Interpretador

En general, los lenguajes se componen de un conjunto de reglas gramaticales. Se pueden construir diferentes oraciones siguiendo estas reglas gramaticales. A veces, una aplicación puede necesitar procesar ocurrencias repetidas de solicitudes similares que son una combinación de un conjunto de reglas gramaticales. Estas solicitudes son distintas pero similares en el sentido de que todas se componen utilizando el mismo conjunto de reglas.

Un ejemplo simple de este tipo sería el conjunto de diferentes expresiones aritméticas enviadas a un programa de calculadora. Aunque cada una de estas expresiones es diferente, todas se construyen utilizando las reglas básicas que componen la gramática del lenguaje de las expresiones aritméticas.

* En lugar de tratar cada combinación distinta de reglas como un caso separado, puede ser beneficioso que la aplicación tenga la capacidad de interpretar una combinación genérica de reglas.

El patrón de intérprete se puede utilizar para diseñar esta capacidad en una aplicación para que otras aplicaciones y usuarios puedan especificar operaciones utilizando un lenguaje simple definido por un conjunto de reglas gramaticales.

* Se puede diseñar una jerarquía de clases para representar el conjunto de reglas gramaticales con cada clase en la jerarquía representando una regla gramatical separada.
* Se puede diseñar un módulo “**interpreter**” para interpretar las oraciones construidas utilizando la jerarquía de clases diseñada anteriormente y realizar las operaciones necesarias.

Debido a que una clase diferente representa cada regla gramatical, el número de clases aumenta con el número de reglas gramaticales. Un idioma con reglas gramaticales extensas y complejas requiere una gran cantidad de clases. El patrón de intérprete funciona mejor cuando la gramática es simple. Tener una gramática simple evita la necesidad de tener muchas clases correspondientes al complejo conjunto de reglas involucradas, que son difíciles de administrar y mantener.

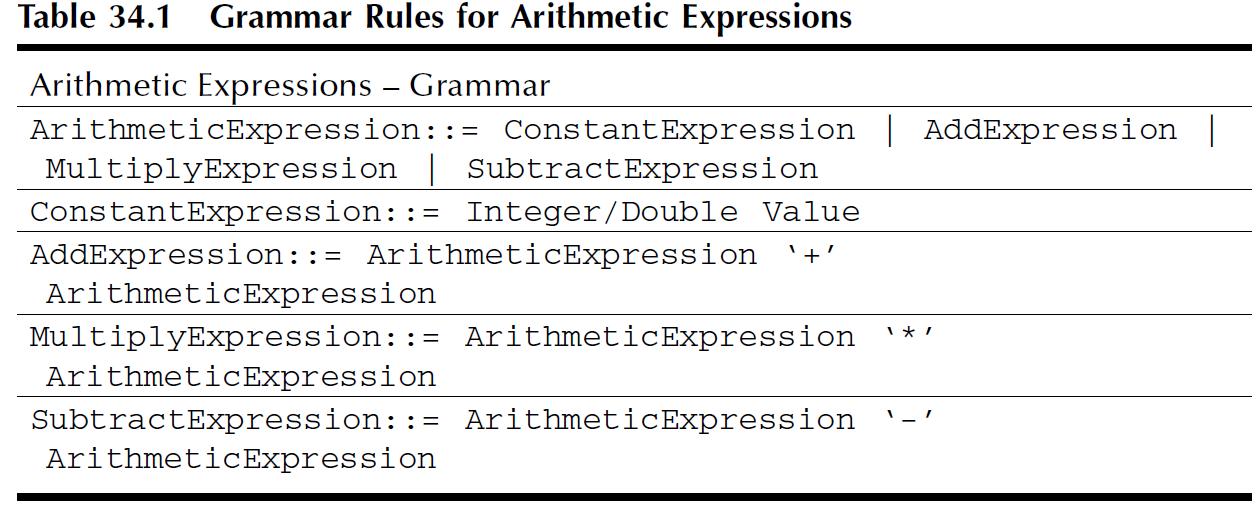
## Ejemplo:

Construyamos una aplicación de calculadora que evalúe una expresión aritmética dada. Para simplificar, consideremos solo operaciones de sumar, multiplicar y restar. El patrón intérprete se puede utilizar para diseñar la capacidad de comprender una expresión aritmética genérica y evaluarla.

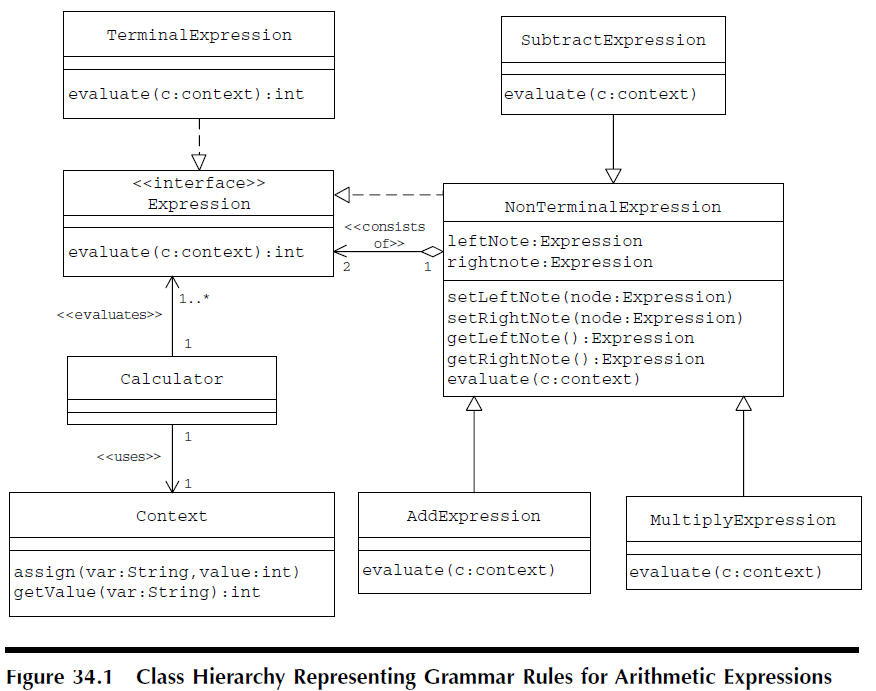
**El patrón de intérprete se puede aplicar en dos etapas:**

1. Definir una representación para el conjunto de reglas que componen la gramática de las expresiones aritméticas.

2. Diseñar un intérprete que haga uso de las clases que representan las diferentes reglas aritméticas para comprender y evaluar una expresión aritmética dada.



En la tabla 34.1, se puede observar que las expresiones aritméticas son de dos tipos: individual (por ejemplo, ConstantExpression) o compuestas (por ejemplo, AddExpression) Estas expresiones se pueden organizar en forma de estructura de árbol, con expresiones compuestas como nodos no terminales y Expresiones como nodos terminales del árbol.



Cada una de las clases que representan diferentes reglas implementa la interfaz Expression y proporciona implementación para el método de **evaluate(c:Context)**.

* **Context**: es un repositorio de información común que almacena los valores de diferentes variables. Si bien cada una de las clases NonTerminalExpression realiza la operación aritmética que representa, la clase TerminalExpression simplemente busca el valor de la variable que representa en el contexto.

El diseño de la aplicación puede evaluar cualquier expresión. Pero para simplificar, el objeto principal **Calculator** usa una expresión aritmética codificada de forma rígida **(a + b) \* (c - d)** como la expresión a interpretar y evaluar.

1. Conversión infijo a sufijo (*Infix-to-postfix)*: la expresión de infijo de entrada se traduce primero en una expresión de sufijo equivalente.

2. Construcción de la estructura de árbol: la expresión de sufijo luego se escanea para construir una estructura de árbol.

3. *Postorder traversal of the tree*: el árbol se recorre de manera transversa para evaluar la expresión.

## Infix-to-Postfix Conversion

Una expresión en la forma estándar es una expresión infija.

* Ejemplo: (a + b) \* (c - d)

Una expresión infija es más fácil de entender para los humanos, pero no es adecuada para evaluar expresiones por computadora.

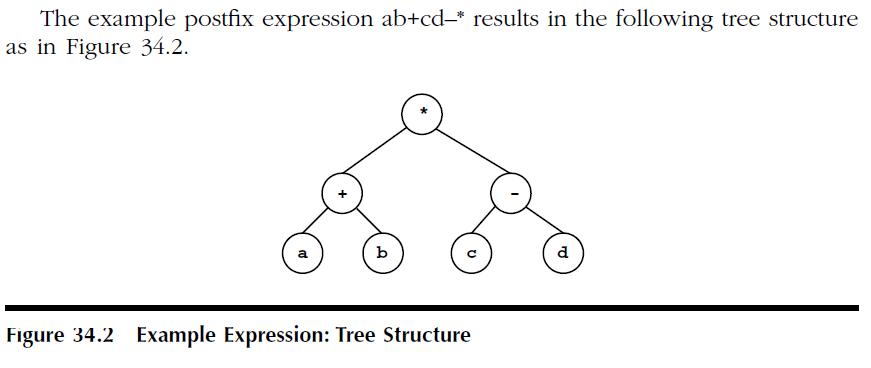
El uso de reglas de precedencia y paréntesis en el caso de expresiones complejas dificulta la evaluación por computadora de estas expresiones. Una expresión de sufijo no contiene paréntesis, no incluye reglas de precedencia y es más adecuada para la evaluación por computadora. El equivalente de sufijo de la expresión de ejemplo anterior es

* ab + cd - \*.

## Construcción de la estructura del árbol

El equivalente de sufijo de la expresión infijo de entrada se escanea de izquierda a derecha y se construye una estructura de árbol utilizando el siguiente algoritmo:

1. Inicialice una pila vacía.
2. Escanee la cadena de sufijo de izquierda a derecha.
3. Si el carácter escaneado es un operando:
   1. Cree una instancia de la clase TerminalExpression pasando el carácter escaneado como argumento.
   2. Envié el objeto TerminalExpression a la pila.
4. Si el carácter escaneado es un operador:
   1. Saque dos elementos superiores de la pila.
   2. Cree una instancia de una subclase apropiada de NonTerminalExpression pasando los dos elementos de la pila recuperados anteriormente como argumentos.
5. Repita el Paso 3 y el Paso 4 para todos los caracteres de la cadena de sufijo.
6. El único elemento que queda en la pila es la raíz de la estructura del árbol.



## Postorder Traversal of the Tree

La calculadora recorre la estructura de árbol y evalúa diferentes objetos de expresión en su ruta de recorrido después del ordenamiento. Hay cuatro técnicas principales de recorrido de árbol. Debido a que el árbol binario en el ejemplo actual es una representación de una expresión posfija, se sigue la técnica de recorrido “postorder” para evaluar la expresión. El objeto Calculadora hace uso de un objeto Context auxiliar para compartir información con diferentes objetos Expression que constituyen la estructura de árbol.

* En general, un objeto Contexto se utiliza como un depósito global de información.

En el ejemplo actual, el objeto Calculator almacena los valores de diferentes variables en El contexto, que son utilizados por cada uno de los diferentes objetos Expression para evaluar la parte de la expresión que representa. El recorrido posterior a la estructura de árbol en la figura 34.2 da como resultado la evaluación del subárbol más a la izquierda de una manera recursiva, seguida por el subárbol más a la derecha, luego el Nodo NonTerminalExpression que representa un operador.