Wzorzec Projektowy: Kompozyt

Agnieszka Mach
Robert Wojtaszek

05.09.2020

Czym jest wzorzec projektowy

Wzorzec projektowy (ang. design pattern) to uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. Pokazuje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie, modyfikację oraz utrzymanie kodu źródłowego. Jest opisem rozwiązania, a nie jego implementacją.
Wzorce projektowe stosowane są w projektach wykorzystujących *programowanie obiektowe*.

Kompozyt

Jego głównym zadaniem jest składanie obiektów w strukturę drzewa *(ang. tree structures)*, w celu przedstawienia hierarchii częściowej. Dzięki temu, między innymi, klient widzi zamiast wielu obiektów tylko jeden (klient może traktować wiele obiektów jak jeden). Aplikacje graficzne (jak np. edytory rysunków i systemy przechwytywania schematów) pozwalają użytkownikom budować **złożone diagramy z prostych komponentów**. Użytkownik może grupować je w celu uformowania "większych komponentów", które mogą w następstwie być grupowane w jeszcze większe składniki.

Zastosowanie

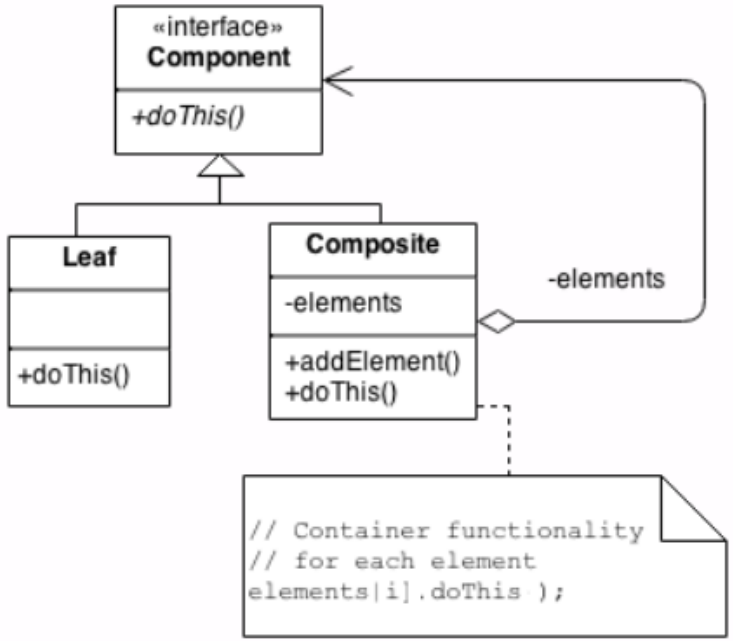
Wzorzec ten stosuje się, gdy wygodniej jest korzystać z pewnych operacji dla danego obiektu w ten sam sposób jak dla grupy obiektów, np. rysując na ekranie prymitywy lub obiekty złożone z prymitywów; zmieniając rozmiar zarówno pojedynczych prymitywów jak i obiektów złożonych z prymitywów (z zachowaniem proporcji).

Cele

- Utworzenie obiektów w tzw. *struktury drzewiaste*
- Składanie obiektów w taki sposób, aby klient widział wiele z nich jako jeden obiekt
- Kompozycja rekurencyjna

Struktura - diagram klas wzorca UML

Kompozyt operuje na wielu obiektach. Poniżej znajduje się diagram klas ów wzorca projektowego UML.



Budowa Kompozytu

1. **Component** – klasa abstrakcyjna lub interfejs reprezentujący pojedyncze obiekty *Leaf*
2. **Leaf** – typ prosty (nie posiada potomków)
3. **Composite** – przechowuje obiekty proste (*Leaf*), implementuje zachowanie elementów które zawiera.

Rzecz jasna **Kompozyt** składa się także z klienta, który operuje na danych we wzorcu. Ponadto *Composite* oraz *Leaf* dziedziczy po tym samym interfejsie co pozwala na dostęp do obiektów prostych w ten sam sposób jak do grupy tych obiektów. Użytkownik może przeprowadzać operacje na pojedynczym obiekcie, jak i na grupie obiektów reprezentowanych tym wzorcem.

Następstwa stosowania

- Możliwość definiowania hierarchii z obiektów prostych i złożonych
- Upraszczanie kodu klientów
- Ułatwianie dodawania komponentów nowego rodzaju
- Szansa na zanadto ogólny projekt

Przykład 1.

```
#include <iostream>
using System;
using System.Collections.Generic;
using System.Text;

interface IMinister
{
    void InformacjeMinister();
}

class Minister : IMinister  //tworzenie klasy
{
    private string Imie, Nazwisko, Funkcja;  //tworzenie trzech zmiennych, które są ciągiem znaków

    public Minister(string Imie, string Nazwisko, string Funkcja)
    {
        Imie = imie;            //
        Nazwisko = nazwisko;    // rownowaznosc zapisow
        Funkcja = funkcja;      //
    }

    public void InformacjeMinister()   // definicja funkcji zawierająca informacje o ministrach
    {
        Console.WriteLine(Imie + " " + Nazwisko + " " + Funkcja);  //Zapisuje określoną wartość ciągu, a po niej bieżący terminator wiersza
    }
}

class Premier : IMinister // Tworzenie klasy; chcemy mieć sytuację jak w drzewie, gdzie wyżej jest premier, a niżej natomiast można znaleźć ministrów
{
    private string Imie, Nazwisko, Funkcja;
    private List<IMinister> lista_ministrowie = new List<IMinister>(); //tworzenie listy ministrów

    public Premier(string Imie, string Nazwisko, string funkcja) // tutaj podajemy wszystkie dane na temat premiera
    {
        Imie = imie;
        Nazwisko = nazwisko;
        Funkcja = funkcja;
    }
```

Przykład 1.

```
39
40      public void AddMinister(IMinister minister) // ta funkcja ma na celu dodanie nowych ministrów do istniejącej już listy
41      {
42          ministrowie.Add(minister);
43      }
44
45      public void InformacjeMinister()
46      {
47          Console.WriteLine("Wsparcie_premiera: " + Imie + " " + Nazwisko);  //dodadnie nowej funkcji, ktora jest w srodku hierarchii
48          foreach(IMinister minister w ministrowie)
49          {
50              minister.InformacjeMinister();
51          }
52      }
53  }
54
```

Przykład 1.

```
1
2   static void Main(string[] args)
3   {
4       Premier premier = new Premier("Adam", "Kowalski", "premier");
5       premier.AddMinister(new Minister("Józef", "Sadełko", "minister zdrowia"));
6       premier.AddMinister(new Minister("Jacek", "Milion", "minister aktywów państwowych"));
7       premier.AddMinister(new Minister("Gustaw", "Kowalski", "minister szkolnictwa"));
8
9       premier.InformacjeMinister();
10
11      Console.ReadKey();
12  }
```

Przykład 2.

```
namespace Shapes //chcemy narysować parę figur, ale nie chcemy się z nimi obchodzić pojedynczo, tylko chcemy np rysować jakieś figury i zmieniać je np na kolor czerwony jak jeden obiek
{
    public interface IShape
    {
        void draw(string fillColor);
    }

    public class Triangle : IShape //utworzenie klasy, która określa trójkąt
    {
        public void draw(string fillColor)
        {
            Console.WriteLine("Drawing Triangle with color " + fillColor);
        }
    }

    public class Circle : IShape //utworzenie klasy, któta określa okrąg
    {
        public void draw(string fillColor)
        {
            Console.WriteLine("Drawing Circle with color " + fillColor);
        }
    }

    public class Drawing : IShape
    {

        private List<IShape> shapes = new List<IShape>();

        public void draw(string fillColor)
        {
            foreach(IShape sh in shapes)
            {
                sh.draw(fillColor);
            }
        }

```

Przykład 2.

```
37          public void add(IShape s)
38          {
39              shapes.Add(s);
40          }
41
42          public void remove(IShape s)
43          {
44              shapes.Remove(s);
45          }
46
47          public void clear()
48          {
49              Console.WriteLine("Clearing all the shapes from drawing");
50              shapes.Clear();
51          }
52      }
53  }
```

Przykład 2.

```
namespace Graphic //dodajemy figury i zmieniamy kolor wszystkich dodanych figur do listy metodą draw()
{
    class Program
    {
        static void Main(string[] args)
        {
            IShape tri = new Triangle();
            IShape tri1 = new Triangle();
            IShape cir = new Circle();

            Drawing drawing = new Drawing();
            drawing.add(tri1);
            drawing.add(tri1);
            drawing.add(cir);

            drawing.draw("Red");

            drawing.clear();

            drawing.add(tri);
            drawing.add(cir);
            drawing.draw("Green");

            Console.ReadKey();
        }
    }
}
```

Bibliografia

- "Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku- Enrich Gamma, Richard Helm, Ralp Johnson, John Vissides
- https://devman.pl/pl/techniki/wzorce-projektowe-9-kompozytcomposite/
- https://pl.wikipedia.org/wiki/Wzorzecprojektowy(informatyka)
- https://pl.wikipedia.org/wiki/Kompozyt(wzorzecprojektowy)