# **C# Composite**

Шаблон дизайну "Композиція" об'єднує об'єкти у деревоподібні структури для представлення ієрархій "частина-ціле". Цей патерн дозволяє клієнтам однаково поводитися з окремими об'єктами та композиціями об'єктів.

До класів та об'єктів, що беруть участь у цьому паттерні, належать такі:

* **Component (DrawingElement) -** оголошує інтерфейс для об'єктів у композиції,

реалізує поведінку за замовчуванням для інтерфейсу, спільного для всіх класів, за необхідності, оголошує інтерфейс для доступу та керування дочірніми компонентами,

* (необов'язково) визначає інтерфейс для доступу до батька компонента у рекурсивній структурі та реалізує його, якщо це доречно.
* **Leaf (PrimitiveElement) -** представляє об'єкти типу leaf у композиції. Лист не має дочірніх об'єктів, визначає поведінку примітивних об'єктів у композиції.
* **Composite (CompositeElement) -** визначає поведінку для компонентів, що мають дочірні елементи, зберігає дочірні компоненти, реалізує операції, пов'язані з дочірніми компонентами, в інтерфейсі **Component**.
* **Client (CompositeApp) -** маніпулює об'єктами в композиції через інтерфейс **Component**.

## **Structural code in C#**

using System;

using System.Collections.Generic;

namespace Composite.Structural

{

/// <summary>

/// Composite Design Pattern

/// </summary>

public class Program

{

public static void Main(string[] args)

{

// Create a tree structure

Composite root = new Composite("root");

root.Add(new Leaf("Leaf A"));

root.Add(new Leaf("Leaf B"));

Composite comp = new Composite("Composite X");

comp.Add(new Leaf("Leaf XA"));

comp.Add(new Leaf("Leaf XB"));

root.Add(comp);

root.Add(new Leaf("Leaf C"));

// Add and remove a leaf

Leaf leaf = new Leaf("Leaf D");

root.Add(leaf);

root.Remove(leaf);

// Recursively display tree

root.Display(1);

// Wait for user

Console.ReadKey();

}

}

/// <summary>

/// The 'Component' abstract class

/// </summary>

public abstract class Component

{

protected string name;

// Constructor

public Component(string name)

{

this.name = name;

}

public abstract void Add(Component c);

public abstract void Remove(Component c);

public abstract void Display(int depth);

}

/// <summary>

/// The 'Composite' class

/// </summary>

public class Composite : Component

{

List<Component> children = new List<Component>();

// Constructor

public Composite(string name)

: base(name)

{

}

public override void Add(Component component)

{

children.Add(component);

}

public override void Remove(Component component)

{

children.Remove(component);

}

public override void Display(int depth)

{

Console.WriteLine(new String('-', depth) + name);

// Recursively display child nodes

foreach (Component component in children)

{

component.Display(depth + 2);

}

}

}

/// <summary>

/// The 'Leaf' class

/// </summary>

public class Leaf : Component

{

// Constructor

public Leaf(string name)

: base(name)

{

}

public override void Add(Component c)

{

Console.WriteLine("Cannot add to a leaf");

}

public override void Remove(Component c)

{

Console.WriteLine("Cannot remove from a leaf");

}

public override void Display(int depth)

{

Console.WriteLine(new String('-', depth) + name);

}

}

}