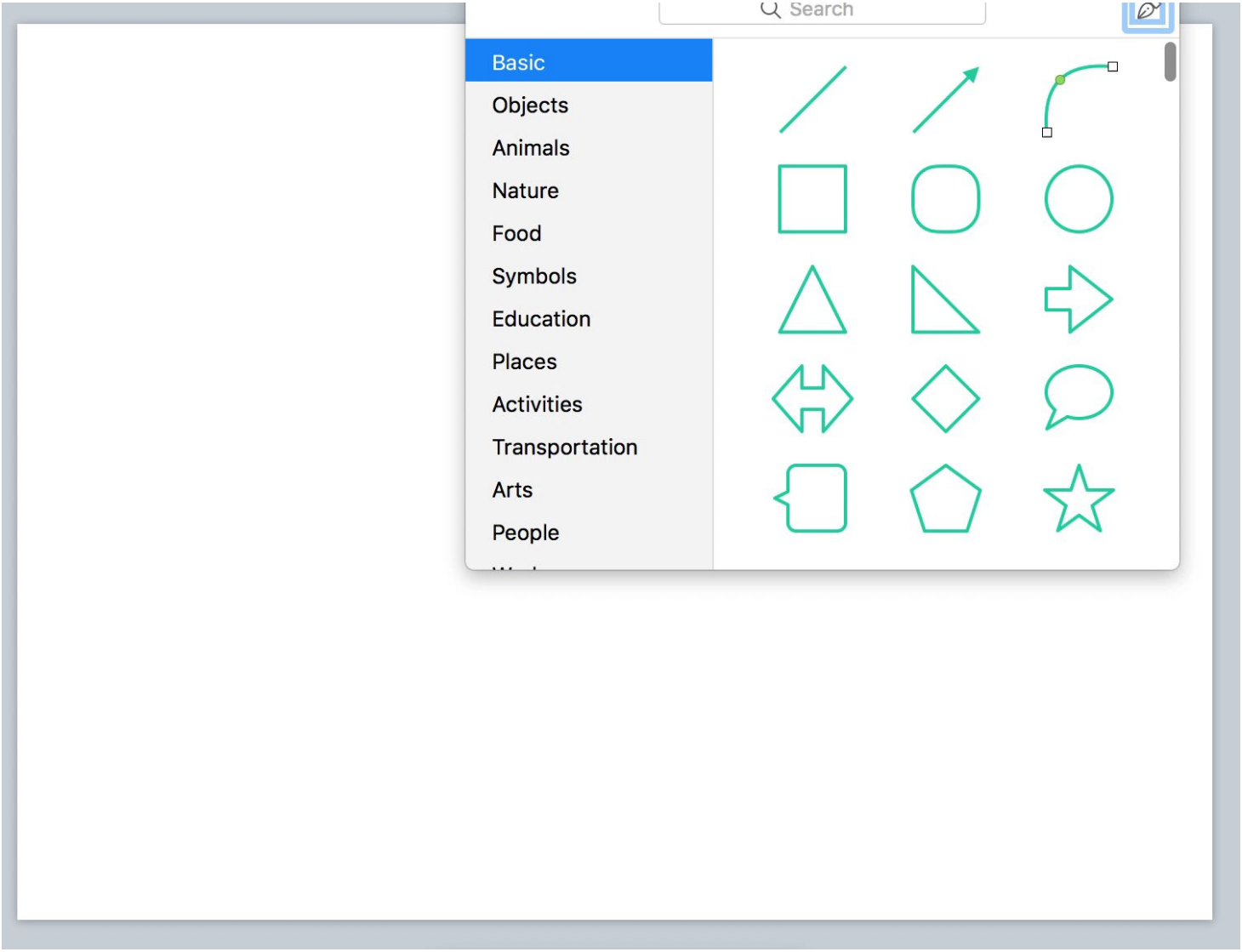
Основы объектно-ориентированного программирования

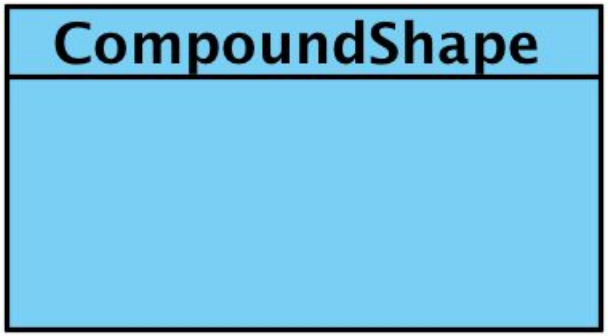
Чернойван Василий Александрович
vchernoivan@gmail.com
http://chernoivan.ru/oop/

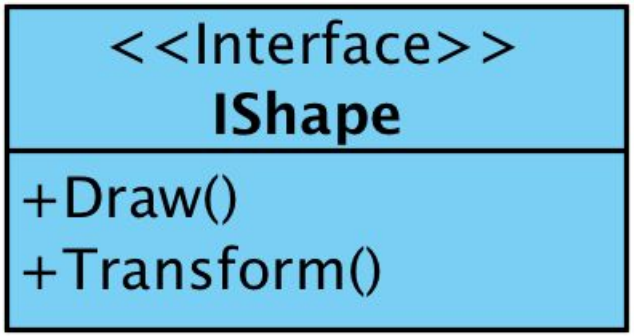
Связи
(отношения, ассоциации)

Связь это физическое или концептуальное соединение между объектами

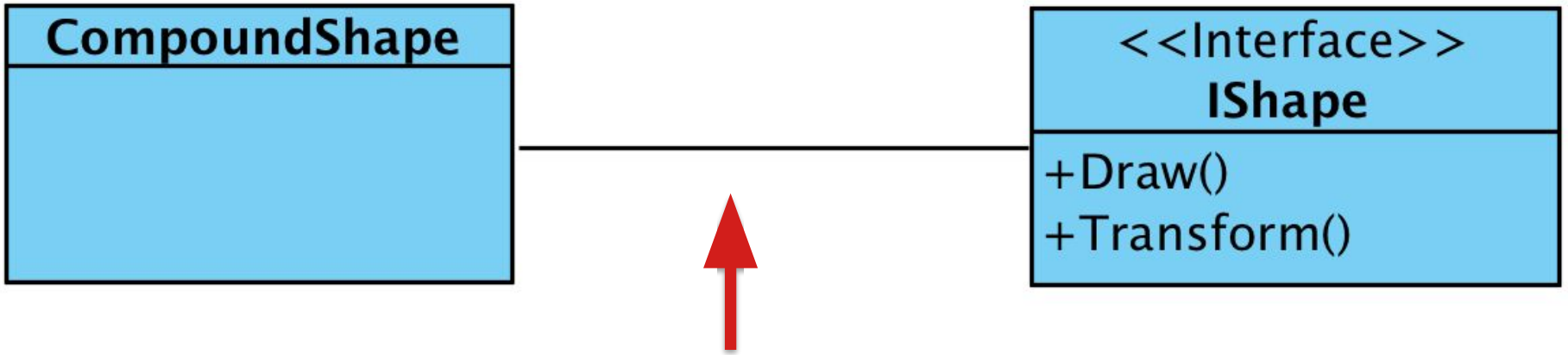

Search
Basic
Objects
Animals
Nature
Food
Symbols
Education
Places
Activities
Transportation
Arts
People

Составная фигура объединяет множество других фигур в одну

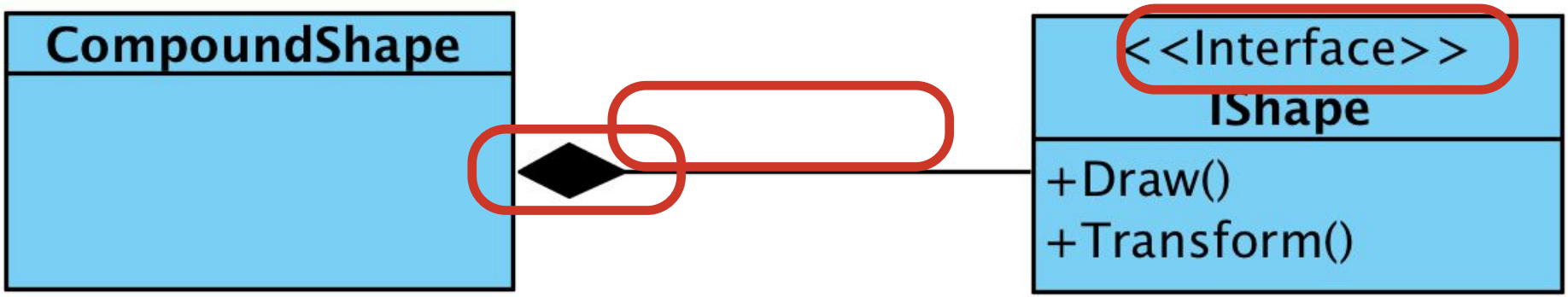




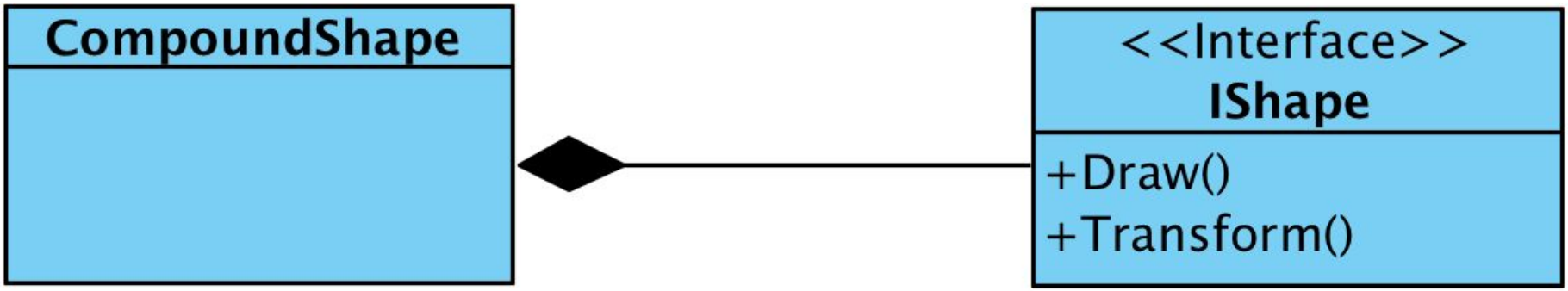
Ассоциация



Стереотип



Направленность



Видимость

- *Глобальная*: один из объектов ***глобален*** по отношению к другому (глобальная переменная или статические методы)
- *Параметр*: один из объектов (или ссылка на него) передан другому в качестве ***параметра*** операции.
- *Поле класса*: один из объектов (или ссылка на него) ***является частью*** другого.
- *Локальная*: один из объектов ***локально порождается*** другим в ходе выполнения какой-либо операции.

Видимость

Глобальная

```
public static class Picture {
    public static IShape[] Shapes { get; set; }
}

public class CompoundShape {
    public void Draw() {
        foreach (var shape in Picture.Shapes) {
            shape.Draw();
        }
    }
}
```

Видимость

Параметр

```
public class CompoundShape {
    public void Draw(IShape[] shapes) {
        foreach (var shape in shapes) {
            shape.Draw();
        }
    }
}
```

Видимость

Поле объекта

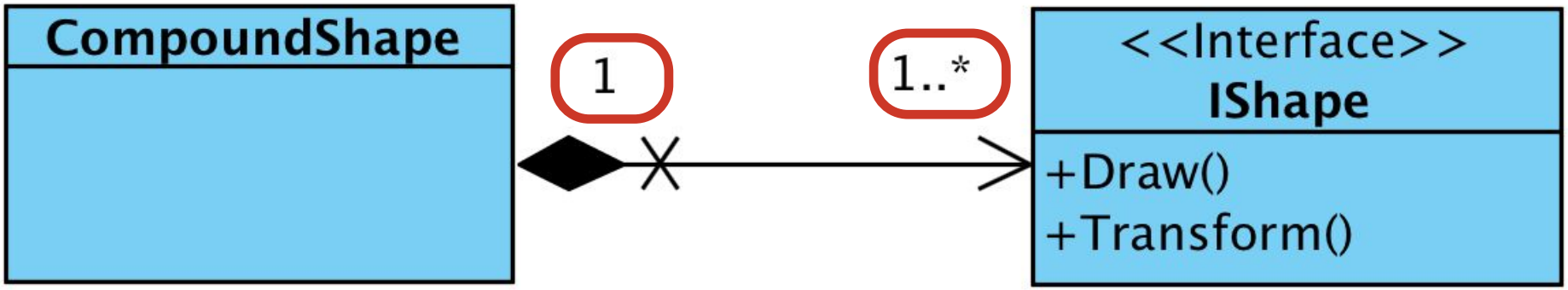
```
public class CompoundShape {
    private IShape[] shapes;
    public void Draw() {
        foreach (var shape in shapes) {
            shape.Draw();
        }
    }
}
```

Видимость

Локальная

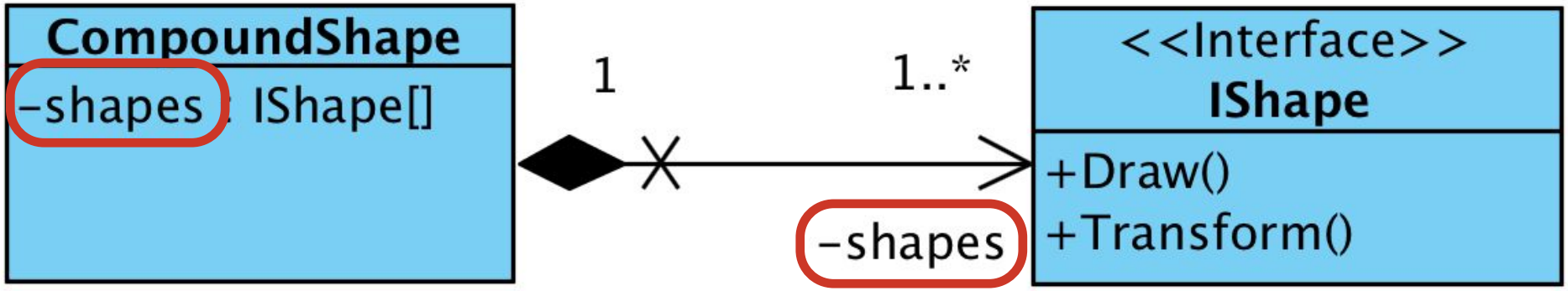
```
public class CompoundShape {
    public void Draw() {
        var shapes = new IShape[0];
        foreach (var shape in shapes) {
            shape.Draw();
        }
    }
}
```

Кратность (мощность)



1 – в точности один объект
0..1 – ни одного либо один объект
***** – множество объектов
0..* – множество объектов, возможно 0
1..* – множество объектов, но хотя бы один

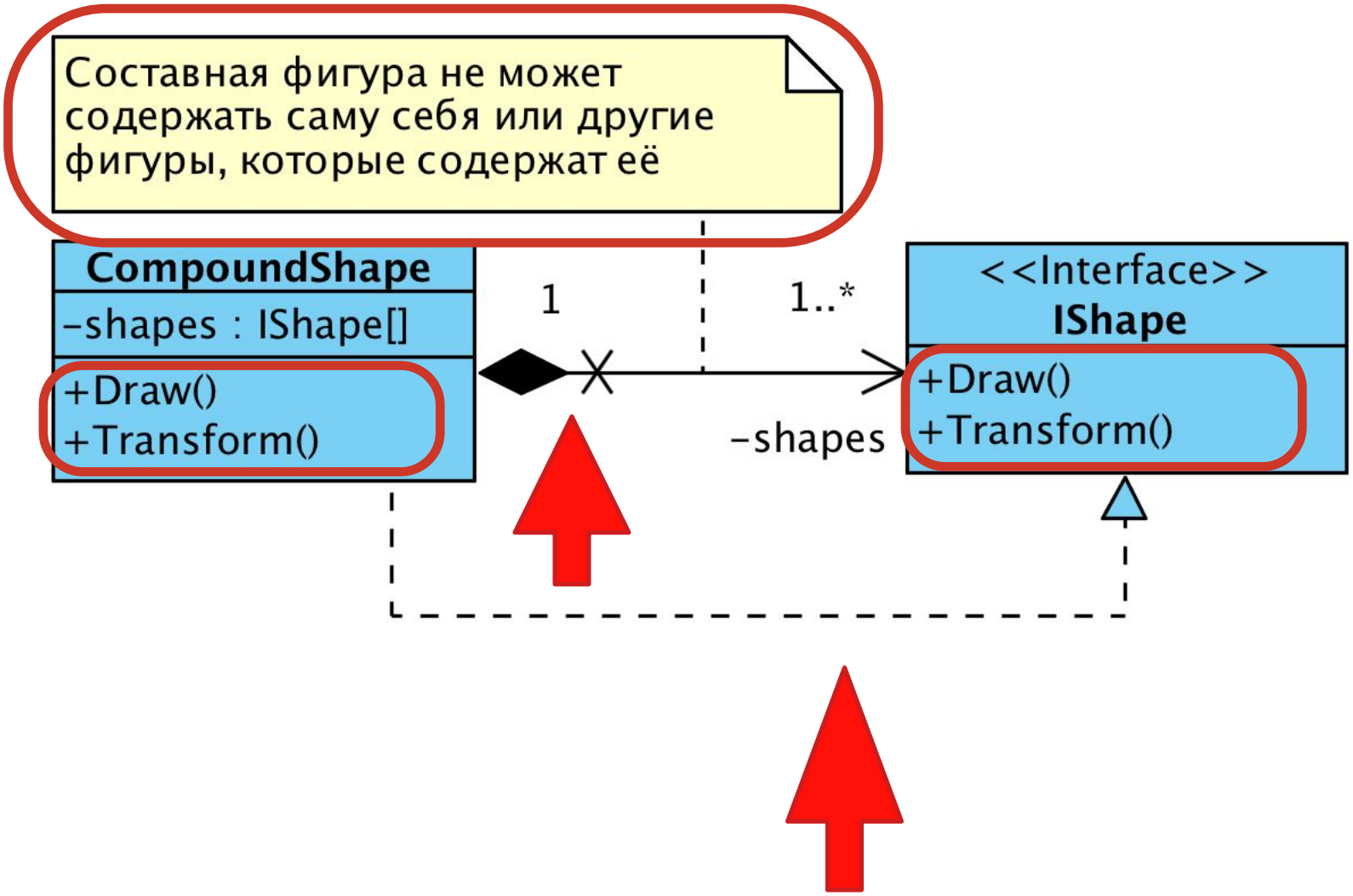
Имена участников



Атрибуты связей

- *Стереотип* – устойчивый характер (смысл, семантика) связи
- *Мощность* – сколько объектов может участвовать в связи с обеих сторон
- *Направленность* – определяет кто из двух участников «знает» о другом
- *Видимость* – каким образом один объект «видит» другой
- *Имена участников*

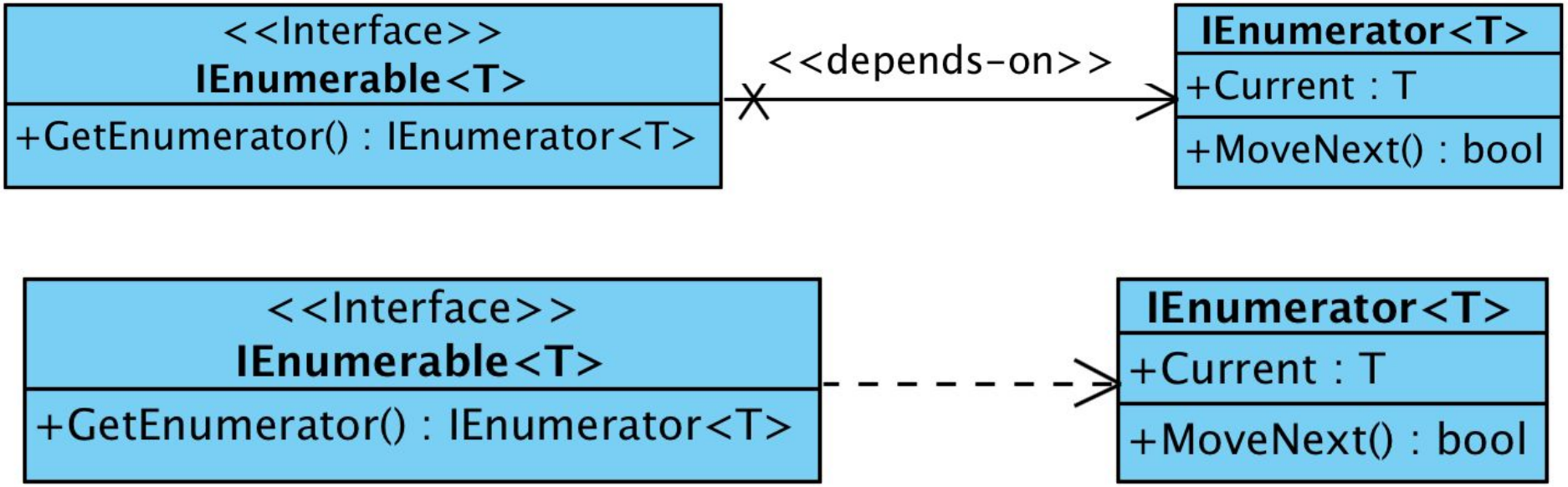
Ещё кое-что


Составная фигура не может содержать саму себя или другие фигуры, которые содержат её
CompoundShape
-shapes : IShape[]
+Draw()
+Transform()
1
1..*
-shapes
<<Interface>>
IShape
+Draw()
+Transform()

Типы отношений

- Ассоциация
- Зависимость
- Использование
- Наследование
- Реализация
- Агрегация
- Конкретизация
- Класс-ассоциация

Зависимость

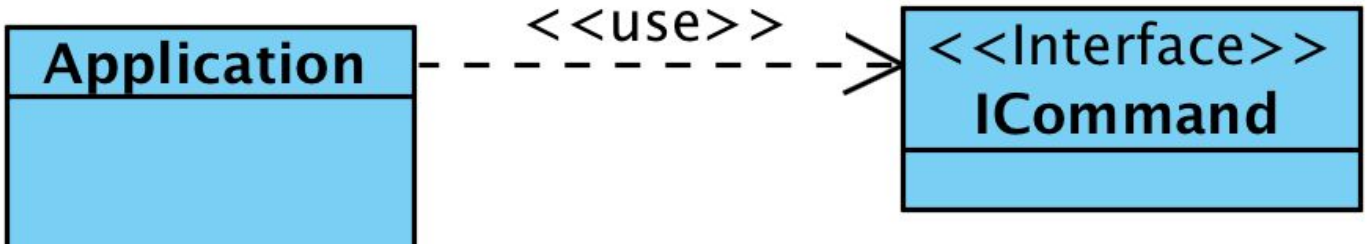



```
public interface IEnumerator<T> { }
public interface IEnumerable<T> {
    IEnumerator<T> GetEnumerator();
}
```

Зависимость

- **Мощность:** определяется контекстом
- **Направленность:** от зависимого к независимому
- **Стереотип связи: зависимость,** поменял «независимого» — нужно менять «зависимого»
- **Видимость:** определяется конкретикой реализации

Использование

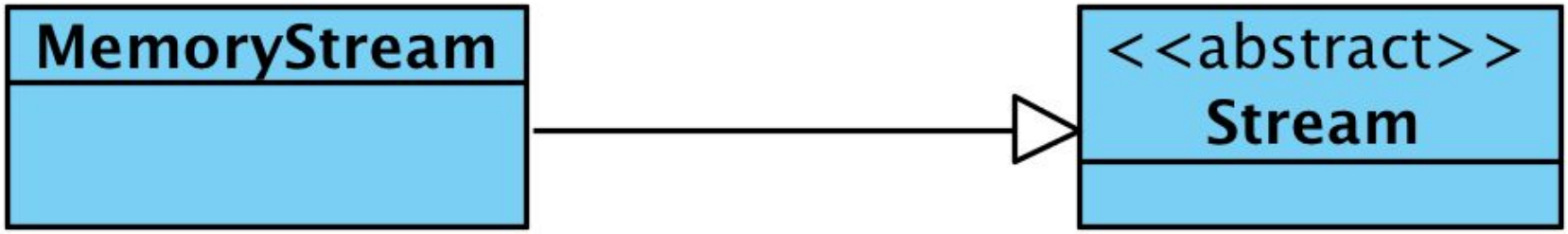



```
public void Run(TextReader reader) {
    string[] cmdline;
    while (keepRunning)
    {
        Console.Write("> ");
        cmdline = reader.ReadLine().Split(' ');
        ICommand cmd = FindCommand(cmdline[0]);
        cmd.Execute(ExtractParams(cmdline));
    }
}
```

Использование

- **Мощность:** определяется реализацией
- **Направленность:** от пользователя к используемому (от клиента к серверу)
- **Стереотип связи:** использование
- **Видимость:** определяется конкретикой реализации, часто локальная
- **Имена участников:** пользователь, используемый (клиент, сервер)

Наследование



MemoryStream это Stream

```
public class MemoryStream : Stream {
    public override void Flush() {
```

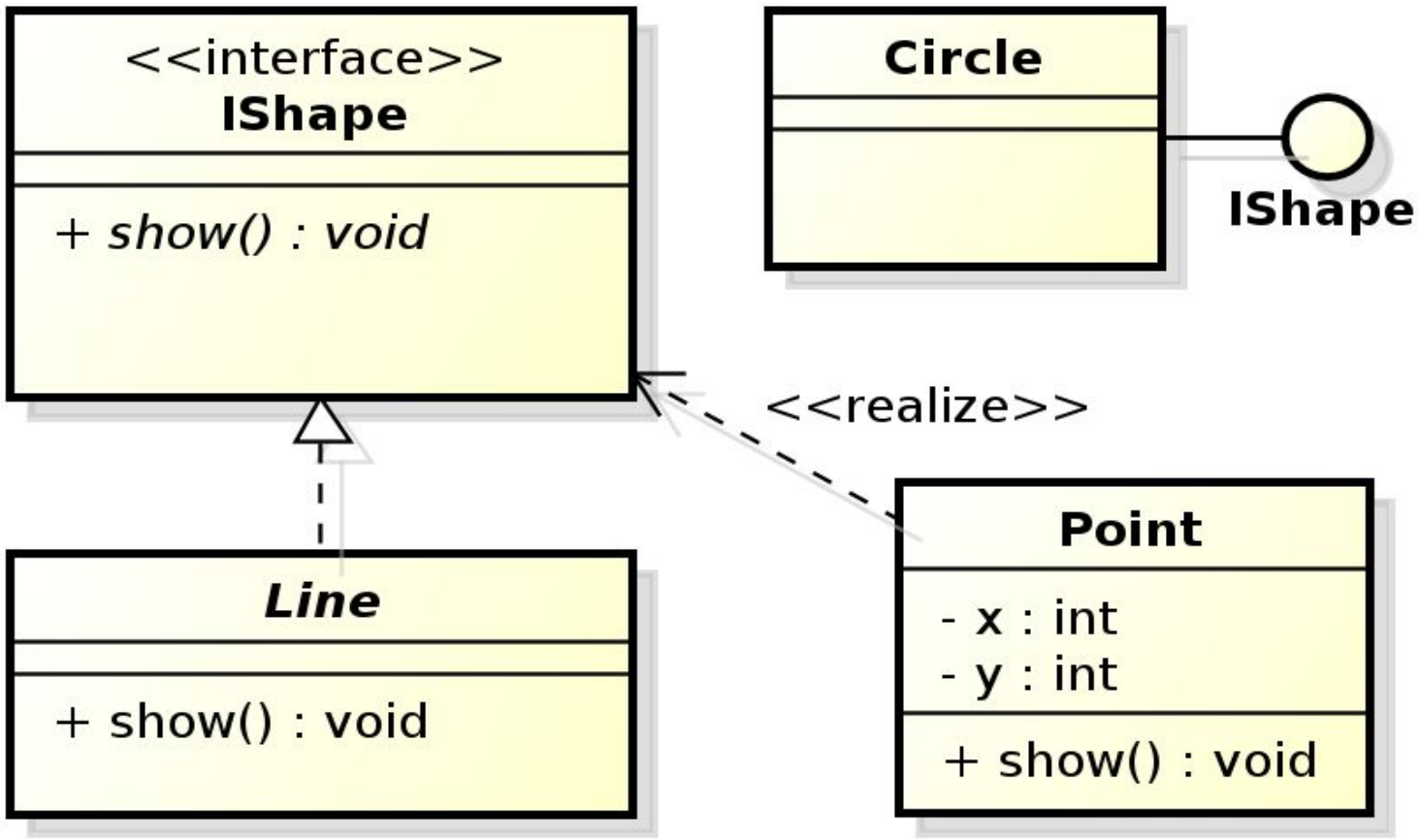
Наследование

- **Мощность —** 1:1
- **Имена участников —** базовый класс, производный класс
- **Направленность** — от потомка к предку
- **Стереотип связи** – являться (is-a)
- **Видимость:** неприменимо, т.к. наследование — отношение между классами, в отношении участвует «один и тот же» объект

Реализация

- Классы `Circle, Line` и `Point` **реализуют интерфейс** `IShape`

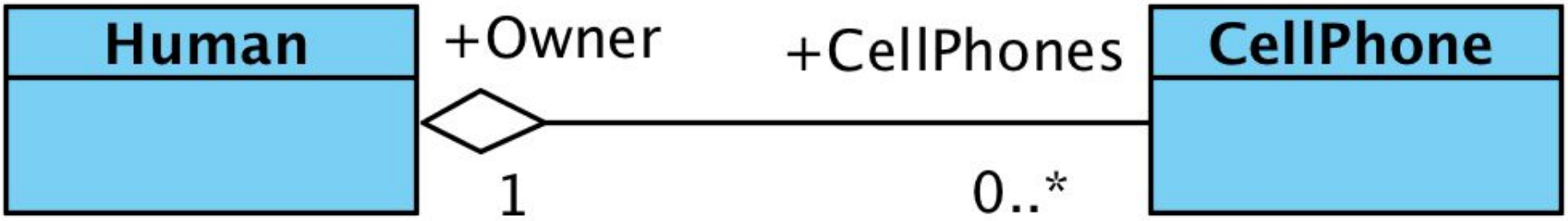
```
public class Line : IShape
```



Реализация

- Связь между классами и интерфейсами
- **Мощность** — 1:1
- **Имена участников** — реализующий класс, интерфейс
- **Направленность** — от реализующего класса к интерфейсу
- **Стереотип связи** — реализация (is-a, реализуем поведение)

Агрегация



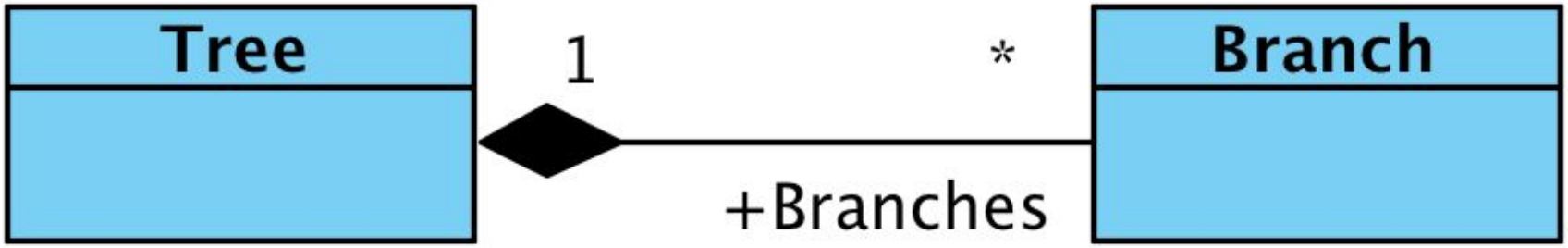

```
public class CellPhone {
    public Human Owner { get; set; }
}

public class Human {
    public List<CellPhone> CellPhones { get; }
}
```

Агрегация

- Агрегация — связь между *объектами*
- **Мощность, Имена участников, Направленность** — определяются контекстом
- **Стереотип связи** — также определяется контекстом, например, «владение», «быть частью»
- **Видимость**: поля класса

Композиция




```
public class Branch{}

public class Tree {
    public IEnumerable<Branch> Branches { get; }
}
```

Композиция

- Композиция – связь между *объектами*
- **Мощность, Имена участников, Направленность** - определяются контекстом
- **Стереотип связи** – «быть частью», «part-of»
- **Видимость**: поля класса

УНИВЕРСАМ ЗАО ТД "ПЕРЕКРЕСТОК"
УЛ. ДОМЛДЕДОВСКАЯ, Д.28

МАГАЗИН 0041 КАССА 0001 КАССИР 0007
1:00679 КАПУСТА БЕЛОКАЧАННАЯ
17.20* 2.020 СКД -1.22 33.52
2:61047 ПРИПРАВА CYKORIA
4.40* 2.000 СКД -0.31 8.49
3:61032 ПЕРЕЦ CYKORIA КРАСНЫЙ
6.15* 1.000 СКД -0.22 5.93
4:79660 МАЙОНЕЗ МЕЧТА ХОЗЯЙКИ
13.90* 1.000 СКД -0.49 13.41
5:38835 ВОДА СМИРНОВСКАЯ МИНЕ
13.90* 2.000 СКД -0.97 26.83
6:49600 БАТОН ПОДМОСКОВНЫЙ
5.65* 2.000 СКД -0.40 10.90
7:41758 СЫР ГОЛЛАНДСКИЙ ШАР 4
123.50* 0.345 СКД -1.49 41.12
10:49587 ХЛЕБ ДАРНИЦКИЙ
7.70* 1.000 СКД -0.27 7.43

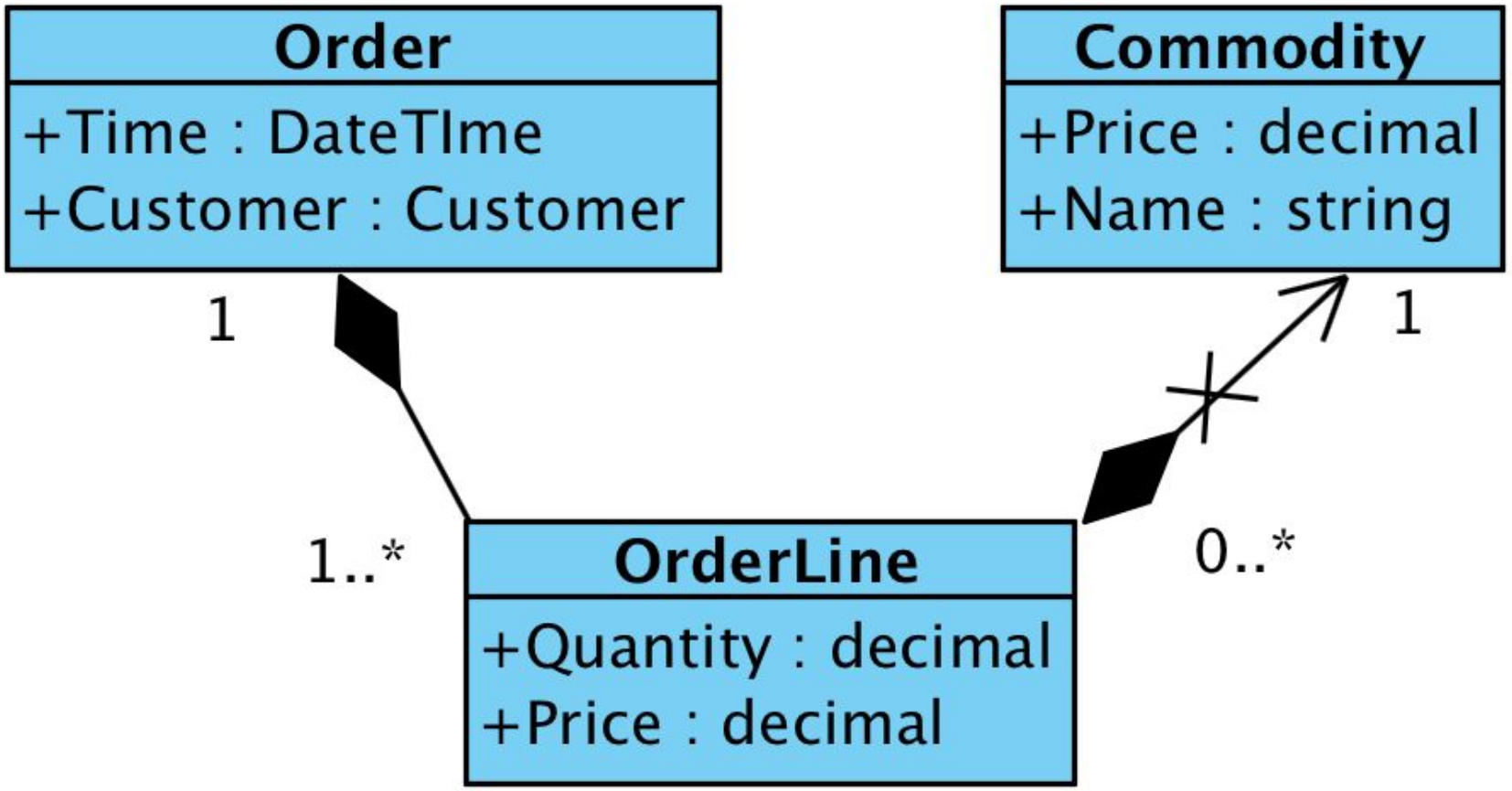
ИТОГ РУБ 232,66
ПОЛУЧЕНО 502,66 РУБ

Класс-ассоциация

Класс-ассоциация

```
public class Commodity {
    public string Name { get; set; }
    public decimal Price { get; set; }
}
public class Order {
    public DateTime Time { get; set; }
    public Customer Customer { get; }
    public OrderLine[] Lines { get; }
}
public class OrderLine {
    public decimal Price { get; set; }
    public decimal Quantity { get; set; }
    public Order Order { get; }
    public Commodity Commodity { get; }
}
```

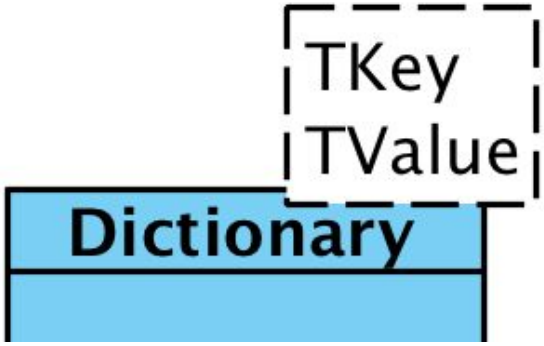
Класс-ассоциация: альтернативная интерпретация



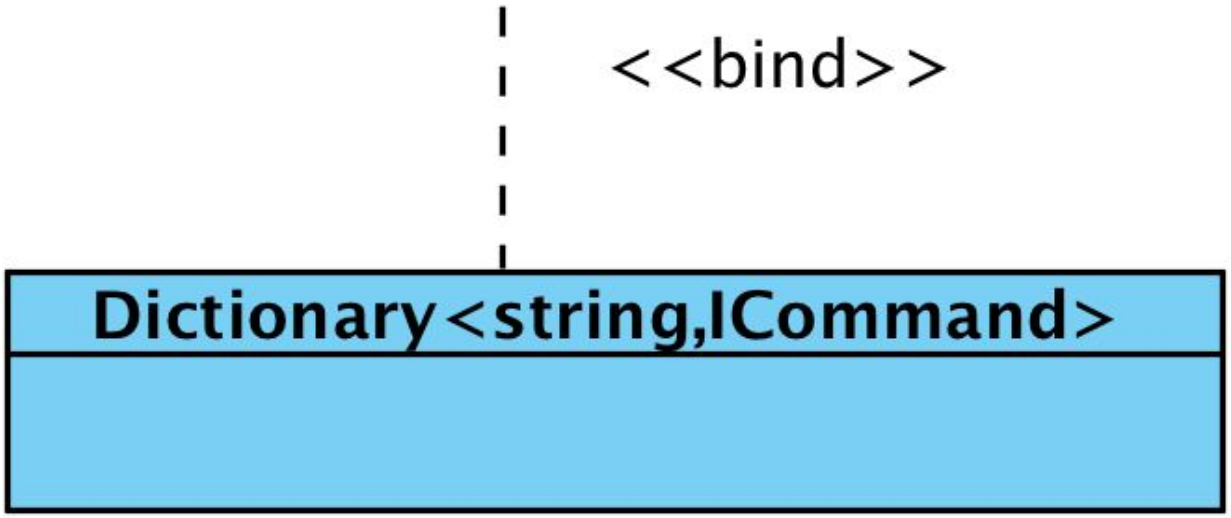
Класс-ассоциация

- **Имена участников, направленность, мощность:** определяется конкретикой задачи
- **Стереотип связи: «класс-ассоциация» -** дополнительная информация, которая характеризует связь
- **Видимость:** поля класса

Конкретизация




```
public class Application {
    readonly Dictionary<string, ICommand> commandMap;
```



Конкретизация

- Отношение между классами
- **Мощность 1:1**
- **Имена участников:** клас-щаблон (параметризованный класс), конкретный (конкретизованный) класс
- **Стереотип связи: «конкретизация» -**

  конкретизируем класс подставляя вместо классов-параметров конкретные классы
- **Видимость:** неприменимо

Задача: для всех объектов внутри контейнера проделать некую операцию.

Варианты:

Перебор объектов

- Сделать реализацию публичной
- Предоставить обратный вызов для обработки

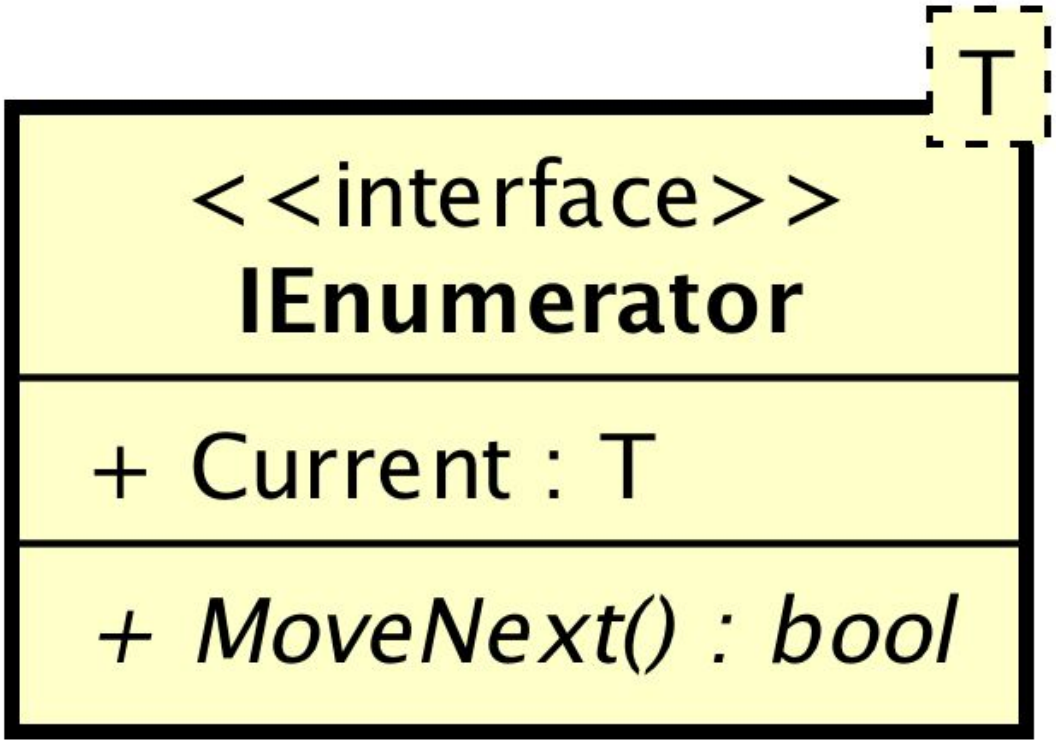
Перебор объектов в контейнере

```
var i = 0;
stack.Iterate(
    element =>
{
    Console.WriteLine(
        "{0}th element is {1}", i++, element
        );
});
```

- Довольно просто, но
- Немного, хм..., необычно

Итераторы — объекты для перебора объектов

Итераторы



Итератор для стека

```
public class Stack<T> {
    public class StackIterator : IEnumerator<T> {
        private readonly Stack<T> stack;
        private Node current;
        public StackIterator(Stack<T> stack) {
            this.stack = stack;
            Reset();
        }
        public T Current { get { return current.Data; } }
        public bool MoveNext() {
            current = current.Next;
            return current != null;
        }
        public void Reset() {
            current = stack.head;
        }
```

Итератор для стека

```
var stack = new Stack<int>();
stack.Push(2);
var i = 0;
var iterator = new Stack<int>.StackIterator(stack);
while (iterator.MoveNext()) {
    Console.WriteLine(
        "{0}th element is {1}", i++, iterator.Current
    );
}
```

- Обычный цикл, но
  - Странноватая инициализация
  - Нужно писать итератор

Итератор для стека

```
var stack = new Stack<int>();
stack.Push(2);
var i = 0;
foreach (var element in stack) {
    Console.WriteLine(
          "{0}th element is {1}", i++, element
          );
}
```

Бо́льшим магии!

Единственный минус — необходимость писать итератор

yield break

yield return

yield

```
public class Stack<T>
    public IEnumerator
        var current =
        while (current
            yield retu...
```

```
foreach (var element in stack) {
    Console.WriteLine(element);
}
```

```
public class Stack<T> {
    public void Iterate(Action<T> action) {
        var current = head;
        while (current != null) {
            action(current.Data);
            current = current.Next;
        }
    }
```

```
stack.Iterate(element =>
    Console.WriteLine(element)
);
```

foreach и IEnumerable<T>

- Для того, чтобы использовать Ваш класс в конструкции foreach необходимо реализовать интерфейс IEnumerable<T>
- Конструкция **yield return** позволяет синхронизировать цикл обработки и перебор элементов, не нарушая инкапсуляции и не усложняя код
- Конструкция **yield break** заканчивает перебор
- Метод, в котором используются эти конструкции должен возвращать IEnumerable или IEnumerator

Спасибо за внимание. Вопросы?