ÇOKBİÇİMLİLİK (POLİMORFİZM), JENERİKLİK

Çokbiçimlilik Kavramı

Bir nesne yönelimli programlama kavramı olarak ele alındığında çokbiçimlilik, bir temel sınıftan miras alan farklı sınıfların, almış oldukları mirasın belirli bir kısmını kendi ihtiyaçları dâhilinde özelleştirerek değiştirmelerine verilen isimdir. Aşağıdaki örneği inceleyelim:

```
using System;

class Matematik
{
    protected int x;
    protected int y;

    public Matematik(int a, int b)
    {
        x = a;
        y = b;
    }
    public virtual void Hesapla()
    { // virtual: miras alan siniflar bu metodu gecersiz kilabilsin
        Console.WriteLine("Temel Sinif Hesapla Metodu");
    }
}
class Toplama:Matematik
{
    public Toplama(int a, int b):base(a,b)
    {
        // Toplama sinifinin iki arguman ("a" ve "b") alan yapici metodu.
        // temel sinifin 2 arguman alan yapici metodunun aynisini
        // gercekle; ilk argumanina "a" yi ver, ikincisine "b" yi ver.
        // SONUC: x=a, y=b
    }
    public override void Hesapla()
    { // mirasla alinan "Hesapla" metodu gecersiz kilinsin
        Console.WriteLine("Toplama : {0}", x+y);
    }
}
class Cikarma:Matematik
{
    public Cikarma(int a, int b):base(a,b)
    {
    }
    public override void Hesapla()
    { // mirasla alinan "Hesapla" metodu gecersiz kilinsin
        Console.WriteLine("Cikarma : {0}", x-y);
    }
}
class Uygulama
{
    static void IslemYap(Matematik m)
    {
        // m: temel siniftan miras alan herhangi bir sinifin referansini
        // temsil eder. Cokbicimlilik sayesinde bu fonksiyona arguman olarak
        // gerek "Toplama" tipinden, gerekse "Cikarma" tipinden referanslar verilebilir.
        m.Hesapla();
    }
    static void Main(string[] args)
    {
        Toplama t = new Toplama(10, 2);
        Cikarma c = new Cikarma(10, 2);

        IslemYap(t);
        IslemYap(c);
        Console.ReadLine();
    }
}
```

Çıktı:

```
Toplama : 12
Cikarma : 8
```

Yukarıdaki örnekte, **Matematik** isminde bir soyut sınıf verilmiştir. Soyut sınıflar; miras bırakabilen ancak nesneleri yaratılamayan sınıflardır. Bu sınıfın içerisinde, **protected** erişimli **x** ve **y** üyelerinin yanı sıra **Hesapla** isminde bir fonksiyon bulunmaktadır. Fonksiyonun gerçekleştiriminde, dönüş değerinden önce **virtual** anahtar sözcüğü kullanılmıştır. Bu, "Bu sınıftan miras alan diğer sınıflar **Hesapla** fonksiyonunu yeniden gerçekleyerek, **Matematik** sınıfından mirasla aldıkları **Hesapla** fonksiyonunu geçersiz kılabilirler." anlamını taşımaktadır.

Matematik sınıfından miras alan **Toplama** sınıfında bir yapıcı metot ve bir de **Hesapla** metodu yer almaktadır. Burdaki yapıcı metot gerçeklenirken, temel sınıf olan **Matematik** sınıfının iki argüman alan yapıcı metodun yaptığı işlemin aynısını yapması istenmiştir. Bu sebeple **base** anahtar sözcüğü yardımıyla temel sınıfın yapıcı metodunun aynısının gerçeklenerek (bir bakıma metotlararası miras alınarak) kullanılması sağlanmıştır. **Matematik** sınıfının **a** ve **b** üyelerinin **Toplama** sınıfına mirasla geçtiği düşünüldüğünde, **Toplama** sınıfının yapıcı metodu da **x** ve **y**' ye sırasıyla argüman olarak almış olduğu **a** ve **b** ile taşınan değerleri atayacaktır. Bu sınıf içerisinde tanımlanan **Hesapla** metodunun dönüş değerinin önünde de **override** anahtar sözcüğü bulunmaktadır. Bu da, temel sınıftan miras olarak alınan **Hesapla** metodunun geçersiz kılındığı anlamına gelmektedir. Benzer durum, **Cikarma** sınıfı için de geçerlidir.

Çokbiçimlilik kavramını daha somut bir biçimde anlamak için, **Uygulama** sınıfı içerisindeki **IslemYap** fonksiyonunu incelemek gerekir. Bu fonksiyon 1 argüman almakta ve değer döndürmemektedir. Aldığı argümanın tipi ise, temel sınıf olan **Matematik**' tir. Fonksiyon, **Matematik** tipinden olan bu argümanı bir nesne tutacağı gibi kullanmış, bu nesnenin bulunduğu sınıfta **Hesapla** isminde ve statik olmayan bir fonksiyon varmışçasına bu fonksiyonu çağırmıştır. Oysa dikkatli düşünüldüğünde **Matematik** sınıfı bir soyut sınıftır ve nesnesi yaratılamamaktadır.

IslemYap fonksiyonunun argüman olarak aldığı ve **Matematik** tipinden olan **m** argümanı, aslında **Matematik** sınıfından miras alan herhangi bir sınıfın nesnesini gösteren referans değişkenini temsil edebilecek nitelikte temel bir tutacaktır. **Uygulama** sınıfı içerisindeki **Main** fonksiyonu incelendiğinde bu durum netlik kazanacaktır. **Main** fonksiyonunda **Toplama** ve **Cikarma** sınıflarının birer nesnesi yaratılmış ve bu nesneler sırasıyla **t** ve **c**

referans tutacaklarıyla refere edilmişlerdir. Sonrasında ise, iki farklı tipten olan **t** ve **c** değişkenleri, ikisi de aynı tipe sahipmişçesine **IslemYap** fonksiyonuna verilmiştir.

IslemYap fonksiyonunun **Toplama** ve **Cikarma** sınıfları içindeki iki ayrı **Hesapla** fonksiyonundan hangisini çağıracağı, programın çalışma anında belli olacaktır. **IslemYap** fonksiyonuna argüman olarak verilen referansın tipi **Toplama** ise **Toplama** sınıfına ait **Hesapla** fonksiyonu, **Cikarma** ise de **Cikarma** sınıfına ait **Hesapla** fonksiyonu çağrılmış olacaktır. Böylelikle; **Toplama** sınıfı nesneleri için ayrı, **Cikarma** sınıfı nesneleri içinse ayrı olmak üzere iki farklı fonksiyon tanımlanmamış, **IslemYap** fonksiyonu ***çokbiçimlilik*** sayesinde hem **Toplama** sınıfı nesnelerini, hem de **cikarma** sınıfı nesnelerini argüman olarak kabul etmiş ve bu nesneler üzerinden **Hesapla** fonksiyonunu çağırmıştır.

```
class Cikarma:Matematik
{
    public Cikarma(int a, int b):base(a,b)
    {
    }
    public new void Hesapla()
    {
        Console.WriteLine("Cikarma : {0}", x-y);
    }
}
```

Çıktı:

```
Toplama : 12
Temel Sinif Hesapla Metodu
```

Eğer temel sınıftan (**Matematik**) miras alan **Cikarma** sınıfındaki **Hesapla** metodunun geçersiz kılınması istenirse, bu metodun dönüş değerinin önündeki **override** anahtar sözcüğü kaldırılarak yerine **new** anahtar sözcüğü koyulur. **override** ve **new** anahtar sözcüklerinin hiçbirinin yazılmaması durumunda miras alan sınıf içerisindeki metot gene geçersiz kılınmış olacaktır (Ancak derleyici uyarı mesajı verecektir.):

```
class Cikarma:Matematik
{
    public Cikarma(int a, int b):base(a,b)
    {
    }
    public void Hesapla()
    {
        Console.WriteLine("Cikarma : {0}", x-y);
    }
}
```

Çıktı:

```
Toplama : 12
Temel Sinif Hesapla Metodu
```

Aşağıda verilen örnekte ise **TestSinifi** isminde dışsal bir sınıf içerisinde **Boyutlar** isminde bir içsel sınıf ve bu içsel sınıftan miras alan **Daire**, **Kure** ve **Silindir** isminde üç adet içsel sınıf bulunmaktadır. Temel sınıf olan **Boyutlar** sınıfı soyut olmadığı halde, bir önceki örnektekine benzer bir kalıtlama ilişkisi kurulabilmektedir (**virtual** ve **override** anahtar sözcüklerine dikkat ediniz.).

```
using System;

class TestSinifi
{
    public class Boyutlar
    {
        public const double PI = Math.PI; // sabit
        protected double x, y;

        public Boyutlar()
        { // x ve y' ye varsayilan tamsayi degerleri (0) atanacak
        }
        public Boyutlar(double x, double y)
        { // this.x --> sinifin uyesi olan "x"; x --> arguman olan "x"
            this.x = x;
            this.y = y;
        }
        public virtual double Alan()
        { // sadece dikdortgen icin gecerli bir alan fonksiyonu
            return x * y;
        }
    }
    public class Daire : Boyutlar
    {
        // x ve y mirasla geldi
        public Daire(double r)
            : base(r, 0)
        {
            // Daire sinifinin tek arguman ("r") alan yapici metodu.
            // temel sinifin 2 arguman alan yapici metodunun aynisini
            // gercekle; ilk argumanina "r" yi ver, ikincisine "0" ver.
            // SONUC: x=r, y=0
        }
        public override double Alan()
        { // Daire sinifi icin ozellesmis bir Alan fonksiyonu
            return PI * x * x;
        }
    }
    class Kure : Boyutlar
    {
        public Kure(double r)
            : base(r, 0)
        {
        }
        public override double Alan()
        {
            return 4 * PI * x * x;
        }
    }
    class Silindir : Boyutlar
    {
        public Silindir(double r, double h)
            : base(r, h)
        {
        }
        public override double Alan()
        {
            return 2 * PI * x * x + 2 * PI * x * y;
        }
    }
    static void Main()
    {
        double r = 3.0, h = 5.0;
        Boyutlar d = new Daire(r);
        Boyutlar k = new Kure(r);
        Boyutlar s = new Silindir(r, h);

        Console.WriteLine("Daire alani    = {0:F2}\tbirimkare", d.Alan());
        Console.WriteLine("Kure alani     = {0:F2}\tbirimkare", k.Alan());
        Console.WriteLine("Silindir alani = {0:F2}\tbirimkare", s.Alan());
        Console.ReadLine();
    }
}
// kaynak: http://msdn.microsoft.com/en-us/library/9fkccyh4(v=vs.80).aspx
```

Çıktı:

```
Daire alani    = 28,27  birimkare
Kure alani     = 113,10 birimkare
Silindir alani = 150,80 birimkare
```

Main fonksiyonuna bakıldığında; **Daire**, **Kure** ve **Silindir** sınıflarının birer nesnelerinin yaratıldığı ve bu nesnelerin üçünün de **Boyutlar** (temel sınıf) tipinden referans tutacaklarıyla (**d**, **k**, **s**) refere edilebildiği görülmektedir. Bu referans tutacaklarının her üçünün de tipi **Boyutlar** olsa da, refere ettikleri nesnelerin tipleri birbirinden farklı olduğu için, bu tutacaklara mesaj gönderilmesiyle çağrılan **Alan** fonksiyonları da farklı farklı olacaktır.

Jeneriklik

Jeneriklik özetle, belli bir sınıfa ait nesneler içerisinde hangi tipte veriler tutulacağına çalışma anında karar verilmesine olanak sağlayan bir programlama yaklaşımıdır. Kodun yeniden kullanılmasına olanak sağlar. Aşağıdaki programı inceleyelim:

```
using System;

public class DiziInt
{ // 32 bit tamsayilar icin tasarlanmis DiziInt sinifi
    private int indeks;
    private int boyut;
    private int[] dizi;

    public DiziInt(int b)
    { // yapici metot
        indeks = 0;
        boyut = b;
        dizi = new int[boyut];
    }
    public void SonaEkle(int eleman)
    {
        if (indeks < boyut)
        {
            dizi[indeks++] = eleman;
        }
    }
    public int SondanCikar()
    {
        if (indeks > 0)
        {
            return dizi[--indeks];
        }
        else
            return 0;
    }
    public void Yazdir()
    {
        for (int i = 0; i < boyut; i++)
            Console.WriteLine("{0}", dizi[i]);
    }
}
public class Sinif1
{
    static void Main()
    {
        DiziInt TamsayiDizisi = new DiziInt(3);
        TamsayiDizisi.SonaEkle(7);
        TamsayiDizisi.SonaEkle(12);
        int i1 = TamsayiDizisi.SondanCikar();
        Console.WriteLine("####### i1 : {0}", i1);
        TamsayiDizisi.SonaEkle(24);
        TamsayiDizisi.SonaEkle(25);
        TamsayiDizisi.SonaEkle(26);
        TamsayiDizisi.Yazdir();

        Console.ReadLine();
    }
}
```

Çıktı:

```
######## i1 : 12
7
24
25
```

Verilen programda, basit bir veri yapısı tasarlanmıştır. **Sinif1** sınıfındaki **Main** fonksiyonunun içeriğine bakıldığında; **DiziInt** sınıfının bir nesnesi yaratılarak bu nesne içerisinde **int** tipinden tamsayıların belli bir kurala göre tutulduğu görülmektedir. **DiziInt** sınıfı nesnesinin **dizi** isimli alt alanı, belirli bir kapasiteye sahip bir 32 bitlik tamsayıları tutmak için kullanılan **int** tipinden bir dizidir. Bu nesne üzerinden çağrılan **SonaEkle** fonksiyonu dizinin ilk elemanından başlamak kaydıyla boş olan en küçük indeksine argüman olarak aldığı tamsayı değerini eklemekte, **SondanCikar** fonksiyonu ise diziye en son eklenmiş olan değeri döndürerek bu değerin bulunduğu indeksi "yeni eleman eklenebilir" kılmaktadır. **yazdir** fonksiyonu ise dizinin içeriğini ekrana yazdırmaktadır.

Jeneriklik söz konusu olmadığında, yukarıdaki örnekte verilen ve **int** tipinden verileri saklamak için kullanılan veri yapısını **string** tipinden veriler için kullanmak istediğimizde, aşağıdaki gibi yeni bir sınıf tasarlamak gerekecektir:

```
using System;

public class DiziString
{ // karakter katarlari icin tasarlanmis DiziString sinifi
    private int indeks;
    private int boyut;
    private string[] dizi;

    public DiziString(int b)
    { // yapici metot
        indeks = 0;
        boyut = b;
        dizi = new string[boyut];
    }

    public void SonaEkle(string eleman)
    {
        if (indeks < boyut)
        {
            dizi[indeks++] = eleman;
        }
    }

    public string SondanCikar()
    {
        if (indeks > 0)
        {
            return dizi[--indeks];
        }
        else
            return null;
    }

    public void Yazdir()
    {
        for (int i = 0; i < boyut; i++)
            Console.WriteLine("{0}", dizi[i]);
    }
}
```

Çıktı:

```
######## s1 : oniki
yedi
yirmidort
yirmibes
yirmialti
```

(devam)

```
public class Sinif1
{
    static void Main()
    {
        DiziString KaraterKatariDizisi = new DiziString(5);
        KaraterKatariDizisi.SonaEkle("yedi");
        KaraterKatariDizisi.SonaEkle("oniki");
        string s1 = KaraterKatariDizisi.SondanCikar();
        Console.WriteLine("####### s1 : {0}", s1);
        KaraterKatariDizisi.SonaEkle("yirmidort");
        KaraterKatariDizisi.SonaEkle("yirmibes");
        KaraterKatariDizisi.SonaEkle("yirmialti");
        KaraterKatariDizisi.Yazdir();

        Console.ReadLine();
    }
}
```

Görüldüğü gibi, **DiziInt** sınıfı ile **DiziString** sınıfının gerçeklenmesinde kullanılan kodlar, tip belirleyicileri (**int**, **string**) haricinde hemen hemen aynıdır. Bahsedilen programlama yaklaşımı, jenerik olmayan bir yaklaşımdır. **int** tipinden verileri saklamak için kullanılan **DiziInt** nesneleri, **string** tipindeki verileri saklayamamakta, bu durum da **string** tipinden verileri saklamak için **DiziString** sınıfının oluşturulmasını ve bu sınıfa ait nesnelerin kullanılmasını zorunlu kılmaktadır.

Jenerik programlama yaklaşımıyla, yukarıda bahsedilen "iki farklı veri tipi için iki farklı sınıf tasarlama zorunluluğu" ve bu durumun doğurmuş olduğu "benzer gerçekleştirimlerin tekrar tekrar yapılması" durumu ortadan kaldırılmış olacaktır. Aşağıdaki örneği inceleyelim:

```
using System;

public class Dizi<T>
{ // Jenerik dizi (T: herhangi bir tip)
    private int indeks;
    private int boyut;
    private T[] dizi;

    public Dizi(int b)
    { // yapici metot
        indeks = 0;
        boyut = b;
        dizi = new T[boyut];
    }
    public void SonaEkle(T eleman)
    {
        if (indeks < boyut)
        {
            dizi[indeks++] = eleman;
        }
    }
    public T SondanCikar()
    {
        if (indeks > 0)
        {
            return dizi[--indeks];
        }
        else
            return default(T);
        // T tipinin varsayilan degeri (int ise "0", string ise "null")
    }
```

Çıktı:

```
####### i1 : 12
7
24
25
**********
####### s1 : oniki
yedi
yirmidort
yirmibes
yirmialti
```

(devam)

```
    public void Yazdir()
    {
        for (int i = 0; i < boyut; i++)
            Console.WriteLine("{0}", dizi[i]);
    }
}

public class Sinif1
{
    static void Main()
    {
        Dizi<int> TamsayiDizisi = new Dizi<int>(3);
        TamsayiDizisi.SonaEkle(7);
        TamsayiDizisi.SonaEkle(12);
        int i1 = TamsayiDizisi.SondanCikar();
        Console.WriteLine("####### i1 : {0}", i1);
        TamsayiDizisi.SonaEkle(24);
        TamsayiDizisi.SonaEkle(25);
        TamsayiDizisi.SonaEkle(26);
        TamsayiDizisi.Yazdir();

        Console.WriteLine("********************");

        Dizi<string> KaraterKatariDizisi = new Dizi<string>(5);
        KaraterKatariDizisi.SonaEkle("yedi");
        KaraterKatariDizisi.SonaEkle("oniki");
        string s1 = KaraterKatariDizisi.SondanCikar();
        Console.WriteLine("####### s1 : {0}", s1);
        KaraterKatariDizisi.SonaEkle("yirmidort");
        KaraterKatariDizisi.SonaEkle("yirmibes");
        KaraterKatariDizisi.SonaEkle("yirmialti");
        KaraterKatariDizisi.Yazdir();

        Console.ReadLine();
    }
}
```

Yukarıdaki örnekte, jeneriklik özelliğinden yararlanılmıştır. **Dizi** sınıfının tanımlandığı satıra bakıldığında, sınıf isminin hemen sağında "**<T>**" ifadesine rastlanacaktır. Bu, ilgili sınıf içerisinde "**T**" adında bir değişken olacağı ve bunun **tip** (int, string, long vs.) tutan bir değişken olarak görev yapacağı anlamını taşımaktadır. Sınıfın içerisinde değişken tanımlamaları yapılırken kimi yerlerde **T**' nin kullanıldığına dikkat ediniz. Bu sınıfın nesneleri yaratılırken de, **T**' nin hangi tipe karşılık geleceği bildirilecektir.

Sinif1 sınıfı içerisinde, **Dizi** sınıfının **TamsayiDizisi** ve **KarakterKatariDizisi** isimlerinde iki farklı nesnesi yaratılmıştır. **TamsayiDizisi** isimli nesne yaratılırken, **Dizi** sınıfında kullanılan **T** değişkeninin **int** tipine karşılık geldiği; **KarakterKatariDizisi** isimli nesne yaratılırken de **T**' nin **string** tipie karşılık geldiği belirtilmiştir.

Sonuç olarak, **int** tipinden verileri saklamak için ayrı, **string** tipinden verileri saklamak içinse ayrı bir sınıf gerçekleştirimi yapılması zorunluluğu ortadan kalkmış, jeneriklik sayesinde tek bir sınıf tasarımıyla her iki tipte verilerin saklanabileceği nesnelerin üretilebilmesi mümkün

kılınmıştır. **TamsayiDizisi** nesnesi **int** tipinden verileri saklayabilirken **KarakterKatariDizisi** nesnesi de **string** tipinden verileri saklayabilmektedir.

Jeneriklik söz konusu olduğunda, yukarıda **int**, **string** gibi tiplere sahip olan veriler için geçerli olan durum, sınıf referansları için de geçerli olmaktadır:

```
using System;

public class Dizi<T>
{ // Jenerik dizi (T: herhangi bir tip)
    private int indeks;
    private int boyut;
    public T[] dizi;

    public Dizi(int b)
    { // yapici metot
        indeks = 0;
        boyut = b;
        dizi = new T[boyut];
    }
    public void SonaEkle(T eleman)
    {
        if (indeks < boyut)
        {
            dizi[indeks++] = eleman;
        }
    }
    public T SondanCikar()
    {
        if (indeks > 0)
        {
            return dizi[--indeks];
        }
        else
            return default(T);
        // T tipinin varsayilan degeri (int ise "0", string ise "null")
    }
}

public class Sinif1
{
    public int a, b;
    public Sinif1()
    {
        a = 0;
        b = 0;
    }
    public Sinif1(int arg1, int arg2)
    {
        a = arg1;
        b = arg2;
    }
}

public class Sinif2
{
    public string x, y;
    public Sinif2()
    {
        x = "";
        y = "";
    }
    public Sinif2(string arg1, string arg2)
    {
        x = arg1;
        y = arg2;
    }
}
```

Çıktı:

```
a : 5           b : 6
**************************
x : bes         y : alti
```

(devam)

```
public class Sinif3
{
    static void Main()
    {
        Sinif1 nesne1 = new Sinif1(1, 2);
        Sinif1 nesne2 = new Sinif1(3, 4);
        Sinif1 nesne3 = new Sinif1(5, 6);
        Sinif1 nesne4 = new Sinif1(7, 8);

        Dizi<Sinif1> Sinif1Dizisi = new Dizi<Sinif1>(3); // T = Sinif1
        Sinif1Dizisi.SonaEkle(nesne1);
        Sinif1Dizisi.SonaEkle(nesne2);
        Sinif1Dizisi.SonaEkle(nesne3);
        Sinif1Dizisi.SonaEkle(nesne4);
        Console.WriteLine("a : {0}\t\tb : {1}", Sinif1Dizisi.dizi[2].a, Sinif1Dizisi.dizi[2].b);

        Console.WriteLine("************************");

        Sinif2 nesne5 = new Sinif2("bir", "iki");
        Sinif2 nesne6 = new Sinif2("uc", "dort");
        Sinif2 nesne7 = new Sinif2("bes", "alti");
        Sinif2 nesne8 = new Sinif2("yedi", "sekiz");

        Dizi<Sinif2> Sinif2Dizisi = new Dizi<Sinif2>(3); // T = Sinif2
        Sinif2Dizisi.SonaEkle(nesne5);
        Sinif2Dizisi.SonaEkle(nesne6);
        Sinif2Dizisi.SonaEkle(nesne7);
        Sinif2Dizisi.SonaEkle(nesne8);
        Console.WriteLine("x : {0}\t\ty : {1}", Sinif2Dizisi.dizi[2].x, Sinif2Dizisi.dizi[2].y);

        Console.ReadLine();
    }
}
```

Hazır Jenerik Sınıflar

System.Collections.Generic isim uzayı içerisinde, jenerik yapıda tasarlanmış olan hazır sınıflara rastlamak mümkündür.

Jenerik Kuyruk (Queue) Sınıfı

"kuyruk" veri yapısının gerçeklenmesi amacıyla tasarlanmış olan jenerik kuyruk sınıfının (**Queue<>**) kullanıldığı bir örnek program aşağıda verilmiştir:

```
using System;
using System.Collections.Generic;

public class Sinif1
{

    static void Main()
    {
        // jenerik Queue sinifinin "int"
        // tipinden eleman kabul eden nesnesi
        Queue<int> q_int = new Queue<int>();

        for (int i = 0; i < 10; i++)
            q_int.Enqueue(i*i);

        for (int i = 0; i < 6; i++)
            Console.WriteLine("q_int --> {0}. eleman\t: {1}", i+1, q_int.Dequeue());

        Console.WriteLine("********************************");

        string alfabe = "abcdefghijklmnopqrstuvwxyz";

        // jenerik Queue sinifinin "string" tipinden eleman kabul eden nesnesi
        Queue<string> q_str = new Queue<string>();

        for (int i = 0; i < 8; i++)
            q_str.Enqueue(alfabe.Substring(i, 4));

        for (int i = 0; i < 8; i++)
            Console.WriteLine("q_str --> {0}. eleman\t: {1}", i + 1, q_str.Dequeue());

        Console.ReadLine();
    }
}
```

Çıktı:

```
q_int --> 1. eleman     : 0
q_int --> 2. eleman     : 1
q_int --> 3. eleman     : 4
q_int --> 4. eleman     : 9
q_int --> 5. eleman     : 16
q_int --> 6. eleman     : 25
********************************
q_str --> 1. eleman     : abcd
q_str --> 2. eleman     : bcde
q_str --> 3. eleman     : cdef
q_str --> 4. eleman     : defg
q_str --> 5. eleman     : efgh
q_str --> 6. eleman     : fghi
q_str --> 7. eleman     : ghij
q_str --> 8. eleman     : hijk
```

Jenerik Yığıt (Stack) Sınıfı

"yığıt" veri yapısının gerçeklenmesi amacıyla tasarlanmış olan jenerik kuyruk sınıfının (**Stack<>**) kullanıldığı bir örnek program aşağıda verilmiştir:

```
using System;
using System.Collections.Generic;

public class Sinif1
{
    static void Main()
    {
        Stack<int> stk = new Stack<int>();

        for (int i = 0; i < 10; i++ )
            stk.Push(i * i);

        for (int i = 0; i < 5; i++)
            stk.Pop();

        Console.WriteLine("Tepedeki eleman\t:\t{0}", stk.Peek());
        Console.ReadLine();
    }
}
```

Çıktı:

```
Tepedeki eleman :       16
```

Aşağıda ise, nispeten daha karmaşık bir örnek verilmiştir:

```
using System;
using System.Collections; // jenerik olmayan Stack sinifi burada
using System.Collections.Generic; // jenerik Stack sinifi burada

public class Sinif1
{
    static void Main()
    {
        // "jenerik olmayan Stack sinifi" tipinden uyeleri tutmak
        // uzere bir jenerik Stack<> sinifi nesnesi yarat
        Stack<Stack> JenStk = new Stack<Stack>();

        Stack IntStk = new Stack();

        for (int i = 0; i < 10; i++ )
            IntStk.Push((object) (i * i));

        string alfabe = "abcdefghijklmnopqrstuvwxyz";
        Stack StrStk = new Stack();

        for (int i = 0; i < 5; i++)
            StrStk.Push((object) alfabe.Substring(i, 3));

        // JenStk yigitinda IntStk ve StrStk referanslari olacak
        JenStk.Push(IntStk);
        JenStk.Push(StrStk);

        Console.WriteLine("{0}", JenStk.Peek().Pop());
        Console.WriteLine("{0}", JenStk.Peek().Pop());
        Console.WriteLine("{0}", JenStk.Peek().Pop());
        Console.WriteLine("{0}", JenStk.Pop().Peek());
        Console.WriteLine();
        Console.WriteLine("{0}", JenStk.Peek().Pop());
        Console.WriteLine("{0}", JenStk.Peek().Pop());
        Console.WriteLine("{0}", JenStk.Peek().Pop());

        Console.ReadLine();

    }
}
```

Çıktı:

```
efg
def
cde
bcd

81
64
49
```

Jenerik Liste (List) Sınıfı

"liste" veri yapısının gerçeklenmesi amacıyla tasarlanmış olan jenerik liste sınıfının (**List<>**) kullanıldığı bir örnek program aşağıda verilmiştir:

```
using System;
using System.Collections; // jenerik olmayan Stack sinifi burada
using System.Collections.Generic; // jenerik Stack sinifi burada

public class Sinif1
{
    static void Main()
    {
        List<double> lst = new List<double>();

        for (int i = 0; i < 10; i++)
            lst.Add(0.111 * i);
        Console.WriteLine("1. durumda lst[4] : {0}", lst[4]);

        lst.RemoveAt(4);
        Console.WriteLine("2. durumda lst[4] : {0}", lst[4]);

        lst.Insert(1, 0.678);
        Console.WriteLine("3. durumda lst[4] : {0}", lst[4]);

        lst.Sort();
        Console.WriteLine("4. durumda lst[4] : {0}", lst[4]);

        lst.Reverse();
        Console.WriteLine("5. durumda lst[4] : {0}", lst[4]);

        Console.ReadLine();

    }
}
```

Çıktı:

```
1. durumda lst[4] : 0,444
2. durumda lst[4] : 0,555
3. durumda lst[4] : 0,333
4. durumda lst[4] : 0,555
5. durumda lst[4] : 0,666
```

Jenerik Sözlük (Dictionary) Sınıfı

System.Collections.Generic isim uzayı içerisindeki jenerik sınıflardan birisi de **Dictionary<>** sınıfı olup, "sözlük" veri yapısının kullanımına olanak sağlamak amacı ile tasarlanmıştır. **Dictionary<>** sınıfında iki tane tip değişkeni (anahtar tipi ve değer tipi için) olduğuna dikkat ediniz:

```
using System;
using System.Collections.Generic;

public class Sinif1
{
    static void Main()
    {
        Dictionary<string, string> TurIng = new Dictionary<string, string>();

        TurIng["kedi"] = "cat";
        TurIng["tavuk"] = "chicken";
        TurIng["ari"] = "bee";
        TurIng["kaplumbaga"] = "turtle";
        TurIng["tilki"] = "fox";
        TurIng["tavuk"] = "hen";

        Console.WriteLine("[TR] tilki\t: [EN] {0}", TurIng["tilki"]);
        Console.WriteLine("[TR] tavuk\t: [EN] {0}", TurIng["tavuk"]);
        Console.WriteLine();


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

        MetinSayi["sifir"] = 0;
        MetinSayi["bir"] = 1;
        MetinSayi["iki"] = 2;
        MetinSayi["uc"] = MetinSayi["iki"] + MetinSayi["bir"];

        Console.WriteLine("uc\t:\t{0}", MetinSayi["uc"]);
        Console.WriteLine();


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

        SayiMetin[3] = "uc";
        SayiMetin[7] = "yedi";
        SayiMetin[9] = "dokuz";
        SayiMetin[-9] = "eksi_" + SayiMetin[9];
        // dizilerde "negatif indeks" olamaz ancak sozluklerde durum farklidir

        Console.WriteLine("-9\t:\t{0}", SayiMetin[-9]);
        Console.WriteLine();

        Dictionary<short, bool> AsalMi = new Dictionary<short, bool>();

        AsalMi[0] = false;
        AsalMi[2] = true;
        AsalMi[7] = true;
        AsalMi[21] = false;
        AsalMi[83] = true;

        Console.WriteLine("2  --> {0}", AsalMi[2]);
        Console.WriteLine("21 --> {0}", AsalMi[21]);

        Console.ReadLine();

    }
}
```

Çıktı:

```
[TR] tilki      : [EN] fox
[TR] tavuk      : [EN] hen

uc      :       3

-9      :       eksi_dokuz

2  --> True
21 --> False
```