Iterator

Autor: Philipp Haller, 7AKIF

Datei: Iterator.md

Namespace: System.Collections

Classification: Behavioural Design Pattern

Interfaces: IEnumerable, IEnumerator, …

Als Iterator wird ein Zeiger bezeichnet, welcher das sequentielle Durchlaufen einer komplexen Datenstruktur ermöglicht, ohne die internen Details preiszugeben. Ein Beispiel für die Funktionsweise des Iterators ist die foreach-Schleife, mit der über die Element eines Arrays oder einer Collection iteriert werden kann.

```
foreach (int x in list)
        Console.WriteLine(x);
```

Dieses Codebeispiel wird vom Compiler folgendermaßen übersetzt:

```
IEnumerator e=list.GetEnumerator();
while (e.MoveNext())
        Console.WriteLine(e.Current)
```

Die Methode GetEnumerator liefert einen Zeiger des Typs IEnumerator, welcher über die Methode MoveNext() und das Property Current verfügt. Mit MoveNext() wird der Zeiger zum nächsten Element der Liste bewegt und Current liefert das Element der Liste. Wird der Zeiger mit MoveNext über das letzte Element der Liste hinausbewegt, erhält man als Rückgabewert false, bis er mit der Reset() Methode wieder auf das erste Element der Auflistung gesetzt wird. GetEnumerator ist eine Methode des Interfaces IEnumerable. Somit können alle Objekte durchlaufen werden, welche dieses Interface implementieren. Ergänzend wird das Interface IEnumerable <T> für generische Collections angeboten. Um eine benutzerdefinierte Auszählung durchlaufen zu können, müssen beide Interfaces implementiert werden.

```
namespace System.Collections
{
   public interface IEnumerable
   {
      IEnumerator GetEnumerator();
   }
}
```

```
namespace System.Collections
{
   public interface IEnumerator
   {
      object? Current { get; }

      bool MoveNext();

      void Reset();
   }
}
```

Folgendes Codebeispiel veranschaulicht die Implementierung:

```
public static void Main(string[] args) {

   foreach (var coffee in new CoffeeCollection()) {
      Console.WriteLine(coffee);
   }
}

public class CoffeeCollection : IEnumerable {
   private CoffeeEnumerator enumerator;

   public CoffeeCollection() {
      enumerator = new CoffeeEnumerator();
   }

   public IEnumerator GetEnumerator() {
      return enumerator;
   }

   public class CoffeeEnumerator : IEnumerator {
      string[] beverages = new string[3] { "espresso", "macchiato", "latte" };
      int currentIndex = -1;

      public object Current {
         get {
            return beverages[currentIndex];
         }
      }

      public bool MoveNext() {
         currentIndex++;

         if (currentIndex < beverages.Length) {
            return true;
         }

         return false;
      }
```

```
    public void Reset() {
      currentIndex = 0;
    }
  }
}
```

Da die Implementierung von IEnumerator aufwendig sein kann, gibt es eine weitere und einfachere Möglichkeit mit dem IEnumerator eine Menge von Elementen zu durchlaufen. Mit Hilfe der yield-Anweisung wird der gelieferte Wert an eine Folge von Ergebniswerten angehängt, welche später mit einer foreach-Schleife durchlaufen werden kann.

```
class Domain
{
  string name1 = “dotnet.jku.at”;
  string name2 = “csharp.jku.at”;
  string name3 = “www.ssw.uni-linz.ac.at”;
  …
  public  IEnumerator GetEnumerator<string> GetEnumerator() {
    yield return name1;
    yield return name2;
    yield return name3;
  }
}
```

```
Domain domain = new Domain();
foreach (string name in domain) Console.WriteLine(name);
```