



- erlauben es, die Elemente einer Kollektion nacheinander abzurufen
- werden meist implizit im Rahmen einer **foreach**-Schleife verwendet:

```
int[] arr_array = new int[] { 0, 1, 2, 3, 4, 5, }; // array creation
```

```
foreach (int i in arr_array) // foreach loop begins and runs till last item  
{  
    Console.WriteLine(i);  
}
```

- Realisierungs-Varianten:
 - benannte Iteratoren
 - Implementierung von `IEnumerable` (vorhanden für `List<T>`, Arrays, ... – nicht für selbsterstellte Listen)

Benannte Iteratoren (Iteratormethode)

Die Iteratormethode verwendet die [yield return](#)-Anweisung, um jedes Element einzeln nacheinander zurückzugeben.

„yield return“ bewahrt die aktuelle Codeposition plus aller lokalen Variablen wird auf dem Stack auf. Wenn die Iteratorfunktion das nächste Mal aufgerufen wird, wird die Ausführung von dieser Position gestartet.

- In dieser Methode darf kein return ohne yield verwendet werden
- „yield break“ bricht die Speicherung in einem Zyklus ab

```
static void Main()
{
    foreach (int number in NextNumber() )
    {
        Console.Write(number.ToString() + " ");
    } // Output: 3 5 8
}
```

```
public static System.Collections.IEnumerable<T> NextNumber()
{
    yield return 3;
    yield return 5;
    yield return 8;
}
```

Benannte Iteratoren (Iteratormethode)

```
static void Main()
{
    foreach (int number in EvenSequence(3, 12) )
    {
        Console.WriteLine(number.ToString() + " ");
    } // Output: 4 6 8 10 12
}
```

```
public static System.Collections.Generic.IEnumerable<int> EvenSequence(int firstNumber, int lastNumber)
{
    for (int number = firstNumber; number <= lastNumber; number++)
    {
        if (number % 2 == 0)
        {
            yield return number;
        }
    }
}
```



Iteratoren: aufzählbare Klasse

```
public class CPerson // Simple business object.
```

```
{  
    public CPerson(string fName, string lName)  
    {  
        FirstName = fName;  
        LastName = lName;  
    }  
  
    public string FirstName {get; set; };  
    public string LastName {get; set; };  
}
```

```
using System.Collections;
```

```
class App  
{  
    static void Main()  
    {  
        CPerson[] peopleArray = new CPerson[3] { new CPerson("John", "Smith"),  
                                                    new CPerson("Jim", "Johnson"),  
                                                    new CPerson("Sue", "Rabon")  
                                                    };  
        CPeople peopleList = new CPeople(peopleArray);  
  
        foreach (CPerson p in peopleList) Console.WriteLine(p.firstName + " " + p.lastName);  
    }  
}
```

```
// This code produces output similar to the following:
```

```
John Smith  
Jim Johnson  
Sue Rabon
```

Iteratoren: aufzählbare Klasse

```
public class CPerson // Simple business object.
```

```
{
    public CPerson(string fName, string lName)
    {
        FirstName = fName;
        LastName = lName;
    }
}
```

```
public string FirstName {get; set; }
public string LastName {get; set; }
}
```

```
public class CPeople
```

```
{
    private CPerson[] people;           // array of persons

    public CPeople (CPerson[] pArray) // ctor copies values
    {
        people = new CPerson[pArray.Length];
        for (int i = 0; i < pArray.Length; i++)    people[i] = pArray[i];
    }
}
```

```
using System.Collections;
```

```
class App
```

```
{
    static void Main()
    {
        CPerson[] peopleArray = new CPerson[3]
        { new CPerson("John", "Smith"),
          new CPerson("Jim", "Johnson"),
          new CPerson("Sue", "Rabon")
        };

        CPeople peopleList = new CPeople(peopleArray);
        foreach (CPerson p in peopleList)
            Console.WriteLine(p.FirstName + " " + p.LastName);
    }
}
```

CPeople

CPerson[]

John Smith

Jim Johnson

Sue Rabon

Program.cs(37, 28): [CS1579] Eine foreach-Anweisung kann nicht für Variablen vom Typ "App.CPeople" verwendet werden, weil "App.CPeople" keine öffentliche Instanz- oder Erweiterungsdefinition für "GetEnumerator" enthält.

Iteratoren: aufzählbare Klasse

```
public interface IEnumerable<out T> : IEnumerable // generische Variante
{
    IEnumerator<T> GetEnumerator(); // Interfacemethode
}
```

zu implementieren: Methoden `IEnumerator<T> GetEnumerator()`

plus: Eigenen Typ (Klasse) von `IEnumerator<T>` ableiten

CPeople: IEnumerable

CPerson[]

John Smith

Jim Johnson

Sue Rabon

Wir erweitern die eigene Klasse CPeople, die ihre Elemente des privaten Arrays namens data in einem sortierten Zustand hält, um die vorgeschriebene Enumeratomethode:

```
public IEnumerator<T> GetEnumerator()
{
    for (int i = 0; i < size; i++) yield return data[i];
    // yield return gibt aktuelles Datenelement zurück und
    // speichert aktuelle (Iterator-)Position in die Laufzeitumgebung für nächsten Aufruf
}
```

Iteratoren: aufzählbare Klasse

```
public class CPeople
{
    private CPerson[] people;           // array of persons
    public CPeople (CPerson[] pArray) // ctor copies values
    {
        people = new CPerson[pArray.Length];
        for (int i = 0; i < pArray.Length; i++) { people[i] = pArray[i]; }
    }

    // Implementation for the GetEnumerator method.
    public IEnumerator<CPerson> GetEnumerator()
    {
        for (int i=0; i<people.Length; i++)
            yield return people[i]; // Konvertierung von CPerson[] in
                                     // IEnumerator<CPerson> muss möglich sein
    }
}
```

Wenn der Compiler einen generischen Iterator erkennt, generiert er automatisch die Methoden *MoveNext()* und *Dispose()* und die Property *Current()*.

Ansehen nach erfolgreicher Übersetzung mit:

ildasm: Intermediate Language Disassembler, öffnen mit:

->Tools ->IL Viewer ->Navigate ->Navigate to -> IL Code , Mode umschalten auf „Low level C#“

Iteratoren: aufzählbare Klasse

jetzt mit IEnumerable-Interface

// Collection of CPerson objects. This class implements
// IEnumerable so that it can be used with ForEach syntax.

```
public class CPeople : IEnumerable<CPerson>
{
    private CPerson[] people;          // array of persons
    public CPeople (CPerson[] pArray) // ctor copies values
    {
        people = new CPerson[pArray.Length];
        for (int i = 0; i < pArray.Length; i++) { people[i] = pArray[i]; }
    }

    // Implementation for the GetEnumerator method.
    public IEnumerator<CPerson> GetEnumerator()
    {
        for (int i=0; i<people.Length; i++)
            yield return people[i];
    }

    IEnumerator IEnumerable.GetEnumerator() // IEnumerable<T> erbt von IEnumerable
    {
        return GetEnumerator();
    }
}
```


Iteratoren: Eigener Enumerator

```
static void Main()
{
    CPerson[] peopleArray = new CPerson[3]
    {
        new CPerson("John", "Smith"),
        new CPerson("Jim", "Johnson"),
        new CPerson("Sue", "Rabon")
    };

    CPeople peopleList = new CPeople(peopleArray);
    foreach (CPerson p in peopleList)
        Console.WriteLine(p.FirstName + " " + p.LastName);

    // es soll nur jedes 2. Element ausgegeben werden
    // This code shall produce output similar to the following:
}
```

John Smith
Jim Johnson
~~Sue Rabon~~

```
public class CPeopleEnum : IEnumerator
{
    public CPerson[] _people;
```

// Enumerators are positioned before the first element until the first MoveNext() call.
int position = -1;

```
public CPeopleEnum (CPerson[] list)
{
    _people = list;
}
```

```
public bool MoveNext()
{
    position++; // neu
    return (position < _people.Length);
}
```

```
public void Reset() { position = -1; }
```

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

```
public CPerson Current
{
    get { return _people[position]; }
}
```

```
public class CPeople:IEnumerable
{
    private CPerson[] people;
    // ...

    public IEnumerator GetEnumerator()
    {
        return new CPeopleEnum(people);
    }
}
```



Erstellen Sie eine Variante *CPeopleClone* unserer Klasse *CPeople*, welche die Schnittstelle **ICloneable** implementiert .

```
public class CPerson // Simple business object.
{
    public CPerson(string fName, string lName)
    { this.firstName = fName; this.lastName = lName; }

    public string firstName;
    public string lastName;
}

public class CPeople
{
    private CPerson[] people;           // array of persons
    public CPeople (CPerson[] pArray) // ctor copies values
    {
        people = new CPerson[pArray.Length];
        for (int i = 0; i < pArray.Length; i++) { people[i] = pArray[i]; }
    }
}
```



DHBW
Duale Hochschule
Baden-Württemberg
Heidenheim

Lösung 1



Wie unterscheiden sich Interfaces von abstrakten Klassen?



In welchen Teilen kann die .NET-Interfacetechnik die Mehrfachvererbung von C++ ersetzen?