Interpreter パターン

目的

Interpreter パターンの基本的な考えは、定義された種類の問題を素早く解くために、ドメインに特化した言語を実装することです。

効果

Interpreter パターンの特徴の 1 つに、「1 つの規則を 1 つのクラスで表す」というものが挙げられます。つまり、新しい規則を追加する場合は AbstractExpression クラスのサブクラスを追加するだけで良くなります。

背景

Interpreter とは、英語で「解釈者・説明者」を意味する単語です。 何らかのフォーマットで書かれたファイルの中身を、解析した結果に則って何らかの処理を行いたい場合があります。 Interpreter パターンとは、このような「解析した結果」得られた手順に則った処理を実現するために最適なパターンです。

Interpreter パターンのコード

InterpreterBuilder の活用例として以下のような構文を実現する算術演算インタープリタを実装してみます。

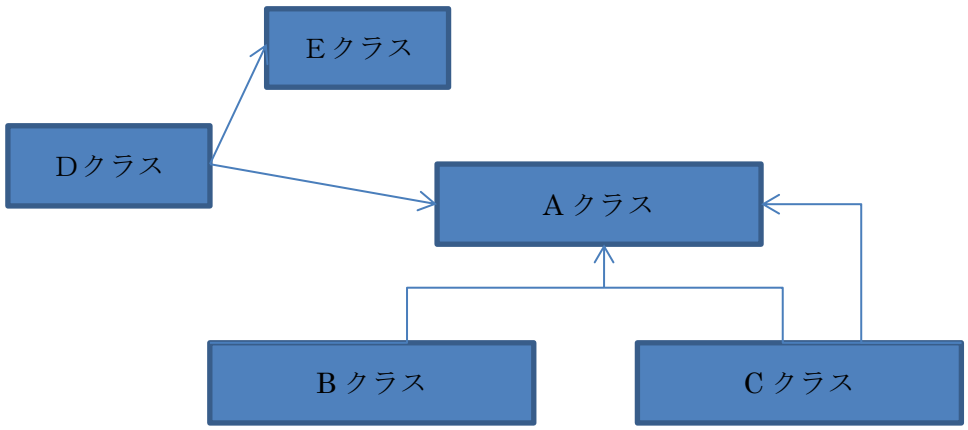
```
exp = divide( plus(2, multiple(3,4)), 3 )
exp.to_i
```

算術演算では終端は数字になるので、self を返す Fixnum#to_i を iterpret メソッドとして使います。

```
require_relative "interpreter_builder"
module Calc
        extend InterpreterBuilder

        class Expression
        end

        nonterminals = {
                plus: :+,
                minus: :-,
                multiple: :*,
                divide: :/
        }

        nonterminals.each do |name, op|
                define_nonterminal(name, Expression, :to_i, op)
        end
end
```

演算子の定義を追加することで他の算術演算も可能になります。

Interpreter パターンのまとめ



A クラス

構文木の要素に共通な API を定義します。

Bクラス

A クラスのサブクラスで、構文木の葉に相当する末端の規則を表します。また、A クラスで定義されたメソッドを実装します。

Cクラス

A クラスのサブクラスで、構文木の節に相当します。内部には、他の規則へのリンクを保持しています。また、A クラスで定義されたメソッドを実装します。

Dクラス

構文木を処理するために必要な情報を保持するクラスです。

Eクラス

A クラスを利用するクラスです。処理する構文木を作成したり、外部から与えられたりします。

総括

Interpreter パターンは用途が非常に具体的なパターンと言えます。