Generik

Mittwoch, 1. Februar 2017

14:52

* Oft müssen ähnliche Operationen / Funktionalitäten für unterschiedliche Datentypen bereitgestellt werden
* Um nicht gleichen Code mehrfach zu schreiben nutzt man Platzhalter (template) für Datentypen
* Beispiel: List<T>

**generische Klasse definieren**

|  |  |
| --- | --- |
| **generische Klasse benutzen**   |  | | --- | |  | |

|  |
| --- |
| Generic types will be provided by the compiler |

>

Ähnlich für:

* struct
* interfaces
* delegates

**Generische Methoden**

Generik funktioniert genauso für Methoden

|  |
| --- |
| class MyClass {  public static void Swap<T>(ref T l, ref T r)  {  T temp = r;  r = l;  l = temp;  } } |

|  |
| --- |
| int a = 3; int b = 4; MyClass.Swap(ref a, ref b); |

**Template Constraints**

Constraints = Vorgaben an die Templates

Interface Constraint

|  |
| --- |
| public static void Swap<T>(ref T l, ref T r) where T : IComparable  {  if (l.CompareTo(r) < 0) {  T temp = r;  r = l;  l = temp;  }  } |

Ebenso für Klassen

class MyGenericClass<T> where T : IComparable

Vererbungs - Constraints

class MyGenericClass<T> where T : Random

class MyGenericClass<T, V> where T : List<V>

class MyGenericClass<T, V> where T : List<V> where V : Random

Instanzierbarkeits - Constraint

T CreateNew<T>() where T : new()  
{  
 return new T();*//Works*  
}

|  |
| --- |
|  |