I. INFORMACION GENERAL			
TEMA:	Ejemplo: Patrón Interprete o Interpreter		
ASIGNATURA	Patrones de Software	**NIVEL**	Quinto "A"
UOC	Básica	**CICLO ACADEMICO**	Abril 2023 – Septiembre 2023
DOCENTE	Ing. Marco Guachimboza, Mg.		**FECHA:**
INTEGRANTES	Caiza Angel , Vichecela Kevin, Pincha Diego.		

II. DESARROLLO

Patrón Interprete o Interpreter

El patrón de diseño interpreter 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.

Interpreter es uno de los patrones de diseño más complejos debido a que para su funcionalidad debe combinar técnicas de programación orientada a objetos avanzada y su interpretación puede ser algo confusa, las principales cosas con las que nos enfrentaremos son la Herencia, Polimorfismo y la Recursividad.

Clasificación: Comportamiento

Propósito: Intérprete o Interpreter en inglés es un patrón, que pertenece a la clasificación de Comportamiento, porque su principal función es incluir elementos como moverse a través de una secuencia o interpretar en lenguaje, que es básicamente, agregar nuevos comportamientos a nuestro software.

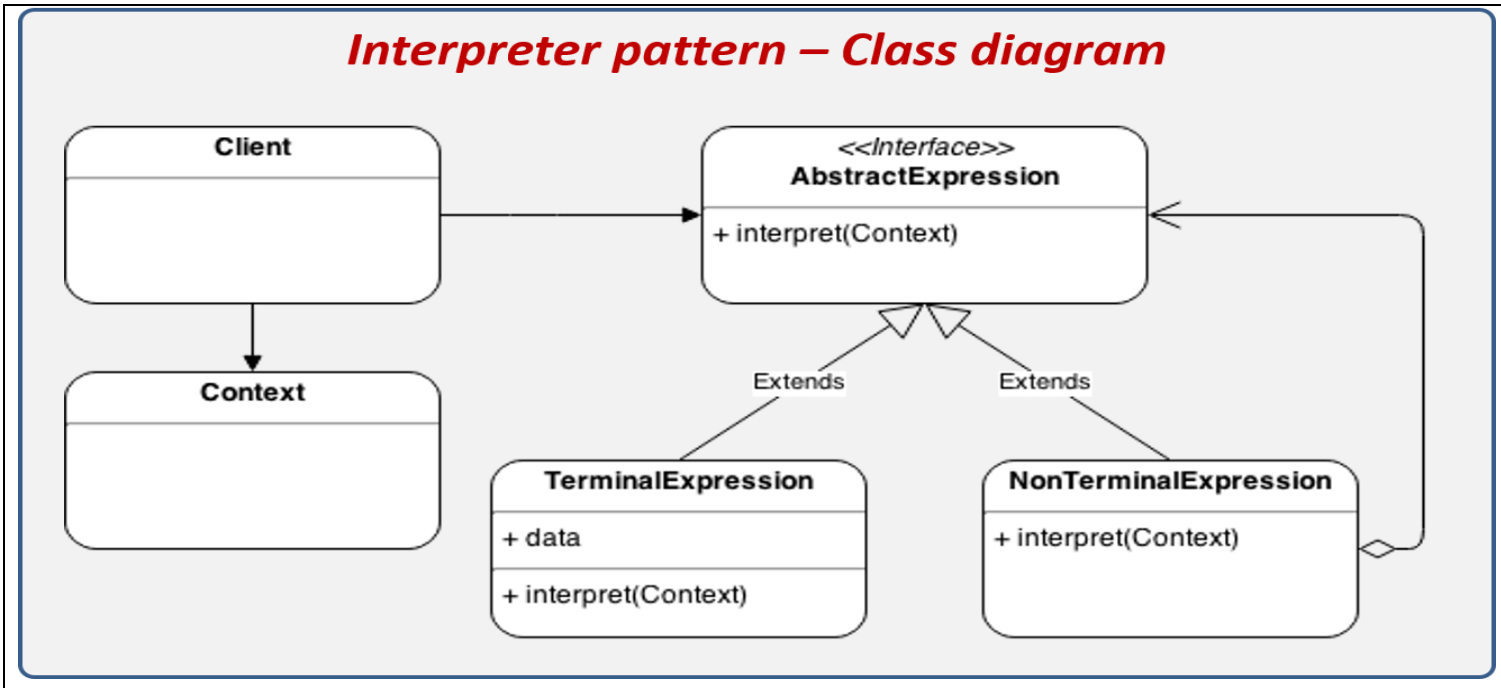
Motivación

Existen problemas particulares que pueden expresarse en función de algún lenguaje. Es necesario construir un intérprete de dicho lenguaje. El patrón define reglas gramaticales del lenguaje, así como la realización y comprensión de sentencias del mismo.

Aplicabilidad

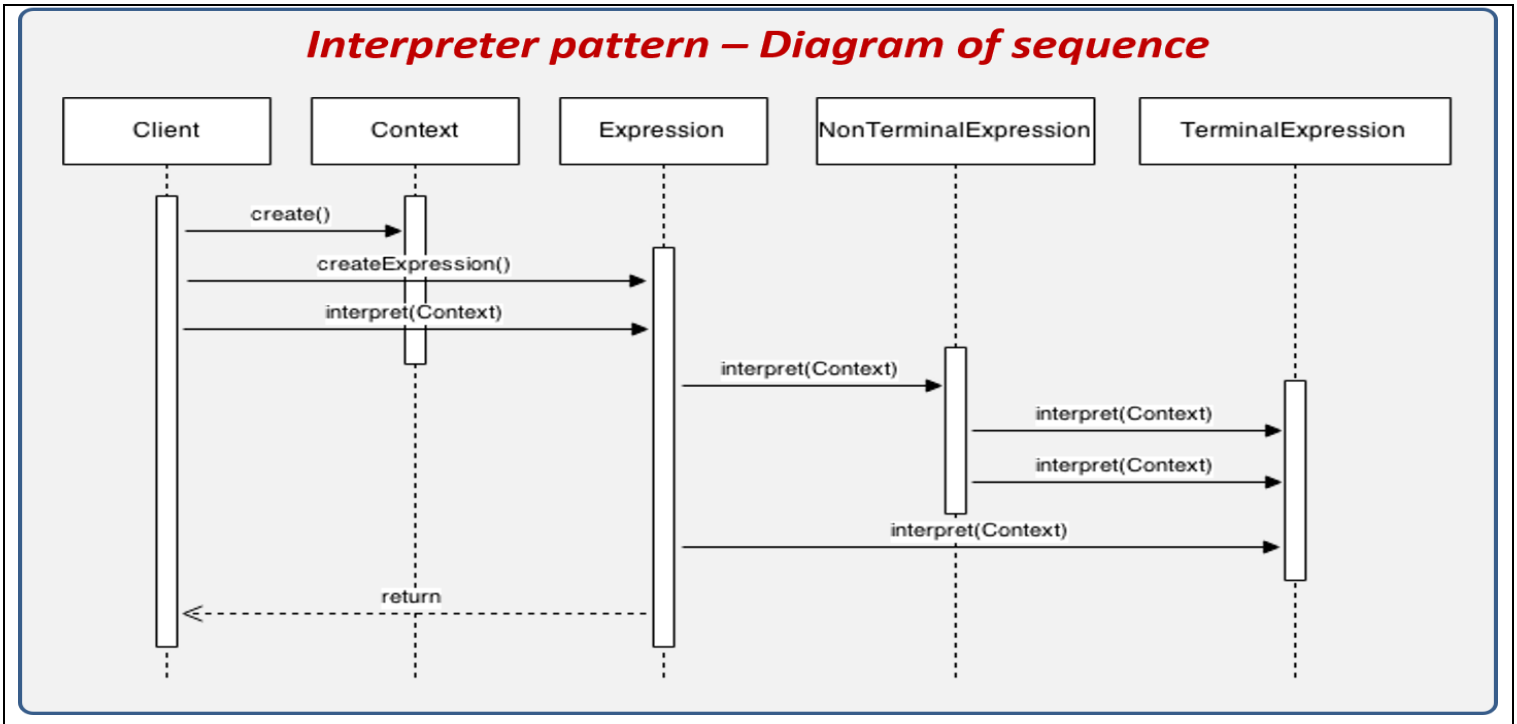
- ✓ Cuando tratamos con gramáticas simples, en caso contrario la mejor opción es utilizar parsers.
- ✓ La eficiencia no es uno de los aspectos más importantes. Hay que traducir el input a una forma inmediata.

Estructura



Los componentes del patrón Interpreter se explican a continuación:

Client: Actor que dispara la ejecución del interpreter.
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, este es enviado al interpreter el cual lo replica por toda la estructura.
AbstractExpression: Interface que define la estructura mínima de una expresión.
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.
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.



1.El cliente crea el contexto para la ejecución del interpreter.
2.El cliente crea u obtiene la expresión a evaluar.
3.El cliente solicita la interpretación de la expresión al interpreter y le envía el contexto.
4.La Expresión manda llamar a las Expresiones No Terminales que contiene.
5.La Expresión No Terminal manda llamar a todas las Expresiones Terminales.
6.La Expresión Raíz solicita la interpretación de una Expresión Terminal.
7.La expresión se evalúa por completo y se tiene un resultado de la interpretación de todas las expresiones terminales y no terminales.

Pros

- ✓ 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).

Contras

- ✓ No es muy eficiente.
- ✓ No cubre gramáticas complejas.

Implementación

Para poder implementar el patrón de manera correcta se debe:

- ✓ Al momento de crear el árbol sintáctico de nuestro lenguaje no es necesario especificarlo, ya que se puede realizar a través de Parser o crearlo directamente en cliente.
- ✓ No es necesario implementar la operación de interprete en cada clase de las expresiones terminales y no terminales, por ese motivo es recomendable usar el patrón visitante.

- ✓ Para que el patrón no sea demasiado pesado es recomendable usarlo con el patrón peso ligero o Flyweight.

Cuando utilizarlo

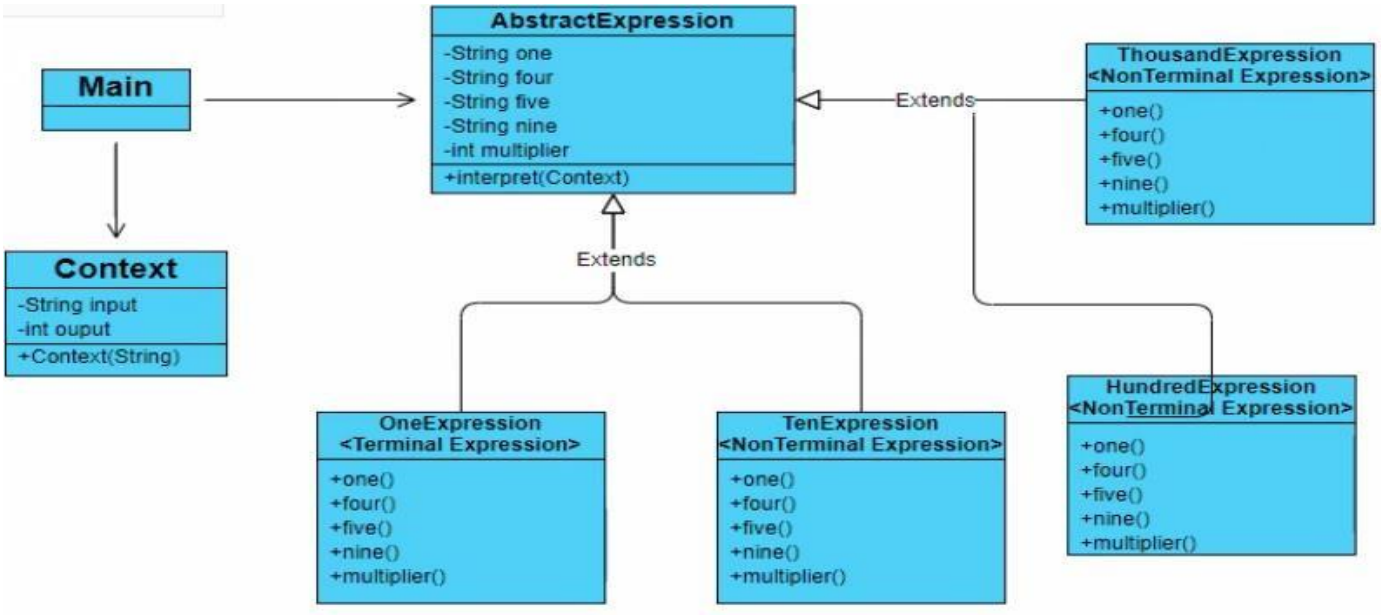
- ✓ Cuando observemos que la gramática de nuestro lenguaje es simple.
- ✓ Cuando se pueda analizar el lenguaje y su gramática para poder crear arboles de sintaxis abstractos.
- ✓ Cuando la eficiencia no tenga mucha relevancia, ya que puede ser difícil de interpretar lenguajes más complicado.

Patrones relacionados

- ✓ Composite o Compuesto: El uso de un árbol sintáctico hace referencia a una jerarquía una estructura fundamental para este patrón.
- ✓ Flyweight o Peso Ligero: Comparte las partes comunes del estado entre varios objetos en lugar de mantener toda la información en cada objeto.
- ✓ Visitor o Visitante: Se usa para mantener un comportamiento compartido entre los diferentes nodos o hojas del árbol de sintaxis dentro de una sola clase.

Ejemplo

Se pensó en un aplicativo que reciba números romanos en forma de texto para interpretarlos y devolver números en escala decimal, teniendo en cuenta la estructura de los números romanos, es decir, las unidades de mil, centenas, decenas y unidades respectivamente. Primero se crea el contexto, esto es una clase donde tiene un input (un String ya que los números romanos son letras) y un output o salida (este viene siendo un int ya que la respuesta es un numero). Después de esto es puro algoritmo, pensando en cómo funcionan los números romanos, encontraremos un patrón: de 1 a 3 ocupa un espacio por vez, el 4 ocupa la combinación del 5 y el 1, es decir, dos lugares (IV), y el próximo signo que hace combinación es el 9 (IX)que utiliza un signo de una escala mayor



III. CONCLUSIONES DEL TRABAJO.

El patrón Interpreter o Intérprete es un patrón muy interesante que nos ayuda a la traducción de formas gramaticales simples usando arboles sintácticos, lo que nos ayudaría al desarrollo de la lógica de programación y para algunos ejercicios puntuales que tengamos que realizar, pero dentro de un contexto real no es un patrón que se use, ya que al no ser muy eficiente es normal recurrir a otro tipo de herramientas para poder traducir patrones más complejos, ya que este patrón no cuenta con mucha escalabilidad.

IV. BIBLIOGRAFÍA:

[1] R. Joshi, «Interpreter Design Pattern Example - Java Code Geeks - 2022», *Java Code Geeks*, 30 de septiembre de 2015. https://www.javacodegeeks.com/2015/09/interpreter-design-pattern.html (accedido 27 de enero de 2022).

[2] «Design Patterns and Refactoring». https://sourcemaking.com (accedido 28 de enero de 2022).

[3] «Interpreter». https://reactiveprogramming.io/blog/en/design-patterns/interpreter (accedido 27 de enero de 2022).

[4] S. L. Programacion en Castellano, «Patrones de Diseño (XVI): Patrones de Comportamiento - Interpreter», *Programación en Castellano.* http://programacion.net/articulo/patrones_de_diseno_xvi_patrones_de_comportamiento_interprete r_1019 (accedido 28 de enero de 2022).