# template vs. generics

―― 総称型のからくりと功罪 ――

2006.12.09
わんくま同盟

επιστημη
Microsoft MVP : Visual C++
episteme@cppll.jp

総称性って･･･?

ぶっちゃけていえば

# 十把ひとからげ

# 毎度おなじみStack

- string の Stack
- int の Stack
- long の Stack
- Button の Stack
- ……

**ぜんぶまとめて T の Stack**

→ T は使うときに決めることにする

# 総称性の言語サポート

- C#, VB.Net では generics
- C++ では template
- C++/CLI では template と generics

※ STL/CLRはこのタイプのManaged コンテナ

# VB による T の Stack (generics)

```
Public Class Stack(Of T)

    Private data_() As T
    Private size_ As Integer

    Public Sub New(ByVal n As Integer)
        ReDim data_(n)
        size_ = 0
    End Sub

    Public ReadOnly Property Empty() As Boolean
        Get
            Return size_ = 0
        End Get
    End Property

    Public Sub Push(ByVal item As T)
        data_(size_) = item
        size_ = size_ + 1
    End Sub

    Public Function Pop() As T
        size_ = size_ - 1
        Return data_(size_)
    End Function

End Class
```

# C# による T の Stack (generics)

```
public class Stack<T> {

    private T[] data_;
    private int size_;

    public Stack(int n) {
        data_ = new T[n];
        size_ = 0;
    }

    public bool Empty {  get { return size_ == 0; }  }

    public void Push(T item) {  data_[size_++] = item; }

    public T Pop() {  return data_[--size_]; }

}
```

# C++/CLI による T の Stack (generics)

```
generic<typename T>
public ref class Stack {
private:
    array<T>^ data_;
    int size_;

public:
    Stack(int n) {
        data_ = gcnew array<T>(n);
        size_ = 0;
    }

    property bool Empty { bool get() { return size_ == 0; } }

    void Push(T item) { data_[size_++] = item; }

    T Pop() { return data_[--size_]; }
};
```

# C++/CLI による T の Stack (template)

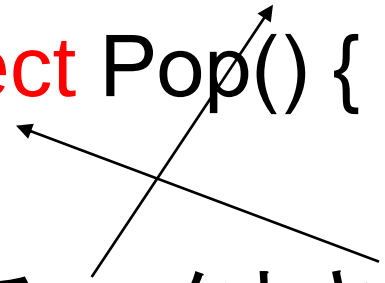
```
template<typename T>
public ref class Stack {
private:
    array<T>^ data_;
    int size_;

public:
    Stack(int n) {
      data_ = gcnew array<T>(n);
      size_ = 0;
    }

    property bool Empty { bool get() { return size_ == 0; } }

    void Push(T item) { data_[size_++] = item; }

    T Pop() { return data_[--size_]; }
};
```

# generics と template の違い : generics

generics

- object とみなしてコンパイル → ライブラリ可

```
class Stack {
  private object[] data_;
  public void Push(object item) { … }
  public object Pop() { … }
}
```



- 使う側で型のチェックとキャストを行う

※ 値型の場合、型ごとに生成する(実行時)

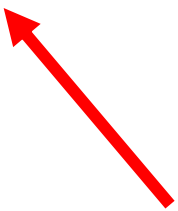
# generic と template の違い : template

template

- 使われるまでコンパイルしない/できない
  → ソースによるライブラリ
  ( .lib や .dll は *ありえねー* )
- 与える型ごとに異なる実装を生成
  → 膨張/爆発の危険性アリ
- コンパイル時につじつまが合えばいい
  → めっさ自由!

# generic の制約

```
// C#
public class PrintUtil<T> {
  private TextWriter writer_;
  public void Print(T item) {
    item.PrintOn(writer_);
  }
}
```

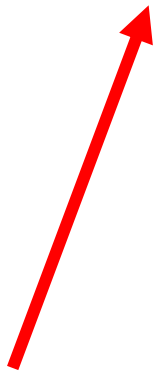


エラー : T に PrintOn なんてメソッドねーよ!

(だってobjectだと思ってんだもん)

# generic の制約

```
public interface IPrint {
  void PrintOn(TextWriter writer);
}

public class PrintUtil<T> where T : IPrint {
  private TextWriter writer_;
  public void Print(T item) {
    item.PrintOn(writer_);
  }
}
```



**※ IPrint じゃない T はコンパイル・エラー**

# コード、読んでみる?