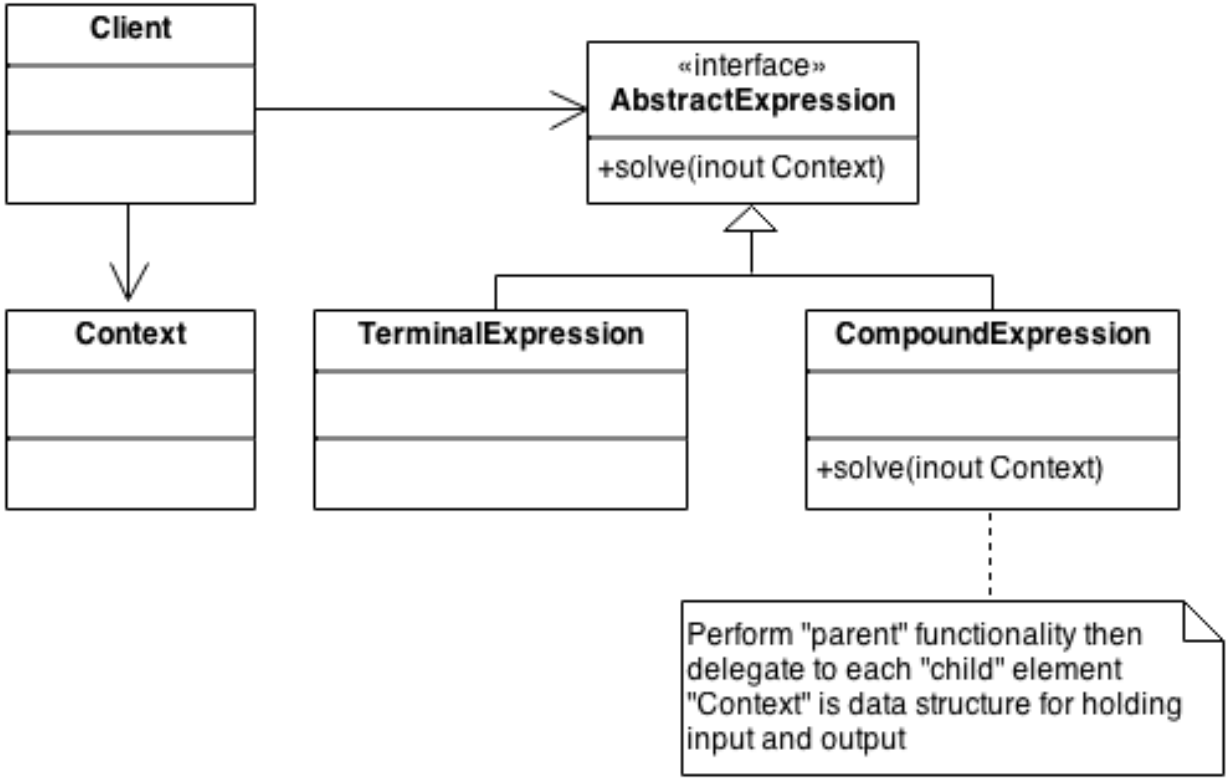
Interpreter

Interpreter is a **behavioral** design pattern that defines a grammatical representation for a language and provides an interpreter that uses the representation to interpret sentences in the language.

Problem

- A class of problems occurs repeatedly in a well-defined and well-understood domain. If the domain were characterized with a "language", then problems could be easily solved with an interpretation "engine".

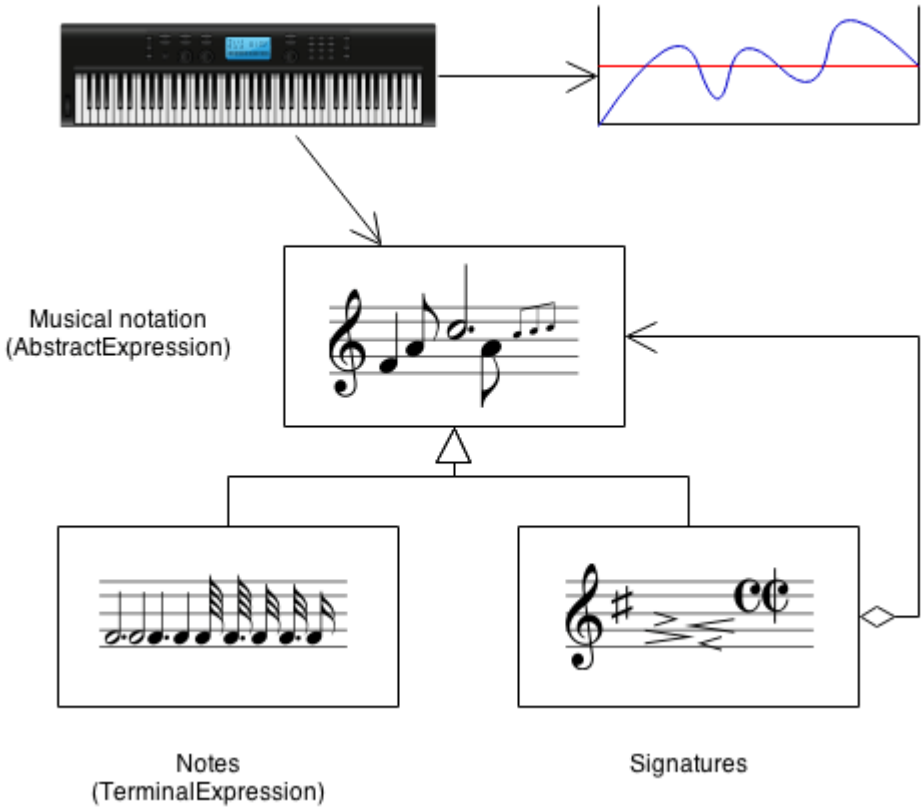
Structure


Client
«interface»
AbstractExpression
+solve(inout Context)
Context
TerminalExpression
CompoundExpression
+solve(inout Context)
Perform "parent" functionality then delegate to each "child" element
"Context" is data structure for holding input and output

Contd.

- **Abstract Expression** (Expression): Declares an interpret() operation that all nodes (terminal and nonterminal) in the Abstract Syntax Tree (AST) overrides.
- **Terminal Expression** (Number Expression): Implements the interpret() operation for terminal expressions.
- **Nonterminal Expression** (Addition Expression, Subtraction Expression, and Multiplication Expression): Implements the interpret() operation for all nonterminal expressions.
- **Context** (String): Contains information that is global to the interpreter. It is this String expression with the Postfix notation that has to be interpreted and parsed.
- **Client** (Expression Parser): Builds (or is provided) the AST assembled from Terminal Expression and Non Terminal Expression. The Client invokes the interpret() operation.

Implementation



Application

- This pattern is used in SQL parsing, symbol processing engine etc.

Pros & Cons

- Using this pattern, it’s easy to change and extend the grammar
- Using this pattern, implementing the grammar is easy
- In this approach, complex grammars are hard to maintain.
- The pattern doesn't address parsing. When the grammar is very complex, other techniques (such as a parser) are more appropriate.