Patrones de comportamiento

Interpreter

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"
[Christopher Alexander].

Por: Brayan Estiven Aguirre Aristizábal – 20231020156
Santiago Zamudio Diaz – 20231020128
Daniel Josué Rozo Rodríguez – 20222020006

1. Introducción

Los patrones de comportamiento en el desarrollo de software tienen como propósito principal proporcionar soluciones estándar y reutilizables para problemas específicos relacionados con la interacción y la distribución de responsabilidades entre objetos. Se centran en mejorar la comunicación y la colaboración entre objetos de un sistema, asegurando que el software sea más flexible, mantenible y adaptable a cambios. Además, promueven la separación de responsabilidades claras entre componentes del sistema, mejoran la escalabilidad y facilitan la implementación de nuevas funcionalidades sin afectar otras partes del sistema. Están diseñados para mejorar la estructura interna del software y cómo sus componentes interactúan entre sí.

El Intérprete es un patrón de diseño que, dado un lenguaje, define una representación para su gramática junto con un intérprete del lenguaje. Se usa para definir un lenguaje para construir expresiones regulares que representen cadenas a buscar dentro de otras cadenas. Además, en general, para definir un lenguaje que permita representar las distintas instancias de una familia de problemas.

2. Interpreter

Nombre del patrón:

Interpreter

Clasificación del patrón:

Comportamiento

Intención:

Definir una gramática para un lenguaje y proporcionar un medio para evaluar sentencias en ese lenguaje. Permite interpretar expresiones complejas o estructuras de datos definidas mediante una gramática formal, convirtiéndolas en acciones concretas

dentro del sistema.

También conocido por:

Intérprete

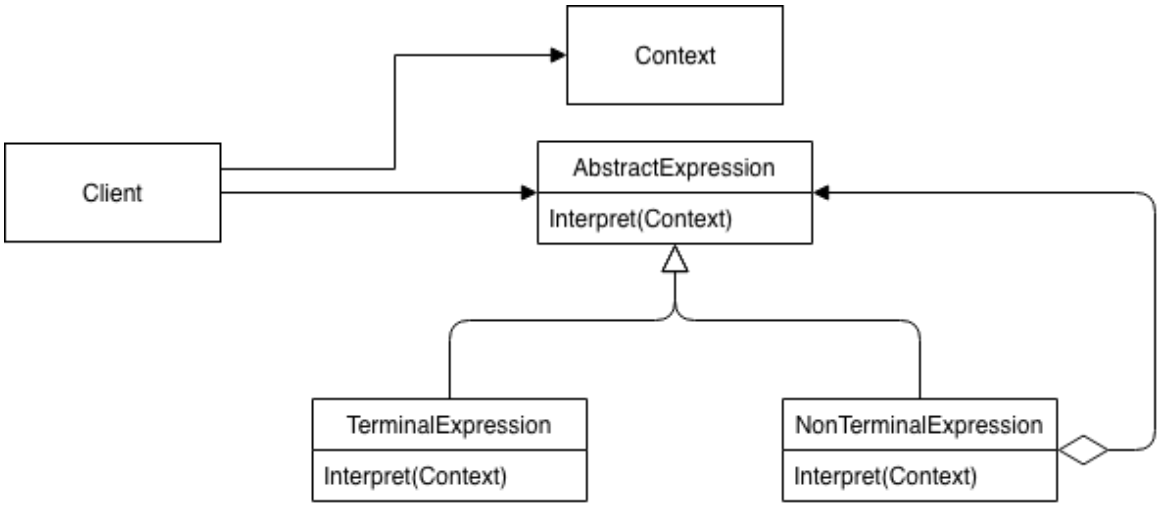
Motivación:

La motivación para desarrollar un intérprete de patrón de comportamiento radica en la necesidad de abordar problemas específicos expresados en un lenguaje determinado. Este intérprete se encarga de analizar y ejecutar reglas gramaticales definidas por el patrón, permitiendo la interpretación precisa y la comprensión de sentencias dentro de dicho lenguaje. Además, el intérprete facilita la implementación de soluciones efectivas para problemas que requieren un entendimiento profundo y estructurado del comportamiento según las normas establecidas por el patrón de comportamiento.

Aplicabilidad:

- **Interpretación de Lenguajes Específicos:** Para analizar y ejecutar comandos o consultas en lenguajes diseñados para tareas específicas.
- **Procesamiento de Formatos de Datos:** Para leer, validar y manipular datos estructurados como JSON, XML, CSV, etc.
- **Personalización de Comportamientos:** Permite ajustar comportamientos de aplicaciones mediante configuraciones externas.
- **Desarrollo de DSLs:** Facilita la creación de lenguajes específicos para dominios particulares.
- **Automatización de Tareas:** Automatiza procesos complejos basados en reglas y datos de entrada.
- **Validación de Reglas de Negocio:** Verifica y aplica reglas de negocio definidas para decisiones coherentes.
- **Integración de Sistemas:** Facilita la interoperabilidad entre sistemas al interpretar y transformar datos según necesidades específicas.

Estructura:



Participantes:

- **ExpresiónAbstracta:** Declara una operación abstracta Intérprete que es común a todos los nodos del árbol de sintaxis abstracta.
- **ExpresiónTerminal:** Una instancia es requerida por cada aparición en una sentencia. implementa una operación Intérprete asociada a cada símbolo terminal.
- **ExpresiónNoTerminal:** Para cada regla es necesario un tipo de clase.
- **Cliente:** Construye el árbol sintáctico de ExpresionesNoTerminales, e instancias de la clase ExpresiónTerminal.
- **Contexto:** Contiene información global para el interpretador.

Colaboraciones:

- Cliente construye una sentencia compuesta de ExpresionesTerminales y ExpresionesNoTerminales. Luego incializa el Contexto e invoca al interpretador
- Cada nodo correspondiente a ExpresionesNoTerminales define al interpretador en función de subexpresiones
- Las operaciones en cada nodo utilizan el Contexto para almacenar y acceder al estado del interpretador

Consecuencias:

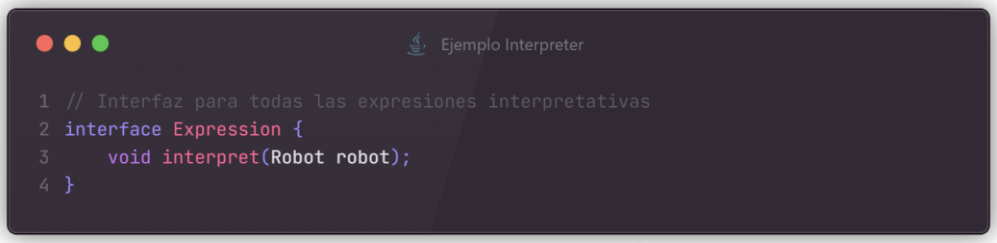
- Es fácil cambiar y extender la gramática.
- Implementar la gramática es sencillo.
- Las gramáticas complejas son difíciles de mantener.
- Se puede añadir nuevas interpretaciones de los símbolos.

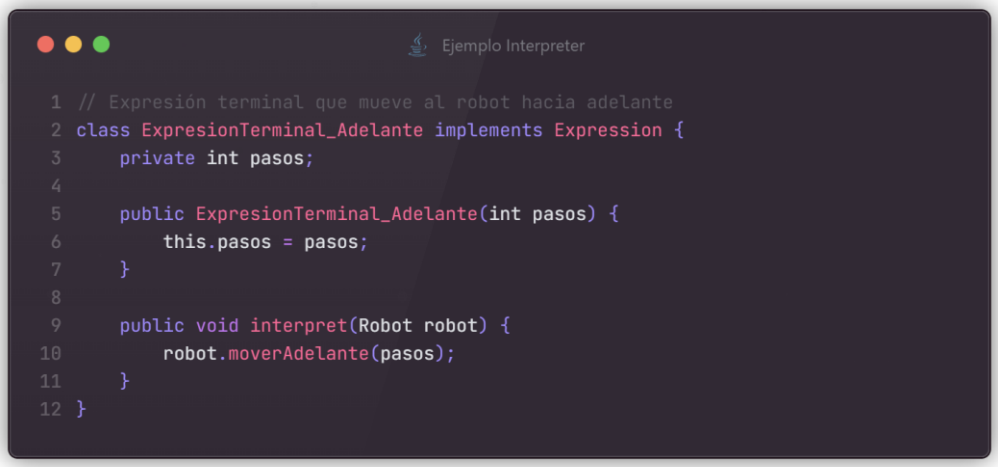
Implementación:

Para realizar una correcta implementación del patrón es recomendable:

- La creación del árbol sintáctico no se especifica. Se puede usar un parser o realizarlo en cliente
- Definiendo la operación de interprete. No hace falta implementarla en cada clase de expresiones. Es recomendable el uso del patrón Visitante.
- Compartiendo símbolos terminales mediante el patrón Peso Ligero.

Código ejemplo:





Ejemplo Interpreter

```java
// Expresión terminal que gira al robot hacia la izquierda
class ExpresionTerminal_Izquierda implements Expression {
    public void interpret(Robot robot) {
        robot.girarIzquierda();
    }
}
```

Ejemplo Interpreter

```java
// Expresión terminal que gira al robot hacia la derecha
class ExpresionTerminal_Derecha implements Expression {
    public void interpret(Robot robot) {
        robot.girarDerecha();
    }
}
```

Ejemplo Interpreter

```java
import java.util.*;

// Clase Parser que construye el árbol de expresiones a partir de una cadena
class Parser {
    private List<Expression> parseTree = new ArrayList<>();

    public Parser(String s) {
        for (String token : s.split(" ")) {
            if (token.equals("adelante")) {
                parseTree.add(new ExpresionTerminal_Adelante(1)); // Por defecto, mueve 1 paso
            } else if (token.equals("izquierda")) {
                parseTree.add(new ExpresionTerminal_Izquierda());
            } else if (token.equals("derecha")) {
                parseTree.add(new ExpresionTerminal_Derecha());
            } else if (token.startsWith("adelante")) {
                int pasos = Integer.parseInt(token.substring(8)); // 'adelanteX' donde X son los pasos
                parseTree.add(new ExpresionTerminal_Adelante(pasos));
            }
        }
    }

    public void ejecutar(Robot robot) {
        for (Expression e : parseTree) {
            e.interpret(robot);
        }
    }
}
```

Ejemplo Interpreter

```java
// Clase Robot que simula las acciones del robot virtual
class Robot {
    private int x, y;
    private String direccion;

    public Robot(int x, int y, String direccion) {
        this.x = x;
        this.y = y;
        this.direccion = direccion;
    }

    public void moverAdelante(int pasos) {
        if (direccion.equals("norte")) {
            y += pasos;
        } else if (direccion.equals("sur")) {
            y -= pasos;
        } else if (direccion.equals("este")) {
            x += pasos;
        } else if (direccion.equals("oeste")) {
            x -= pasos;
        }
    }

    public void girarIzquierda() {
        if (direccion.equals("norte")) {
            direccion = "oeste";
        } else if (direccion.equals("sur")) {
            direccion = "este";
        } else if (direccion.equals("este")) {
            direccion = "norte";
        } else if (direccion.equals("oeste")) {
            direccion = "sur";
        }
    }

    public void girarDerecha() {
        if (direccion.equals("norte")) {
            direccion = "este";
        } else if (direccion.equals("sur")) {
            direccion = "oeste";
        } else if (direccion.equals("este")) {
            direccion = "sur";
        } else if (direccion.equals("oeste")) {
            direccion = "norte";
        }
    }

    public String obtenerPosicion() {
        return "(" + x + ", " + y + ") dirección " + direccion;
    }
}
```

Usos Conocidos:

- **Intérpretes de lenguajes**: Los intérpretes de lenguajes como Python, Ruby, PHP, etc. utilizan el patrón Interpreter para analizar y ejecutar el código fuente.
- **Compiladores:** Los compiladores también utilizan el patrón Interpreter para analizar la sintaxis del lenguaje y generar código objeto.
- **SQL:** Los motores de bases de datos relacionales como MySQL, PostgreSQL, Oracle, etc. utilizan el patrón Interpreter para analizar y ejecutar consultas SQL.
- **NoSQL**: Algunas bases de datos NoSQL como MongoDB, Cassandra, etc. también utilizan el patrón Interpreter para analizar y ejecutar consultas.

Relaciones con otros patrones:

- **Composite**: El árbol sintáctico es un Composite.
- **Peso Ligero**: Permite compartir símbolos terminales del árbol sintáctico.
- **Visitor**: Puede usarse para mantener el comportamiento de cada nodo del árbol de sintaxis abstracta en una sola clase.

Referencias Bibliográficas

[1] Junta de Andalucía. (s.f.). Intérprete. Madeja. https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/192

[2] Reactive Programming. (s.f.). Interpreter. Patrones de diseño. https://reactiveprogramming.io/blog/es/patrones-de-diseno/interpreter