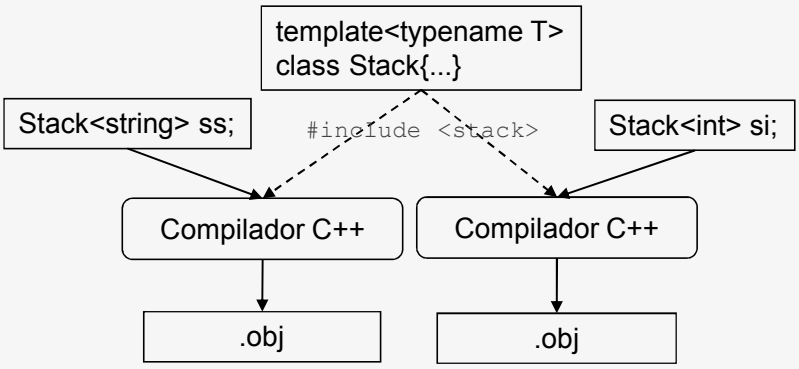
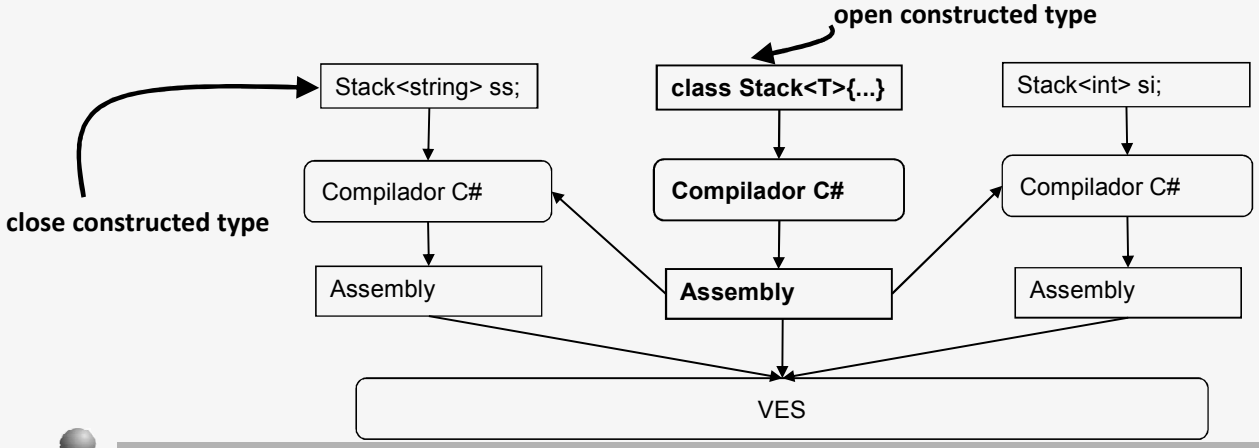
Genéricos ≠ *Templates*

- *Templates*
  - Não originam directamente código intermédio nem nativo
    - por cada instanciação de um *template*, a sua definição é combinada com a dos tipos-parâmetro para gerar código nativo específico
  - O compilador conhece as interfaces dos tipos-parâmetro usados na instanciação do *template*
    - verificações realizadas durante a compilação das várias instâncias



Genéricos ≠ *Templates*

- *Genéricos*
  - O código genérico é compilado para IL, que fica com informação genérica de tipos
    - representação intermédia ainda é genérica
    - genérico é usável na forma compilada CIL
  - O compilador não conhece a interface dos tipos que vão ser usados na instanciação do genérico
    - limita as acções realizáveis sobre objectos dos tipos-parâmetro



Genéricos .NET e Genéricos Java

- Em Java, os tipos e métodos genéricos passam por um processo de *type erasure*
  - Processo que traduz uma classe parametrizada numa outra não parametrizada mais informação na metadata que a classe originalmente era genérica e os correspondentes parâmetros

```
class Generic<T> {
   T value;
   Generic(T value) {
      this.value=value;
   }

   T getValue() {
      return value;
   }

   void setValue(T t) {
      value=t;
   }
}
```

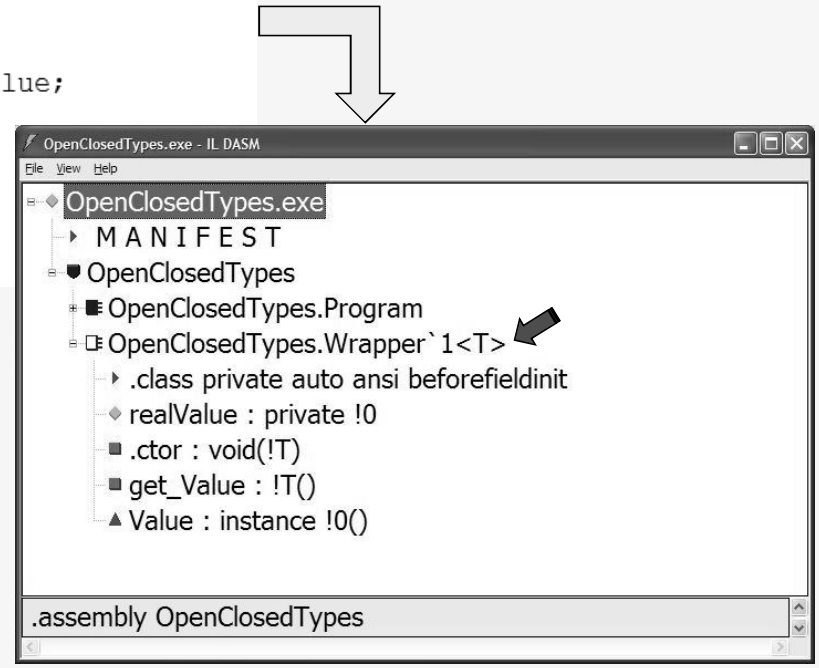
Erasure

```
class GenericObj {
   Object  value;
   GenericObj(Object value) {
      this.value=value;
   }

   Object  getValue() {
      return value;
   }

   void setValue(Object t) {
      value=t;
   }
}
```


Genéricos ≠ *Templates*

- Templates C++
  - Verificação e instanciação em tempo de compilação na utilização
  - Expansão de código
- Genéricos Java
  - Verificação em tempo de compilação na declaração
  - Uma única instanciação (type erasure)
  - Partilha de código
- Genéricos .NET
  - Verificação em tempo de compilação na declaração
  - *Dynamic Code Expansion and Sharing*
    - Técnica levada a cabo pelo JIT e que mistura as ideias de C++ e Java
    - expansão de código nativo à medida
    - partilha de código nativo quando argumentos são tipos referência

```
namespace OpenClosedTypes
{
    class Wrapper<T>
    {
        private T realValue;
        public Wrapper(T realValue)
        {
            this.realValue = realValue;
        }
        public T Value{
            get { return realValue;
        }
    }
```




```
static void closedType()
{
    Wrapper<int> w = new Wrapper<int>(1);
    // ...
}
```