ITERATOR PATTERN IN C#

COLLECTIONS

WAS IST EINE COLLECTION?

- Funktionen mit unterschiedlichen Graden
- Hinzufügen von Elementen unterstützen (optional)
- Entfernen von Elementen unterstützen (optional)
- Indizierung von Elementen unterstützen (optional)
- Entfernen von allen Elementen (optional)
- **Möglichkeit zur Aufzählung der einzelnen Elemente unterstützen**

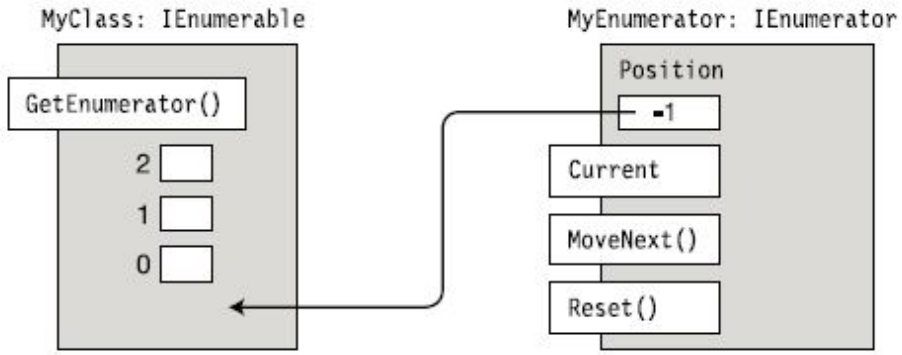
AUFZÄHLUNG

.Net bietet für alle Sammlungen einen Aufzählmechanismus

```
1 List<int> nums = new List<int> { 2, 3, 7, 5, 8, 12, 18 };
2
3 foreach (int num in nums)
4 {
5     Console.WriteLine(num);
6 }
```

WIE FUNKTIONIERT DIE AUFZÄHLUNG

mit dem **Iterator Pattern**



ITERATOR PATTERN

Eine Collection, wie z.B. eine Liste, soll eine Möglichkeit bieten, auf ihre Elemente sequentiell (d.h. eines nach dem anderen) zuzugreifen, ohne dass die interne Struktur der Collection bekannt sein muss.

ITERATOR PATTERN

- Der Zugriff kann dabei z.B. durch folgende Methoden erfolgen (in C# definiert im Interface IEnumerator):
  - `Reset()`: Positionszeiger vor Beginn der Liste stellen
  - `bool MoveNext()`: Auf nächstes Listenelement gehen (true wenn vorh.)
  - `Current`: Aktuelles Element zurückgeben

ITERATOR PATTERN

- Eine naheliegende Idee wäre es, diese Methoden direkt in der Collection (z.B. ListOfPupil) zu implementieren. Dies würde jedoch u.a. immer nur eine Traversierung (= Durchlaufen der Liste) gleichzeitig erlauben, da die Liste nur einen Positionszeiger gleichzeitig führen würde (welcher durch MoveNext um eins weitergesetzt wird.)

ITERATOR PATTERN

- Mit dem Iterator Pattern wird die Logik für das Durchlaufen nicht im Collection-Objekt selbst implementiert, sondern in einem eigenen Iterator-Objekt (C# Enumerator). Dadurch können mehrere Enumeratoren für ein und dasselbe Listen-Objekt erstellt werden. D.h. es können gleichzeitig mehrere Traversierungen erfolgen.

ZWEI INTERFACES

- IEnumerator: Interface für die Iterator
  - `IEnumerator GetEnumerator()`: Methode, die den Iterator zurückgibt
- IEnumerable: Interface für den Collection
  - `Object Current`: Aktuelles Element zurückgeben
  - `bool MoveNext()`: Auf nächstes Listenelement gehen (true wenn vorh.)
  - `Reset()`: Positionszeiger vor Beginn der Liste stellen

BEISPIEL: EIGENE GENERISCHE LINKED LIST

MYLIST.CS

```
1 public IEnumerator<T> GetEnumerator()
2 {
3     return new MyEnum<T>(head);
4 }
5
6 IEnumerator IEnumerable.GetEnumerator()
7 {
8     return GetEnumerator();
9 }
```

MYENUM.CS

VARIABLES

```
public class MyEnum<T> : IEnumerator<T>
{
    private Node<T> head;
    private Node<T> current;
```

CONSTRUCTOR

```
public MyEnum(Node<T> head)
{
    this.head = head;
    current = null;
}
```

CURRENT

```
public T Current
{
    get { return current.Value; }
}

object IEnumerator.Current
{
    get { return Current; }
}
```

MOVENEXT

```
public bool MoveNext()
{
    if (current == null)
    {
        current = head;
    }
    else
    {
        current = current.Next;
    }
    return current != null;
}
```

RESET

```
public void Reset()
{
    current = null;
}
```

VERWENDUNG

```
MyList<int> list = new MyList<int>();

list.Add(1);
list.Add(2);
list.Add(3);

foreach (int i in list)
{
    Console.WriteLine(i);
}
```

OUTPUT:

```
1
2
3
```