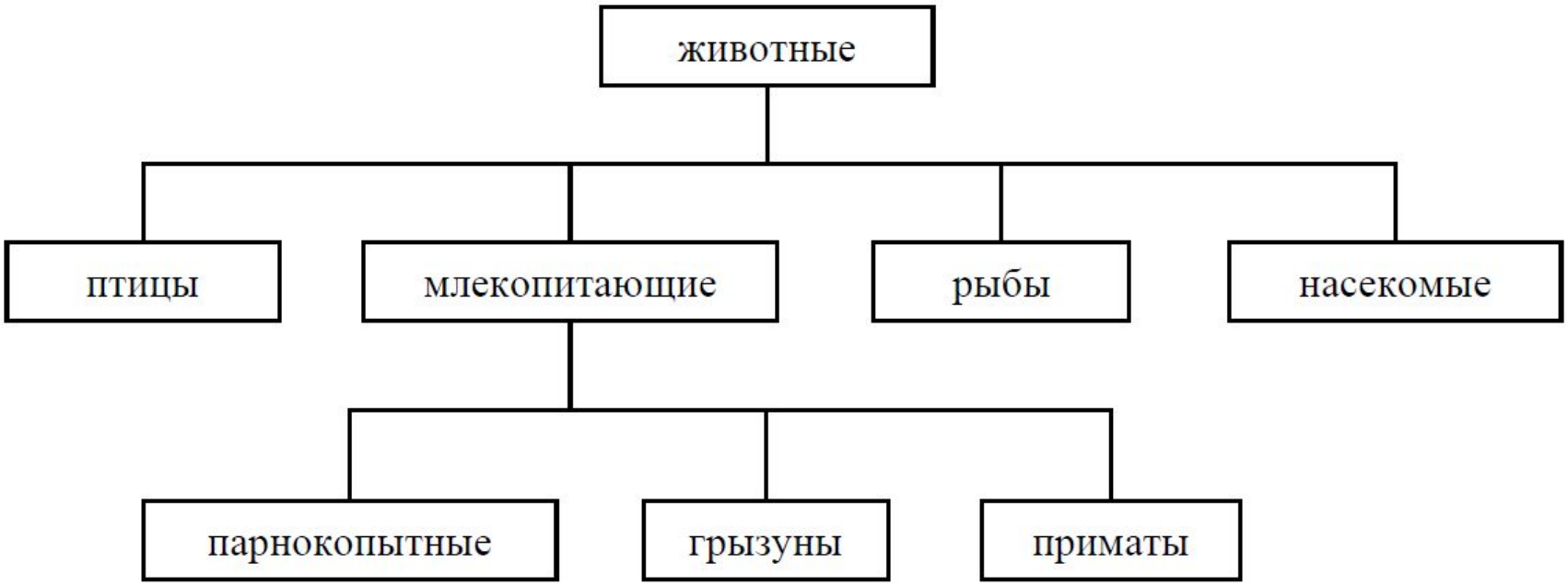
Основные принципы (парадигмы) ООП:

1) абстрагирование (абстракция) данных
2) инкапсуляция
3) **наследование**
4) полиморфизм


Типы отношений между понятиями предметной области
«Целое – часть» (отношение принадлежности)
«Является» (отношение обобщения)
композиция/агрегация
наследование

# Пример: обобщение и иерархия

Млекопитающее – подтип животного, примат – подтип млекопитающего, и т.д.




Общие свойства и поведение наследуются от предка, а специфические особенности отделяют одного потомка от другого.

# Синтаксис наследования

При объявлении производного класса после его имени указываются:

1) знак двоеточия

2) спецификатор доступа (public, private)

3) имя базового класса, от которого производится наследование

```
class имя_класса1 : спецификатор имя_класса2
{
    <объявления полей и методов класса2>
}
```

# Пример: живо

```
class animal
{
  public:
   int size;
   int weight;

   void eat();
   void sleep();
   void breath(
}
```

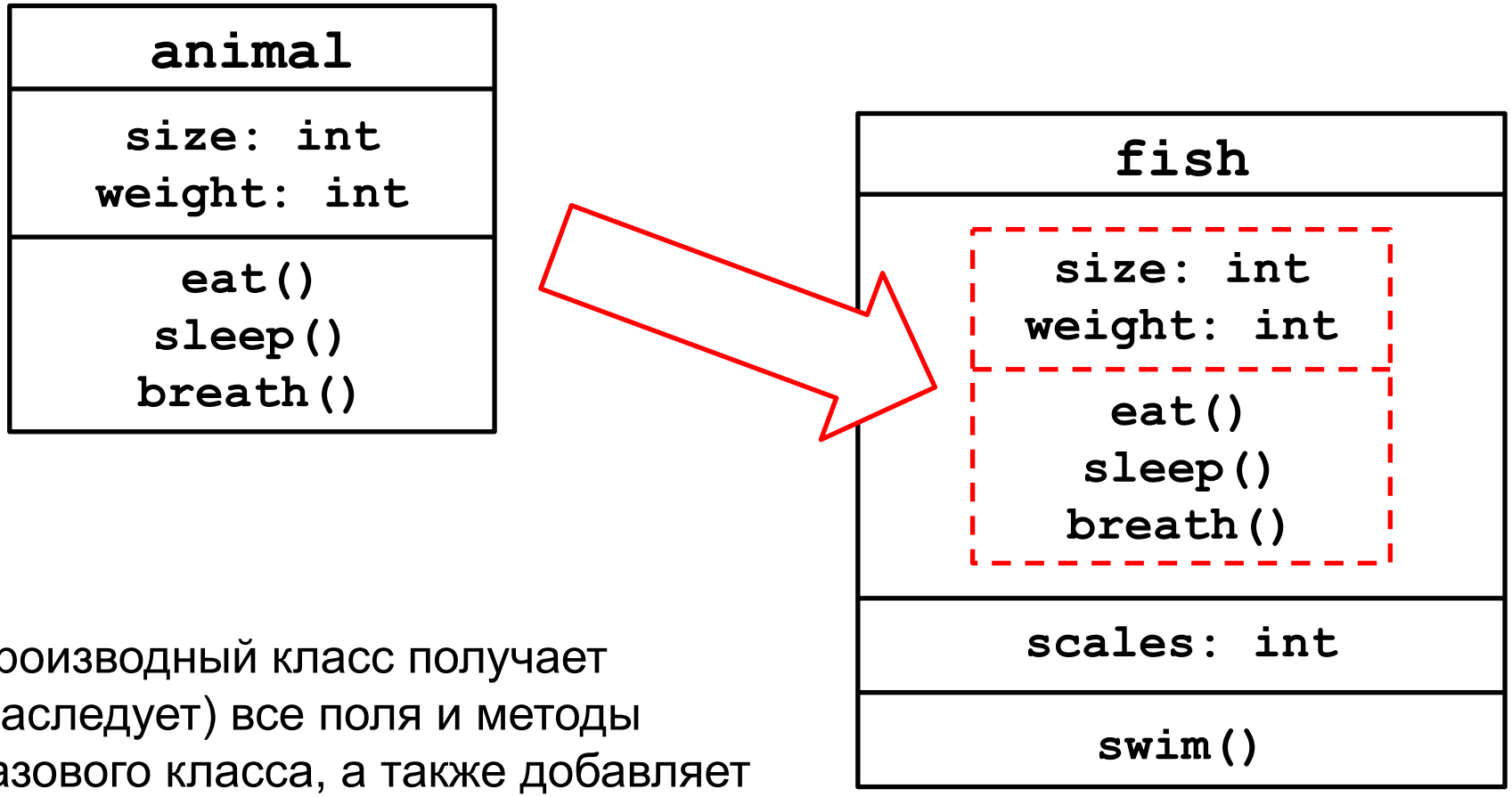
```
class bird: public animal
{
  public:
   int wings; // крылья

   void fly();
   void sing();
}
```

```
class fish: public animal
{
  public:
   int scales; // чешуя

   void swim();
}
```

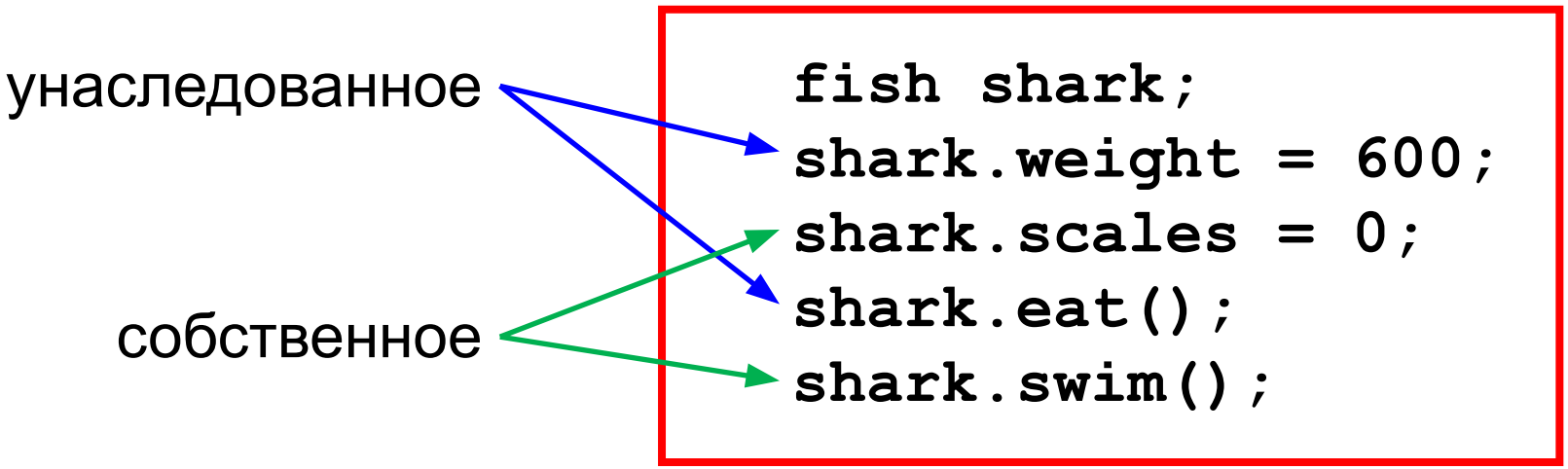
# Механизм наследования




Производный класс получает (наследует) все поля и методы базового класса, а также добавляет свои собственные поля и методы.

# Пример использования классов

```
animal yeti;
yeti.weight = 200;
yeti.eat();
yeti.sleep();
```



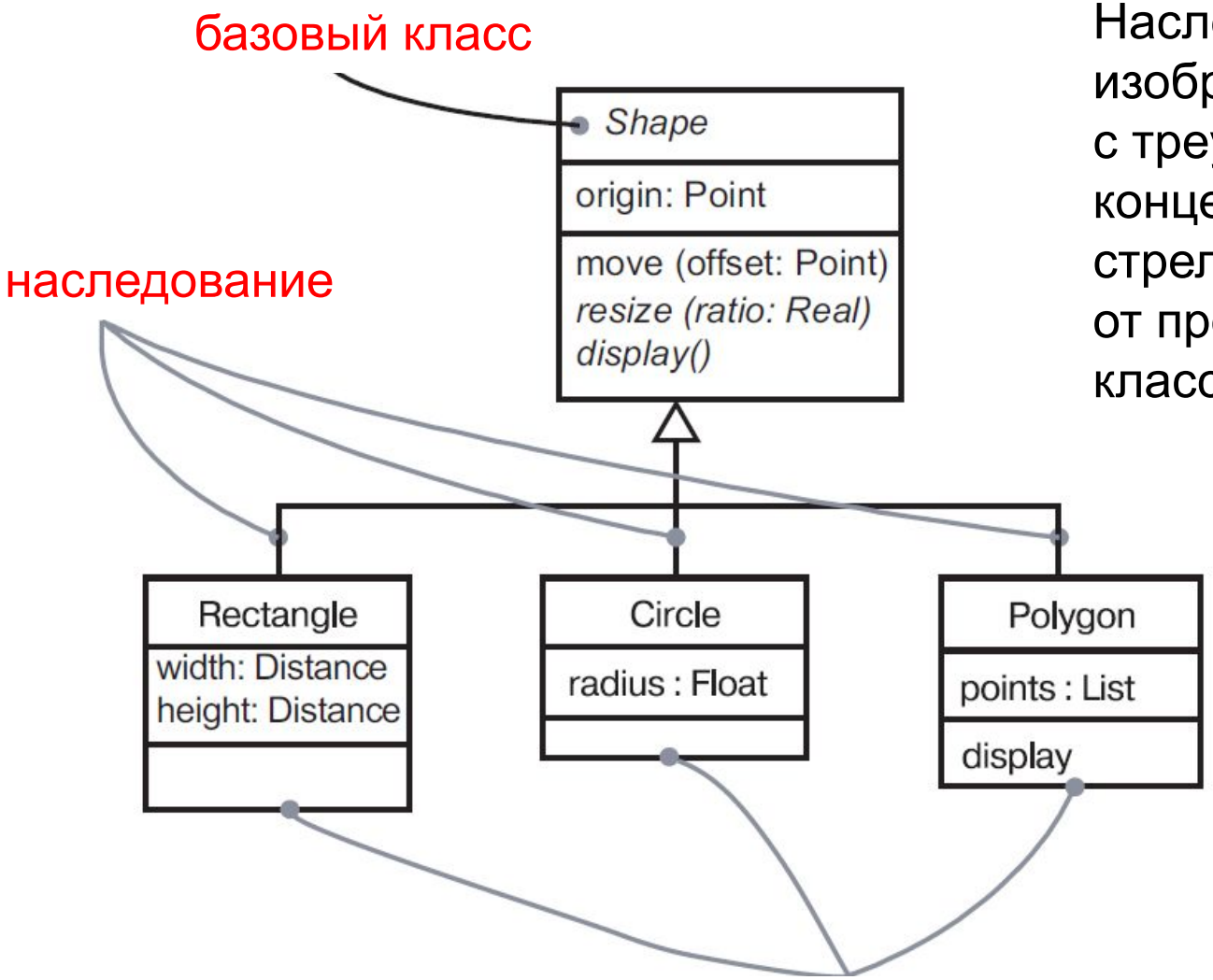
# Зачем нужно наследование?

Наследование дает еще один механизм повторного использования кода (кроме стандартных функций).

Один и тот же метод может использоваться как базовым классом, так и всеми его производными классами.

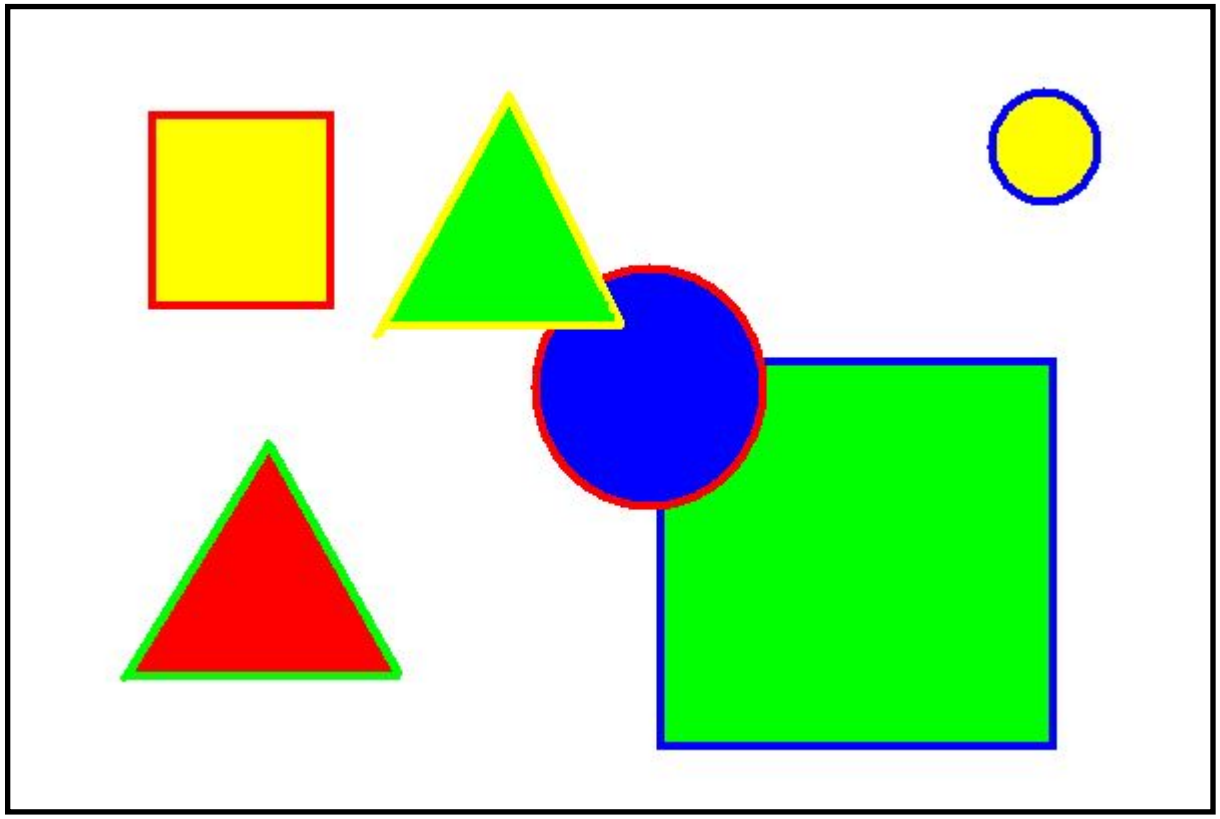
```
yeti.eat();
shark.eat();
elephant.eat();
spider.eat();
```

# Наследование на диаграммах UML




Наследование изображается линией с треугольником на конце. При этом стрелка направлена от производного класса к базовому.

# Пример: абстрактная живопись и ООП



# Объектно-ориентированный анализ и проектирование

Основные понятия предметной области

1) *геометрические фигуры* разной формы – круги, квадраты, треугольники
2) *холст* (окно программы)

Холст и фигуры находятся между собой в отношении «целое/часть» – фигуры являются компонентами холста (также можно сказать, что «принадлежат» ему).

# Свойства объектов (поля данных в классах)

*геометрическая фигура* (класс graph)

- форма объекта
- положение (координаты центра)
- размеры
- цвет заливки
- цвет линии

*холст* (класс canvas)

- массив объектов (композиция!)
- текущее число объектов
- другие (при необходимости)

# Действия над объектами (методы классов)

класс graph

- функция «создать» (конструктор/-ры)
- функция «нарисовать» (draw)
- функция «сдвинуть» (move)
- функция «масштабировать» (resize)

класс canvas

- функция «создать» (конструктор/-ры)
- функция «добавить объект» (add)
- функция «нарисовать» (draw)

# Геометрические фигуры на плоскости: отношение обобщения

Различные геометрические формы (круги, квадраты, треугольники) являются родственными понятиями, так как все они – фигуры на плоскости.

Понятие фигура (graph) является *обобщающим* для понятий круг (circle), квадрат (square), треугольник (triangle). Поэтому класс graph мы далее делаем базовым классом, а классы circle, square и triangle – производными от него.

# Итоговая диаграмма классов