**Strategy Pattern**

Saluti, colleghi sviluppatori! Bentornati a .Net Core Central. Nel post del blog di oggi, ci addentriamo nell'affascinante mondo dei design pattern, con un'attenzione particolare allo Strategy Design Pattern, un membro versatile dei design pattern Gang of Four.

**Understanding the Essence of Strategy Pattern**

Lo Strategy Design Pattern, classificato nei behavioral design pattern, ruota attorno all'idea di definire una famiglia di algoritmi e di incapsulare ciascuno di essi. L'obiettivo primario è rendere questi algoritmi facilmente intercambiabili, consentendo variazioni nel comportamento indipendentemente dal client che li utilizza.

Mentre i dettagli diventano più chiari nella fase di implementazione, il concetto di alto livello prevede la creazione di un contratto, l'implementazione di diversi algoritmi su tale contratto e la possibilità per i clienti di passare senza problemi da un'implementazione all'altra.

**Implementing Strategy Pattern in .NET Core**

Adottiamo un approccio pratico per comprendere meglio questo pattern. Uno scenario in cui lo Strategy Design Pattern brilla è quando si ha a che fare con più algoritmi, come vari metodi di crittografia o tecniche di ordinamento. Nel nostro esempio, ci concentreremo sugli algoritmi di ordinamento: Quick Sort e Bubble Sort.

**Quick Sort Implementation**

Innanzitutto, creiamo un'interfaccia denominata ISortStrategy con un singolo metodo, Sort. La classe QuickSort implementa quindi questa interfaccia, mostrando un semplice algoritmo Quick Sort. Questa tecnica di ordinamento comporta la divisione dell'array in base a un punto medio, la creazione di raccolte a sinistra e a destra e il loro ordinamento ricorsivo.

internal class QuickSort : ISortStrategy

{

public int[] Sort(int[] inputArray)

{

if (inputArray == null || inputArray.Length <= 1) return inputArray;

var centerIndex = inputArray.Length / 2;

var centerValue = inputArray[centerIndex];

var leftPart = new List<int>();

var rightPart = new List<int>();

for (var i = 0; i < inputArray.Length; i++)

{

if (i == centerIndex) continue;

if (inputArray[i] <= centerValue)

leftPart.Add(inputArray[i]);

else

rightPart.Add(inputArray[i]);

}

var sorted = Sort(leftPart.ToArray()).ToList();

sorted.Add(centerValue);

sorted.AddRange(Sort(rightPart.ToArray()));

return sorted.ToArray();

}

}

**The Client: Sorted Numbers**

Per legare il tutto insieme, introduciamo una classe client denominata SortedNumbers. Questa classe utilizza lo Strategy Pattern accettando un ISortStrategy nel suo costruttore. Il metodo Sort di questa classe delega quindi l'operazione di ordinamento alla strategia iniettata, fornendo flessibilità nella scelta dell'algoritmo di ordinamento.

**Running the Strategy Pattern**

Nella classe Program, creiamo un array di numeri casuali e istanziamo la classe SortedNumbers due volte. Una volta con Bubble Sort come strategia e una volta con Quick Sort. Chiamando il metodo Sort, osserviamo l'output ordinato in base alla strategia scelta.

public void Innesco()

{

var items = new[] { 88, 54, 34, 87, 15, 8, 105 };

var sortedNumbers = new SortedNumbers(new BubbleSort());

sortedNumbers.Sort(items);

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

sortedNumbers = new SortedNumbers(new QuickSort());

sortedNumbers.Sort(items);

}

**Conclusion: Where Strategy Shines**

Lo Strategy Design Pattern eccelle in scenari in cui sono necessari algoritmi multipli. Nel nostro esempio, abbiamo dimostrato come gestisce senza sforzo l'ordinamento con strategie diverse. Seguendo i principi SOLID, in particolare il Single Responsibility Principle, l'implementazione è pulita, manutenibile ed estensibile.

Ed ecco fatto: una visione più approfondita dello Strategy Design Pattern in .NET Core. Se hai trovato questa esplorazione illuminante, non dimenticare di leggere il resto dei miei post del blog. Per altri contenuti illuminanti, prendi in considerazione di andare sul mio canale YouTube e di iscriverti a .Net Core Central. Grazie per esserti unito a me in questo viaggio attraverso i design pattern!

https://dotnetcorecentral.com/blog/strategy-pattern/