

Generics mit "Zirkulärer Typreferenz"

Source:

http://www.tech-archive.net/Archive/German/Entwicklung/microsoft.public.de_german.entwickler.dotnet.csharp/2008

- *From:* Thomas Schremser <vbng01.20.tschremser@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 22 Jul 2008 14:57:47 +0200
-

Hallo allerseits!

In meinem Programm möchte ich Objekte als Weak Reference in einem generischen Cache speichern. Diese Objekte sollen folgendes generische Interface implementieren:

```
internal interface ICacheable<T> where T : class
{
    Cache<T> Cache
    {
        get;
        set;
    }

    int Key
    {
        get;
        set;
    }
}
```

Über den Key wird das Objekt identifiziert, der Verweis auf den Cache ermöglicht es, wenn das Objekt vom GC entsorgt wird, den Verweis aus dem Cache zu löschen:

```
public class SomeObject : ICacheable<SomeObject>
{
    ~SomeObject()
    {
        ((ICacheable<SomeObject>)this).Cache.Remove(
            ((ICacheable<SomeObject>)this).Key);
    }

    // ICacheable<SomeObject>-Member
}
```

Der Cache dazu sieht – vereinfacht – so aus:

Generics mit "Zirkulärer Typreferenz"

```
internal class Cache<T> where T : class
{
private Dictionary<int, WeakReference> mItems;

public void Add(ICacheable<T> item)
{
mItems.Add(item.Key, new WeakReference(item));
item.Cache = this;
}

public bool Remove(int key)
{
return mItems.Remove(key);
}
}
```

Grundsätzlich funktioniert das auch recht gut. Nur würde ich bei der Definition von `ICacheable<T>` und `Cache<T>` noch folgendes sicherstellen:

? Bei `ICacheable<T>` soll T immer vom Typ der Klasse sein, die dieses Interface implementiert. Folgendes soll also nicht gehen:

```
class SomeObject : ICacheable<SomeOtherObject>
```

? `Cache<T>` soll als Typ nur Klassen zulassen, die auch `ICacheable<T>` implementieren.

```
class Cache<T> where T : class, ICacheable<T>
```

führt leider zu einem Fehler.

Gibt es eine Möglichkeit, das zu realisieren?

TIA

Grüße
Thomas

--

Any problem in computer science can be solved with another layer of indirection. But that usually will create another problem.

David Wheeler

.