**Collection Sınıflar**

C# Programlama Dili Kursunda collection sınıf olarak yalnızca ArrayList ve bunn generic biçimi olan List<T> sınıfı görülmüştü. Burada önemli bazı diğer colelction sınıflar ele alınacaktır.

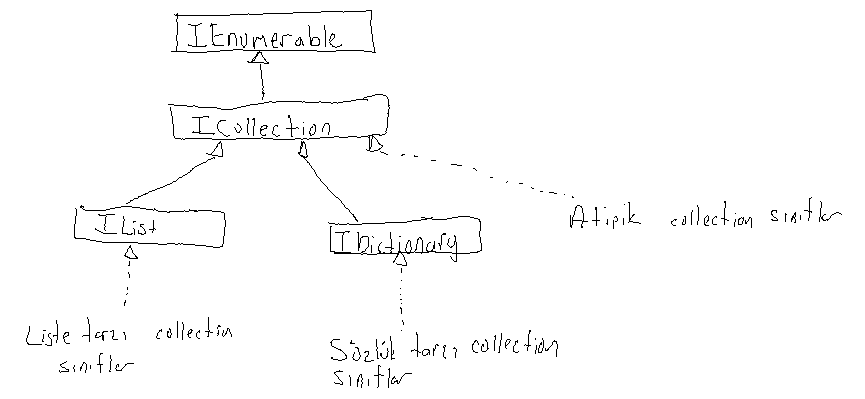
.NET'te collection sınıflar üç gruba ayrılmaktadır:

1) Liste tarzı collection sınıflar

2) Sözlük tarzı collection sınıflar

3) Atipik collection sınıflar

.NET'teki collection sınıflar bazı arayüzleri desteklemektedir. Bu nedenle önce o arayüzlerin ele alınması anlatını kolaylaştıracaktır.



IEnumerable arayüzünün GetEnumerator isimli tek bir metodu vardır:

interface IEnumerable

{

IEnumerator GetEnumerator();

}

IEnumerator arayüzü bir collection'ı türden bağımsız dolaşmak için kullanılmaktadır. Bu arayüzün üç elemanı vardır: MoveNext, Current ve Reset.

MoveNext metodu imleci sonraki elemana kaydırır. Current property'si object türündendir ve read-only'dir. Enumerator ilk alındığında imleç ilk elemanın bir gerisindedir. Yani bizim imleci ilk elemana konumlandırmamız için bir kez MoveNext yapmamaız gerekir. MoveNext metodunun parametrik yapısı şöyledir:

bool MoveNext()

MoveNext her çağrıldığında imleç bir sonraki elemana konumlanır. MoveNext ile imleç eğer son elemandan sonraya konumlandırılmaya çalışılırsa MoveNext false ile geri döner. Bu da collection'ın sonuna geldiğimiz gösterir. Bu durumda collection'ın sonuna kadar dolaşmak için şöyle basit bir while döngüsü yeterlidir.

IEmumerator ie = col.GetEnumerator();

while (ie.MoveNext())

{

//...

}

Reset metodu imleci ilk duruma yani ilk elemanın bir gerisine konumlandırır.

IEnumerable arayüzü sayesinde biz her türlü collection sınıfı onun özelliklerini bilmeden, türden bağımsız olarak dolaşabiliriz. Genel dolaşım kalıbı aşağıdaki gibi olabilir:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

ArrayList al = new ArrayList();

for (int i = 0; i < 10; ++i)

al.Add(i \* 10);

IEnumerator ie = al.GetEnumerator();

while (ie.MoveNext())

{

int val = (int)ie.Current;

Console.Write("{0} ", val);

}

Console.WriteLine();

ie.Reset();

while (ie.MoveNext())

{

int val = (int)ie.Current;

Console.Write("{0} ", val);

}

Console.WriteLine();

}

}

}

Aşağıdaki örnekte Walk isimli metot IEnumerable arayüzü türünden bir referansı parametre olarak almaktadır. Biz de bu metoda o arayüzü destekleyen herhangi bir sınıf referansını ya da yapı nesnesini parametre olarak geçirebiliriz:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

ArrayList al = new ArrayList();

for (int i = 0; i < 10; ++i)

al.Add(i \* 10);

Sample.Walk(al);

}

}

class Sample

{

public static void Walk(IEnumerable ie)

{

IEnumerator ien = ie.GetEnumerator();

while (ien.MoveNext())

Console.Write("{0} ", (int)ien.Current);

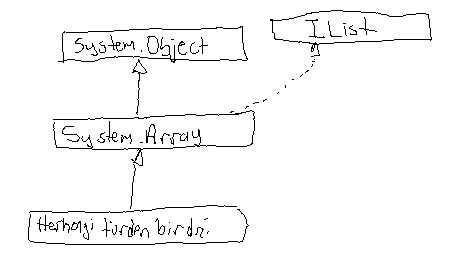
Console.WriteLine();

}

}

}

C#'ta tüm diziler de sanki liste tarzı bir collection sınıf gibidir. Şöyle ki: C# standartlarına göre tüm dizilerin System.Array isimli bir sınıftan türetildiği varsayılmaktadır. System.Array sınıfı da IList arayüzünü destekler ve System.Object sınıfından türetilmiştir:



Örneğin:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

double[] d = new double[] { 1, 2, 3, 4.5 };

int[] i = new int[] { 10, 20, 30, 40, 50 };

string[] s = new string[] { "ali", "veli", "selami", "ayşe", "fatma" };

ArrayList al = new ArrayList();

al.Add("Ankara");

al.Add("İzmir");

al.Add("Adana");

Sample.Walk(d);

Sample.Walk(i);

Sample.Walk(s);

Sample.Walk(al);

}

}

class Sample

{

public static void Walk(IEnumerable ie)

{

IEnumerator ien = ie.GetEnumerator();

while (ien.MoveNext())

Console.Write("{0} ", ien.Current.ToString());

Console.WriteLine();

}

}

}

C#'taki foreach deyimi aslında kendi içerisinde IEnumearable arayüzüyle dolaşımı yapmaktadır. Yani biz foreach deyimini kullandığımızda derleyicinin ürettiği koda bakarsak onun dolaşımı IEnumerable arayüzü ile yaptığını görürüz. Başka bir deyişle:

foreach (T x in a)

{

//...

}

Döngüsü aşağıdakiyle eşdeğerdir:

IEnumerator ien = a.GetEnumerator();

while (ien.MoveNext())

{

T x = (T)ien.Current;

//...

}

Peki biz IEnumerable arayüzünü destekleyen bir sınıf yazabilir miyiz? Böyle bir örnek şöyle oluşturulabilir:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

MyCollection mc = new MyCollection(10, 20);

IEnumerator ien = mc.GetEnumerator();

while (ien.MoveNext())

{

int val = (int)ien.Current;

Console.Write("{0} ", val);

}

Console.WriteLine();

foreach (int x in mc)

Console.Write("{0} ", x);

Console.WriteLine();

}

}

class MyCollection : IEnumerable

{

private int m\_low;

private int m\_high;

public MyCollection(int low, int high)

{

m\_low = low;

m\_high = high;

}

public IEnumerator GetEnumerator()

{

return new MyEnumerator(this);

}

private class MyEnumerator : IEnumerator

{

private MyCollection m\_mc;

private int m\_curVal;

public MyEnumerator(MyCollection mc)

{

m\_mc = mc;

m\_curVal = mc.m\_low - 1;

}

public bool MoveNext()

{

if (m\_curVal == m\_mc.m\_high)

return false;

++m\_curVal;

return true;

}

public object Current

{

get { return m\_curVal;}

}

public void Reset()

{

m\_curVal = m\_mc.m\_low - 1;

}

}

}

}

**Anahtar Notlar:** Alçak seviyeli bir dilden yüksek seviyeli bir dile dönüştüren tersine mühendislik programlarına "decompiler" denilmektedir. Örneğin .NET'in .exe ve .dll dosyalarını C#'a dönüştüren programlar birer "decompiler" programlardır. .NET'in için decompiler "Salamander" ismindeki paralı üründü. Sonra "Reflector" isimli "open source" yazılım kullanılmaya başlandı. Ancak "reflector"ü yazanlar "Redgate" firmasına sattılar. Tabi eski kodlara erişim devam etti. Yeni bir proje grubu da "ILSpy" ismiyle bunu devam ettirdi. Maalesef .NET gibi Java gibi dillerin arakodları "decompile" edilebilmektdir. Ancak C/C++ gibi dillerle saf makina diline dönüştürülmüş kodlar halen etkin bir biçimde "decompile" edilememektedir.

**ICollection Arayüzü**

ICollection arayüzünün dört elemanı vardır. CopyTo metodu collection sınıf içerisindekileri bir diziye aktarmak için kullanılır. Metodun parametrik yapısı şöyledir:

void CopyTo(Array array,int index)

Metodun birinci parametresi aktarımınm yapılacağı diziyi, ininci parametresi bu dizide kopyalamanın hangi indeksten itibaren yapılacağıdır. Örneğin:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

ArrayList al = new ArrayList();

for (int i = 0; i < 10; ++i)

al.Add(i);

Foo(al);

}

public static void Foo(ICollection ic)

{

int[] a = new int[ic.Count];

ic.CopyTo(a, 0);

foreach (int x in a)

Console.Write("{0} ", x);

Console.WriteLine();

}

}

}

Burada aktarılacak dizinin oradaki elemanın türüyle aynı türden olması gerekir. Aksi halde exception oluşur.

ICollection arayüzünün Count isimli read-only property elemanı collection içerisindeki eleman sayısını vermektedir. IsSynchronized property'si collection'ın thread güveli (thread safe) olup olmadığı bilgisini bize verir. SyncRoot property'si yine thread senkronizasyonu için kullanılan bir elemandır.

**IList Arayüzü**

IList arayüzü liste tarzı collection sısnıfların desteklediği ortak bir arayüzdür. İçerisinde pek çok önemli eleman vardır. Bu elemanların hepsi bu arayüzü destekleyen sınıflarda bulunmak zorundadır.

Add metodu collection'ın sonuna eleman eklemek için kullanılır:

[int](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ed425922-7a7b-5232-1fbc-5e4ac9680de6.htm) Add([Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) value)

Metot eklenecek elemanı bizden parametre olarak alır. Eklemenin yapıldığı indekse geri döner.

**Anahtar Notlar:** Bilindiği gibi bir dizeye ekleme yapılamaz. Peki hani diziler Array sınıfından türetilmişti de o sınıf da IList arayüzünü destekliyordu? Bu durumda biz bir dizi için Add yapabilir miyiz? İşte Array sınıfı içerisinde Add ve birkaç metot açıkça (explicit) desteklnemiştir. Array sınıfının Add metodunda doğrudan exception throw edilmektedir. Yani biz istesek de diziye bu metotlarla ekleme yaapamayız.

ILıst arayüzünün Clear metodu collection'daki tüm elemanları silmektedir. Contains metodu belli bir eleman colleciton'da var mı diye bakar. IndexOf da benzer bir arama yapar fakat elemanı bulursa onun collection'daki indeks numarasına bulamasa -1 değerine geri döner. Insert metodu diğer elemanları kaydırarak belli bir indekse ekleme yapmaktadır. Metodun parametrik yapısı şöyledir:

void Insert([int](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ed425922-7a7b-5232-1fbc-5e4ac9680de6.htm) index, [Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) value)

Collection'ın Remove metodu elemanı bularak siler, RemoveAT ise indeks ile belirtilen elemanı siler. Silme sırasında collection'daki elemanlar birer kaydırılır (shrink işlemi).

IList arayüzünün object türünden read/write indeskleyicisi vardır. Yani her liste collection sınıfı biz köşeli parantez operatörüyle kullanabiliriz. IList arayüzünün IsFixedSize property'si colelction'ın sabit uzunlukta olup olmadığını belirlemek için, IsReadOnly property elemanı da collection'ın read-only olup olmadığını belirlemek için kullanılmaktadır.

IList arayüzünü destekleyen en önemli sınıf ArrayList ve bunun generic biçimi olan List<T> sınıfıdır. İleride GUI işlemlerinde bu arayüzü destekleyen pek çok collection sınıfla karşılaşılacaktır.

**IDictionary Arayüzü**

Sözlük tarzı collection sınıflar IDictionary arayüzünü desteklemektedir. Bu collection sınıflar anahtar-değer çiftlerini bizden alırlar. Ona anahtarı verdiğimizde bize değeri verirler. Tabi bu işlemi özel algoritmik yöntemlerle çok hızlı yaparlar.

IDictionary arayüzünün Add metodu collection'a eleman eklemek için kullanılır:

void Add([Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) key, [Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) value)

Metodun birinci parametresi anahtarı ikinci parametresi değeri belirtir. Clear metodu collection'daki tüm elemanları silmektedir. Contains metodu algoritmik olarak belli bir anahtarın collection'da olup olmadığına bakar. Remove belli bir anahtarı parametre olarak alır ve onu colelction'dan siler.

IDicitionary arayüzünün object parametreli object türünden indeksleyicisi belli bir anahtara karşılık bize değeri vermektedir. Arayüzün Keys property'si bize tüm anahtarları, Values property'si tüm değerleri ICollection arayüzü ile verir.

Genel olarak IDictionary arayüzünü destekleyen sınıflarda Add metodu ile aynı anahtarı ikinci kez ekleyemeyiz. Bu durumda Exception oluşur. Ancak indeskleyici yoluyla ekleme yapmak istersek eski değer gider yeni değer atanır.

**Hashtable Sınıfı**

En çok kullanılan sözlük tarzı collection sınıf Hashtable sınıfıdır. Bu sınıf "hash tablosu" ya da "hashing" denilen algoritmik yöntemi kullanmaktadır. Örneğin:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

Hashtable ht = new Hashtable();

ht.Add("Ali Serçe", 123);

ht.Add("Kaan Aslan", 512);

ht.Add("Necati Ergin", 512);

int val = (int)ht["Ali Serçe"];

Console.WriteLine(val);

ht["Selami Karakelle"] = 654;

ht["Selami Karakelle"] = 657;

val = (int)ht["Selami Karakelle"];

Console.WriteLine(val);

foreach (string key in ht.Keys)

Console.Write("{0} ", key);

Console.WriteLine();

foreach (int v in ht.Values)

Console.Write("{0} ", v);

Console.WriteLine();

}

}

}

Hashtable şu durumlarda tercih edilmelidir:

- Eleman sayısı az olmayan durumlarda (en az 20, fakat daha fazla olsa daha iyi)

- Arama işleminin çok fazla yapıldığı sistemlerde

Eğer eleman sayısı çok azsa (tipik olarak 20'den az) bu durumda SortedList isimli collection tercih edilmelidir. SortedList iki paralel dizi biçiminde gerçekletirilmiştir. Dolayısıyla anahtar dizisinde eleman sıralı aramayla bulunur. Sonra da değer dzisinde bunun karşılığı olan değer verilir. Eleman sayısı az olduğu için sıralı arama çok etkindir. Örneğin:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

SortedList sl = new SortedList();

sl.Add("Ali Serçe", 123);

sl.Add("Kaan Aslan", 512);

sl.Add("Necati Ergin", 512);

int val = (int)sl["Ali Serçe"];

Console.WriteLine(val);

sl["Selami Karakelle"] = 654;

sl["Selami Karakelle"] = 657;

val = (int)sl["Selami Karakelle"];

Console.WriteLine(val);

foreach (string key in sl.Keys)

Console.Write("{0} ", key);

Console.WriteLine();

foreach (int v in sl.Values)

Console.Write("{0} ", v);

Console.WriteLine();

}

}

}

Hashtable sınıfının generic versiyonun ismi Dictionary<Key, Value> sınıfıdır. Ayrıca bir de HashSet<T> biçiminde bir versiyonu da vardır. HashSet<T> sınıfı bizden ekleme sırasında anahtar ve değeri ayrı ayrı istemez. Tek bir nesne olarak ister. Onu bulurken eşitlik karılaştırması yapmaktadır. Dictionary<Key, Value> ise Hashtable gibi kullanılır. Örneğin:

using System;

using System.Collections.Generic;

namespace CSD

{

class App

{

public static void Main()

{

Dictionary<string, int> dict = new Dictionary<string, int>();

dict.Add("Ali Serçe", 123);

dict.Add("Kaan Adictan", 512);

dict.Add("Necati Ergin", 512);

int val = (int)dict["Ali Serçe"];

Console.WriteLine(val);

dict["Selami Karakelle"] = 654;

dict["Selami Karakelle"] = 657;

val = (int)dict["Selami Karakelle"];

Console.WriteLine(val);

foreach (string key in dict.Keys)

Console.Write("{0} ", key);

Console.WriteLine();

foreach (int v in dict.Values)

Console.Write("{0} ", v);

Console.WriteLine();

}

}

}

Örneğin:

using System;

using System.Collections.Generic;

namespace CSD

{

class App

{

public static void Main()

{

SortedList<string, int> sl = new SortedList<string, int>();

sl.Add("Ali Serçe", 123);

sl.Add("Kaan Aslan", 512);

sl.Add("Necati Ergin", 512);

int val = (int)sl["Ali Serçe"];

Console.WriteLine(val);

sl["Selami Karakelle"] = 654;

sl["Selami Karakelle"] = 657;

val = (int)sl["Selami Karakelle"];

Console.WriteLine(val);

foreach (string key in sl.Keys)

Console.Write("{0} ", key);

Console.WriteLine();

foreach (int v in sl.Values)

Console.Write("{0} ", v);

Console.WriteLine();

}

}

}

SortedList<Key, Value> sınıfı System.dll içerisinde bulunmaktadır. Bu nedenle bu sınıf kullanılmadan önce bu dll'e referans edilmesi gerekir.

**Queue Sınıfı**

Queue kuruk veri yapısını destekleyen atipik bir collection sınıftır. Doğrudan IColelction arayüzünü desteklemektedir. Kuyruklar FIFO prensibiyle çalışan tamponlardır. Kuyruk için iki önemli işlem söz konusudur: Kuyruğa eleman yerleştirilir ve kuyruktan eleman alınır. Örneğin biz X, Y ve Z elemanlarını kuyruğa yerleştirmiş olalım. Bunları almak istediğimizde yine X, Y ve Z sırasına göre alırız. Kuyruk veri yapısı bilgileri geçici süre sırası bozulmadan bekletmek için kullanılmaktadır. Queue sınıfının Queue<T> biçiminde generic bir versiyonu da vardır.

Kuruğa eleman yerleştirmek için Enqueue metodu, eleman almak için Dequeue metodu kullanılır:

public virtual void Enqueue([Object](file:///E:\Dropbox\Kurslar\Beatles-App1\Doc\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) o)

public virtual [Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) Dequeue()

Count property'si kuyrukta o anda bulunan eleman sayısını bize verir. Kuyruk boşken Dequeue yapılmak istenirse exception oluşur. Örneğin:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

Queue q = new Queue();

for (int i = 0; i < 10; ++i)

q.Enqueue(i);

while (q.Count > 0)

{

int val = (int)q.Dequeue();

Console.Write("{0} ", val);

}

Console.WriteLine();

}

}

}

Queue sınıfının Peek metodu kuyruğun önündeki elemanı bize verir fakat onu kuyruktan atmaz.

Diğer bir kuruk sistemine de öncelik kuyruğu (priority queue) denilmektedir. Bu tür kuyruklara elemanlar bir öncelik derecesi verilerek eklenir. Eleman alınırken en yüksek önceliğe sahip eleman alınır. Maalesef .NET kütüphensinde Microsoft tarafından gerçekleştirilmiş böyle bir sınıf hazır biçimde yoktur.

**Stack Sınıfı**

LIFO (Last In First Out) prensibiyle çalışan kuyruk sistemlerine Stack denilmektedir. Stack sistemleriyle günlük hayatta karşılaşılmaktadır. Örneğin tabaklar üst üste konulduğunda son konulan önce alınır. Undo işlemi son yapılanı geri alır. Stack veri yapısı bazı tipik algoritmaları gerçekleştirmek için kullanılabilmektedir. Stack sisteminde kuyruğa eleman eklemeye geleneksel olarak Push işlemi, kuyruktan eleman almaya da Pop işlemi denilmektedir:

public virtual void Push([Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) obj)

public virtual [Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) Pop()

Yine Stack'teki eleman sayısı Count property'si ile elde edilmektedir. Stack de atipik bir collection sınıfıdır. Doğrudan ICollection arayüzünü desteklemektedir. Örneğin:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

Stack s = new Stack();

for (int i = 0; i < 10; ++i)

s.Push(i);

while (s.Count > 0)

{

int val = (int)s.Pop();

Console.Write("{0} ", val);

}

Console.WriteLine();

}

}

}

Stack sınıfının generic versiyonu benzer biçimde kullanılabilir:

using System;

using System.Collections.Generic;

namespace CSD

{

class App

{

public static void Main()

{

Stack<int> s = new Stack<int>();

for (int i = 0; i < 10; ++i)

s.Push(i);

while (s.Count > 0)

{

int val = s.Pop();

Console.Write("{0} ", val);

}

Console.WriteLine();

}

}

}

**Eşitlik Karşılaştırmaları**

C#'ta iki referans == ve != operatörleriyle karşılaştırılmak istendiğinde önce referanslara ilişkin sınıflarda bu karşılaştırmayı yapabilecek operatör metodu var mı diye bakılır. Varsa onlar çağrılır. Fakat böyle bir operatör metodu yoksa bu durum referanslar içerisindeki adreslerin karşılaştırılacağı anlamına gelir. Yani böylelikle iki referansın aynı nesneyi gösterip göstermediğine bakılacaktır. Ancak operatör metodu söz konusu olmadığında bizim == ve != operatörleriyle eşitlik karşılaştırması yapabilmemiz için iki referansın ya aynı sınıf türünden olması ya da aralarında türetme ilişkisi olması gerekir (yani biri diğerinin taban sınıfı olm alıdır). Biz operatör metodu söz konusu değilse >, >=, < ve <= operatörleriyle hiçbir durumda karşılaştırma yapamayız.

Örneğin biz iki string'i karşılaştırmak istediğimizde string sınıfının == ve != operatör metotları olduğu için onların içerisindeki adresleri değil, onların gösterdiği yerdeki nesne içerisindeki yazıların aynı olup olmadığını karşılaştırırız.

Bazen sınıfın == ve != operatör metotları olduğu halde biz yine de adres karşılaştırması yapmak isteyebiliriz. Bunun için object sınıfının static ReferenceEquals metodu kullanılmaktadır. Örneğin:

using System;

namespace CSD

{

class App

{

public static void Main()

{

string s = "ankara";

string k = "an";

k += "kara";

Console.WriteLine(s == k); // True

Console.WriteLine(object.ReferenceEquals(s, k)); // False

}

}

}

**Anahtar Notla**r: C# standartlarına göre taamamen aynı karakterlerden oluşan özdeş string'ler için her defasında ayrı bir string nesnesi oluşturulmaz. Aynı assemblideki özdeş string2ler için tek bir string nesnesi oluşturulur. Hepsine aynı nesne adresi atanır. Örneğin:

using System;

namespace CSD

{

class App

{

public static void Main()

{

string s = "ankara";

string k = "ankara";

Console.WriteLine(s == k); // True

Console.WriteLine(object.ReferenceEquals(s, k)); // True

}

}

}

**Object sınıfının Equals Metotları**

object sınıfının aşağıdaki gibi virtual bir Equals metodu vardır:

public virtual [bool](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ff35b1f1-386c-370b-2c36-a48e7dcbc147.htm) Equals([Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) obj)

Bu metot int, long, double gibi temel türlere ilişkin yapılarda ve pek sınıfta override edilmiştir. Equals metodu bazı metotlar tarafından kullanılmaktadır. Örneğin ArrayList sınıfının Equals gibi, IndexOf gibi metotları collection içerisinde arama yaparken Equals sanal metodundan faydalanmaktadır. Bizim bu metotları sağlıklı llarak kullanabilmemiz için ilgili sınıfta bu metodu override etmemiz gerekir. Örneğin:

using System;

using System.Collections;

namespace CSD

{

class App

{

public static void Main()

{

ArrayList al = new ArrayList();

for (int i = 0; i < 10; ++i)

al.Add(new Number(i));

Number x = new Number(5);

if (al.Contains(x))

Console.WriteLine("Var");

else

Console.WriteLine("Yok");

}

}

class Number

{

private int m\_val;

public Number(int val)

{

m\_val = val;

}

public override bool Equals(object obj)

{

Number n = (Number)obj;

return n.m\_val == m\_val;

}

public int Val

{

get { return m\_val; }

}

}

}

ArrayList sınıfında Contains metodu muhtemelen aşağıdakine benzer biçimde yazılmıştır:

class MyArrayList

{

private object[] m\_objs;

private int m\_count;

private int m\_capacity;

public MyArrayList()

{

m\_objs = new object[2];

m\_capacity = 2;

m\_count = 0;

}

public int Add(object o)

{

if (m\_count == m\_capacity)

{

m\_capacity \*= 2;

object[] newObjs = new object[m\_capacity];

for (int i = 0; i < m\_count; ++i)

newObjs[i] = m\_objs[i];

m\_objs = newObjs;

}

m\_objs[m\_count] = o;

++m\_count;

return m\_count - 1;

}

public int Count

{

get { return m\_count; }

}

public int Capacity

{

get { return m\_capacity; }

}

public bool Contains(object obj)

{

for (int i = 0; i < m\_count; ++i)

if (obj.Equals(m\_objs[i]))

return true;

return false;

}

public object this[int index]

{

get { return m\_objs[index]; }

set { m\_objs[index] = value; }

}

//...

}

Eğer biz ilgili sınıfta Equals metodunu override etmezsek bu durumda object sınıfının Equals metodu çağrılır. O da referans eşitliğine bakmaktadır.

object sınıfında static bir Equals metodu daha vardır:

public static [bool](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ff35b1f1-386c-370b-2c36-a48e7dcbc147.htm) Equals([Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) objA,[Object](file:///C:\Users\csystem\AppData\Roaming\Microsoft\Word\ee2c26d9-17cc-ab19-8a9c-6fca33a3c7ad.htm) objB)

Bu fonksiyon tatamen kendi içerisinde objA.Equals(objB) işlemini yapmaktadır. Yani:

object.Equals(a, b)

ile,

a.Equals(b)

tamamen işlevsel olarak eşdeğerdir.