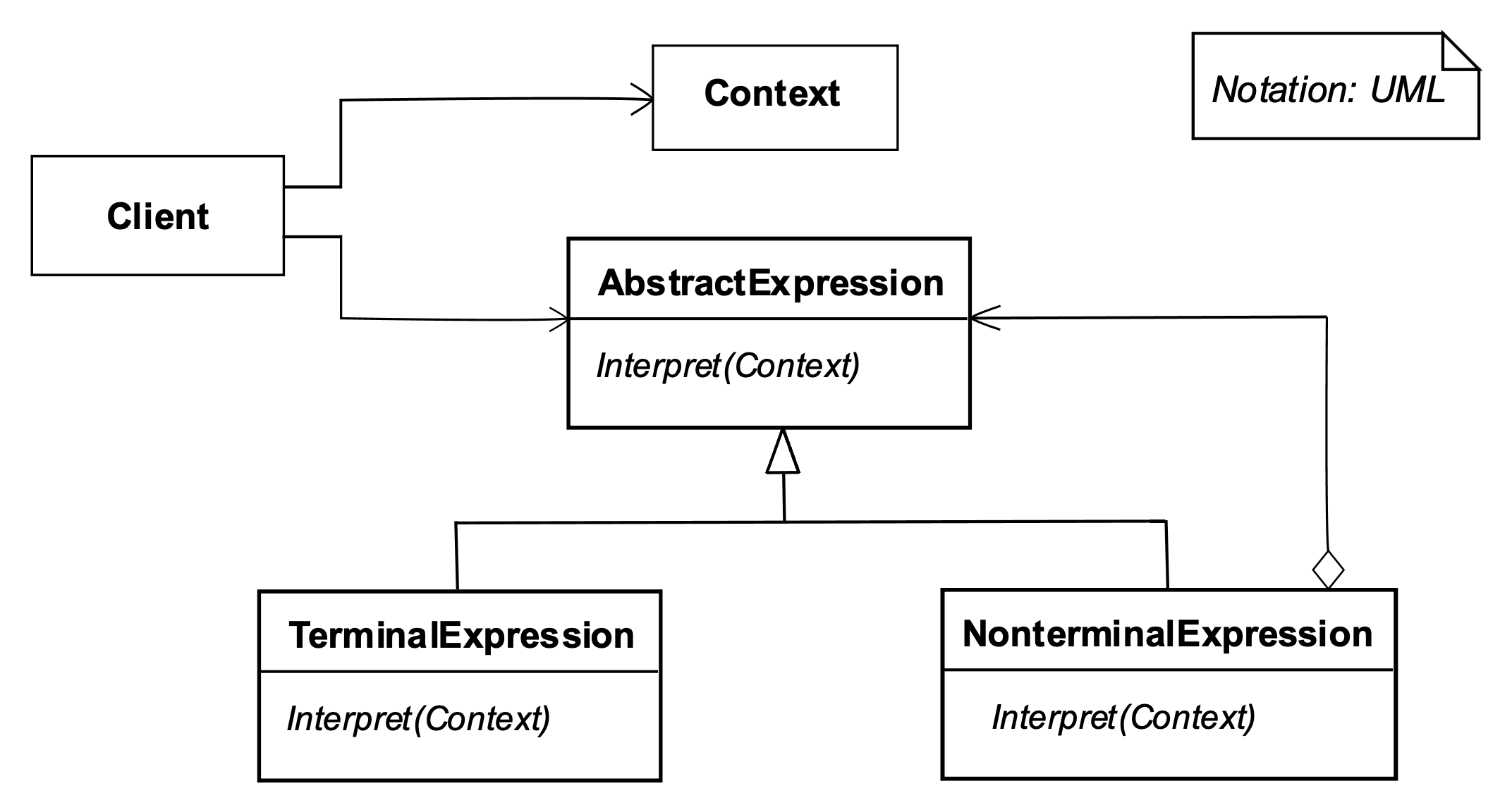
**Interpreter**

Das Interpreter-Entwurfsmuster ist ein Verhaltensmuster, das eine Möglichkeit definiert, Sprachgrammatik oder Ausdrücke zu interpretieren und auszuwerten. Es bietet einen Mechanismus zur Auswertung von Sätzen in einer Sprache, indem ihre Grammatik als eine Reihe von Klassen dargestellt wird. Jede Klasse repräsentiert eine Regel oder einen Ausdruck in der Grammatik, und das Muster ermöglicht es, diese Klassen hierarchisch zu komponieren, um komplexe Ausdrücke zu interpretieren.

Das Muster umfasst die Definition einer Hierarchie von Ausdrucksklassen, sowohl terminaler als auch nicht-terminaler, um die Elemente der Grammatik der Sprache zu repräsentieren.

Terminalausdrücke repräsentieren grundlegende Bausteine, während nicht-terminale Ausdrücke Zusammensetzungen dieser Bausteine repräsentieren.

*Components:*



Die abstrakte Klasse **AbstrakterAusdruck** schreibt eine Methode Interpretiere() vor, die von allen abgeleiteten, konkreten Klassen implementiert werden muss und den entsprechenden Ausdruck auswertet.

Ein **TerminalerAusdruck** steht für einen Ausdruck, der keinen Unterausdruck hat, d. h. für einen festen Wert innerhalb eines gemäß der Grammatik geformten Satzes. Z. B. steht Zahl für einen Zahlenwert innerhalb einer mathematischen Formel.

Ein **NonterminalerAusdruck** steht für einen Ausdruck, der aus Unterausdrücken besteht, zum Beispiel *Addition* oder *Multiplikation*, die beide zwei Operanden als Unterausdrücke benötigen. Ein Unterausdruck kann sowohl ein **TerminalerAusdruck** als auch ein NichtterminalAusdruck sein.

Für den zu interpretierenden Satz wird gemäß der Grammatik ein Syntaxbaum aus *Nichtterminal-* und *Terminalausdrücken* aufgebaut. Dies kann durch einen externen Parser oder den Klienten selbst geschehen. Der Klient wertet diesen Syntaxbaum aus, indem er für den obersten Ausdruck die Methode Interpretiere() aufruft.

Im Kontext werden die konkreten Werte der Terminalausdrücke gekapselt, mit denen der Satz interpretiert werden soll, z. B. die Belegung von Variablen.

*Vor- und Nachteile:*

Vorteile:

Einfache Implementierung: Interpreter sind im Allgemeinen einfacher zu schreiben als Compiler, da sie keinen Maschinencode generieren müssen.

Portabilität: Da Interpreter den Quellcode direkt ausführen, können sie auf jeder Maschine ohne Neukompilierung des Codes ausgeführt werden.

Debugging: Interpreter können detaillierte Fehlerberichte liefern, da sie das Programm Zeile für Zeile ausführen. Dies erleichtert das Debuggen.

Dynamische Typisierung: Interpreter werden oft mit dynamisch typisierten Sprachen verwendet, die flexibler und einfacher zu verwenden sein können für bestimmte Aufgaben.

Interaktive Ausführung: Interpreter können Code interaktiv ausführen, was nützlich ist für Skripterstellung, Testen und Programmierlernen.

Nachteile:

Leistung: Interpretierter Code ist im Allgemeinen langsamer als kompilierter Code, da der Interpreter jede Codezeile bei jeder Ausführung analysieren muss.

Speicherverbrauch: Interpreter verwenden oft mehr Speicher als kompilierte Programme, da sie den Quellcode und den Programmzustand speichern müssen.

Startzeit: Interpreter können eine längere Startzeit haben, da sie den Interpreter initialisieren und den Quellcode laden müssen.

Sicherheit: Quellcode in interpretierten Sprachen ist leichter umkehrbar, was ein Sicherheitsrisiko darstellen kann.

Abhängigkeit: Auf der Zielmaschine muss der entsprechende Interpreter installiert sein, um den Code auszuführen.

*Beispiele für Verwendung:*

Bei der Arbeit mit domänenspezifischen Sprachen:

Wenn Sie Ausdrücke oder Befehle in einer domänenspezifischen Sprache (DSL) interpretieren und ausführen müssen, kann das Interpreter-Muster eine flexible und erweiterbare Möglichkeit bieten, die Grammatik und Semantik der Sprache zu implementieren.

Wenn Sie eine Grammatik zum Interpretieren haben:

Wenn Sie eine klar definierte Grammatik für Ausdrücke oder Befehle haben, die interpretiert werden müssen, kann das Interpreter-Muster dabei helfen, diese Strukturen effizient zu parsen und auszuwerten.

Wenn das Hinzufügen neuer Operationen häufig ist:

Wenn Ihre Anwendung häufig das Hinzufügen neuer Operationen oder Befehle erfordert, ermöglicht das Interpreter-Muster das Hinzufügen neuer Ausdrucksklassen einfach, ohne den bestehenden Code zu ändern, was die Wartbarkeit und Erweiterbarkeit fördert.

Wenn Sie komplexe Grammatikparser vermeiden möchten:

Wenn das Erstellen und Pflegen von komplexen Grammatikparsern für Ihren Anwendungsfall entmutigend oder unnötig erscheint, bietet das Interpreter-Muster eine einfachere Alternative zum direkten Interpretieren von Ausdrücken.

Source :

1) https://de.wikipedia.org/wiki/Interpreter\_(Entwurfsmuster)  
2) https://www.geeksforgeeks.org/interpreter-design-pattern/#reallife-analogy-of-interpreter-design-pattern