UNIVERSIDAD TECNOLÓGICA NACIONAL

Patrón Interpreter

Cátedra:

- **Diseño de sistemas**

Docentes:

- **Juan Pablo Ferreyra**
- **Pablo Pioli**

Alumno:

- **Musso Tulio (14502)**

Año: 2021

Índice

Patrón Interpreter

Se trata de un patrón de diseño de comportamiento el cual es utilizado para evaluar un lenguaje definido como Expresiones, este patrón nos permite interpretar un lenguaje como Java, C#, SQL o incluso un lenguaje inventado por nosotros el cual tiene un significado, y darnos una respuesta tras evaluar dicho lenguaje.

Este patrón propone un intérprete para dicho lenguaje, es decir que crea un sistema que entiende y traduce ese lenguaje.

El patrón nos pide encontrar un lenguaje (gramática y sus reglas) que modele el problema y luego se procede a representar esa gramática con herramientas de software utilizando programación orientada a objetos.

De estos objetos se obtiene el intérprete el cual va a solucionar nuestro problema.

Se aplica cuando:

- Hay un lenguaje que interpretar y se pueden representar las sentencias del lenguaje como arboles sintácticos abstractos (se trata de una instancia del patrón composite).
- La gramática es simple.
- La eficiencia no es una preocupación critica o maxima.

Ventajas y desventajas de su uso

Ventajas:

- Fácil modificación y ampliación de los elementos gramaticales (al ser representados mediante una jerarquía de clases).
- Su implementación es sencilla (en comparación con otros métodos para implementar esta funcionalidad).
- La implementación de los métodos para cada elemento del lenguaje es dinámica (puede cambiarse en tiempo de ejecución).

Desventajas:

- No es muy eficiente: Se utilizan una gran cantidad de bucles y llamadas recursivas por lo que puede significar una eficiencia baja de ejecución.
- Limitación en el tipo de gramáticas: no cubre gramáticas complejas o son difíciles de mantener.

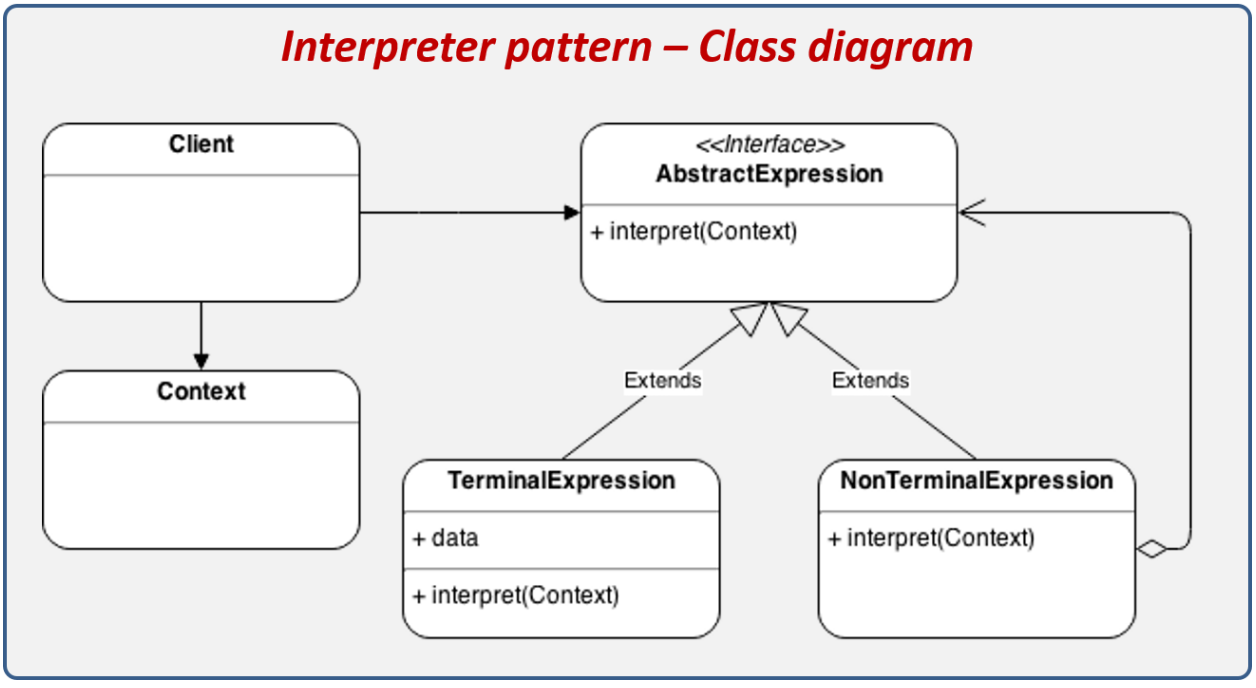
Solución y Estructura

La solución es representar la gramática del lenguaje (previamente definida) mediante una jerarquía de objetos. Los nodos terminales se representan creando clases TerminalExpression y los nodos no terminales con NonterminalExpression. Ambas clases implementan una interfaz común (o heredan de una clase abstracta común) llamada AbstractExpression y que contendrá la declaración del método interpreter(), que se encargará de evaluar el nodo en concreto.

Además puede existir un contexto común a todas las expresiones que defina ciertos valores, funciones o características del lenguaje que queremos interpretar. Este contexto será representado con la clase Context. Client se encargará de construir el árbol sintáctico de la expresión y asignar el contexto en caso de haberlo.

La estructura es la siguiente:

Estructura del patrón de diseño Interpreter



- **Client**: Actor que dispara la ejecución del interpreter. (son las herramientas para hacer la interpretación junto con el context)
- **Context:** Objeto con información global que será utilizada por el intérprete para leer y almacenar información global entre todas las clases que conforman el patrón y este es enviado al interpreter.
- **AbstractExpression:** Declara una interfaz para la ejecución de una operación. Se trata de una clase Abstracta.
- **TerminalExpression:** Se refiere a expresiones que no tienen más continuidad y al ser evaluadas o interpretadas terminan la ejecución de esa rama. Estas expresiones marcan el final de la ejecución de un sub-árbol de la expresión. OPERA CON LOS SIMBOLOS TERMINALES DE LA GRAMATICA.
- **NonTerminalExpression:** Son expresiones compuestas y dentro de ellas existen más expresiones que deben ser evaluadas. Estas estructuras son interpretadas utilizando recursividad hasta llegar a una expresión Terminal. OPERA CON LOS SIMBOLOS NO TERMINALES DE LA GRAMATICA

Se adjunta a continuación un ejemplo del patrón Interpreter en C#.