Interpreter:

The same thing can be different based on its context,

**Other name (if any)**

placeholder

**What it does**

It is behavioral design pattern. The Interpreter design pattern is a behavioral design pattern that facilitates the interpretation and evaluation of expressions or language grammars. It has hierarchy kind of structure. Terminal expressions represent basic building blocks, while nonterminal expressions represent compositions of these building blocks. The tree structure of the Interpreter design pattern is somewhat similar to that defined by the composite design pattern with terminal expressions being leaf objects and non-terminal expressions being composites.

**Where to use**

1. Interpreter pattern provides a way to evaluate language grammar or expression.  
2. This pattern is used in SQL parsing, symbol processing engine etc.  
3. It is used in in simple rule engines.

**Steps**

1. Concrete Context Class: Create a concrete context class. It will have a hashmap. Which will contain the value of each literals. To set value a function setValue will be used and to get the value function getValue will be used.  
2. Interpreter Interface: Create an interpreter interface with at least one function which is interpret. The function will take context as parameter.  
3. Concrete Interpreter Class: Create concrete interpreter class for terminal and nonterminal both kind of nodes. These will implement the interpret function of the interpreters interface. In the terminal node, it would evaluate to their respective values directly without further decomposition. A single variable will be passed to the constructor of terminal node. In the non-terminal, the whole expression will be decomposed. Non-terminal expressions facilitate the traversal of expression trees during the interpretation process. As part of this traversal, they recursively interpret their sub-expressions, ensuring that each part of the expression contributes to the overall interpretation. Multiple object of interpret interface will be passed to the constructor of the non-terminal node.  
4. Client Code: Create a context object using concrete context class. Set the value of the context object using the setValue function of the code concrete context class. Create object of concrete interpreter class using different combination of concrete interpreter class. Pass the context to the interpret function of the object of the concrete interpreter class.

**Special cases (if any)**

placeholder

**Advantages**

1. New operations or language constructs can be added without modifying existing code, promoting code reuse and maintainability.

**Disadvantages**

1. Interpreting expressions through the Interpreter pattern might introduce overhead compared to other approaches, especially for complex expressions or large input sets. In performance-critical applications, a more optimized solution, such as compilation to native code, may be preferable.  
2. If your grammar is highly complex, with numerous rules and exceptions, implementing it using the Interpreter pattern may lead to a proliferation of expression classes and increased code complexity. In such cases, a dedicated parser generator or compiler may be more suitable.

**Code**

Coding Concept

**Difference with similar pattern**

placeholder

**Diagram**

Coding Concept + Geekforgeeks