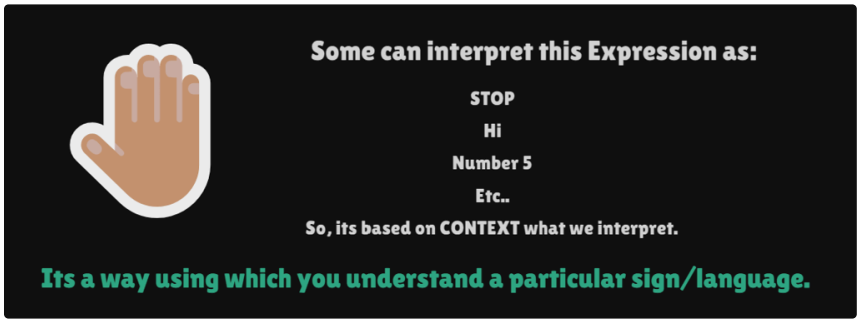
Interpreter Pattern

Resources

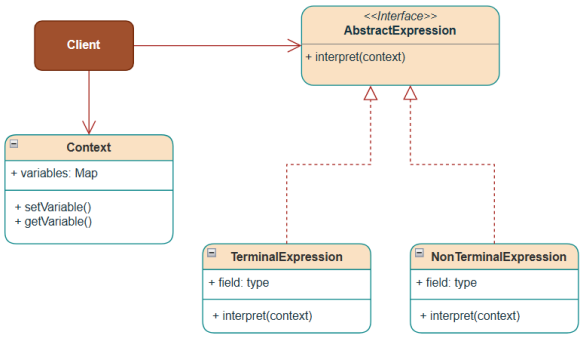
- 41. All Behavioral Design Patterns | Strategy, Observer, State, Template, Command, Visitor, Memento
- 40. Interpreter Design Pattern | LLD System Design | Design pattern explanation in Java

Definition



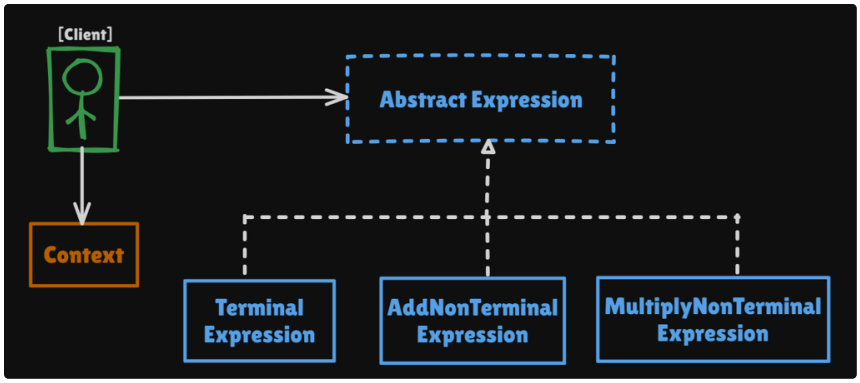
> ***The Interpreter design pattern is a behavioral pattern that defines how to interpret and evaluate expressions.***

Class Diagram



Structure of Interpreter Pattern

Structure of a simple mathematical expression interpreter that can handle addition and multiplication of numbers:



- **Context**: Contains information that's global to the interpreter (like variable values).
- **Abstract Expression**: Defines the interpret operation that all concrete expressions must implement.
- **Terminal Expression**: Implements interpretation for variables in the expressions (leaf nodes).
- **Non-terminal Expression(** `AddNonTerminalExpression`, `MultiplyNonTerminalExpression` **)**: Implements interpretation for non-terminal expressions (composite expressions that contain other expressions).
- **Client**: Builds the expressions and invokes the interpret operation.

Implementation

Let's create a simple mathematical expression interpreter that can handle addition and multiplication of numbers:

```java
// Context class
class Context {
    private final Map<String, Integer> variables = new HashMap<>();

    public void setVariable(String name, int value) {
        variables.put(name, value);
    }

    public int getVariable(String name) {
        return variables.getOrDefault(name, 0);
    }

    @Override
    public String toString() {
        return variables.toString();
    }
}
```

```java
// Abstract Expression interface
public interface AbstractExpression {
    int interpret(Context context);
}
```

```java
// Terminal Expression - represents a variable
public class NumberTerminalExpression implements AbstractExpression {
    String stringValue;

    NumberTerminalExpression(String stringVal) {
        this.stringValue = stringVal;
    }

    @Override
    public int interpret(Context context) {
        return context.getVariable(stringValue);
    }
}
```

```java
// Non-terminal Expression - represents addition
public class AddNonTerminalExpression implements AbstractExpression {
    private final AbstractExpression rightExpression;
    private final AbstractExpression leftExpression;

    public AddNonTerminalExpression(AbstractExpression left,
AbstractExpression right) {
        this.leftExpression = left;
        this.rightExpression = right;
    }

    @Override
    public int interpret(Context context) {
        return leftExpression.interpret(context) +
rightExpression.interpret(context);
    }
}
```

```java
// Non-terminal Expression - represents multiplication
public class MultiplyNonTerminalExpression implements
AbstractExpression {
    private final AbstractExpression leftExpression;
    private final AbstractExpression rightExpression;

    public MultiplyNonTerminalExpression(AbstractExpression left,
AbstractExpression right) {
        this.leftExpression = left;
        this.rightExpression = right;
    }

    @Override
    public int interpret(Context context) {
        return leftExpression.interpret(context) *
rightExpression.interpret(context);
    }
}
```

```java
// Non Terminal Expression - represents an expression with a binary
operator + or *
public class BinaryNonTerminalExpression implements AbstractExpression
{
    AbstractExpression leftExpression;
    AbstractExpression rightExpression;
    char operator;

    public BinaryNonTerminalExpression(AbstractExpression
leftExpression, AbstractExpression rightExpression, char operator) {
        this.leftExpression = leftExpression;
        this.rightExpression = rightExpression;
        this.operator = operator;
    }

    @Override
    public int interpret(Context context) {
        return switch (operator) {
            case '+' -> leftExpression.interpret(context) +
rightExpression.interpret(context);
            case '*' -> leftExpression.interpret(context) *
rightExpression.interpret(context);
            default -> 0;
        };
    }
}
```

```java
// Client code
public class Client {
    public static void main(String[] args) {
        System.out.println("##### Interpreter Design Pattern #####");

        // Context
```

```
        Context context = new Context();
        context.setVariable("a", 12);
        context.setVariable("b", 5);
        context.setVariable("c", 3);
        context.setVariable("d", 9);
        System.out.println("Context: " + context);

        // Expression: a + b
        AbstractExpression expression1 = new AddNonTerminalExpression(
                new NumberTerminalExpression("a"),
                new NumberTerminalExpression("b"));
        System.out.println("Expression: (a+b) = " +
expression1.interpret(context)); // Output: 17

        // Expression: a * b
        AbstractExpression expression2 = new
MultiplyNonTerminalExpression(
                new NumberTerminalExpression("a"),
                new NumberTerminalExpression("b")
        );
        System.out.println("Expression: (a*b) = " +
expression2.interpret(context)); // Output: 60

        // Complex Expression: (a + b) * c
        AbstractExpression expression3 = new
MultiplyNonTerminalExpression(
                new AddNonTerminalExpression(
                        new NumberTerminalExpression("a"),
                        new NumberTerminalExpression("b")
                ),
                new NumberTerminalExpression("c")
        );
        System.out.println("Expression: ((a+b)*c) = " +
expression3.interpret(context)); // Output: 51

        // Expression: ((a*b) + (c*d))
        AbstractExpression expression4 = new
BinaryNonTerminalExpression(
                new BinaryNonTerminalExpression(
                        new NumberTerminalExpression("a"), new
NumberTerminalExpression("b"), '*'),
                new BinaryNonTerminalExpression(
                        new NumberTerminalExpression("c"), new
NumberTerminalExpression("d"), '*'),
                '+');
        System.out.println("Expression: ((a*b) + (c*d)) = " +
expression4.interpret(context));

    }
}
```

Output

```
##### Interpreter Design Pattern #####
Context: {a=12, b=5, c=3, d=9}
Expression: (a+b) = 17
Expression: (a*b) = 60
Expression: ((a+b)*c) = 51
Expression: ((a*b) + (c*d)) = 87

Process finished with exit code 0
```