Behavioral Pattern: Interpreter

Kevin Dockx
Architect

@KevinDockx | www.kevindockx.com

Coming Up

Describing the interpreter pattern

- Translating Arabic numerals to Roman numerals

Structure of the interpreter pattern

Coming Up

Use cases for this pattern

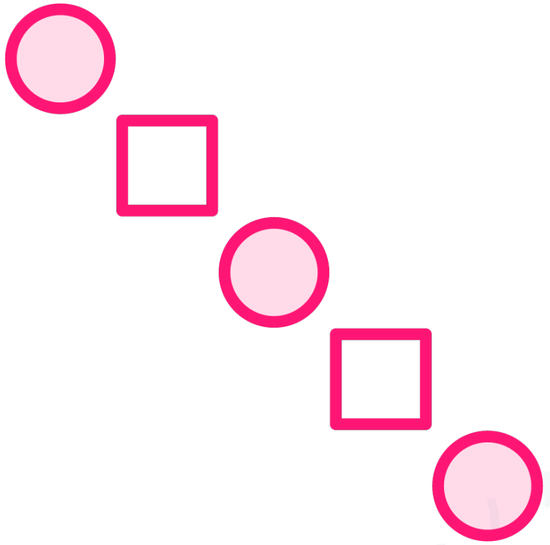
Pattern consequences

Related patterns

Describing the Interpreter Pattern

Creational

Structural

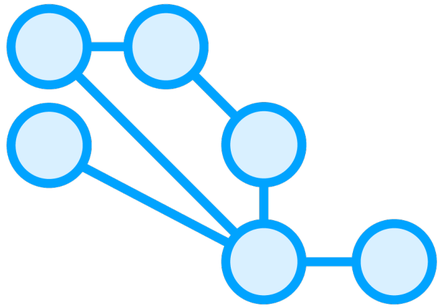


Behavioral

Interpreter

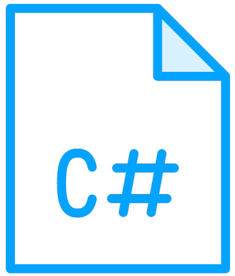
The intent of this pattern is to, given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language

Describing the Interpreter Pattern



LINQ queries

Regular expressions



C# compiler

Describing the Interpreter Pattern

Translating Arabic (181) to Roman (CLXXXI) numerals

- Huge, unmanageable switch statement

Describing the Interpreter Pattern

Look at “181” as a sentence in a grammar

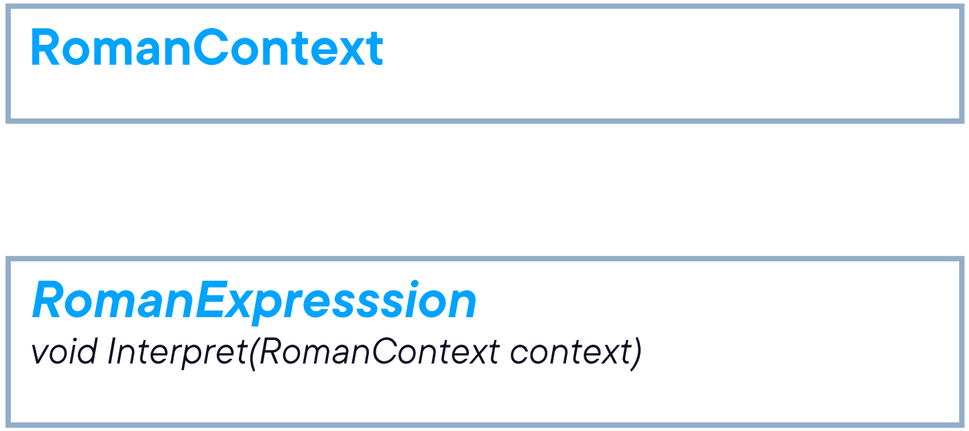
- Grammar consists of a set of expressions
- Each expression is responsible for translating a word

A set of expressions is called a syntax tree

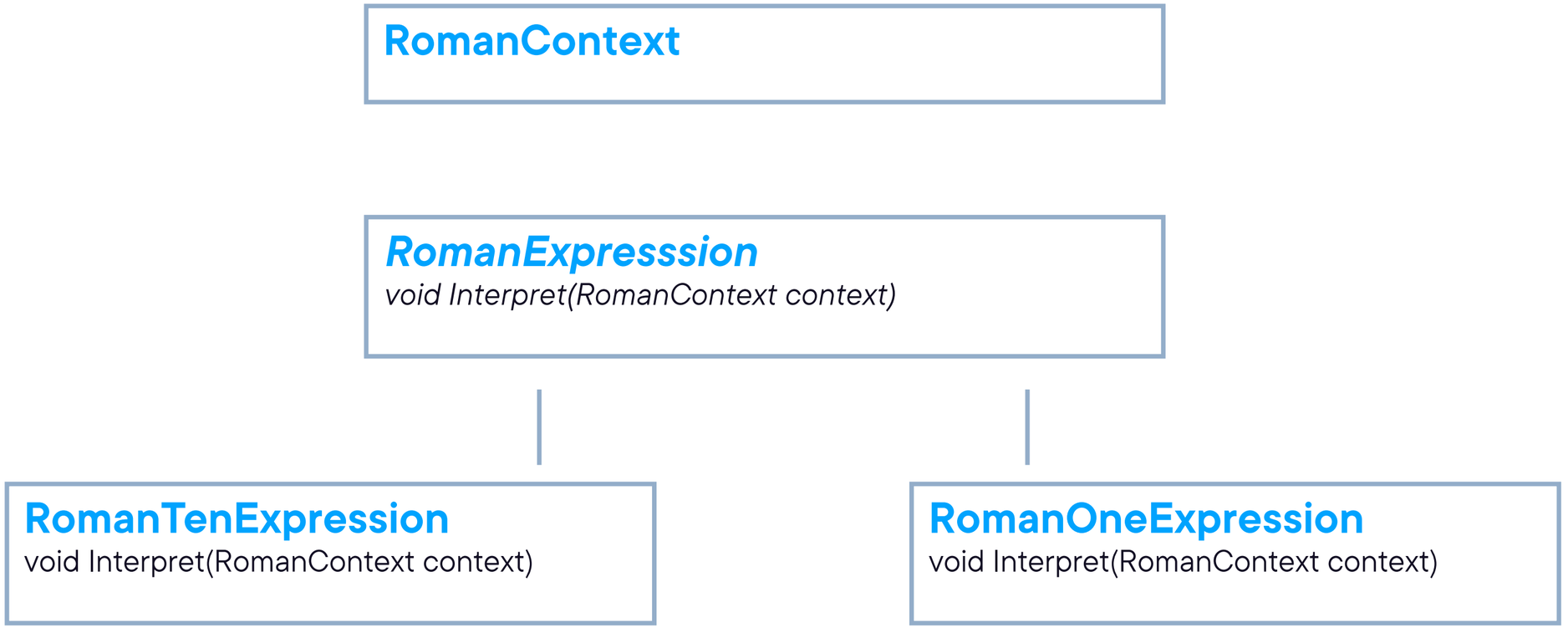
Describing the Interpreter Pattern

Describing the Interpreter Pattern

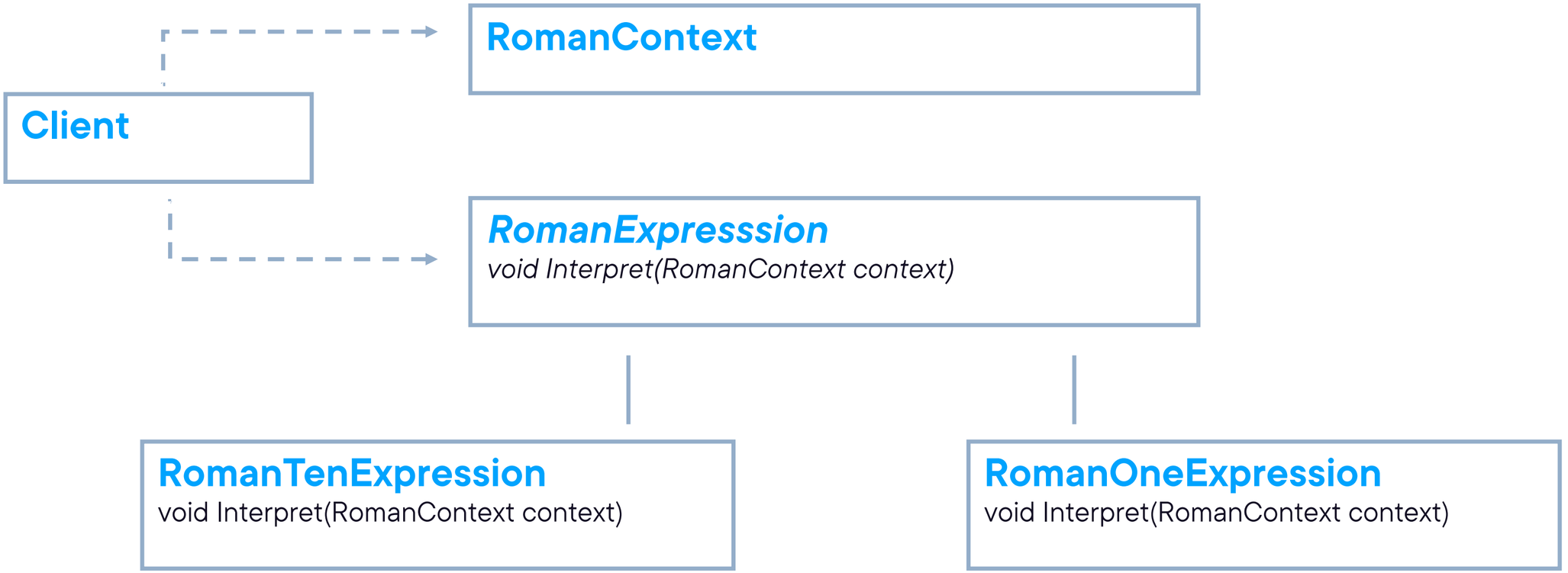
Describing the Interpreter Pattern



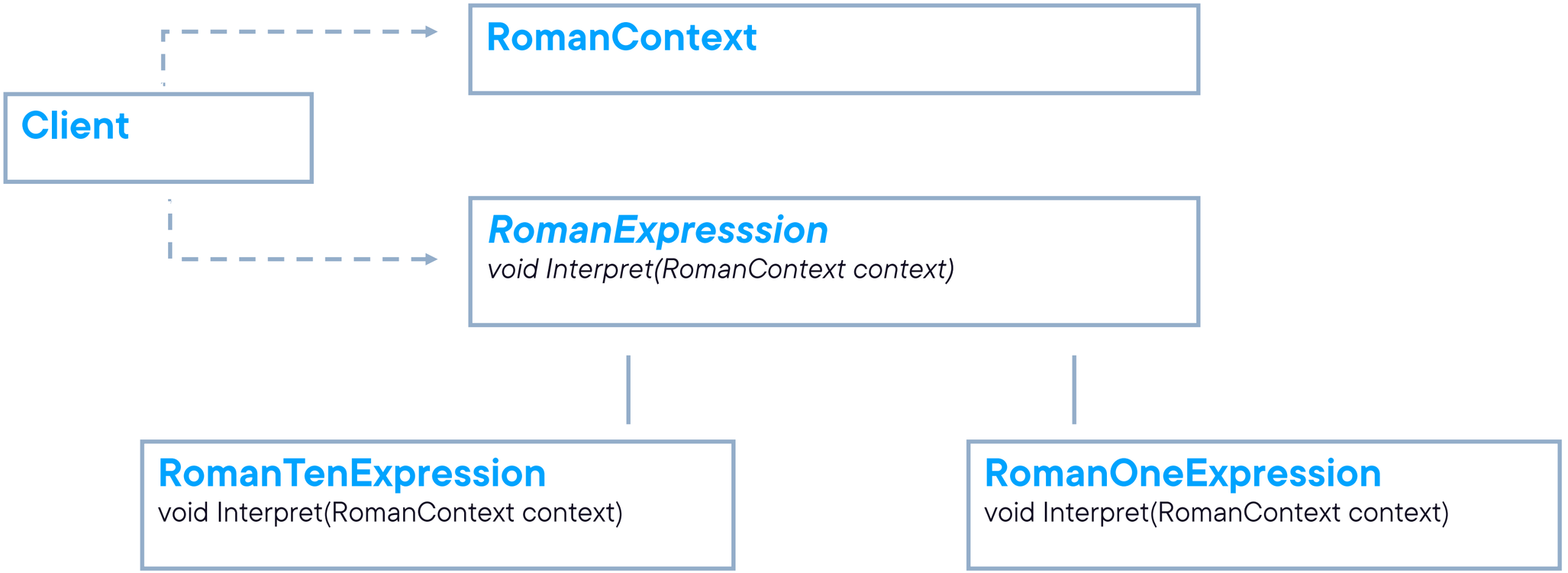
Describing the Interpreter Pattern



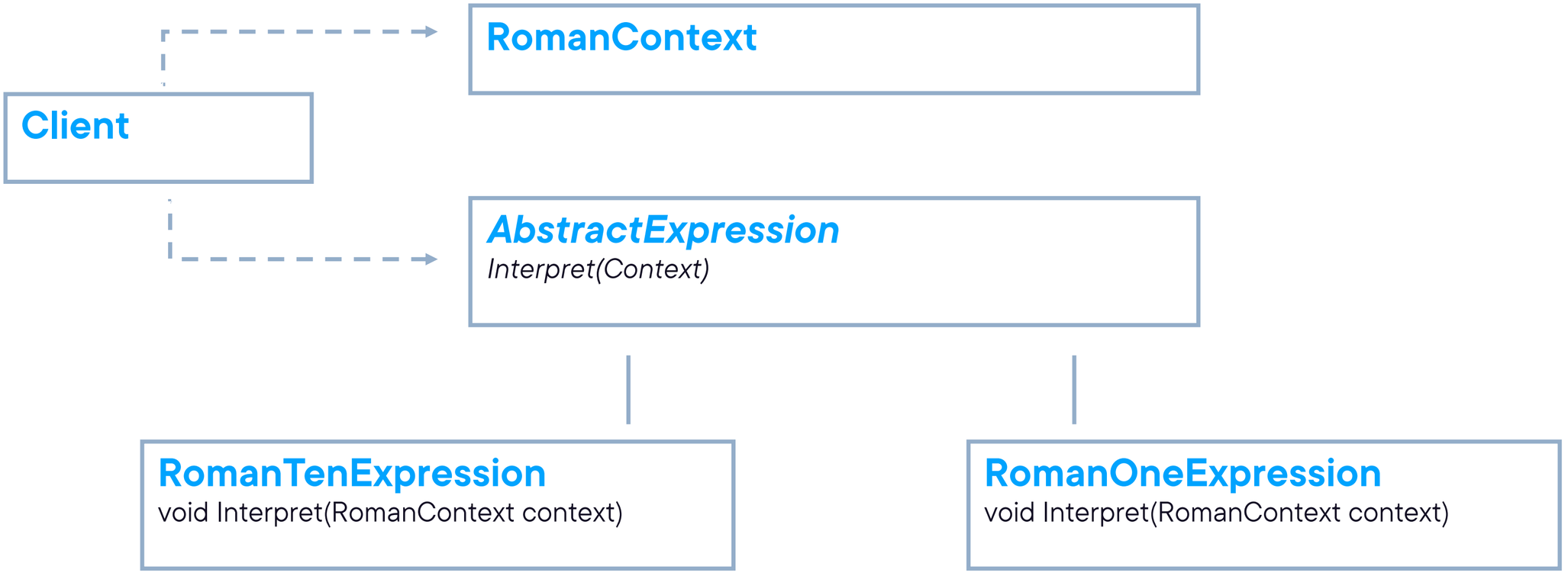
Describing the Interpreter Pattern



Structure of the Interpreter Pattern



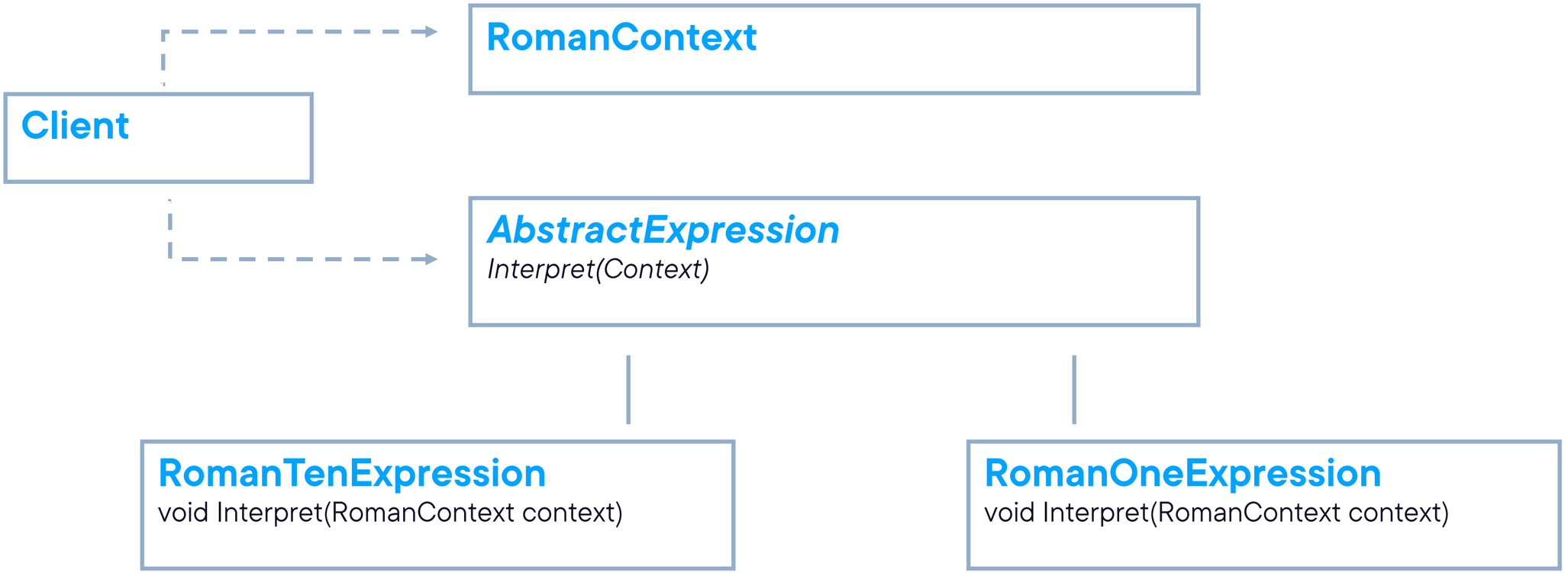
Structure of the Interpreter Pattern



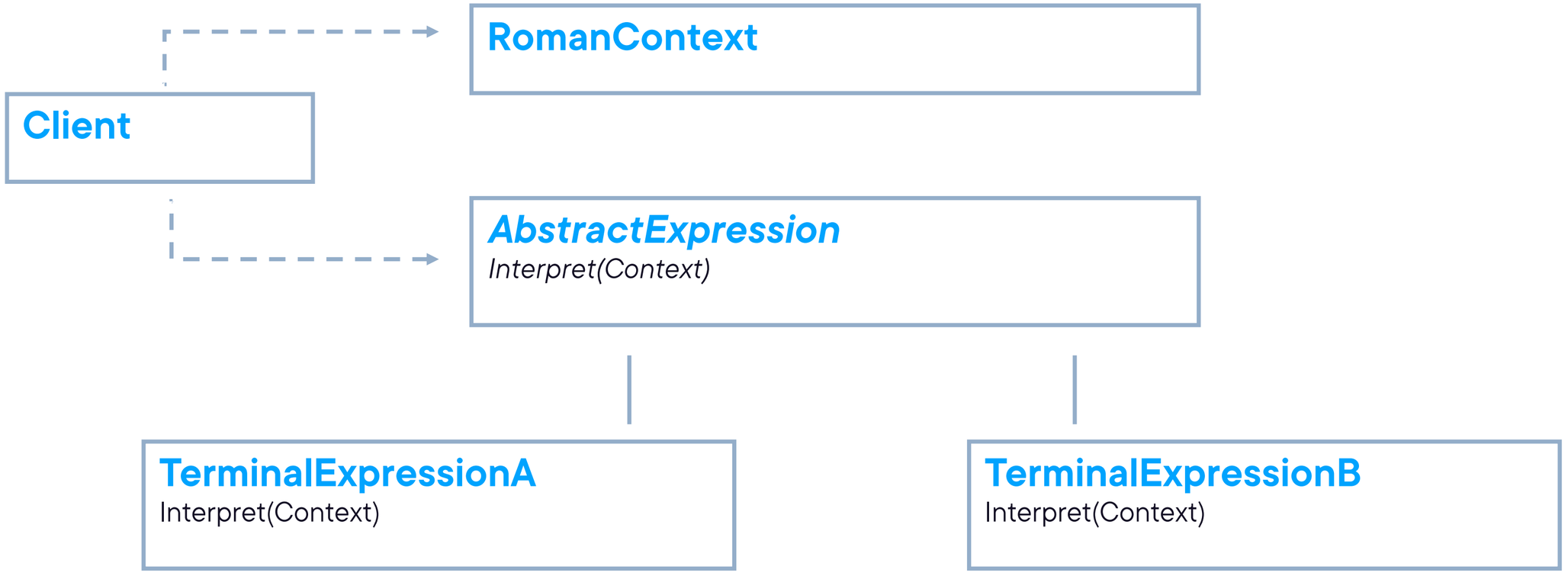
`AbstractExpression` **declares an abstract Interpret operation that is common to all nodes in the abstract syntax tree**

`TerminalExpression` **implements an Interpret operation associated with terminal symbols in the grammar**

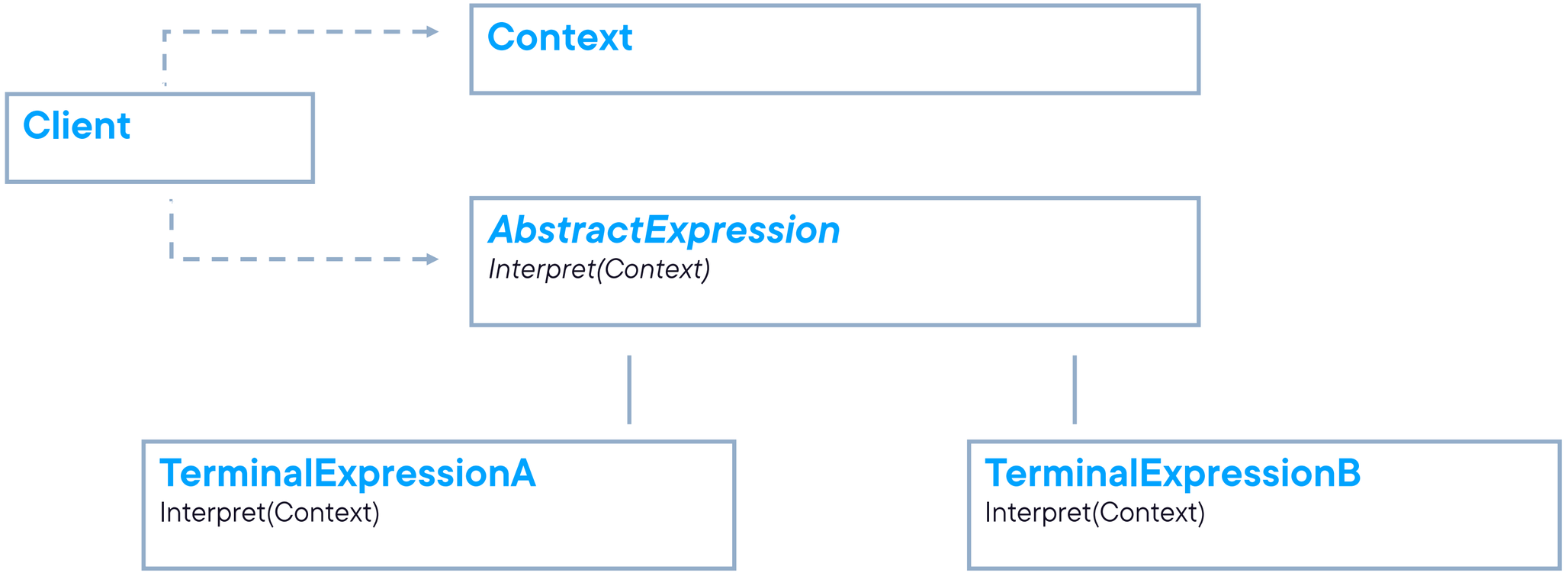
Structure of the Interpreter Pattern



Structure of the Interpreter Pattern

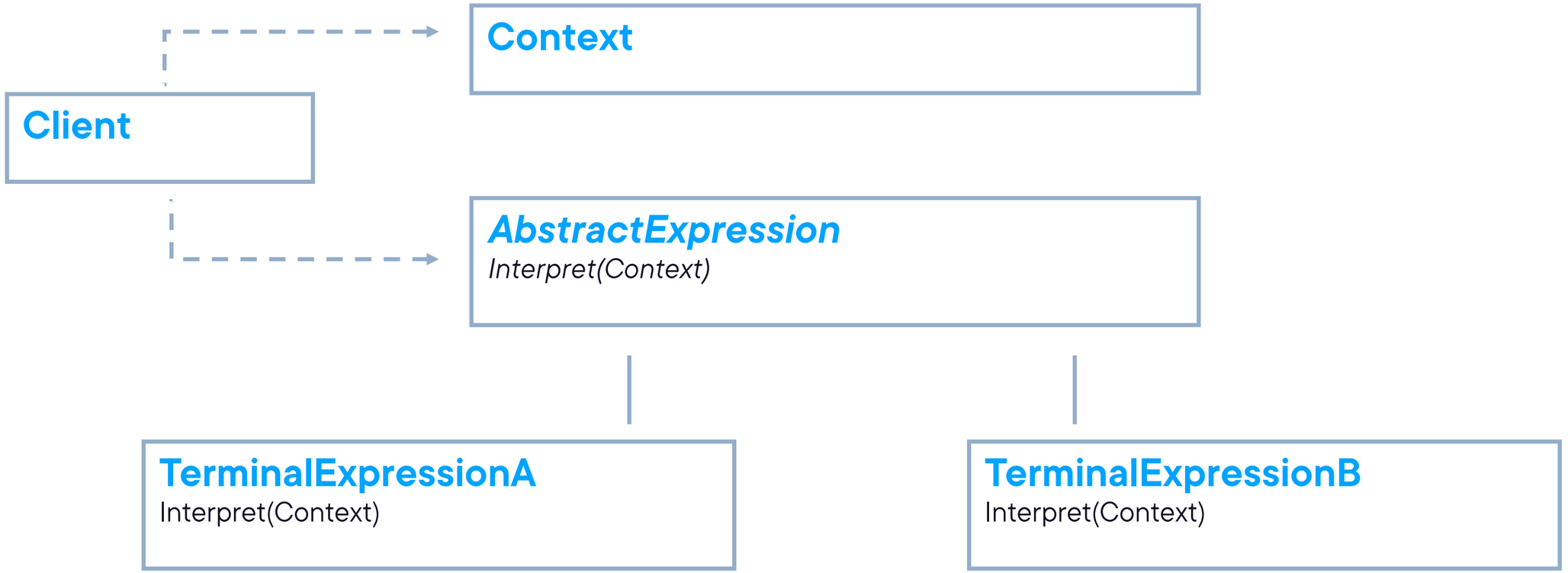


Structure of the Interpreter Pattern



`Context` **contains information that's global to the** `Interpreter`

Structure of the Interpreter Pattern



`Client` **builds or is given the abstract syntax tree which represents a sentence in the language that the grammar defines.** `Client` **invokes the Interpret operation.**

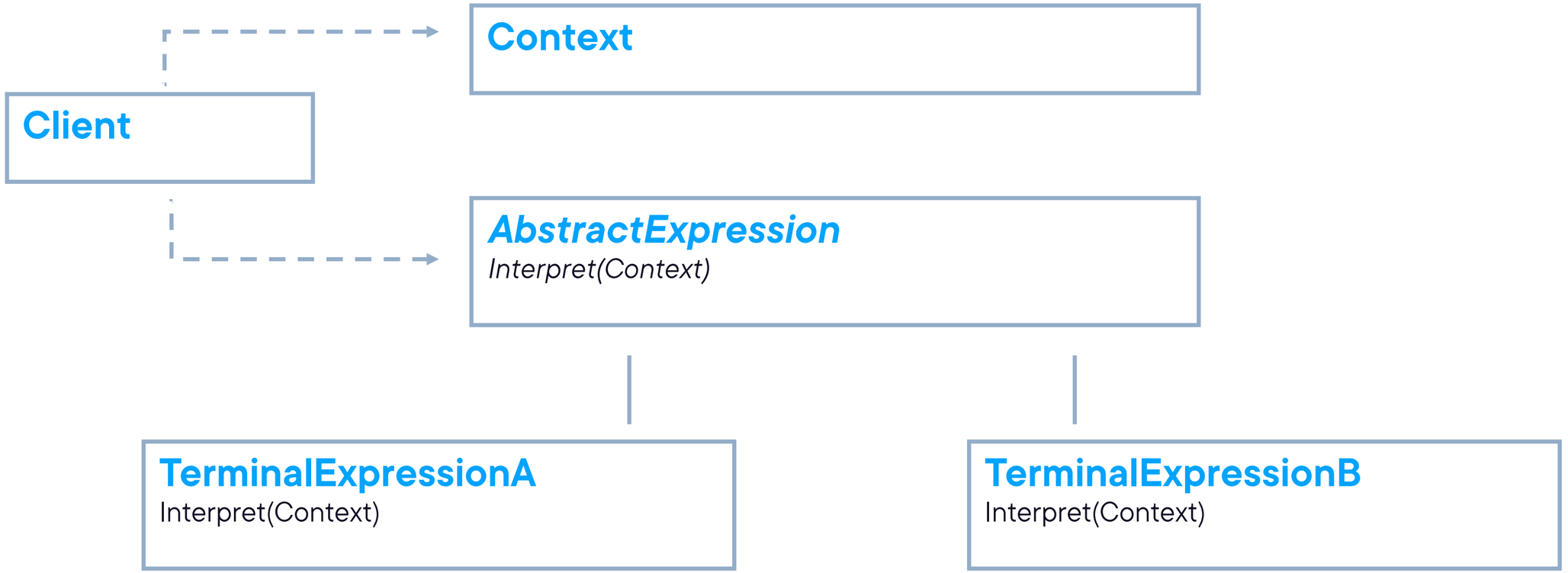
Demo

Implementing the interpreter pattern

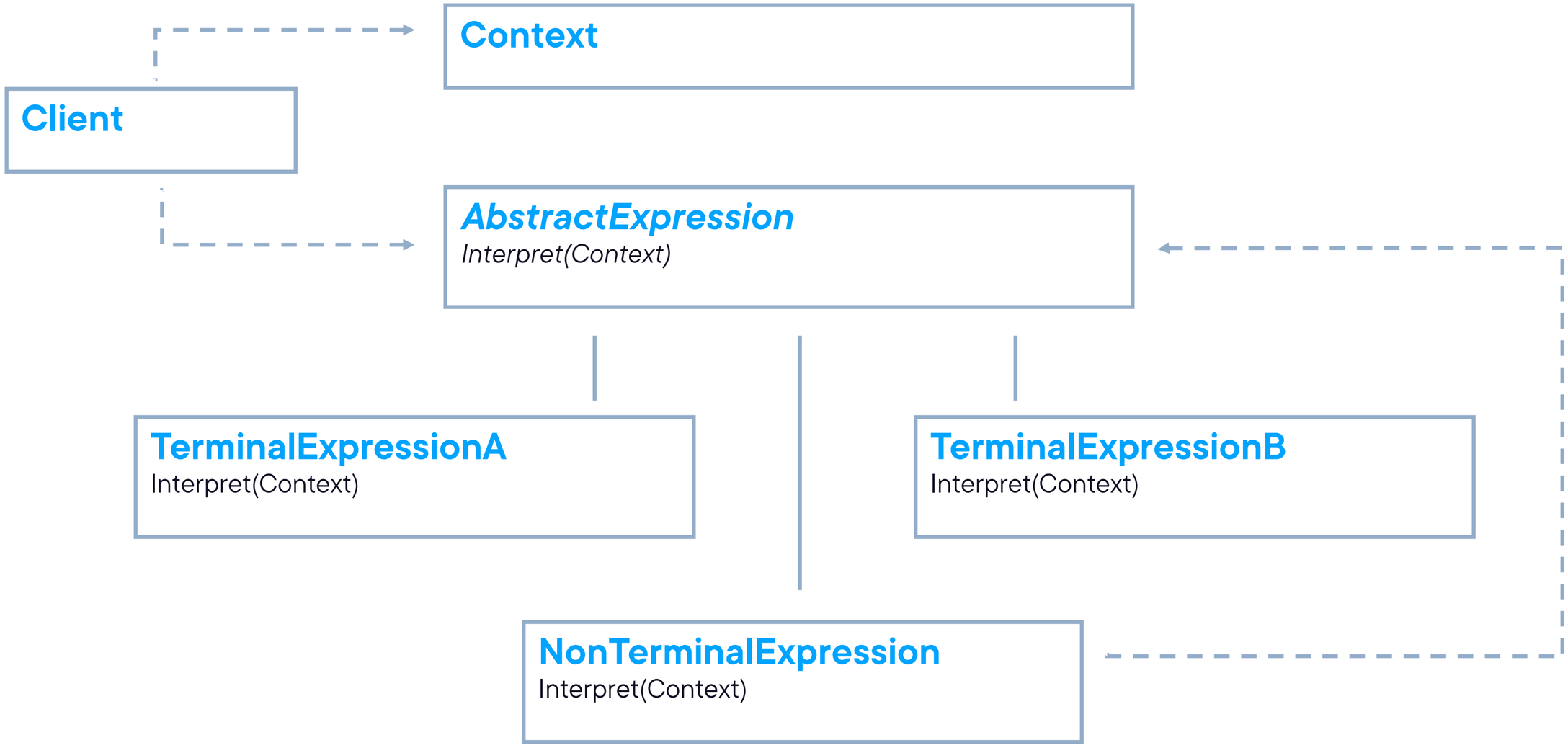
Terminal Versus NonTerminal Expressions

Terminal expression	Vs	NonTerminal expression
A symbol (the smallest meaningful part or unit of a language) is called a Terminal Implemented through a Terminal expression		Statements that are made up of terminals and are allowed by a language are called NonTerminals Can consist of Terminal(s) and/or NonTerminal(s)

Terminal Versus NonTerminal Expressions



Terminal Versus NonTerminal Expressions



Use Cases for the Interpreter Pattern

When there's a language you can interpret, and you can represent statements in the language as abstract syntax trees

AND when the grammar is simple

AND when the best possible efficiency isn't required

Use Cases for the Interpreter Pattern

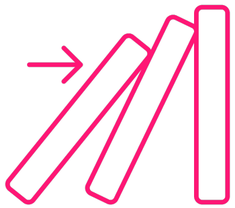
Command line parsers

HTML or XML parsing

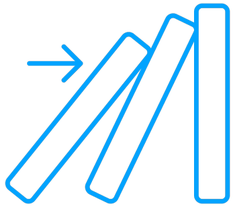
Natural language processors

Mathematical expression evaluation

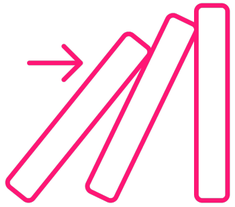
Pattern Consequences



It’s easy to change and extend the grammar

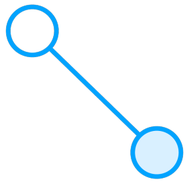


It’s easy to implement the grammar

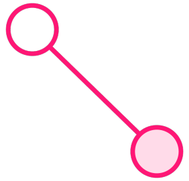


Complex grammars are hard to maintain

Related Patterns



Composite
The abstract syntax tree is an instance of the composite pattern



Iterator
You can use an iterator to traverse the tree

Summary

Intent of the interpreter pattern:

- To, given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language

Summary

A symbol (the smallest meaningful part or unit of a language) is called a Terminal

Statements that are made up of terminals and are allowed by a language are called NonTerminals

The End is Nigh...

Questions?
Use the discussions tab on the course page. Follow me on X (@KevinDockx) for updates.

Please consider rating this course! :)


You're ready
to be
AWESOME!