SAKARYA ÜNİVERSİTESİ

Nesneye Dayalı Programlama

Hafta 12

Prof. Dr. Ümit KOCABIÇAK
Öğr. Gör. Özgür ÇİFTÇİ

Bu ders içeriğinin basım, yayım ve satış hakları Sakarya Üniversitesi'ne aittir. "Uzaktan Öğretim" tekniğine uygun olarak hazırlanan bu ders içeriğinin bütün hakları saklıdır. İlgili kuruluştan izin almadan ders içeriğinin tümü ya da bölümleri mekanik, elektronik, fotokopi, manyetik kayıt veya başka şekillerde çoğaltılamaz, basılamaz ve dağıtılamaz.
Her hakkı saklıdır © 2010 Sakarya Üniversitesi

12.1. Dizi ve ArrayList

- Diziler tanımlanırken elaman sayısı belirtilen aynı türden verileri içersinde barındırabilen yapılardır.
- ArrayList ise dizinin yaptığı tüm işleri yapar ama sınır getirilmediğinden istediğimiz kadar eleman ekleyebiliriz. Aldığı her elemanı boxing işlemi ile object olarak sakladığından her türlü değeri alabilir. Add, Remove, Sort gibi metodlar kullanışlı metodlardır.

Örnekler;

```
class DiziArrayList
  {
    private int k = 0;
    private int[] dizi = new int[5];
    private ArrayList al = new ArrayList();

    public void DiziAta(int a)
    {
      dizi[k] = a;
      k++;
    }
    public void DiziAta(object a)
    {
      al.Add(a);
    }

    public void DiziYazdir(int[] a)
    {
      foreach (int elaman in a)
        Console.WriteLine(elaman);
    }

    public void DiziYazdir()
    {
      foreach (object elaman in al)
        Console.WriteLine(elaman);
    }

  }
```

```
class Program
  {
    static void Main(string[] args)
    {
      int[] yeniDizi={1,2,3,4,5};
      DiziArrayList d = new DiziArrayList();

      for (int i = 0; i < 5; i++)
      {
```

```
            d.DiziAta(yeniDizi[i]);
        }

        d.DiziYazdir(yeniDizi);

        string s = "123";
        int c = 123;

        d.DiziAta((object)s);
        d.DiziAta((object)c);

        d.DiziYazdir();

    }
}
```

Diziler aynı türden verileri tutarken, ArrayList ise object türünden verileri saklayabilmektedir. ArrayListler'de object türüne dönüştürme işlemi performans kaybına yol açmaktadır.

Yukarıdaki örneğimiz de ArrayList'e atamalar yaparken object türüne dönüşüm yapılmaktada, benzeri şekilde bir elamanın listeden çekimi sırasında da tür dönüşümüne gerek vardır.

```
string a = (string)al[0]; // unboxing işlemi
int b = (int)al[1]; //unboxing işlemi
```

Bu noktada çözüm ise içersinde güvenli tiplerin bulunduralabileceği Generic Yapılar'dır.

12.1. Generic Yapılar

İstisnalar haricinde C/C++ taki şablonların geliştirilmiş halidir. C# System.Collections.Generic isim uzayında yer almaktadır. Dolayısıyla Collections gurubuna ait yapılardır. <tür> şekilinde bildirimi yapılır. Yapının içersinde hangi tür verilerin tutulması isteniyorsa "tür" yerine veri tipi yazılır.

```
List<string> gls = new List<string>();
List<int> gli = new List<int>();
```

Generic Tiplerin kullanım amacı:

1- Type Safe (Tip Güvenliği)
2- Performance (Performans)

Generic Tiplerde şu işlemleri yapabiliyoruz:

1- Generic Class
2- Generic Interface
3- Generic Metod
4- Generic Delegate
5- Generic ve Array

Örnek 2:

```
class GenericSinifOrnegi
  {
    public List<string> gls = new List<string>();
    public List<int> gli = new List<int>();

    public void ElamanEkle(string a)
    {
      gls.Add(a);
    }

    public void ElamanEkle(int a)
    {
      gli.Add(a);
    }

    public void ElamanCikart(string a)
    {
      gls.Remove(a);
    }

    public void ElamanCikart(int a)
    {
      gli.Remove(a);
    }


    public void Yazdir(List<string> l)
    {
      foreach (string s in l)
        Console.WriteLine(s);
    }

    public void Yazdir(List<int> l)
    {
      foreach (int s in l)
        Console.WriteLine(s);
    }
  }
```

```
class Program
  {
    static void Main(string[] args)
    {

      GenericSinifOrnegi gs = new GenericSinifOrnegi();

      gs.ElamanEkle("ali");
      gs.ElamanEkle("veli");
      gs.ElamanEkle("zeynep ");
      gs.ElamanEkle("a");

      gs.Yazdir(gs.gls);

      Console.WriteLine("\n**********************\n");
```

```
        gs.ElamanCikart("a");

        gs.Yazdir(gs.gls);

        for (int i=1;i<3;i++)
           Console.WriteLine("************************");

        gs.ElamanEkle(5);
        gs.ElamanEkle(6);
        gs.ElamanEkle(7);

        gs.Yazdir(gs.gli);

        Console.WriteLine("\n**********************\n");
        gs.ElamanCikart(7);

        gs.Yazdir(gs.gli);

    }
  }
```

Örnek 3:

```
class Program
  {
    static void Main(string[] args)
    {
      Dictionary<int, string> yeniListe = new Dictionary<int, string>();

      yeniListe.Add(0, "a");
      yeniListe.Add(1, "b");

      yeniListe.Remove(0);
    }
  }
```

Örnek 4:

```
enum KullanimTuru
  {
    Is,
    ozel,
  };
```

```
enum Tip
  {
    fax,
    telefon,
     modem,
  };
```

```
class Tel
  {

    private KullanimTuru kt;
    private Tip t;
    private string UlkeKodu;
    private string telno;

    public Tel(KullanimTuru kt, Tip t, string UlkeKodu, string telno )
    {
      this.kt = kt;
      this.t = t;
      this.UlkeKodu = UlkeKodu;
      this.telno = telno;
    }
  }
```

```
class Ogrenci
  {
    public string m_ad;
    public string m_soyadi;
    public DateTime m_dt;
    public string numara;
    public List<Tel> Telefon = new List<Tel>();


    ///diğer Kodlar
  }
```

```
class Program
  {
    static void Main(string[] args)
    {
      Tel t1=new Tel(KullanimTuru.Is,Tip.fax,"0","11111111");
      Tel t2 = new Tel(KullanimTuru.ozel, Tip.telefon, "0", "11111111");
      Tel t3 = new Tel(KullanimTuru.ozel, Tip.modem, "0", "11111111");


      Ogrenci o = new Ogrenci();

      o.Telefon.Add(t1);
      o.Telefon.Add(t2);
```

```
        o.Telefon.Add(t3);
    }
  }
```

12.2. Generic Sınıflar

Generic sınıfların[1] bildiriminde sınıf isminden sonra < > işaretleri arasında tür parametresi belirtilir. Generic bir sınıf bildiriminin genel biçimi şöyledir :

```
class Sınıf_İsmi <Tür_Parametresi>
{
   // ...
}
```

Tür parametresi; bildirim sırasında herhangi bir türün karşılığı olmayan ve çoğu zaman T, G, AD gibi bir yer tutucudur. Bu yer tutucu(lar) sınıfın kullanımı noktasında bir tür belirtilmesi ile anlam kazanır(lar).

Örneğin;

```
class Sample <T>
{
   // ...
}

// ....
public static void Main()
{
   Sample <int> obj = new Sample <int> ();
}
```

Yukarıdaki örnekte bildirim sırasında tür parametresi olarak kullanılan <T>, sınıfın kullanımı noktasında <int> türünün belirtilmesi ile anlam kazanmıştır.

NOT : Terminolojide Sample <int> gibi tanımlanan türler; **constructed type** biçiminde adlandırılmaktadır !

Çok tipli generic sınıfların bildiriminde birden fazla tür parametresi kullanılabilir, örneğin :

```
class Sample <AB, CD>
{
   // ...
```

[1] http://www.yazgelistir.com/Makaleler/1000000369.ygpx

```
}

// ....
public static void Main()
{
   Sample <int, string> obj = new Sample <int, string> ();
}
```

Bu örnekte ise; **AB -> int , CD -> string** türlerinin karşılığı olmuşlardır.

Örnek :

```
class Cocuk<T, K>
 {
    T m_id;
    T m_agirligi;
    K m_Adi;
    K m_Soyadi;

    public T Id
    {
       get { return m_id; }
       set { m_id = value; }
    }
    public T Agirligi
    {
       get { return m_agirligi; }
       set { m_agirligi = value; }
    }
    public K Adi
    {
       get { return m_Adi; }
       set { m_Adi = value; }
    }
    public K Soyadi
    {
       get { return m_Soyadi; }
       set { m_Soyadi = value; }
    }
    public Cocuk()
    {
    }

    public Cocuk(T id, T agirligi, K adi, K soyadi)
    {
       this.Id = id;
       this.Adi = adi;
       this.Soyadi = soyadi;
       this.Agirligi = agirligi;
    }
 }
```

```
class Program
{
    static void Main(string[] args)
    {
        Cocuk<byte, string> adam = new Cocuk<byte, string>(1, 250, "mehmet", "gedik");
        Cocuk<string, string> adam2 = new Cocuk<string, string>("100", "450", " nurhan", "özay");
    }
}
```

12.2. Generic Metotlar

```
class Program
{
    static void Main(string[] args)
    {
        Topla<int>(5, 3);
        Topla<byte>(1, 2);
        Topla<int, byte>(5, 9);
        int d = Topla<int>(5);
        int a = 21;
        int b = 45;
        Console.WriteLine(a + " " + b);
    }

    static void Topla<T>(T a, T b)
    {
        Console.WriteLine("Generic Metot Calisti");
    }
    static void Topla<T, K>(T a, K b)
    {
        Console.WriteLine("Farkli tipteki generic Calisti");
    }
    static T Topla<T>(T a)
    {
        Console.WriteLine("Geriye Deger Dondur");
        return a;
    }
}
```

12.4. Kısıtlamalar (Constraints) Kullanımlar

Generic mimarisi gönderilen parametrelerin tipinin seçilmesini kullanıcıya bırakır. Dolayısı ile kullanıcı istediği tipten veri girişi yapabilir. Bu yüzden bazı durumlarda kasıtlamalar koymak gerekebilir.

Koşul	**Syntax**
Değer tipi olma zorunluluğu	where Tip : struct
Referans tipi olma zorunluluğu	where Tip : class
Constructor zorunluluğu	where Tip : new()
Türeme zorunluluğu	where Tip : <Temel Sınıf>
Interface zorunluluğu	where Tip : <Interface>

```
class insan<T, K>
      where K : class
      where T : struct
   {
      public T m_Id;
      public K m_Adi;
      public K m_Soyadi;

      public insan()
      {

      }
   }
```

```
static void BuyukOlaniYaz<T>(T a, T b) where T : IComparable
      {
         T buyukolan = a;
         if (a.CompareTo(b) < 0)
            buyukolan = b;

         Console.WriteLine("Buyuk Olan : {0}", buyukolan);
      }
```

```
public class MyGeneric<Tip> where Tip : MyClass, MyInterface, new()
   {
      // Bazı kodlar
   }
```

Son duruma göre Tip ile belirteceğimiz class tipi, MyClass class'ından türemeli, MyInterface interface'ini implemente etmeli ve mutlaka parametresiz bir constructor' ı olmalı.