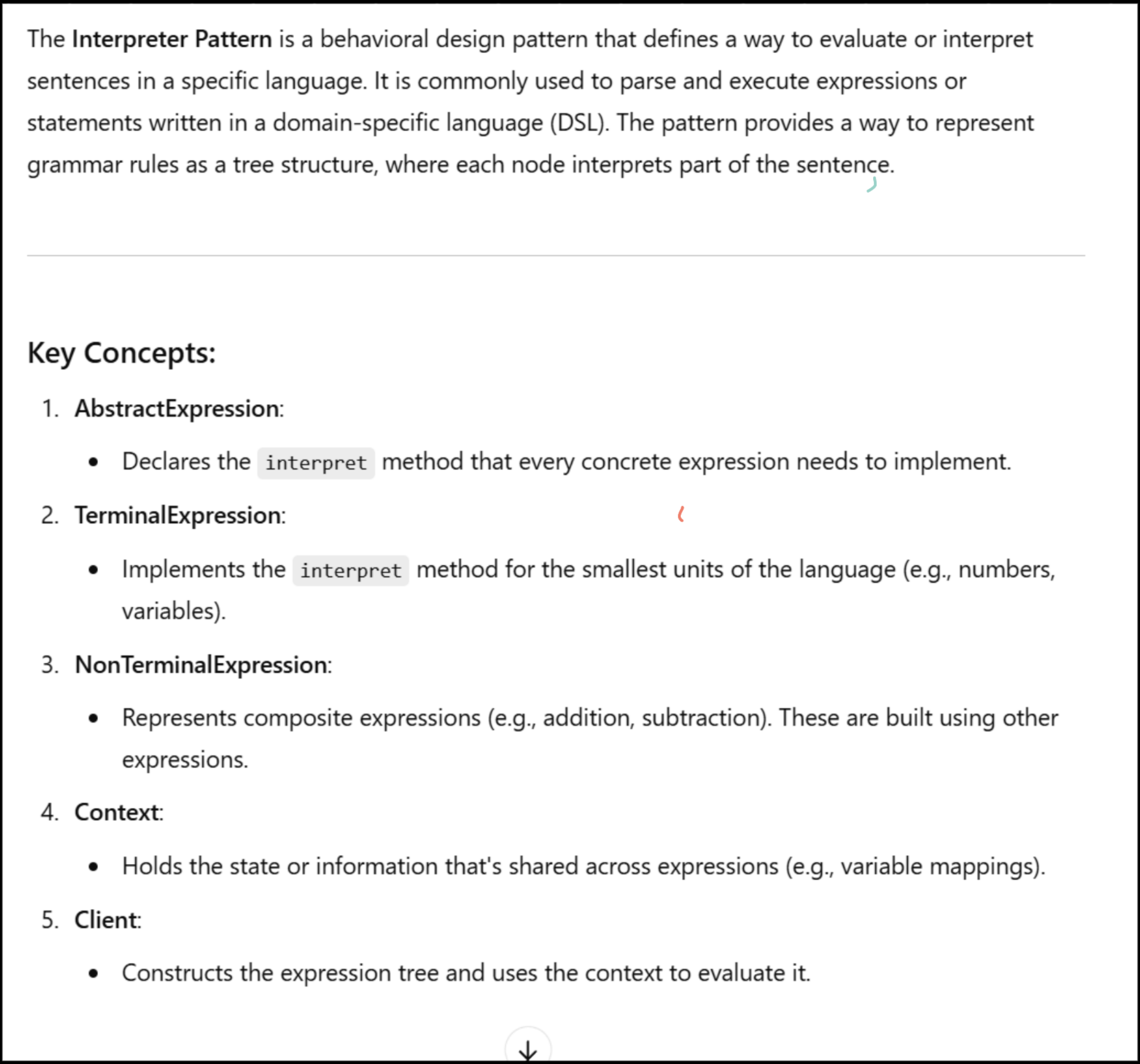
[Interpreter Design Pattern]



The **Interpreter Pattern** is a behavioral design pattern that defines a way to evaluate or interpret sentences in a specific language. It is commonly used to parse and execute expressions or statements written in a domain-specific language (DSL). The pattern provides a way to represent grammar rules as a tree structure, where each node interprets part of the sentence.

Key Concepts:

1. **AbstractExpression**:
   - Declares the `interpret` method that every concrete expression needs to implement.
2. **TerminalExpression**:
   - Implements the `interpret` method for the smallest units of the language (e.g., numbers, variables).
3. **NonTerminalExpression**:
   - Represents composite expressions (e.g., addition, subtraction). These are built using other expressions.
4. **Context**:
   - Holds the state or information that's shared across expressions (e.g., variable mappings).
5. **Client**:
   - Constructs the expression tree and uses the context to evaluate it.