Generische Klassen (C#-Programmierhandbuch)

20.07.2015 7 Minuten Lesedauer Beitragende

In diesem Artikel

Generische Klassen kapseln Operationen, die nicht spezifisch für einen bestimmten Datentyp sind. Generische Klassen werden am häufigsten bei Auflistungen verwendet, z.B. bei verknüpften Listen, Hashtabellen, Stapeln, Warteschlangen, Strukturen usw. Vorgänge wie das Hinzufügen oder Entfernen von Elementen aus der Auflistung werden nahezu auf die gleiche Art und Weise ausgeführt, unabhängig vom Typ der gespeicherten Daten.

Für die meisten Szenarios, die Auflistungsklassen erfordern, wird empfohlen, die in der Klassenbibliothek von .NET bereitgestellten Auflistungsklassen zu verwenden. Weitere Informationen zur Verwendung dieser Klassen finden Sie unter Generic Collections in .NET (Generische Auflistungen in .NET).

In der Regel erstellen Sie generische Klassen, indem Sie von einer vorhandenen konkreten Klasse ausgehen und Typen in Typparameter ändern (einen nach dem anderen), bis das optimale Verhältnis zwischen Verallgemeinerung und Verwendbarkeit gefunden ist. Wenn Sie eigene generische Klassen erstellen möchten, sollten Sie die folgenden wichtigen Aspekte beachten:

- Welche Typen sollen in Typparameter verallgemeinert werden.

  Im Allgemeinen gilt: Je mehr Typen Sie parametrisieren können, umso flexibler und besser wiederverwendbar ist Ihr Code. Jedoch kann ein Zuviel an Verallgemeinerung zu Code führen, der von anderen Entwicklern nur schwer gelesen oder verstanden wird.

- Welche Einschränkungen sollen ggf. auf die Typparameter angewendet werden (siehe Constraints on Type Parameters (Einschränkungen für Typparameter)).

  Es empfiehlt sich, alle Einschränkungen anzuwenden, die maximal möglich sind, und bei denen Sie dennoch sämtliche Typen, die Sie behandeln müssen, auch behandeln können. Wenn Sie zum Beispiel wissen, dass die generische Klasse nur mit Referenztypen verwendet werden soll, dann können Sie die Klasseneinschränkung anwenden. Dadurch wird verhindert, dass die Klasse unbeabsichtigt mit Werttypen verwendet wird. Gleichzeitig können Sie den Operator `as` für `T` verwenden und nach NULL-Werten suchen.

- Ob das generische Verhalten in Basisklassen und Unterklassen zerlegt werden soll.

Da generische Klassen als Basisklassen dienen können, sind hier beim Entwurf die gleichen Aspekte zu berücksichtigen wie bei nicht generischen Klassen. Weitere Informationen bieten die Regeln zum Erben von generischen Basisklassen weiter unten in diesem Thema.

- Ob eine oder mehrere generische Schnittstellen implementiert werden soll.

  Wenn Sie beispielsweise eine Klasse entwerfen, die zum Erstellen von Elementen in einer generisch basierten Auflistung verwendet wird, müssen Sie unter Umständen eine Schnittstelle implementieren, z.B. IComparable<T>, wobei `T` der Typ Ihrer Klasse ist.

Ein Beispiel für eine einfache generische Klasse finden Sie unter Introduction to Generics (Einführung in Generika).

Die Regeln für Einschränkungen und Typparameter haben eine Reihe von Auswirkungen auf das Verhalten einer generischen Klasse, besonders im Hinblick auf Vererbung und Zugriff der Member. Bevor Sie fortfahren, sollten Sie einige Begriffe verstehen. Bei einer generischen Klasse kann der Clientcode `Node<T>,` auf die Klasse verweisen, indem er ein Typargument angibt, um einen geschlossenen konstruierten Typ (`Node<int>`) zu erstellen. Die Alternative besteht darin, den Typparameter nicht anzugeben, z.B. wenn Sie eine generische Basisklasse angeben, um einen offenen konstruierten Typ (`Node<T>`) zu erstellen. Generische Klassen können von konkreten, geschlossenen konstruierten oder offenen konstruierten Basisklassen erben:

C# Kopieren

```
class BaseNode { }
class BaseNodeGeneric<T> { }

// concrete type
class NodeConcrete<T> : BaseNode { }

//closed constructed type
class NodeClosed<T> : BaseNodeGeneric<int> { }

//open constructed type
class NodeOpen<T> : BaseNodeGeneric<T> { }
```

Nicht generische, also konkrete Klassen können von geschlossenen konstruierten Basisklassen erben, aber nicht von offenen konstruierten Klassen oder Typparametern, denn während der Laufzeit ist es für den Clientcode nicht möglich, das erforderliche Typargument bereitzustellen, das zum Instanziieren der Basisklasse benötigt wird.

C# Kopieren

```
//No error
class Node1 : BaseNodeGeneric<int> { }
```

```
//Generates an error
//class Node2 : BaseNodeGeneric<T> {}

//Generates an error
//class Node3 : T {}
```

Generische Klassen, die von offenen konstruierten Typen erben, müssen für sämtliche Basisklassen-Typparameter, die von der erbenden Klasse nicht verwendet werden, Typargumente bereitstellen. Der folgende Code stellt ein Beispiel dafür dar:

C# Kopieren

```
class BaseNodeMultiple<T, U> { }

//No error
class Node4<T> : BaseNodeMultiple<T, int> { }

//No error
class Node5<T, U> : BaseNodeMultiple<T, U> { }

//Generates an error
//class Node6<T> : BaseNodeMultiple<T, U> {}
```

Generische Klassen, die von offenen konstruierten Typen erben, müssen Einschränkungen angeben, die den Einschränkungen des Basistyps übergeordnet sind oder diese implizieren:

C# Kopieren

```
class NodeItem<T> where T : System.IComparable<T>, new() { }
class SpecialNodeItem<T> : NodeItem<T> where T : System.IComparable<T>, new() { }
```

Generische Typen können mehrere Typparameter und Einschränkungen wie folgt verwenden:

C# Kopieren

```
class SuperKeyType<K, V, U>
    where U : System.IComparable<U>
    where V : new()
{ }
```

Offene konstruierte und geschlossene konstruierte Typen können als Methodenparameter verwendet werden:

C# Kopieren

```
void Swap<T>(List<T> list1, List<T> list2)
{
    //code to swap items
```

```
}

void Swap(List<int> list1, List<int> list2)
{
    //code to swap items
}
```

Wenn eine generische Klasse eine Schnittstelle implementiert, können alle Instanzen dieser Klasse in diese Schnittstelle umgewandelt werden.

Generische Klassen sind unveränderlich. Wenn also ein Eingabeparameter eine `List<BaseClass>` angibt, wird Ihnen ein Kompilierungsfehler angezeigt, falls Sie versuchen, eine `List<DerivedClass>` bereitzustellen.

Siehe auch

System.Collections.Generic
C#-Programmierhandbuch
Generika
Saving the State of Enumerators (Speichern des Zustands von Enumeratoren)
An Inheritance Puzzle, Part One (Ein Vererbungs-Puzzle, Teil 1)