POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

16. *Interpreter*

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema
2. Solución
3. Consecuencias
4. Relación

1. Problema

- *Demandar a un anciano que sepa manejar un aparato moderno (p.e. móvil, televisión digital, …) puede ser una tarea muy compleja por su propia naturaleza y contexto.*
- *Una solución que alivia toda la complejidad es que cuando el anciano quiere manipular algún aparato moderno, llama a su nieto para que "interprete" sus deseos: el anciano dicta (p.e. quiero hablar con Pepe, quiero ver el partido, …) y el nieto opera (p.e. marca el teléfono, maneja los 3 mandos a distancia, …).*

- ***Dado un lenguaje, definir una representación de su gramática junto con un intérprete que utiliza la representación para interpretar sentencias del lenguaje***

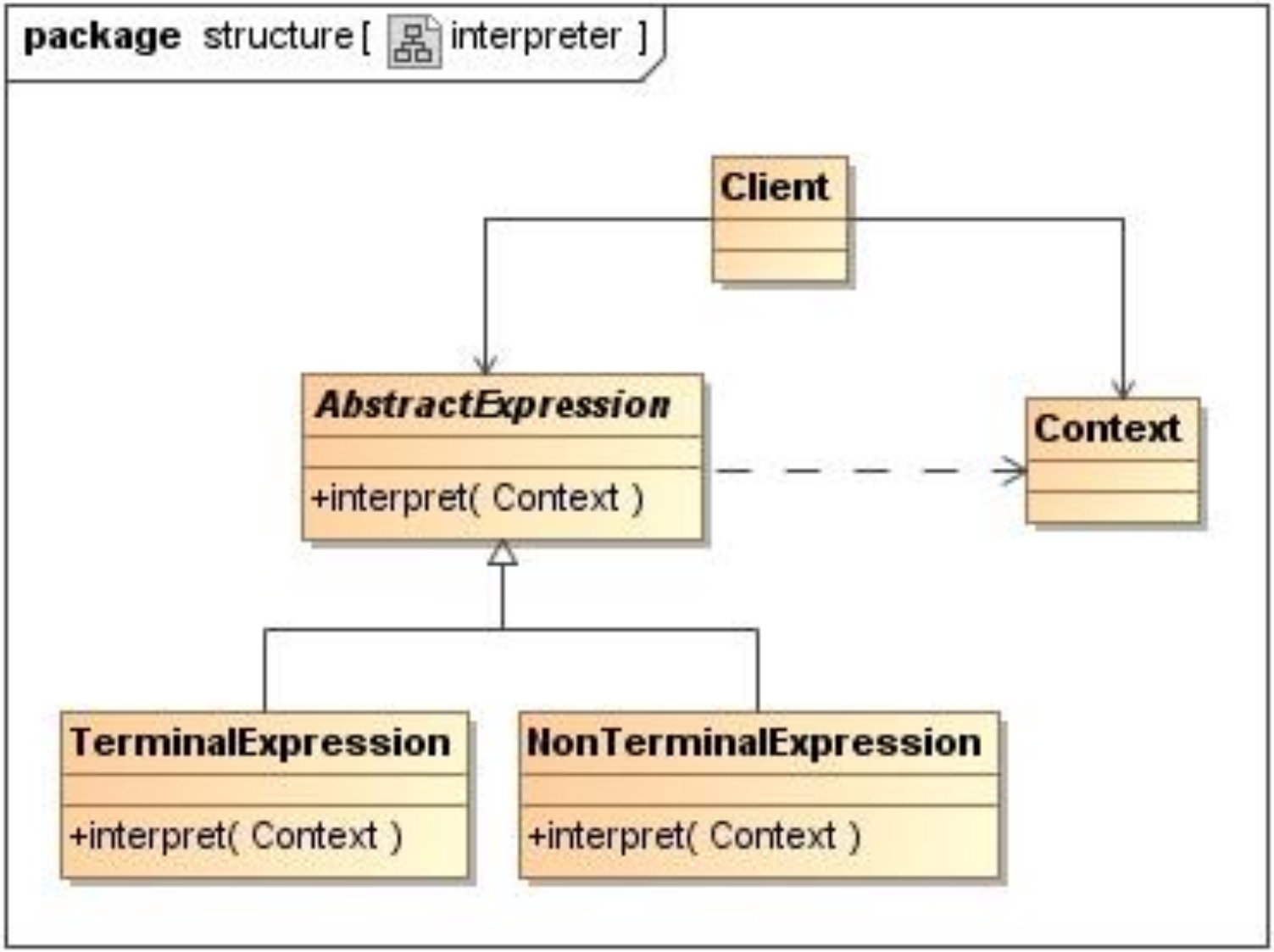
1. Problema

- Si hay un tipo de problemas que ocurren con cierta frecuencia, puede valer la pena expresar las apariciones de ese problema como instrucciones del un lenguaje simple. A continuación puede construirse un intérprete que resuelva el problema interpretando dichas instrucciones.
- Úsese el patrón cuando hay un lenguaje que interpretar y se pueden representar las sentencias del lenguaje como árboles sintácticos o abstractos.

1. Problema

- El patrón *Interpreter* funciona mejor cuando:
  - La gramática es simple. Para gramáticas complejas, la jerarquía de clases de la gramática se vuelve grande e inmanejable. Herramientas como los generadores de analizadores sintácticos se constituyen una alternativa mejor en estos casos. Éstas pueden interpretar expresiones sin necesidad de construir árboles sintácticos abstractos, lo que puede ahorrar espacio y, posiblemente, tiempo.
  - La eficiencia no es una preocupación crítica. Los intérpretes más eficientes normalmente no se implementan interpretando árboles de análisis sintácticos directamente, sino que primero los traducen a algún formato. Por ejemplo, las expresiones regulares suelen transformarse en máquinas de estados. Pero incluso en ese caso, el traductor puede implementarse con el patrón *Interpreter*, de modo que éste sigue siendo aplicable.

2. Solución


package structure [interpreter]
Client
AbstractExpression
+interpret(Context)
Context
TerminalExpression
+interpret(Context)
NonTerminalExpression
+interpret(Context)

2. Solución

- No explica cómo crear un árbol sintáctico abstracto. En otras palabras, no se encarga del análisis sintáctico. El árbol sintáctico abstracto puede ser creado mediante una analizador sintáctico dirigido por una tabla, mediante una analizador sintáctico (normalmente recursivo descendiente) hecho a mano, o directamente por el cliente.
- No tenemos por qué definir la operación Interpretar en las clases de la expresión. Si se van a crear nuevos intérpretes es mejor usar el patrón Visitor para poner Interpretar en un objeto visitante aparte.

2. Solución

- Para la gramáticas cuyas sentencias contienen muchas repeticiones de un mismo símbolo terminal puede ser beneficioso compartir una única copia de dicho símbolo. Las gramáticas de programas de computadora son buenos ejemplos.
  - Los nodos terminales generalmente no guardan información sobre su posición en el árbol sintáctico abstracto.
  - Los nodos padre les pasan cualquier información sobre su posición en el árbol sintáctico abstracto.
  - Por tanto, aquí se da una distinción entre estado compartido (intrínseco) y estado recibido (extrínseco), por lo que es aplicable el patrón *Flyweight*

3. Consecuencias

- Puesto que el patrón usa clases para representar las reglas de la gramática, se puede usar la herencia para cambiar o extender ésta. Se puede modificar incrementalmente las expresiones existentes, y se pueden definir nuevas como variaciones de las antiguas
- Las clases que definen los nodos del árbol sintáctico abstracto tiene implementaciones similares. Estas clases son fáciles de escribir, y muchas veces se pueden generar automáticamente con un compilador o un generador de analizadores sintácticos

3. Consecuencias

- El patrón *Interpreter* define al menos una clase para cada regla de la gramática (las reglas que se hayan definido usando BNF pueden necesitar varias clases). De ahí que las gramáticas que contienen muchas reglas pueden ser difíciles de controlar y mantener. Se pueden aplicar otros patrones de diseño para mitigar el problema. Pero cuando la gramática es muy compleja son más adecuadas otras técnicas como los generadores de analizadores sintáticos o de compiladores.

3. Consecuencias

- El patrón *Interpreter* facilita evaluar una expresión de una manera distinta. Por ejemplo, podríamos permitir imprimir con formato una expresión o realizar una comprobación de tipo en ella definiendo una nueva operación en las clases de expresiones. Si vamos a seguir añadiendo nuevos modos de interpretar una expresión, deberíamos considerar la utilización del patrón *Visitor*

3. Consecuencias

- No explica cómo crear un árbol sintáctico abstracto. En otras palabras, no se encarga del análisis sintáctico. El árbol sintáctico abstracto puede ser creado mediante una analizador sintáctico dirigido por una tabla, mediante una analizador sintáctico (normalmente recursivo descendiente) hecho a mano, o directamente por el cliente.
- No tenemos por qué definir la operación Interpretar en las clases de la expresión. Si se van a crear nuevos intérpretes es mejor usar el patrón Visitor para poner Interpretar en un objeto visitante aparte.

4. Relaciones

- ***Iterator*** para recorrer el árbol sintáctico abstracto
- ***Composite*** para la creación del árbol sintáctico abstracto
- ***Flyweight*** para compartir símbolos terminales dentro del árbol sintáctico abstracto
- ***Visitor*** para mantener el comportamiento de cada nodo del árbol sintáctico abstracto en una clase.
- ***State*** para definir contextos para el análisis.