逆ポーランド電卓のつくりかた

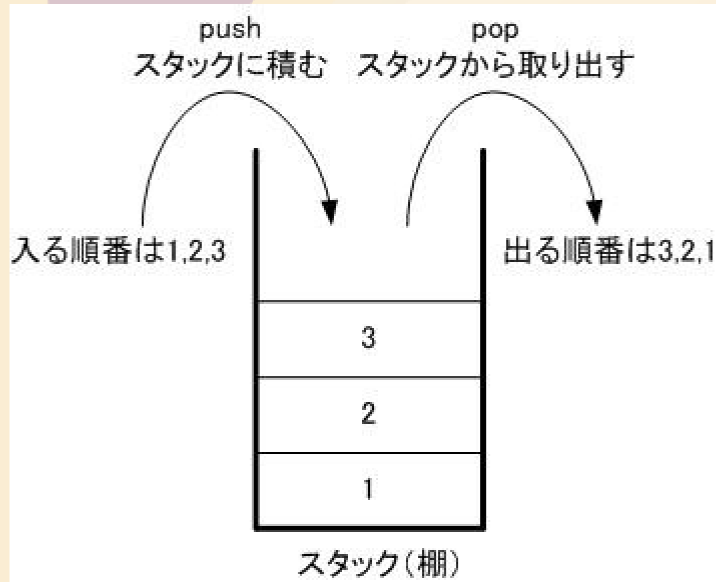
―― 脱ビギナ系 データ構造とアルゴリズム講座「StackとRPN」

わんくま同盟

$\varepsilon\pi\iota\sigma\tau\eta\mu\eta$ episteme@cppll.jp

Microsoft MVP for Visual C++ (2004-)

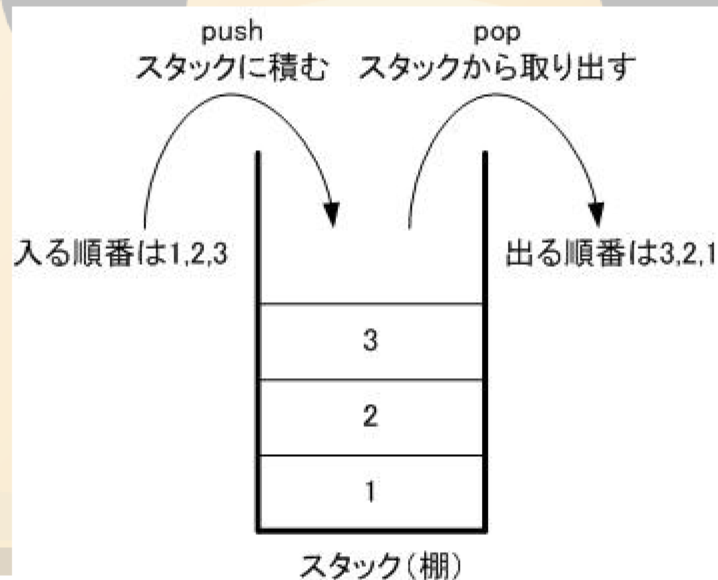
スタック(Stack)てなんぞ？ (1)



Pez(ペッツ) 知ってる?
遠足のおやつに持ってってたアレ。

スタック(Stack)てなんぞ? (2)

- またの名を First-In/Last-Out(FILO)バッファ
  - First-In : 最初に入れたのが
  - Last-Out : 最後に出てくる
- Pushで入れて Popで取り出す



逆ポーランド記法：計算式の表現のひとつ

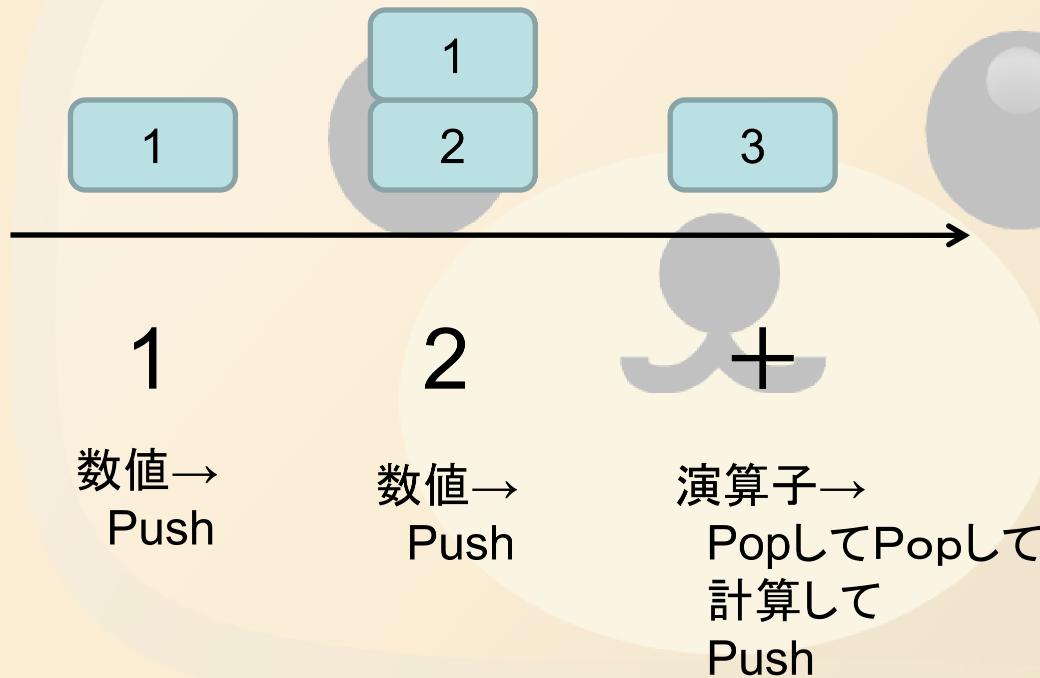
- RPN(Reverse Polish Notation)
  - Jan Łukasiewicz (ヤン・ウカシェヴィチ)
    - 1＋2 を (＋ 1 2) って書いてみた (LISPみたーい♪)
    - 別名「前置記法: prefix notation」
    - このひと論理学者でポーランド人
    - なのでポーランド記法(Polish Notation)
  - ポーランド記法をひっくり返したのがRPN
    - 1 + 2 → (+ 1 2) → 1 2 + って書いてみようよ
    - 別名「後置記法: postfix notation」

※ふつーの数式は「中置記法: infix notation」

RPN をStackで計算(評価)する

- RPN式を左から順に読み…
  - 数値なら
    - StackにPush
  - 演算子なら
    - StackからPopして
    - StackからPopして ← 単項演算なら省略
    - 演算子に応じた計算をして
    - 結果をStackにPush
- この操作を繰り返し、
  最後にStackに残ったのが答

RPN をStackで計算してみよう (1)

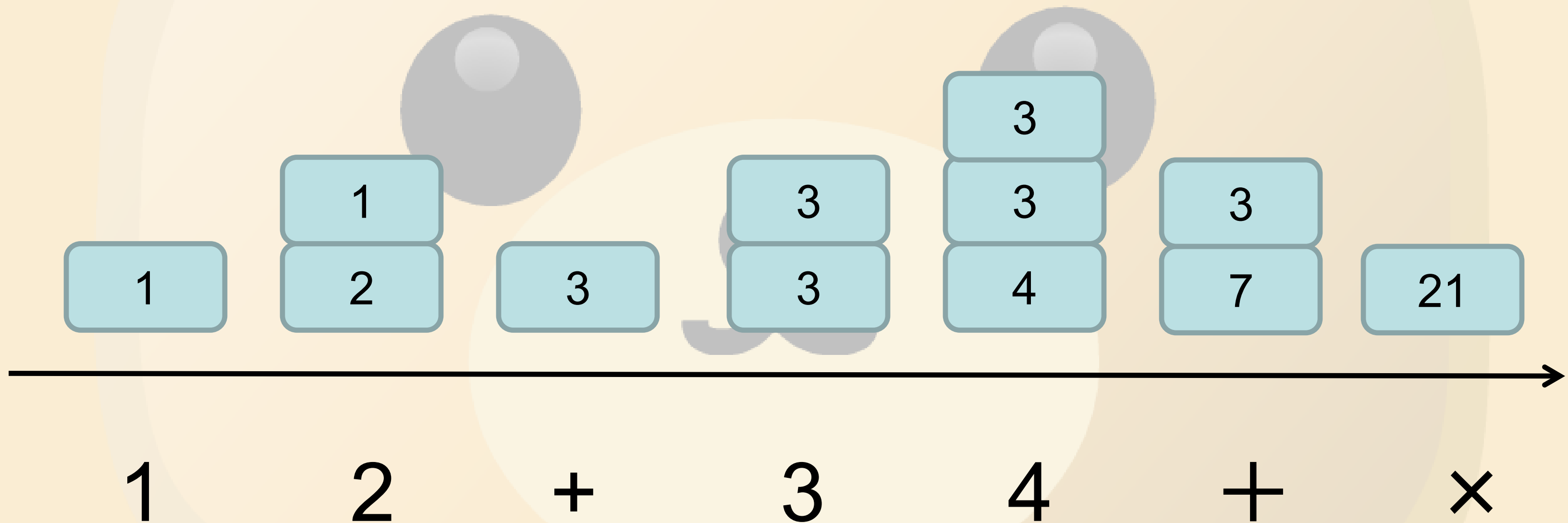


RPN をStackで計算してみよう (2)

(1 + 2) × (3+4) =

計算式 : 1 2 ＋ 3 4 ＋ ×

※ 最下部がStackの先頭



Push/Pop/計算するだけの簡単なお仕事です。

RPNの特徴

- すっごく単純
  - Stack一本で計算できる → 計算機向き
- 演算子に優先順位がない
  演算子が出てきたらすぐさま計算
  - カッコ がいらない
  - ＝ もない
- 日本語と同じ順序
  - (1 ＋ 2) ×(3 ＋ 4) =
  1 に 2 を たして、3 に 4 を たして、かける
  → 1 2 ＋ 3 4 ＋ ×

Stack二本で計算機を作る

- 命令Stackに式を積む
- ひとつずつPopしながら
  - 数値なら計算Stackに積む
  - 演算子なら計算StackからPopしてPopして、計算してPush

命令Stack

命令Stackが空になったとき、
計算Stackに残るのが答

計算Stack

RPN計算手順

Stackは.NET Frameworkに実装済

System.Collections.Stack

```
Sub Push(ByVal Item As Object)
Function Pop() As Object
```

System.Collections.Generic.Stack(Of T)

```
Sub Push(ByVal Item As T)
Function Pop() As T
```

今回、こいつらは使いません

Stackは可変長配列
System.Collections.IList,
System.Collections.Generic.IList(Of T)
で代替可

- Push
  - 末尾に追加

```
リスト.Add(item)
```

- Pop
  - 末尾から取り出し

末尾要素のインデクス

```
Dim item As T = リスト(リスト.Count-1)
リスト.RemoveAt(リスト.Count-1)
Return item
```

DEMO-1 ListBox.ItemsをStackとして利用する

- Public ReadOnly Property Items As ListBox.ObjectCollection
- Public Class ObjectCollection

  Implements IList, ICollection, IEnumerable

IListならStackがわりに使えるちゃん!

拡張メソッドで IList に Push/Pop を追加

```
Imports System.Runtime.CompilerServices
Imports System.Collections.Generic

Namespace Wankuma.Episteme
  Module ListExtensions

    <Extension()>  Public Sub PushBack(ByVal Self As IList, ByVal item As Object)
            Self.Add(item)
        End Sub

    <Extension()>  Public Function PopBack(ByVal Self As IList) As Object
            Dim result As T = Self(Self.Count - 1)
            Self.RemoveAt(Self.Count - 1)
            Return result
        End Function
        …
  End Module
End Namespace
```

Smart UI (利口なUI) アンチパターン

- 層状アーキテクチャの対極をなすアンチパターン。
- ビジネスロジックやデータアクセスのコードが、UIのコードと一緒になってしまっている、いわばスパゲッティな状態。
- 利口なUIと呼ぶのは、ビジネスロジックを含むすべての処理がUIの中で行なわれるから。
- 最もやっつけで手軽なやり方がこれなので、設計を何も考えないとこの状態に陥ってしまう。

System.Collections.ObjectModel

- Collection(Of T)

  Implements IList(Of T), ICollection(Of T), IEnumerable(Of T), IList, ICollection, IEnumerable

- KeyedCollection(Of TKey, TItem)

  Inherits Collection(Of TItem)

IListならStackがわりに使えんぢゃん!

要素の追加/削除/変更を再定義可
→ Model/Viewの分離

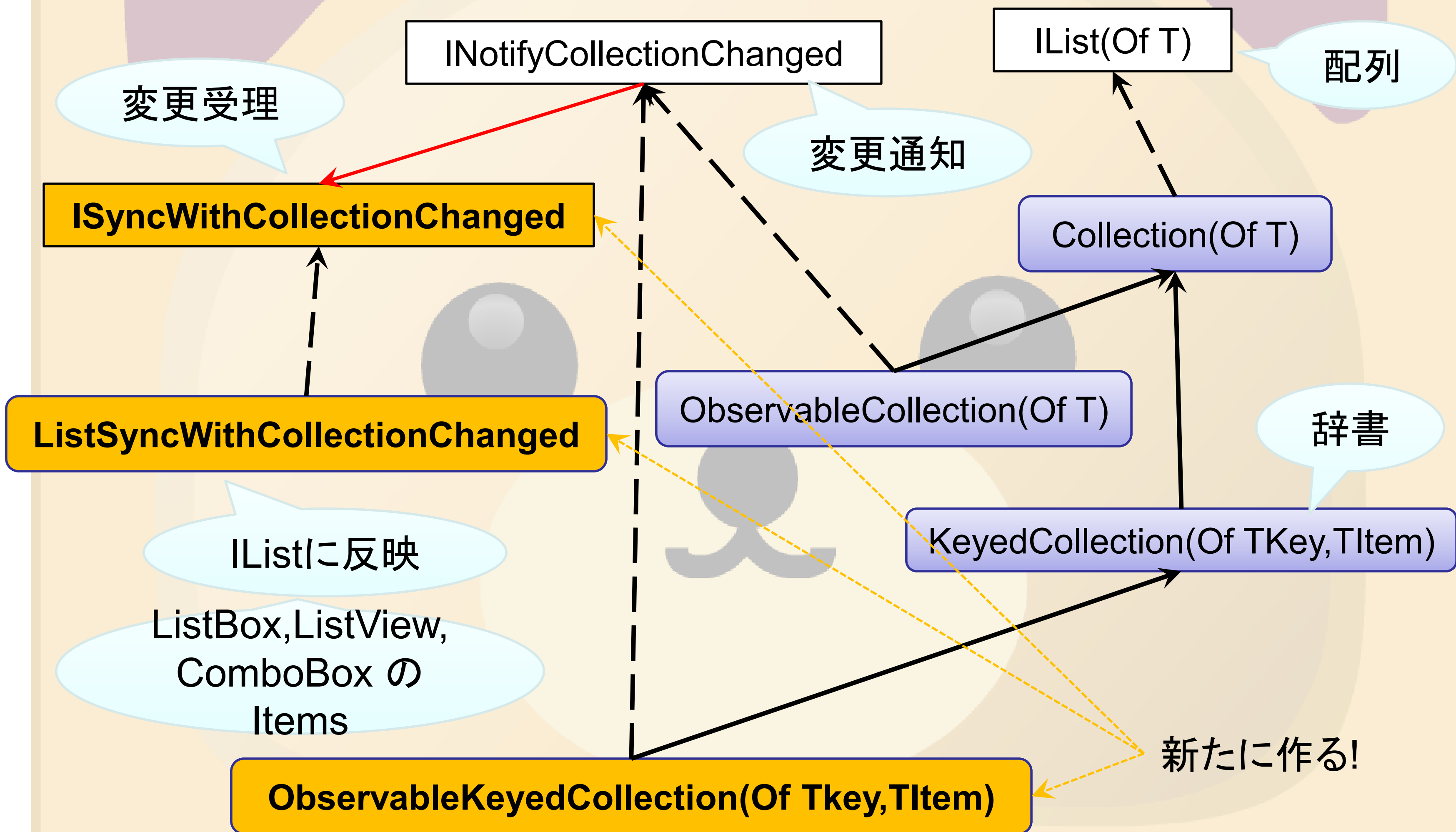
```
Public Class ListCollection(Of T)
    Inherits Collection(Of T)

    Public Target As ListBox

    Protected Overrides Sub InsertItem(ByVal index As Integer,  _
                                                        ByVal newItem As T)
        MyBase.InsertItem(index, newItem)  ‘ もともとの InsertItem を呼ぶ
        Target.Items.Insert(index,newItem) ‘ ListBox にも Insert する
    End Sub

     SetItem,  RemoveItem, ClearItems についても同様。

End Class
```


System.Collections.ObjectModel をタネにして…
INotifyCollectionChanged
IList(Of T)
配列
変更受理
変更通知
ISyncWithCollectionChanged
Collection(Of T)
ListSyncWithCollectionChanged
ObservableCollection(Of T)
辞書
KeyedCollection(Of TKey,TItem)
IListに反映
ListBox,ListView,
ComboBox の
Items
新たに作る!
ObservableKeyedCollection(Of Tkey,TItem)

アナタはいまどこ?

- Wizardが吐くものを使う
- ライブラリから探して使う
- ちょいといじって使う
  (派生/拡張メソッド/ヘルパ)
- なければ作る
  - アプリべったり(作り捨て/使い捨て)なら簡単
  - ツブシの効くものはそれなりのスキルが必要