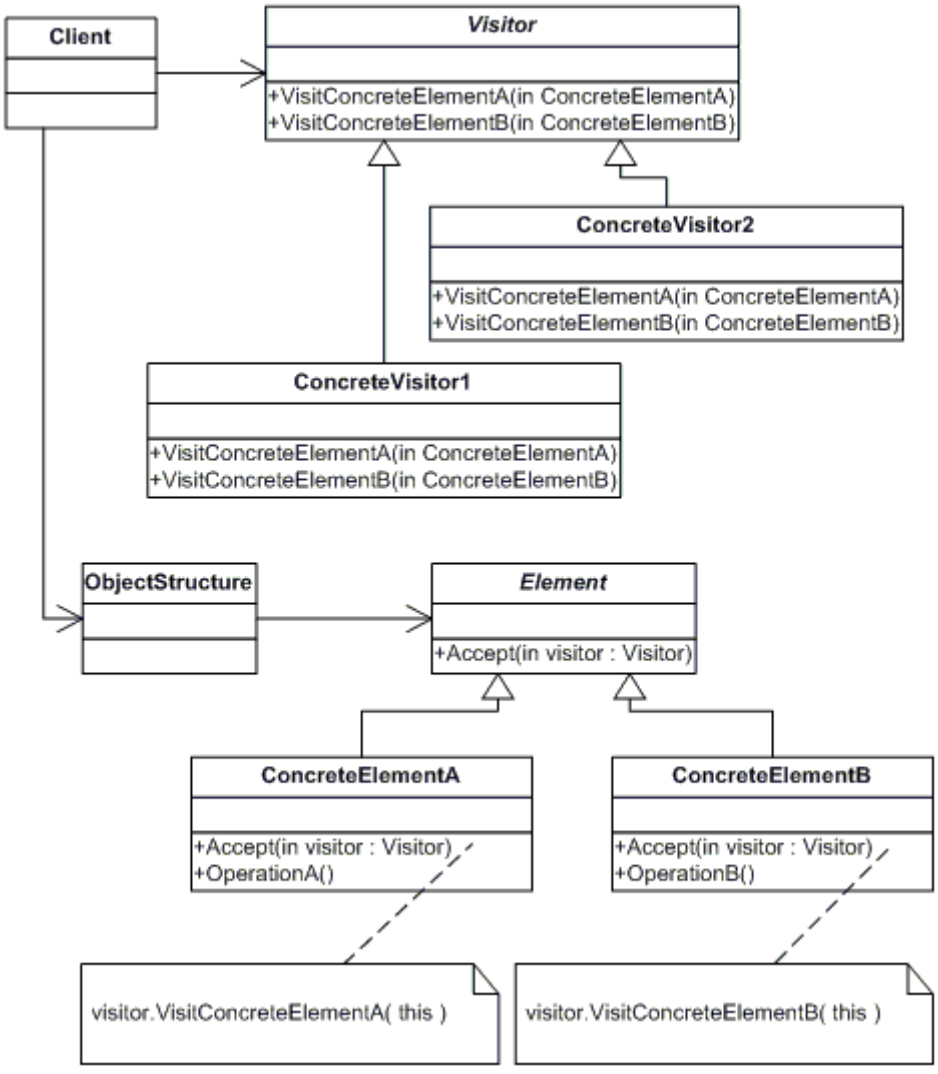
Projektowanie obiektowe oprogramowania
Wykład 6 – wzorce czynnościowe
Wiktor Zychla 2015

1 Od Composite structure do Visitor

Motto: definiowanie nowej operacji bez modyfikowania interfejsu
Kojarzyć: TreeVisitor z wykładu, ExpressionVisitor z biblioteki standardowej .NET



Uwaga! To pierwszy tak złożony wzorzec jaki poznajemy, dlatego warto mieć pod ręką przykład implementacji strukturalnej.

```
class Program
{
    static void Main( string[] args )
    {
        CompositeStructure cs = new CompositeStructure();
        cs.AddElement( new ConcreteElementA() );
        cs.AddElement( new ConcreteElementA() );
        cs.AddElement( new ConcreteElementB() );

        ConcreteVisitor1 cv1 = new ConcreteVisitor1();
        ConcreteVisitor2 cv2 = new ConcreteVisitor2();

        cs.Accept( cv1 );
        cs.Accept( cv2 );

        Console.ReadLine();
    }
}

public abstract class Element
{
    public abstract void Accept( Visitor v );
}

public class ConcreteElementA : Element
{
    public override void Accept(Visitor v)
    {
        v.VisitConcreteElementA( this );
    }
}

public class ConcreteElementB : Element
{
    public override void Accept(Visitor v)
    {
        v.VisitConcreteElementB( this );
    }
}

public class CompositeStructure
{
    private List<Element> elements = new List<Element>();
    public void AddElement( Element e )
    {
        this.elements.Add( e );
    }

    public void Accept( Visitor v )
    {
        foreach ( var e in elements )
            e.Accept( v );
    }
}

public abstract class Visitor
{
    public abstract void VisitConcreteElementA( ConcreteElementA elem );
    public abstract void VisitConcreteElementB( ConcreteElementB elem );
```

```
}

public class ConcreteVisitor1 : Visitor
{
    public override void VisitConcreteElementA( ConcreteElementA elem )
    {
        Console.WriteLine( "cv1 visiting A" );
    }

    public override void VisitConcreteElementB( ConcreteElementB elem )
    {
        Console.WriteLine( "cv1 visiting B" );
    }
}

public class ConcreteVisitor2 : Visitor
{
    public override void VisitConcreteElementA( ConcreteElementA elem )
    {
        Console.WriteLine( "cv2 visiting A" );
    }

    public override void VisitConcreteElementB( ConcreteElementB elem )
    {
        Console.WriteLine( "cv2 visiting B" );
    }
}
```

Komentarz. Podstawowy problem tak skonstruowanej struktury to jej złożoność. Studenci, którzy już rozumieją na czym polega Visitor pytają często „po co jest cała część akceptowania struktury visitora w elementach struktury kompozytowej? Dlaczego nie wystarczy sama hierarchia visitorów?”

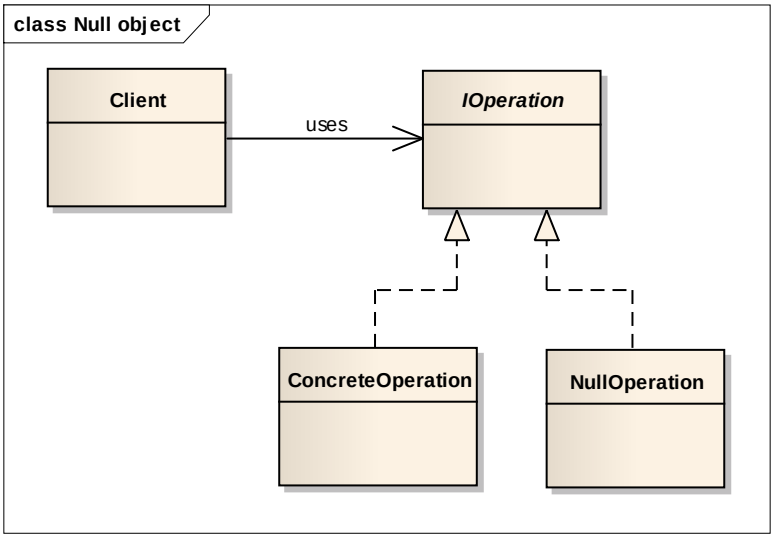
Odpowiedzi na to pytanie udzielimy na wykładzie – wszystko zależy od tego **kto** implementuje algorytm „odwiedzania” struktury kompozytowej.

Jeżeli robi to sam visitor – to wydaje się że nie ma w ogóle potrzeby aby istniała zależność od struktury kompozytowej do visitorów. Takie visitory są jednak bardziej złożone, choć jak pokazuje przykład z biblioteki standardowej, **ExpressionVisitor**, bywają wybierane przy implementacji.

Jeżeli robi to kompozyt – to visitory nie wiedzą jak wygląda wnętrze struktury kompozytoej i jak je odwiedzać i wtedy mamy taki podział odpowiedzialności między strukturą kompozytoową a visitorami jak na diagramie, ponieważ implementacja „odwiedzania” struktury, zawarta w kompozycie, deleguje „odwiedzanie” poszczególnych typów struktury do odpowiednich metod visitora.

2 Null Object

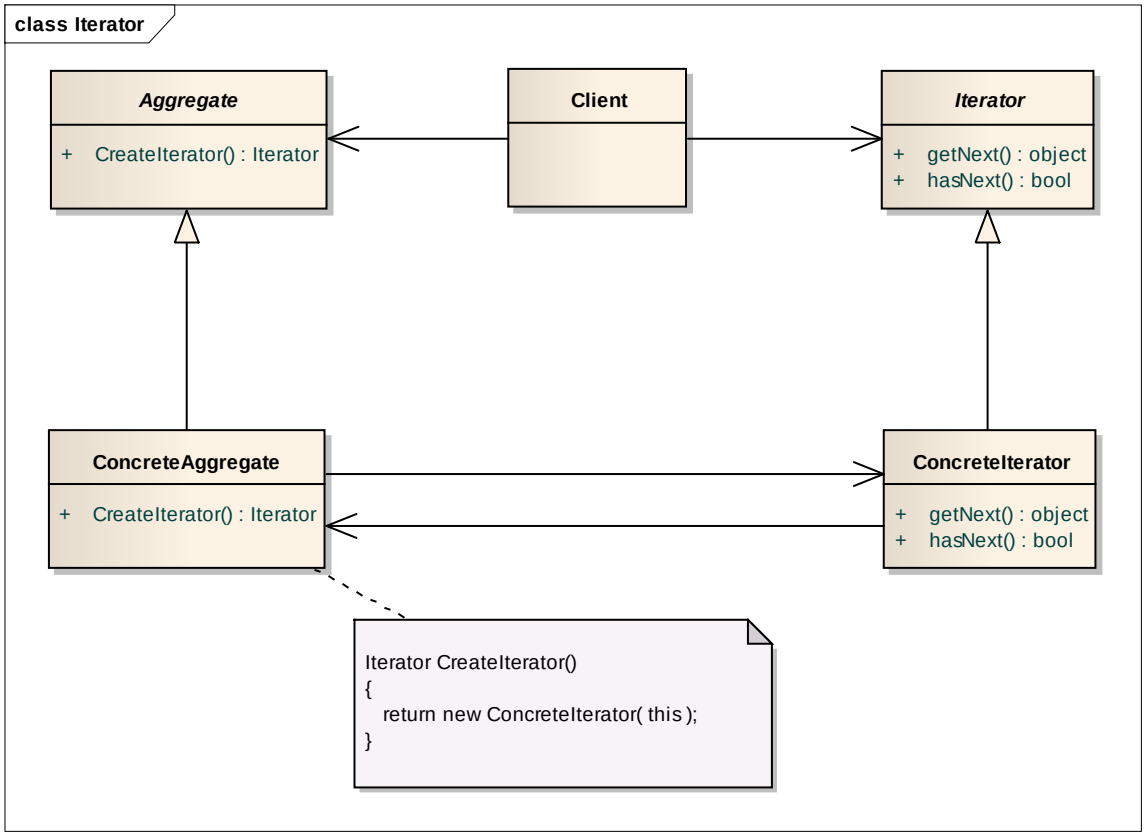
Motto: pusta implementacja zwalniająca klienta z testów **if** na *null*



Komentarz: Null object ma sens w połączeniu z fabryką – przy specyficznych lub niedostatecznych parametrach inicjalizacyjnych fabryka zwraca Null object.

3 Iterator

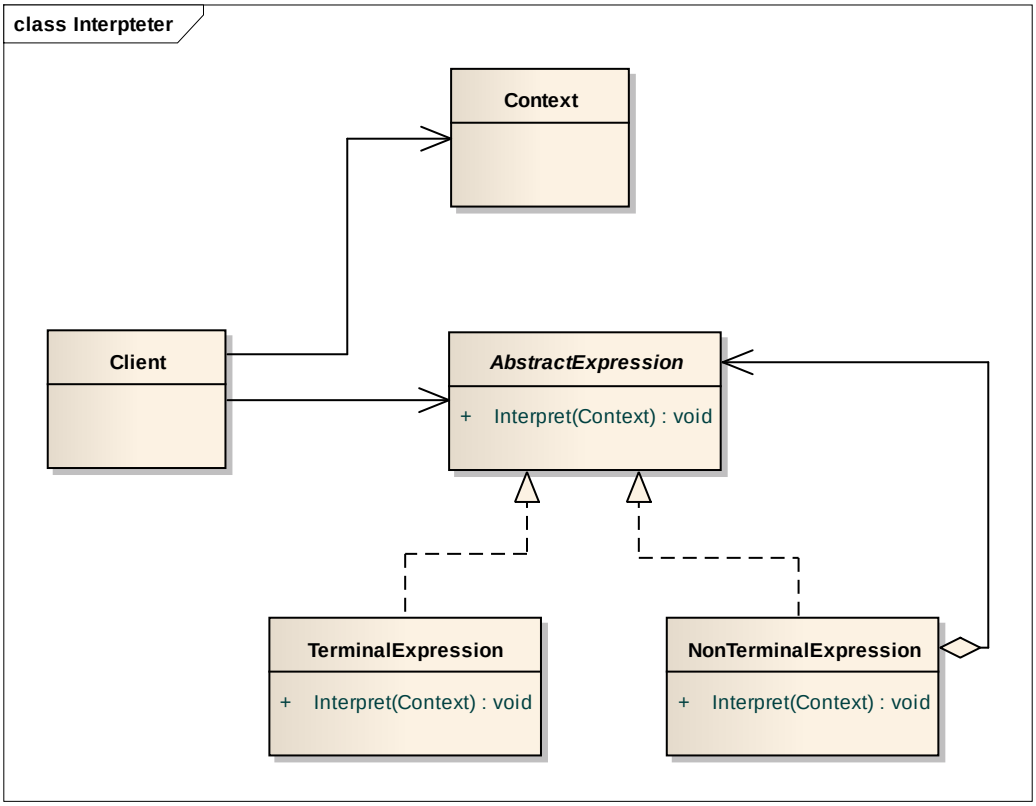
Motto: sekwencyjny dostęp do obiektu zagregowanego bez ujawniania jego struktury
Kojarzyć: *IEnumerator*, *IEnumerable*



Komentarz: ten wzorzec został z powodzeniem włączony do nowoczesnych języków programowania (Java, C#) stanowiąc podstawę dla lukru syntaktycznego (C# - **foreach**). To przykład jak wzorce projektowe silnie wpływają na języki.

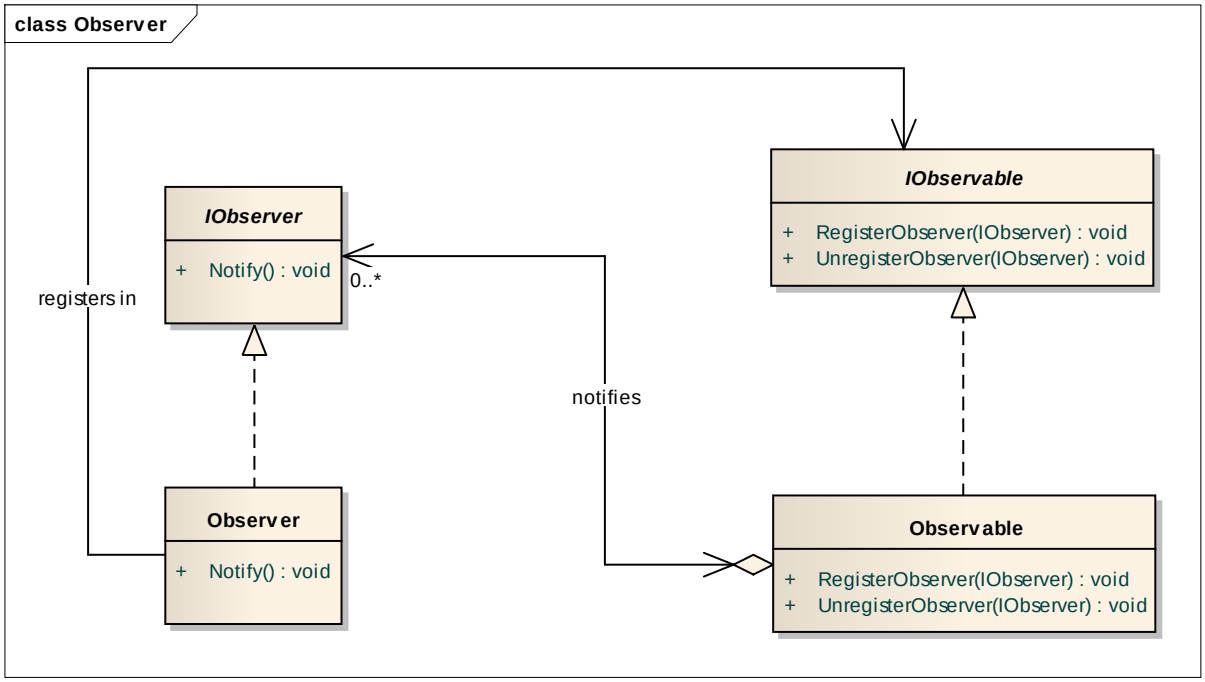
4 Interpreter (Little Language)

Motto: reprezentacja gramatyki języka i jego interpretera
Kojarzyć: kompozyt z interpreterem



5 Observer

Motto: powiadamianie zainteresowanych o zmianie stanu, dzięki czemu nie odwołują się one do siebie wprost.
Kojarzyć: zdarzenia w C#
Przykład z życia: architektura aplikacji oparta o powiadomienia między różnymi widokami (w środku okienka – Mediator, pomiędzy okienkami – Observer)
Jeszcze inaczej – Observer ujednolica interfejs „Colleagues” Mediatora, dzięki czemu obsługuje dowolną liczbę „Colleagues”



Komentarz: kolejny wzorzec który silnie wpływa na rozwój języków – C#-owe zdarzenia (events) to przykład uczynienia ze wzorca projektowego elementu języka.