**Interpreter Pattern**

Saluti, colleghi sviluppatori! Unitevi a me su .Net Core Central per un altro interessante viaggio nel mondo dei design pattern. Nel post del blog di oggi, sveleremo le complessità dell'Interpreter Design Pattern, una preziosa aggiunta ai rinomati design pattern Gang of Four.

Navigating the Landscape of Interpreter Design Pattern

L'Interpreter Design Pattern, incastonato nella categoria dei behavioral design pattern, si distingue come uno strumento potente. Il suo obiettivo principale è quello di fornire un mezzo per interpretare le frasi in un linguaggio definendo una rappresentazione della sua grammatica insieme a un interprete. Questo interprete, a sua volta, sfrutta la rappresentazione grammaticale per comprendere ed eseguire le frasi nel linguaggio dato.

In termini più semplici, immagina di avere un linguaggio, di definirne la grammatica e di impiegare un interprete per comprendere ed eseguire frasi in quel linguaggio. Per illustrare questo schema, ci immergeremo in un esempio del mondo reale: la creazione di una classe che funge da interfaccia a riga di comando per la comunicazione con il software sottostante, concentrandosi sulla conversione di stringhe.

**Crafting the Interpreter Pattern Framework**

Il viaggio inizia con la creazione di una classe Context, una semplice entità che contiene sia un valore che un'espressione, entrambi rappresentati come stringhe. Il valore può essere manipolato dall'interprete, rendendolo un elemento centrale nel nostro pattern.

internal class Context

{

public Context(string expression, string value)

{

Expression = expression;

Value = value;

public string Expression { get; }

public string Value { get; set; }

}

Successivamente, definiamo l'interfaccia IExpression, caratterizzata da un singolo metodo, Evaluate, che accetta un Context per la valutazione. Per il nostro esempio, ci concentriamo su due tipi di espressione: LowercaseExpression e UppercaseExpression, ognuno dei quali gestisce la conversione delle stringhe in minuscolo e maiuscolo, rispettivamente.

internal interface IExpression

{

    void Evaluate(Context context);

}

internal class LowerCaseExpression : IExpression

{

public void Evaluate(Context context)

    {

        context.Value = context.Value.ToLower();

    }

internal class UpperCaseExpression : IExpression

{

public void Evaluate(Context context)

{

context.Value = context.Value.ToUpper();

}

}

The Interpreter Pattern in Action

Il cuore della nostra implementazione risiede nella classe Interpreter. Armata di un singolo metodo Interpret, prende un Context e intraprende il viaggio di interpretazione del messaggio in arrivo. Seziona l'espressione, identifica i tipi di espressione appropriati, li valuta e stampa il valore modificato.

**Interactive Demonstration**

To bring our implementation to life, we step into the program class. Here, we prompt the user for a sentence and an expression. The interpreter then transforms the input based on the specified expression—whether converting to lowercase or uppercase.