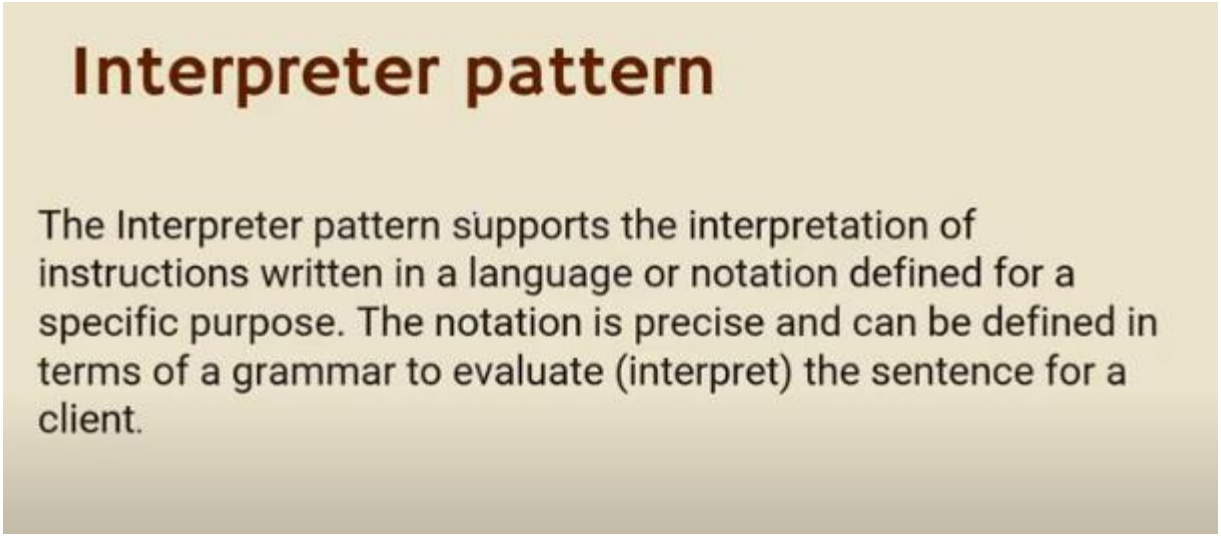
Interpreter Design Pattern

Interpreter design pattern is one of the **behavioral** design pattern.

Interpreter pattern is used to defines a grammatical representation for a language and provides an interpreter to deal with this grammar.

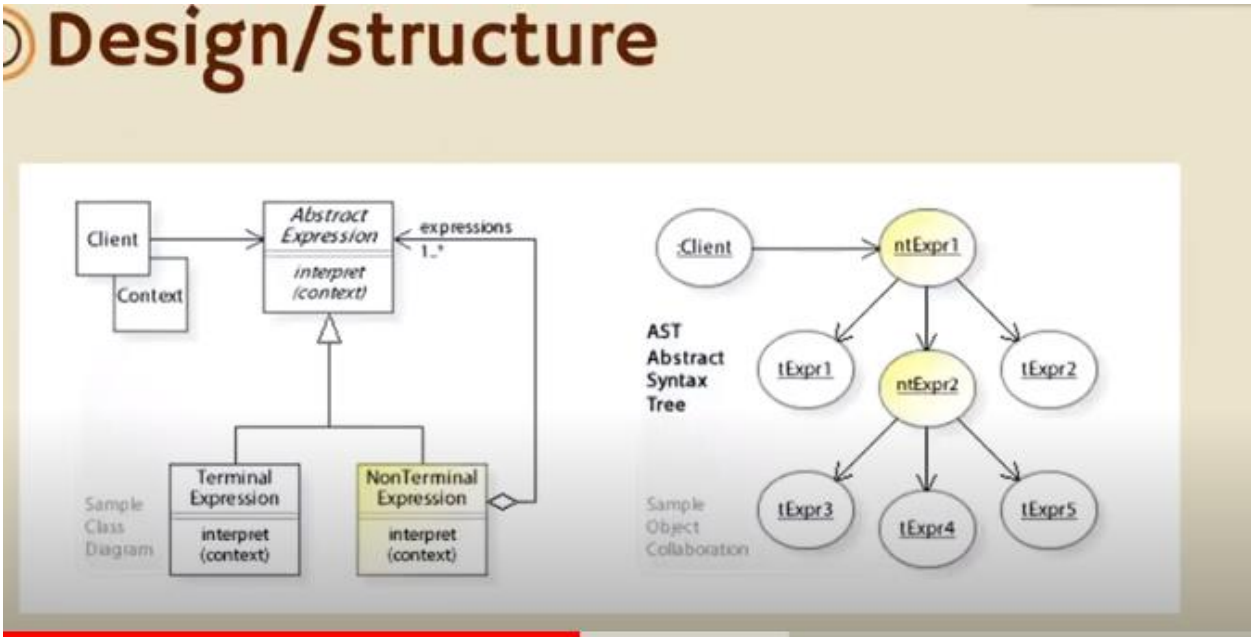
- This pattern involves implementing an expression interface which tells to interpret a particular context. This pattern is used in SQL parsing, symbol processing engine etc.
- This pattern performs upon a hierarchy of expressions. Each expression here is a terminal or non-terminal.
- The tree structure of 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.
- The tree contains the expressions to be evaluated and is usually generated by a parser. The parser itself is not a part of the interpreter pattern.



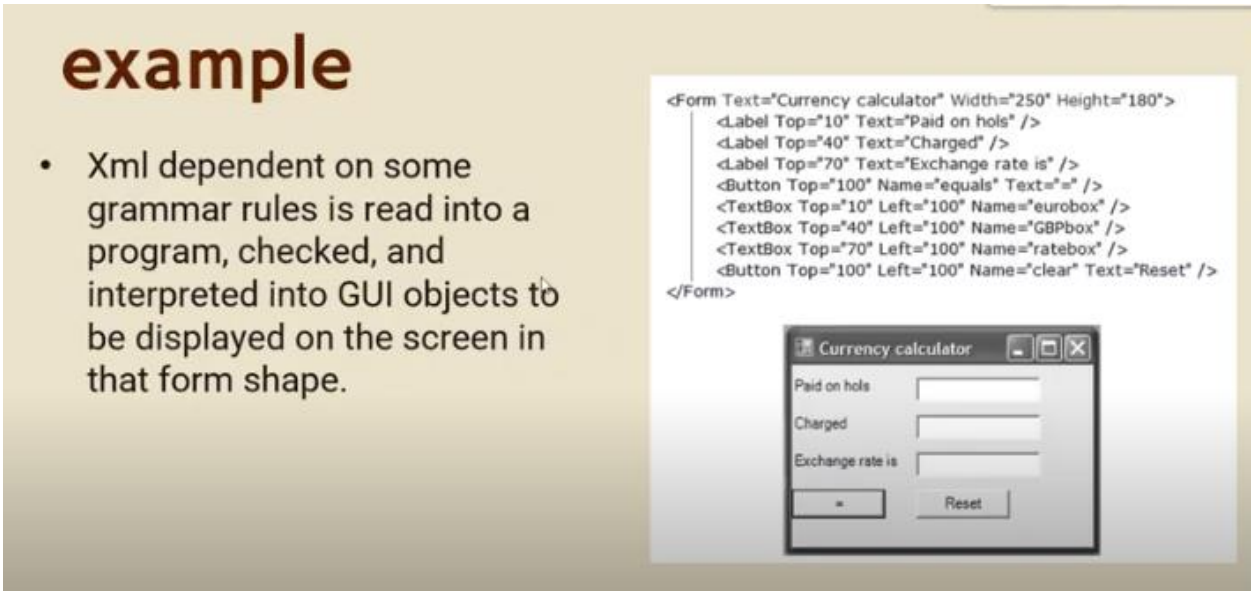
Some applications are so complex that they require advanced configuration. You could offer a basic scripting language which allows the end-user to manipulate your application through simple instructions. The Interpreter pattern solves this particular problem – that of creating a simple scripting language.

Certain types of problems lend themselves to be characterized by a language. This language describes the problem domain which should be well-understood and well-defined. To implement this you need to map the language to a grammar. Grammars are usually hierarchical tree-like structures that step through multiple levels and then end up with terminal nodes (also called literals).

Problems like this, expressed as a grammar, can be implemented using the Interpreter design pattern.



For Example :



Participants

The objects participating in this pattern are:

- **Client** -- In sample code: the run() program.
  - builds (or is given) a syntax tree representing the grammar
  - establishes the initial context
  - invokes the interpret operations
- **Context** -- In sample code: **Context**
  - contains state information to the interpreter
- **TerminalExpression** -- In sample code: **Expression**
  - implements an interpret operation associated with terminal symbols in the grammar
  - one instance for each terminal expression in the sentence
- **NonTerminalExpression** -- In sample code: not used
  - implements an interpret operation associated for non-terminal symbols in the grammar

Reference :-

1) https://www.dofactory.com/javascript/design-patterns/interpreter

2) https://www.geeksforgeeks.org/interpreter-design-pattern/

3) https://www.youtube.com/watch?v=doASLHEmAqM&t=528s