Ruby初級者向けレッスン第 24 回 演習問題解答例

okkez @ Ruby 関西

2008 年 09 月 20 日

演習

スタックを作ろう

実装コード:

```
01: require 'forwardable'
02: class Stack
03:   extend Forwardable
04:
05:   def_delegator(:@stack, :push)
06:   def_delegator(:@stack, :size)
07:   def_delegator(:@stack, :empty?)
08:
09:   def initialize
10:     @stack = []
11:   end
12:
13:   def pop
14:     raise EmptyError if empty?
15:     @stack.pop
16:   end
17:
18:   class EmptyError
19:   end
20: end
```

テストコードは略。(http://jp.rubyist.net/?KansaiWorkshop20 の解答例を参照してください。)

step8 まで実装を行ったあと、もう一段階リファクタリングを行った結果こうなりました。テストは通るはずなのでこういう書き方もあるというのを知ってください。

また使用しているライブラリやメソッドについてはるりまの記事や、リファレンスマニュアルを参照してください。

逆ポーランド記法で使える電卓を作ろう

```
$ rpn.rb 1 2 + 3 4 + \*
21
# * はシェルのワイルドカードなのでエスケープしている
```

のような引数を受け取って計算結果を返すプログラムを書いてください。逆ポーランド記法の詳しい説明は Wikipedia などを参照してください。

- 先ほど作ったスタッククラスを使用する
- テストを先に記述する
- 四則演算ができればよい
- 演算子が受け取る引数は二つまで。
- (追加) 不正な式を受け取った場合は例外を発生させる

以上に注意して進めてください。
解答例のソースコードは別途ダウンロードしてください。

Step 1

- とりあえず $1 + 2 = 3$ をテストする
- 実装コードを書く前にテストを一度失敗させておく
- 失敗を確認したら Fake it でテストを成功させる

Step 2

- Trianguration するために別の式を計算させる
- 実装は Fake it しているために失敗する
- 今回のテストを通すために実装コードを修正する
  - できるだけ実装はシンプルに保つ
- テストを実行して成功するのを確認する

Step 3

ここから先は二つのパターンが考えられます。

- 演算子は一つのまま、長い式に対応させます
- オペランドは二つのまま対応する演算子を増やします

最終的に満たすべき仕様は同じですが、ここでは前者の方針でテストを追加します。前者は機能拡張、後者は機能追加なので機能拡張である前者を行うことにしました。また、今回の場合は前者の方が本質的にはより重要であるということも考えました。

- 長い式を計算するテストを追加します
- 失敗する事を確認します
- テストに成功するように実装コードを修正します
- テストが成功する事を確認します

Step 4

この部分は重要なのでもう少しだけテストを追加します。

- さらに別の長い式を試すテストを追加します
  - 失敗したら、実装を修正します
  - 成功したら、次のステップへ進みます

Step 5

対応する演算子を追加します。演算子は一つずつ追加していきます。

- 別の演算子を使用した式を計算するテストを追加します
- テストが失敗する事を確認します
- 実装コードを修正します
- テストが成功する事を確認します

Step 6

一つずつ追加する予定でしたが、簡単そうなので残り二つの演算子を追加しました。

- 今回追加する演算子を使用する式を計算するテストを追加します
- テストが失敗する事を確認します
- 実装コードを修正します
- テストが成功する事を確認します

Step 7

- リファクタリングします
- テストが成功する事を確認します

Step 8

- さらにリファクタリングします
- テストが成功する事を確認します

Step 9

- 長い式をテストしてみます
- テストが成功する事を確認します

Step 10

不正な式が入力された場合について考えます。不正な式を計算しようとした場合は例外を発生させることにします。

- 不正な式を計算しようとするテストを追加します
- テストが失敗する事を確認します
- 実装コードを修正します
- テストが成功する事を確認します。

最終結果

Test::Unit:

```
01: require 'test/unit'
02: require 'rpn'
03:
04: class TestRpn < Test::Unit::TestCase
05:
06:   def test01_plus
07:     assert_equal(3, rpn(%w[1 2 +]))
08:   end
09:
10:   def test02_plus
11:     assert_equal(4, rpn(%w[2 2 +]))
12:   end
```

```
13:
14:   def test03_plus
15:     assert_equal(6, rpn(%w[2 2 + 2 + ]))
16:   end
17:
18:   def test04_plus
19:     assert_equal(8, rpn(%w[2 2 + 2 2 + +]))
20:   end
21:
22:   def test05_multiple
23:     assert_equal(16, rpn(%w[2 2 + 2 2 + *]))
24:   end
25:
26:   def test06_multiple
27:     assert_equal(8, rpn(%w[3 3 * 3 3 / -]))
28:   end
29:
30:   def test07_multiple
31:     assert_equal(21, rpn(%w[1 2 + 3 4 + *]))
32:   end
33:
34:   def test08_multiple
35:     assert_equal(55, rpn(%w[1 2 3 4 5 6 7 8 9 10 + + + + + + + + + +]))
36:   end
37:
38:   def test09_invalid_expr
39:     assert_raise(RuntimeError){
40:       rpn(%w[1 2])
41:     }
42:   end
43: end
```

RSpec:

```
01: require 'rubygems'
02: require 'spec'
03: require 'rpn'
04:
05: describe 'RPN' do
06:
07:   describe '1 2 +' do
08:     it 'should equal 3' do
09:       rpn(%w[1 2 +]).should == 3
10:     end
```

```
11:   end
12:
13:   describe '2 2 +' do
14:     it 'should equal 4' do
15:       rpn(%w[2 2 +]).should == 4
16:     end
17:   end
18:
19:   describe '2 2 + 2 +' do
20:     it 'should equal 6' do
21:       rpn(%w[2 2 + 2 +]).should == 6
22:     end
23:   end
24:
25:   describe '2 2 + 2 2 + +' do
26:     it 'should equal 8' do
27:       rpn(%w[2 2 + 2 2 + +]).should == 8
28:     end
29:   end
30:
31:   describe '2 2 + 2 2 + *' do
32:     it 'should equal 16' do
33:       rpn(%w[2 2 + 2 2 + *]).should == 16
34:     end
35:   end
36:
37:   describe '3 3 * 3 3 / -' do
38:     it 'should equal 8' do
39:       rpn(%w[3 3 * 3 3 / -]).should == 8
40:     end
41:   end
42:
43:   describe '1 2 + 3 4 + *' do
44:     it 'should equal 21' do
45:       rpn(%w[1 2 + 3 4 + *]).should == 21
46:     end
47:   end
48:
49:   describe '1 2 3 4 5 6 7 8 9 10 + + + + + + + + + +' do
50:     it 'should equal 55' do
51:       rpn(%w[1 2 3 4 5 6 7 8 9 10 + + + + + + + + + +]).should == 55
52:     end
53:   end
```

```
54:
55:   describe '1 2' do
56:     it 'should raise RuntimeError' do
57:       lambda{ rpn(%w[1 2]) }.should raise_error RuntimeError
58:     end
59:   end
60:
61: end
```

実装コード:

```
01: require 'stack'
02:
03: OPERATORS = %w[+ - * /]
04:
05: def rpn(expr)
06:   stack = Stack.new
07:   until expr.empty?
08:     v = expr.shift
09:     a = b = nil
10:     if OPERATORS.include?(v)
11:       b = stack.pop.to_i
12:       a = stack.pop.to_i
13:       op = v
14:       stack.push(a.__send__(op, b))
15:     else
16:       stack.push(v)
17:     end
18:   end
19:   raise 'invalid expression' unless stack.size == 1
20:   stack.pop
21: end
22:
23: def main
24:   puts rpn(ARGV)
25: end
26:
27: if __FILE__ == $0
28:   main
29: end
```