**[Composite Pattern](https://dotnetcorecentral.com/blog/composite-pattern/)**

Saluti, colleghi sviluppatori! Bentornati a Dot Net Core Central. Oggi ci addentriamo nell'affascinante regno dei design pattern, concentrando la nostra attenzione sul Composite Design Pattern, un gioiello strutturale degli illustri design pattern Gang of Four.

**Svelare la potenza del modello composito**

**Il Composite Design Pattern, incastonato nella categoria dei modelli di progettazione strutturale, offre una soluzione solida alle sfide della rappresentazione gerarchica degli oggetti. Il suo intento primario è comporre gli oggetti in strutture ad albero, formando gerarchie di parti e interi. Questo approccio architettonico consente ai clienti di trattare oggetti singoli e composizioni di oggetti in modo uniforme, creando un'esperienza fluida.**

**Navigating Hierarchical Object Structures**

**Nel mondo dello sviluppo software, avere a che fare con oggetti gerarchici è inevitabile. Immagina una gerarchia composta da nodi e foglie, con un nodo base o radice che avvia la struttura. Questa composizione ricorsiva, simile a una struttura di cartelle in un PC, costituisce la base del Composite Design Pattern.**

**Una sfida significativa si presenta nello stabilire una struttura o un contratto uniforme tra foglie e nodi. Qui sta l'essenza del pattern: consentire ai client di trattare oggetti e composizioni individuali in modo uniforme. Tuttavia, è fondamentale riconoscere una lacuna: la esploreremo in dettaglio mentre ci immergiamo nell'implementazione.**

**Creazione del modello di progettazione composito**

**Per illustrare la potenza del Composite Design Pattern, creiamo una gerarchia di classi che rappresenti una struttura di oggetti. La nostra gerarchia comprende nodi compositi e nodi foglia, che emulano una struttura ad albero.** **Il viaggio inizia con la definizione di una classe base astratta, Node, che presenta metodi essenziali per aggiungere e rimuovere nodi, stampare figli e stampare genitori. Questa classe base costituisce il progetto per i nodi compositi e foglia.**

**public abstract class Node**

**{**

**public abstract string name { get; }**

**public virtual void add(Node node) => throw new NotImplementedException();**

**public virtual void Remove(Node node) => throw new NotImplementedException();**

**public virtual void StampaFiglio(Node node) => throw new NotImplementedException();**

**public virtual void StampaPadre(Node node) => throw new NotImplementedException();**

**}**

**Implementing Composite Nodes**

**La prima classe che creiamo è CompositeNode, che eredita dalla classe base Node. CompositeNode consente l'aggiunta e la rimozione di nodi figlio, la stampa dei figli e la stampa del genitore. Abbraccia senza soluzione di continuità la struttura gerarchica.**

**internal class CompositeNode : Node**

**{**

**private List<Node> children = new List<Node>();**

**private readonly string name;**

**private readonly Node parentNode;**

**public override string Name => name;**

**public CompositeNode(string nome, Node parentNode = null)**

**{**

**this.name = nome;**

**this.parentNode = parentNode;**

**}**

**public override void Add(Node node)**

**{**

**children.Add(node);**

**}**

**public override void Remove(Node node)**

**{**

**children.Remove(node);**

**}**

**public override void StampaFiglio(Node node)**

**{**

**if (parentNode != null)**

**Console.WriteLine($"Parent: {parentNode.Name}");**

**else**

**Console.WriteLine("Root node");**

**}**

**public override void StampaPadre(Node node)**

**{**

**Console.WriteLine**

**($"Children: {string.Join(",", children.Select(c => c.Name))}");**

**}**

**}**

**Embracing Leaf Nodes**

**La classe LeafNode, un'altra progenie della classe base Node, rappresenta i nodi foglia nella nostra gerarchia. Con un nodo padre obbligatorio, il focus di LeafNode è sulla stampa del padre e sull'aderenza al contratto uniforme.**

**internal class NodoFiglio : Node**

**{**

**private readonly string name;**

**private readonly Node parentNode;**

**public override string Name => name;**

**public NodoFiglio(string name, Node parentNode)**

**{**

**this.name = name;**

**this.parentNode = parentNode;**

**}**

**public override void StampaPadre(Node node)**

**{**

**Console.WriteLine($"Parent: {parentNode.Name}");**

**}**

**}**

**Building and Testing the Hierarchy**

**Nella classe del programma, costruiamo una struttura gerarchica, che include un nodo radice, nodi compositi e nodi foglia. Con aggiunte e rimozioni attentamente orchestrate, mostriamo il comportamento del Composite Design Pattern in azione. L'output dimostra elegantemente le relazioni gerarchiche, convalidando l'efficacia del pattern.**

**using CompositePattern.Demo;**

**var root = new CompositeNode("Root");**

**var node1 = new CompositeNode("node1", root);**

**var leaf11 = new LeafNode("leaf11", node1);**

**var leaf12 = new LeafNode("leaf12", node1);**

**node1.Add(leaf11);**

**node1.Add(leaf12);**

**var node2 = new CompositeNode("node2", root);**

**var leaf1 = new LeafNode("leaf1", root);**

**root.Add(leaf1);**

**root.Add(node1);**

**root.Add(node2);**

**Console.WriteLine("Root ======");**

**root.PrintParent();**

**root.PrintChildren();**

**Console.WriteLine("leaf1 ======");**

**leaf1.PrintParent();**

**Console.WriteLine("node1 ======");**

**node1.PrintParent();**

**node1.PrintChildren();**

**Console.WriteLine("leaf11 ======");**

**leaf11.PrintParent();**

**leaf11.PrintChildren();**

**Unpacking the Shortcomings**

**Sebbene il Composite Design Pattern fornisca una soluzione elegante, introduce una sfida. Il contratto uniforme, sebbene vantaggioso, porta a una potenziale incoerenza. Ad esempio, il metodo PrintChildren è valido per i nodi compositi ma non valido per i nodi foglia. Raggiungere un equilibrio tra uniformità e comportamento su misura è una considerazione sfumata nell'implementazione di questo pattern.**

**Conclusion: Harnessing the Composite Design Pattern**

**Nell'arazzo dei design pattern, il Composite Design Pattern si erge come uno strumento versatile per la gestione di strutture gerarchiche. Mentre il suo contratto uniforme può presentare delle sfide, i vantaggi in termini di organizzazione del codice e flessibilità sono innegabili. E con questo, concludiamo la nostra esplorazione del Composite Design Pattern in .NET Core. Se questo viaggio ti ha colpito, non dimenticare di leggere il resto dei miei post del blog.**