

IEnumerator Arayüzü:

```
namespace System.Collections
{
    public interface IEnumerator
    {
        object Current { get; }
        bool MoveNext();
        void Reset();
    }
}
```

System.Collections isim alanı içinde bulunan bu arayüz Iteator işlemleri yapmamızı sağlıyor. Bunu kendimiz de yapamaz mıyız ? tabi ki yaparız. Ama belli başlı sınıflar ya da kullanımlar bizden IEnumerator ya da IEnumerable arayüzüne sahip sınıflardan türetilmiş elemanlar ister bu yüzden iterator işlemlerimiz için kendi yapımızı kullanmak yerine IEnumerator ve IEnumerable arayüzlerini implment edersek genel geçer bir yapımız olur.

Bu yapıyı şunlara benzetebiliriz.

Foreach

PHP'de db result fetching olayları.

.Netciler de zaten .GetEnumerator() metodlarını çokça kullanmışlardır.

Neden MovePrev yok dersiniz onu da makaleyi okuduktan sonra eklersiniz. sınıflarımıza (:.

IEnumerable Arayüzü:

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

Tek metod (GetEnumerator) isteyen bu arayüz bir iterator olmanız yerine sizden iterator gorebi gorebilecek bir dongu de alabilir. Bu da cok işimize yarayabilecek ozelliklerden biridir. Hemen bir IEnumerator dönderebilen Sınıf implement ediiip geçmeden önce yield return'dan bahsederlim

Yield Return:

IEnumerator dönüş tipi olan bir fonksiyon (Örn: GetEnumerator()) birden fazla elemanın ard arda dönebileceğini düşünün. Ki zaten .net te Enumerator yapısı bunun üzerine kurulmuştur. IEnumerator<T> yapısıyla dönecek elemanın tipi de belirlenebiliyor. Bu da bizi Current elemanının type casting olayından kurtarıyor. IEnumerator dönüş tipinizse yield return ile birden fazla eleman döndürüp bunu da GetEnumerator() dya tanımlayabilirsiniz bunları yapmışken IEnumerable arayüzünden implement ederseniz yine bazı standartlara uymuş olursunuz.

Şimdi hem IEnumerable türüne örnek yazalım hem de yield return kullanımına örnek göstereyim.

```
Public class Yazilimcilar : IEnumerable
{
    #region IEnumerable Members

    Public static IEnumerator GetEnumerator()
    {
        yield return "Tufan Barış YILDIRIM";
        yield return "Berkant KARDUMAN";
        yield return "Gökmen KOÇ";
        yield return "Süleyman Hilmi ŞAHİN";
        yield return "Savaş YILDIRIM";
        yield return "Ahmet ALTAY";
        yield break;
    }

    #endregion
}
```

Örnek sınıfımız artık bize bir numarator verebiliyor.

```
IEnumerator YazilimciNumarator = (new Yazilimcilar()).GetEnumerator();
while (YazilimciNumarator.MoveNext())
{
    Console.WriteLine(YazilimciNumarator.Current);
}
```

Ve bu kullanıma artık müsait. Tabi ki kullanımımız bu kadar basit olmayacaktır ancak örnek olması açısından düşünülebilir. Bu verilerin bize belli bir kaynaktan geldiğini varsayalım yine hiç bir şey değişmeyecek ve kaynağımızdan alma sırasına göre numaratore gonderebileceğiz. Örnek sınıfımızı hiç bozmadan buna örnek vermek istersek

```
Public static IEnumerator GetEnumerator()
{
    for (int i = 0; i < 10; i++)
    {
        yield return i + " Numaralı Yazılımcı ";
    }
    yield break;
}
```

Yield return anında bir kırılma olabilir mi ? return olsaydı olurdu ancak debug anında şöyle bir şey gözlemledim. Numarator alan fonksiyon ya metodlar beraberinde üretim fonksiyonunda ismini taşıyor "GetEnumerator" tahminimce kaldığı yeri de burdan biliyor

IEnumerator<T> Arayüzü

```
namespace System.Collections.Generic
{
    public interface IEnumerator<T> : IDisposable, IEnumerator
    {
        T Current { get; }
    }
}
```

Kendi tiplerimize iterate özelliği kazandırabilir miyiz. .NET in En sevdiğimiz yapılarından olan Generic Collections bize bura da Numaratorleri object türünden değil de kendi tiplerimizde kullanma şansı veriyor. LINQ içinde from eleman in elemanlar dediğimizde eleman tipini tanıması generic collectionslarda tipin zaten tanımlanıyor olmasıdır. Biz de numaratorlerimizde YazilimciNumarator.Current değimizde cast ihtiyacı duymadan . tipine uygun metodları ya da propertyleri kullanabilmemiz için **Enumerator arayüzüne de sahip** bu arayüzü kullanmalıyız bir yazılımcı sınıfı yazıp numaratore örnek verelim.

```
public class Yazilimci
{
    public string Adi { get; set; }
    public string Soyadi { get; set; }
}
```

IEnumerable<T> Arayüzü:

```
namespace System.Collections.Generic
{
    public interface IEnumerable<T> : IEnumerable
    {
        IEnumerator<T> GetEnumerator();
    }
}
```

IEnumerable Arayüzü ile aynı olan bu arayüz GetEnumerator metodunu ezerek buna bir de tip tanımlanmış olan IEnumerator<T> tipinde bir Generic dönderir. Yine bu metodu biz yazacağız.

Yazılımcı listesini yine bir “yield return” örneğiyle kısaca anlatalım.

```
public class Yazilimcilar: IEnumerable<Yazilimci>
{
    #region IEnumerable<Yazilimci> Members

    public IEnumerator<Yazilimci> GetEnumerator()
    {
        yield return new Yazilimci() { Adi = "Tufan Barış", Soyadi = "YILDIRIM" };
        yield return new Yazilimci() { Adi = "Berkant", Soyadi = "KARDUMAN" };
        yield return new Yazilimci() { Adi = "Gökmen", Soyadi = "KOÇ" };
        yield return new Yazilimci() { Adi = "Süleyman", Soyadi = "ŞAHİN" };
        yield return new Yazilimci() { Adi = "Savaş", Soyadi = "YILDIRIM" };
        yield return new Yazilimci() { Adi = "Ahmet", Soyadi = "Altay" };
    }

    #endregion

    #region IEnumerable Members

    IEnumerator IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }

    #endregion
}
```

Artık tipimiz de belliyse şimdi numarator gibi kullanırken “Current” bize “Yazilimci” gibi davranabilecek davranmaktan ziyade tipi zaten Yazilimci olacaktır fakat bunu cast etme ihtiyacı duymadan kullanabileceğiz.

```
IEnumerator<Yazilimci> Eleman = (new Yazilimcilar()).GetEnumerator();

while (Eleman.MoveNext())
{
    Console.WriteLine(Eleman.Current.Adi + " " + Eleman.Current.Soyadi);
}
```