



GÖRSEL PROGRAMLAMA

Koleksiyonlar, Genel sınıflar ve T

Genel Türler

- Generic
- Birden fazla tür için tanımlanabilir sınıflar
- <T>
- Yeniden kullanılabilirlik
- En çok görülen kullanım alanı
 - Koleksiyonlar

Genel türler bir metodun ya da sınıfın istenilen türle birlikte çalışması için kullanılır.

Metot ya da sınıfın yanına <T> eklemesi yapılarak bir tür ile kullanılabileceği belirtilir.

Kod tekrarını engellemek ve yeniden kullanılabilirliği arttırmak amaçlanmaktadır. Genellikle birden fazla elemanın saklandığı ya da üzerinde toplu işlemler yapıldığı durumlarda kullanılır.

Genel Türler

```
namespace Uygulama
{
    2 references
    class Sinif
    {
        1 reference
        internal void Yazdir(string v)
        {
            Console.WriteLine(v);
        }


        1 reference
        internal void Yazdir(int v)
        {
            Console.WriteLine(v);
        }
    }
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Sinif s = new Sinif();
            s.Yazdir("String");
            s.Yazdir(1);
        }
    }
}
```



```
namespace Uygulama
{
    2 references
    class Sinif
    {
        2 references
        internal void Yazdir<T>(T v)
        {
            Console.WriteLine(v);
        }
    }
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Sinif s = new Sinif();
            s.Yazdir("String");
            s.Yazdir(1);
        }
    }
}
```

Sol tarafta hem string bir v değeri için hem de bir int v değeri için yazdır metodu yazılmak zorundadır. Sağ tarafta ise <T> ile belirtildikten sonra yazdır metodu istenilen türde çağırılabilir. Metodun çağırılmasında Yazdir<string> gibi bir kullanım yapılabileceği gibi otomatik çözümleme de gerçekleştirilebilir.

Genel Türler



```
namespace Uygulama
{
    2 references
    class Sinif
    {
        1 reference
        internal void Yazdir(string v)
        {
            Console.WriteLine(v);
        }

        1 reference
        internal void Yazdir(int v)
        {
            Console.WriteLine(v);
        }
    }
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Sinif s = new Sinif();
            s.Yazdir("String");
            s.Yazdir(1);
        }
    }
}

5 references
class Sinif<T>
{
    T degisken;
    2 references
    public Sinif(T degisken)
    {
        this.degisken = degisken;
    }
    2 references
    internal void Yazdir()
    {
        Console.WriteLine(degisken);
    }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Sinif<string> s = new Sinif<string>("String");
        s.Yazdir();
        Sinif<int> s2 = new Sinif<int>(3);
        s.Yazdir();
    }
}
```

Aynı şekilde sınıflar da <T> tanımlanabilir. Bu sınıfın içerisinde T cinsinden değişkenler de tanımlanabilir. Bu değişkenlerin tür çözümü sınıf tanımlamasında T'ye verilen değişken türüne göre (örnekte string) yapılır.

Genel Türler

- T bir değişken olarak veya dizi olarak kullanılabilir.

```
4 references
class Sinif<T>
{
    T degisken;
    T[] dizi;
    2 references
    internal void Yazdir()
    {
        for (int i = 0; i < dizi.Length; i++)
            Console.Write(dizi[i]);
    }
}
```

T type bir değişken tanımlanabileceği gibi, aynı şekilde T türünde bir dizi de oluşturulabilmektedir. Bu dizi kullanılmadan önce diğer dizilerde olduğu gibi örneğinin alınması (new anahtar kelimesi ile) gereklidir.

Genel Türler

- T üzerinde aritmetik işlemler gerçekleştirilemez.

```
class Sinif<T>
{
    T degisken;
    T[] dizi;
    2 references
    internal void Yazdir()
    {
        T sonuc;
        for (int i = 0; i < dizi.Length; i++)
            sonuc += dizi[i];
    }
}
```

CS0019: Operator '+' cannot be applied to operands of type 'T' and 'T'

- Bunun için ek metotlar yazılması gerekmektedir.

```
class Sinif<T>{
    T[] dizi;
    0 references
    internal void Hesapla(){
        dynamic sonuc=0;
        for (int i = 0; i < dizi.Length; i++){
            sonuc = topla(sonuc, dizi[i]);
        }
    }
    1 reference
    private T topla(T sayi1, T sayi2){
        dynamic a = sayi1;
        dynamic b = sayi2;
        return a + b;
    }
}
```

T türü sınıf tanımlanırken belirleneceği için aritmetik operatörlerin kullanılmasını desteklemez. T türü bir int, string, double olabileceği gibi bir sınıf türünden ve hatta kullanıcı tanımlı bir sınıf ya da yapı türünden olabilir. Bu durumda + - * ve / işlemlerinin ne yapacağı tanımlı olmayacağından dolayı bu işlemler desteklenmezler. Bunun önüne geçebilmek için çalışma anında (runtime) çözümleme yapıp bu şekilde işlem yaptırılmak istenebilir. Bu durumda kod derlenecektir fakat eğer kod içerisinde toplama işlemine izin verilmeyen bir tür kullanılırsa çalışma zamanında hata alınacaktır.

Koleksiyonlar

- Diziler
 - Statik
- Koleksiyonlar
 - Dinamik
 - Farklı ihtiyaçlara çözümler
 - Sınıf - Örnek alınmalı

Koleksiyonlar dizilerden farklı olarak, dinamik olmasıyla beraber hash, dictionary, list gibi farklı çözümler sunan yapılar da içermektedir.

Koleksiyonlar

- **List<T>**
- **Dictionary<TKey,TValue>**
- **HashSet<T>**
- Queue<T>
- SortedList<T>
- Stack<T>

Koleksiyonlar içerisinde en sık kullanılanları list, dictionary ve hashset'tir.

Koleksiyonlar (List)

- Dinamik
- Dizi
- Tek türden elemanların listesi

Listeler dizilerden farklı olarak dinamik ekleme ve silme işlemi gerçekleştirilebilirler. Yeniden boyutlandırma listenin implementasyonu içerisinde otomatik olarak gerçekleştirilir.

Koleksiyonlar (List)

- Temel tip listeleri oluşturulabilir.

```
List<string> stringListesi = new List<string>();  
List<int> intListesi = new List<int>();
```

- Sınıf listeleri oluşturulabilir.

```
List<Sinif> sinifListesi = new List<Sinif>();  
List<FileStream> dosyaListesi = new List<FileStream>();
```

- Sınıf alabildiği için List listeleri oluşturulabilir.

```
var listListesi = new List<List<string>>();  
var listListesiListesi = new List<List<List<string>>>();
```

Listeler T türünde parameter aldıkları için istenilen türde tanımlanabilirler. Bir liste temel türlerde veya tanımlanmış yapı ve sınıf türlerinde açılabilir. Ayrıca bir türün listesi (list<tür>) de bir tür olarak kabul edilebileceği için bir liste tür olarak başka türde tanımlanmış bir listeyi alabilir. Bu durumda listenin her bir elemanı bir liste olacaktır.

Koleksiyonlar (List)

- Add()
- AddRange()
- Remove()
- RemoveAt()
- Count
- ToArray()
- Clear()

```
static void Main(string[] args)
{
    List<string> stringListesi = new List<string>();
    List<string> altListe = new List<string>() { "deger1", "deger2", "deger3" };
    stringListesi.Add("a");
    stringListesi.AddRange(altListe);
    stringListesi.Remove("deger1");
    stringListesi.RemoveAt(2);
    int uzunluk = stringListesi.Count;
    string[] dizi = stringListesi.ToArray();
    stringListesi.Clear();
}
```

Listelerin en çok kullanılan metotları.

Add ekleme işlemi yapar. Listelerde tekillik olmadığı için bir eleman birden fazla kez tekrarlı bir şekilde eklenebilir.

AddRange ile verilen türden olmak üzere birden fazla eleman aynı anda eklenebilir.

Remove ile referansı bilinen bir nesne listeden silinebilir

RemoveAt ile listenin belirli indisindeki bir eleman silinebilir

Count listenin eleman sayısını verir

Bazı durumlarda listeler yerine native diziler ile çalışılmak istenebilir. Bu durumda listeleri dizilere dönüştürmek için toArray metodu kullanılabilir.

Bir listeyi temizlemek için clear metodu kullanılabilir.

Koleksiyonlar (Dictionary)

- Anahtar ve değer alanları

```
int deger=0;  
Dictionary<string, int> degerler = new Dictionary<string, int>();
```

- Her iki alan da <T>

```
var degerler1 = new Dictionary<string, int>();  
var degerler2 = new Dictionary<int, string>();  
var degerler3 = new Dictionary<int, float>();  
var degerler4 = new Dictionary<string, Sinif>();
```

- Anahtar ile değere erişim

```
int deger=0;  
Dictionary<string, int> degerler = new Dictionary<string, int>();  
degerler["anahtar"] = deger;  
deger = degerler["anahtar"];
```

Belirli durumlarda indis yerine bir değişken, bir metin veya bir yapı kullanılma ihtiyacı olabilir. Dictionary veri türünde bir anahtar ve anahtara karşılık gelen bir değer saklanmaktadır. Bu anahtar istenilen türde olabileceği gibi değerler de yine istenilen türde tanımlanabilmektedir. Dictionary değerlerine erişmek için dizi notasyonunda olduğu gibi anahtarın köşeli parantezler içerisinde verilmesi yeterlidir.

Koleksiyonlar (Dictionary)

- Add
- Count
- Keys
- Values
- ContainsKey
- ContainsValue

```
Dictionary<string, int> degerler = new Dictionary<string, int>();  
degerler.Add("Anahtar", 1);  
Console.WriteLine(degerler.Count);  
foreach (var key in degerler.Keys)  
    Console.WriteLine(key);  
foreach (var value in degerler.Values)  
    Console.WriteLine(value);  
Console.WriteLine(degerler.ContainsKey("Anahtar"));  
Console.WriteLine(degerler.ContainsValue(1));
```

Listelerde olduğu gibi sözlüklerde de add ve count metotları bulunmaktadır. Keys özelliği sözlüğün anahtarlarının bir listesini verirken (List type değil), Values ise sözlüğün içerdiği değerleri döndürür. Eğer sözlükte bir anahtar veya bir değer bulunup bulunmadığı öğrenilmek isteniyorsa containskey ve containsvalue metotları kullanılabilir.

Koleksiyonlar (Hashset)

- Tek anahtar alanı <T>

```
HashSet<int> set = new HashSet<int>();
```

- Her eleman tekil

```
bool sonuc = set.Add(1); sonuc = true  
sonuc = set.Add(1); sonuc = false
```

- Elemanların varlığının kontrolü

```
bool varMi = set.Contains(1);
```

- Eleman silme

```
set.Remove(1);  
set.RemoveWhere(x => x > 5);
```

Hashset elemanları tekil olarak saklayan ve elemanların bulunup bulunmadığının kontrol edilmesi için kullanılan bir sınıftır. Hashset sınıfına eleman eklemek için add metodu kullanılır. Eleman eklendikten sonra bu elemanın bulunup bulunmadığının kontrolü contains metodu ile yapılmaktadır. Yine hashset üzerinde bir eleman silinmek isteniyorsa Remove kullanılabilir.

Koleksiyonlar (Sortedlist)

- Sıralama ölçütü ve sıralanacak değer şeklinde iki alan

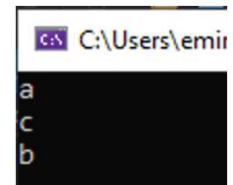
```
SortedList<int, string> siraliListe = new SortedList<int, string>();
```

- Değer eklemek için add metodu

```
siraliListe.Add(1, "a");  
siraliListe.Add(3, "b");  
siraliListe.Add(2, "c");
```

- Sıralanmış değerleri almak için values özelliği

```
List<string> degerler = siraliListe.Values.ToList();  
foreach (var deger in degerler)  
    Console.WriteLine(deger);
```



Sortedlist sıralanacak bir değişken ve karşılık gelen değer tanımlanabilen bir yapıdır. İlk parametrede verilen değişkene göre sıralama yapılmaktadır.

Koleksiyonlar (Queue)

- Kuyruk implementasyonu (ilk giren ilk çıkar fifo)

```
Queue<string> kuyruk = new Queue<string>();
```

- Kuyruğa ekleme

```
kuyruk.Enqueue("Eleman");
```

- Kuyruktan eleman alma

```
string alinan = kuyruk.Dequeue();
```

- Sıradaki elemana kuyruktan çıkarmadan bakma

```
string siradaki = kuyruk.Peek();
```

- Eleman sayısı

```
int sayi = kuyruk.Count;
```

Kuyruk veri yapısının implementasyonudur. İlk giren ilk çıkar mantığı ile çalışmaktadır. Enqueue ile kuyruğa eleman eklenirken dequeue ile kuyruktan eleman çıkartılır. Kuyrukta sıradaki elemanı çıkartmadan görebilmek için peek metodu kullanılabilir.

Koleksiyonlar (Stack)

- Lifo
- <T>
- Kuyruk metotları ile aynı

```
Stack<string> stack = new Stack<string>();  
stack.Push("Eleman");  
string alinan = stack.Pop();  
int sayi = stack.Count;  
string siradaki = stack.Peek();
```

Stack son giren ilk çıkar mantığı ile çalışır. Stack üzerinde push ve pop metotları kullanılabilir.