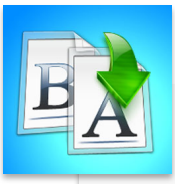
KOLEKSİYONLAR II



İÇİNDEKİLER

- Generic Koleksiyonlar
  - List Sınıfı
  - Stack Sınıfı
  - Queue Sınıfı
  - LinkedList Sınıfı
  - Dictionary Sınıfı

HEDEFLER

- Bu üniteyi çalıştıktan sonra;
  - Generic koleksiyonlar hakkında bilgi sahibi olabilecek,
  - List, Stack, Queuei LinkedList ve Dictionary sınıflarını tanımlayabilecek,
  - Bu sınıflarla beraber sıklıkla kullanılan metotları örnekler eşliğinde öğrenebileceksiniz.

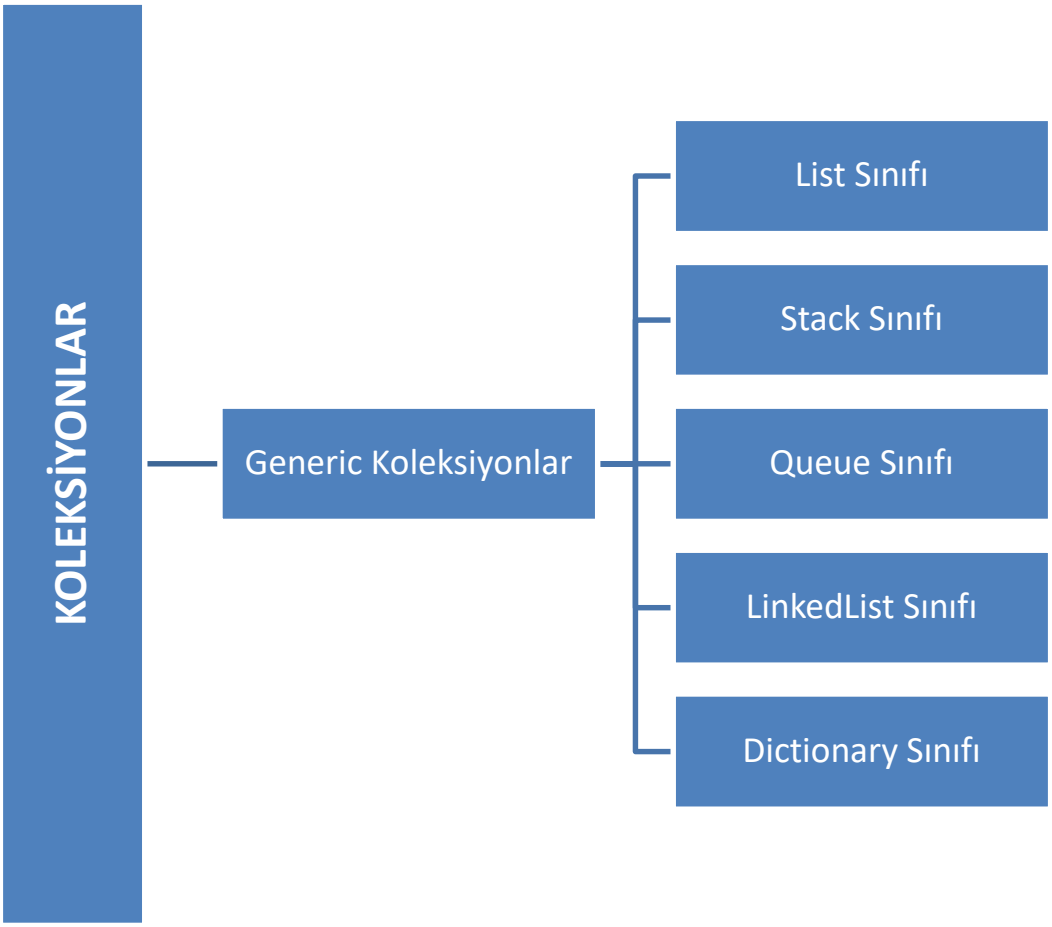
Atatürk Üniversitesi
Açıköğretim Fakültesi

NESNE TABANLI PROGRAMLAMA I

Dr. Orhan ÇELİKER

ÜNİTE

13


KOLEKSİYONLAR
Generic Koleksiyonlar
List Sınıfı
Stack Sınıfı
Queue Sınıfı
LinkedList Sınıfı
Dictionary Sınıfı

GİRİŞ

List, Stack, Queue, LinkedList ve *Dictionary* sınıfları *generic* koleksiyonlar içinde sıklıkla kullanılan yapılardır. Söz konusu bu koleksiyonlar *System.Collections.Generic* isim uzayının altında yer alırlar. Bu sınıfların kullanımları benzerlik göstermekle birlikte kullanım amaçları ve çalışma yapıları arasında ciddi farklılıklar bulunmaktadır. Özellikle yüksek performansın öncelikli olduğu durumlarda söz konusu koleksiyonlar arasında yapılan tercih kritik öneme sahiptir. Koleksiyonlarda tutulan öğelere erişim ve bu öğelerin yönetilmesi nesne yönelimli programlamanın öne çıkan konuları arasındadır. Özellikle verinin işlenmesi ve yönetilmesi noktasında *List, Stack, Queue, LinkedList* ve *Dictionary* yapıları Visual Studio.NET çatısı altında sıklıkla tercih edilmektedir.

Bu ünitede *List, Stack, Queue, LinkedList* ve *Dictionary* yapıları incelenecektir. Ünite kapsamında amaç ve kullanım amacına göre farklı şekillerde uygulamaya konulan koleksiyonların temel kullanım şekillerine yer verilecektir. Verilen örnekler C# dilinde olup, örnekler söz konusu koleksiyon yapılarının temel çalışma prensiplerini anlamaya yöneliktir. Küçük ölçekli programlardan büyük kapsamlı yazılımlara kadar birçok projede kullanılan söz konusu koleksiyonlar, özellikle program içerisinde verilerin yönetilmesi noktasında programcılara büyük esneklikler sunmaktadır. Hatta Silikon Vadisi'nde faaliyet gösteren birçok önemli bilişim şirketi program geliştirici işe alım mülakatlarında bu ünitede anlatılacak olan yapıların kullanımına yönelik sorular sormaktadırlar. Dolayısıyla ünitede yer verilen yapıları kavramak ve uygulamaya geçirmek sizlere programlama becerisi noktasında değerli katkılar sunacaktır.

GENERİC KOLEKSİYONLAR

Generic koleksiyonlar, aynı türden verileri saklayacak şekilde tasarlanan ve bu sayede veri tipi güvenliğini sağlayan koleksiyon türleridir. *Non generic* koleksiyonlardan farklı olarak *generic* koleksiyonlar, tıpkı dizilerde olduğu gibi aynı veri tipinde elemanların tutulmasına izin veriler ve bu koleksiyonlar *System.Collections.Generic* isim uzayının altında yer alırlar. *Generic* koleksiyonlar, *non generic* koleksiyonlarda olduğu gibi *boxing* işlemi ile herhangi bir dönüşümü tabi tutulmazlar. Dolayısıyla bu da *non generic* koleksiyonara göre daha performanslı çalışmasını sağlar.

List sınıfı, aynı türden verileri saklamak için kullanılır. Tıpkı dizi tanımlamada olduğu gibi veri tipi belirtilir ancak eleman sayısı ya da büyüklük belirtilmez.

Bu ünitede, *generic* koleksiyonlar arasında sıklıkla kullanılan *List, Stack, Queue, LinkedList* ve *Dictionary* sınıflarına yer verilecektir.

List Sınıfı

List sınıfı, aynı türden verileri saklamak için kullanılır. Tıpkı dizi tanımlamada olduğu gibi veri tipi belirtilir ancak eleman sayısı ya da büyüklük belirtilmez. *List* sınıfı *non generic* bir koleksiyon olan *ArrayList* sınıfıyla birçok ortak metoda ve özelliği sahiptir. Ayrıca bu sınıf için *ArrayList* sınıfının *generic* karşılığı denilebilir. Ancak *List* tanımlanırken veri tipi belirtildiğinden dolayı veriler *boxing* kullanılarak

herhangi bir dönüşüme tabi tutulmaz. Dolayısıyla *ArrayList* sınıfına göre daha performanslı çalışır.

List sınıfı *System.Collections.Generic* isim uzayının altında yer alır ve genel sözdizimi aşağıdaki gibidir.

List<Değişken Türü> Değişken Adı= new List<Değişken Türü>()

Sözdiziminden de görüldüğü gibi *List* sınıfı tanımlanırken koleksiyonun içereceği elemanların veri tipi *< >* operatörleri arasına yazılır. Böylece List'e eklenecek elemanların veri tipi belirlenir ve *< >* operatörleri arasına yazılan tip dışında elemanın eklenmesine izin vermez. Bir *List* nesnesinin tanımlanması ve bu nesneye eleman eklemeyle ilgili kod örneği aşağıdaki gibidir.

```
List<string> mevsimler = new List<string>();
mevsimler.Add("İlkbahar");
mevsimler.Add("Yaz");
mevsimler.Add("Sonbahar");
mevsimler.Add("Kış");
```

Yukarıdaki örnekte *string* tipinde mevsimler isimli bir *List* nesnesi oluşturulmuştur. Bu *List* nesnesine *Add()* metodu kullanılarak *string* tipinde 4 eleman eklenmiştir. Eleman eklenirken *string* tipinde değil de başka tipte bir elemanın eklenmesi istenirse hata mesajı alınırdı. Bunun sebebi *List* nesnesi sadece tanımlandığı andaki veri tipini kabul eder. Bu özelliğinden ötürü *List generic* bir koleksiyon türü olarak kategorize edilir ve veri tipi güvenliğini sağlar.

List sınıfı birçok metodu *ArrayList* sınıfıyla ortak kullanır. Bu metotlardan bazıları açıklamalarıyla beraber Tablo 13.1'de verilmiştir.

Tablo 13.1. List sınıfına ait bazı metotlar

Metot Adı	Açıklama
Add	List içerisine eleman eklemek için kullanılır.
AddRange	List'in sonuna eleman eklemek için kullanılır.
Clear	List nesnesinin bütün elemanlarını siler ve eleman sayısı 0 olur.
Contains	List içerisinde arama yapmak için kullanılır.
ConvertAll	List içerisindeki elemanları başka bir türe çevirir ve dönüştürülmüş elemanları içeren bir List döndürür.
CopyTo	List nesnesinin içerdiği elemanları aynı tipteki başka bir koleksiyona kopyalamak için kullanılır.
IndexOf	List içerisinde indeks numarasına göre arama yapar.
Remove	Parametre olarak değeri verilen elemanı List'ten çıkarır.
RemoveAt	Parametre olarak indeks numarası verilen elemanı List'ten çıkarır.
Reverse	List'in içeriğini terse çevirmek için kullanılır.
Sort	List elemanlarını sıralamak için kullanılır.

Bireysel Etkinlik

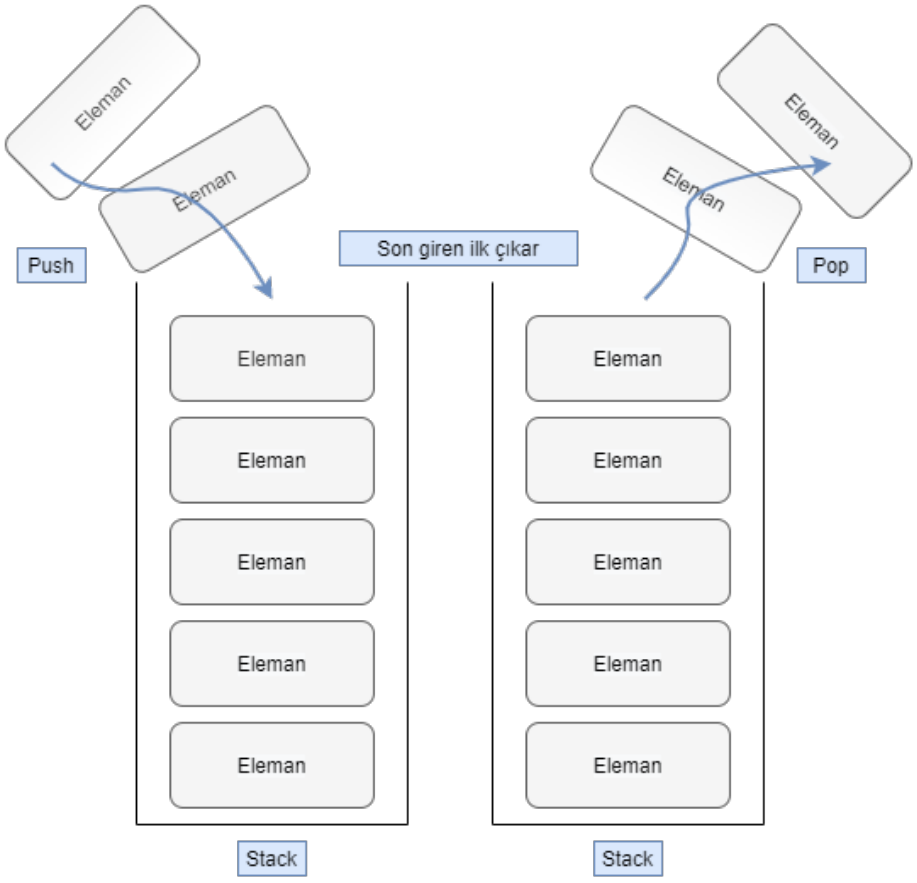
Bir list koleksiyonu tanımlayarak Tablo 13.1'deki List sınıfına ait metotlarla ilgili birer örnek yapınız.

Stack Sınıfı

Koleksiyonlarla ilgili şimdiye kadar anlatılan sınıflarda basit veri ekleme, silme, sıralama ya da saklama gibi işlemler yapılmaktaydı. Koleksiyonlar sadece bu tür basit işlemleri gerçekleştirmek veya performansı artırmak amacıyla kullanılmamaktadır. *System.Collections.Generic* isim uzayının altında yer alan *Stack (Yığın)* sınıfı derleyici, sistem veya yapay zeka yazılımlarında sıklıkla kullanılan bir koleksiyon türüdür.

Stack sınıfı, son giren ilk çıkar (last in first out - LIFO) yöntemini kullanır ve boyutu dinamik olarak büyüyebilen bir koleksiyondur.

Stack sınıfı, *son giren ilk çıkar (last in first out -LIFO)* yöntemini kullanır ve boyutu dinamik olarak büyüyebilen bir koleksiyondur. *Stack* sınıfının çalışma mantığını anlamak için öncelikle *LIFO* yönteminin anlaşılması gerekmektedir. Bu yönteme örnek olarak 20 kitabın veya herhangi bir nesnenin üst üste bir yığın şeklinde dizilmesi düşünülebilir. Bu yığına eklenen son kitap en üstte olacaktır ve *son giren ilk çıkar* yöntemine göre son kitap erişilebilen ilk eleman olacaktır. Yığına eklenen ilk kitap ise yığının yani *Stack* koleksiyonunun son elemanı olacaktır. *LIFO* için verilen bu örnek Şekil 13.1' de görselleştirilmiştir.



Şekil 13.1. LIFO yöntemi çalışma mantığı

Stack sınıfı, diğer *generic* koleksiyonların kullandığı birçok metodu ortak olarak kullanmaktadır. Bu metotlar daha önce anlatıldığı için burada değinilmeyecektir. Ancak *Stack* sınıfını kullanırken bu sınıfa ait olan *Push(), Pop()* ve *Peek()* metotlarının bilinmesi gerekmektedir.

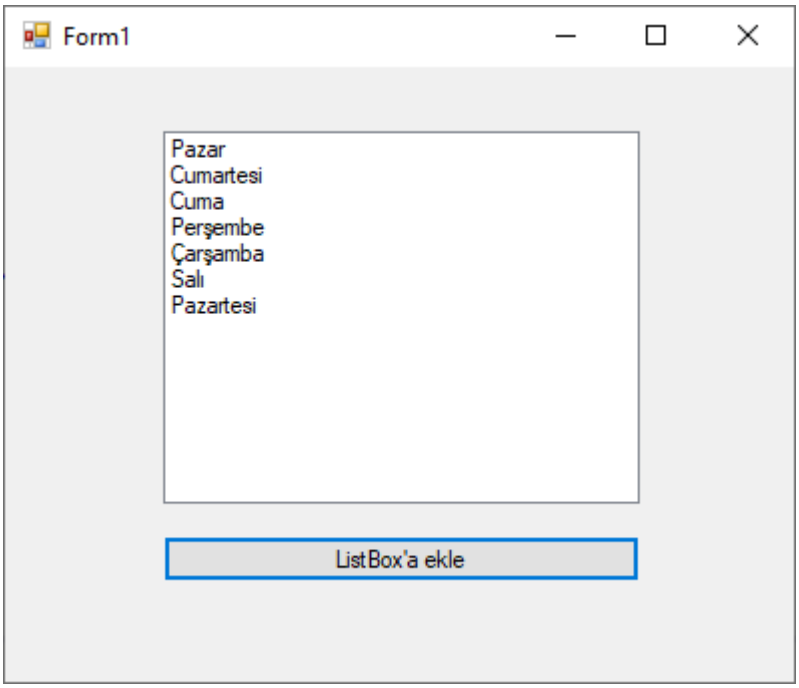
Push() metodu kullanılarak eklenen en son eleman Stack nesnesinin en üstünde bulunur ve ilk eleman değerini alır. Pop() metodu da Stack nesnesinden eleman okumak ve bu elemanı silmek için kullanılmaktadır.

Push() metodu *Stack* koleksiyonuna eleman eklemek için kullanılır. Eklenen elemanlar *object* türünde bir parametre alır ve dolayısıyla *Stack* nesnesine istenilen tipte eleman eklenebilir. *Push()* metodu kullanılarak eklenen en son eleman *Stack* nesnesinin en üstünde bulunur ve ilk eleman değerini alır. *Pop()* metodu da *Stack* nesnesinden eleman okumak ve bu elemanı silmek için kullanılmaktadır. Bir başka ifadeyle *Pop()* metodu her kullanıldığında *Stack* nesnesine en son eklenen elemanı geri döndürür ve bu elemanı *Stack* nesnesinden çıkarır. *Peek()* metodu ise *Pop()* metoduna benzer olarak *Stack* nesnesindeki en son elemanı geri döndürür ancak bu elemanı *Stack* nesnesinden silmez.

Stack nesnesine *Push()* metodu kullanılarak eleman ekleme ve *Pop()* metodu yardımıyla bu elemanların bir *ListBox* ögesine yazdırılmasıyla ilgili örnek kod aşağıda verilmiştir.

```
Stack<string> gunler = new Stack<string>();
gunler.Push("Pazartesi");
gunler.Push("Salı");
gunler.Push("Çarşamba");
gunler.Push("Perşembe");
gunler.Push("Cuma");
gunler.Push("Cumartesi");
gunler.Push("Pazar");
int elemanlar = gunler.Count;
for(int i=0;i<elemanlar;i++)
{
   listBox1.Items.Add(gunler.Pop());
}
```

Yukarıdaki örnekte *gunler* isimli bir *Stack* nesnesi tanımlanmış ve *Push()* metodu kullanılarak bu nesneye haftanın günleri eleman olarak eklenmiştir. Ardından *Count* özelliği kullanılarak *Stack* nesnesindeki toplam eleman sayısı elemanlar isimli değişkene aktarılmıştır. Son olarak *for* döngüsü kullanılarak *Stack* koleksiyonunda yer alan elemanlar *ListBox* nesnesine eklenmiştir. Örnekteki kod çalıştırıldığında Şekil 13.2'deki gibi bir görüntü elde edilecektir.



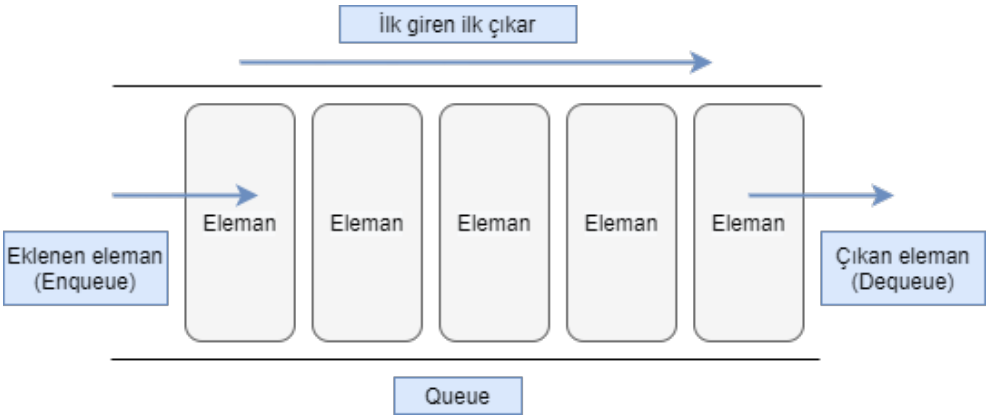
Şekil 13.2. Örnek kod ekran görüntüsü

Yukarıda verilen örnekte *ListBox* nesnesine eleman ekleme işlemi *Pop()* metoduyla yapılmıştır. Dolayısıyla döngüden çıkıldıktan sonra *gunler* koleksiyonunun eleman sayısı kontrol edildiğinde sonucun 0 (sıfır) olduğu yani koleksiyonun hiçbir elemana sahip olmadığı görülecektir. Bunun sebebi *Pop()* metodu, yukarıda da değinildiği gibi her çalıştırıldığında geri döndürdüğü elemanı *Stack* nesnesinden çıkarır.

Queue Sınıfı

Queue sınıfı ilk giren ilk çıkar (first in first out - FIFO) çalışma yöntemini kullanır.

Queue (kuyruk), System.Collections.Generic isim uzayının altında yer alan bir koleksiyon türü olup genelde ağ trafiği yönetimi, simülasyonlar ve veri tabanı işlemlerinde kullanılmaktadır. *Stack* sınıfı, *son giren ilk çıkar (LIFO)* yöntemine göre çalışmaktayken, *Queue* sınıfı *ilk giren ilk çıkar (first in first out - FIFO)* çalışma yöntemini kullanır. Yani *Queue* koleksiyonuna eklenen ilk elemana ilk sırada erişim sağlanır. *FIFO* yöntemi, çalışma mantığının daha iyi anlaşılması açısından Şekil 13.3'te görselleştirilmiştir.



Şekil 13.3. FIFO yöntemi çalışma mantığı

Dequeue(): Queue içerisinden eleman silmek ve bu elemanı geri döndürmek için kullanılır.

Queue sınıfı, diğer *generic* koleksiyonların kullandığı birçok metodu ortak olarak kullanmaktadır. Aşağıda *Queue* sınıfında sıkça kullanılan bazı metotlar açıklanmıştır.

- *Enqueue():* Bir Queue nesnesine eleman eklemek için kullanılır. Bu metot Object tipinde parametre kabul eder ve bu parametreyi kuyruğun sonuna ekler.
- *Dequeue():* Queue içerisinden eleman silmek ve bu elemanı geri döndürmek için kullanılır.
- *GetEnumerator():* Queue nesnesinde yer alan bütün elemanları bir IEnumerator referansı oluşturarak döndürür. Böylece Queue içerisindeki tüm elemanlara ulaşılabilir.
- *Peek():* Queue nesnesinde en üst sırada yer alan elemanı döndürür.
- *ToArray():* Queue koleksiyonunun elemanlarını klasik bir diziye aktarır.
- *TrimToSize():* Queue koleksiyonunun kapasitesini kuyruktaki eleman sayısına düşürür.

Aşağıda *Queue* koleksiyonuyla ilgili bir örnek verilmiştir. Bu örnekte ay isimlerinin iki *ListBox* arasında butonlar aracılığıyla geçiş yaptırılmış ve bu işlem için *Queue* sınıfı kullanılmıştır. Örneğin kod kısmına geçmeden önce ilk olarak Şekil 13.4'teki gibi bir form tasarımı yapılmıştır.

Şekil 13.4. Form Tasarımı

Tasarım işlemi bittikten sonra *Form* nesnesinin *Load* olayına aşağıdaki örnek kod yazılır.

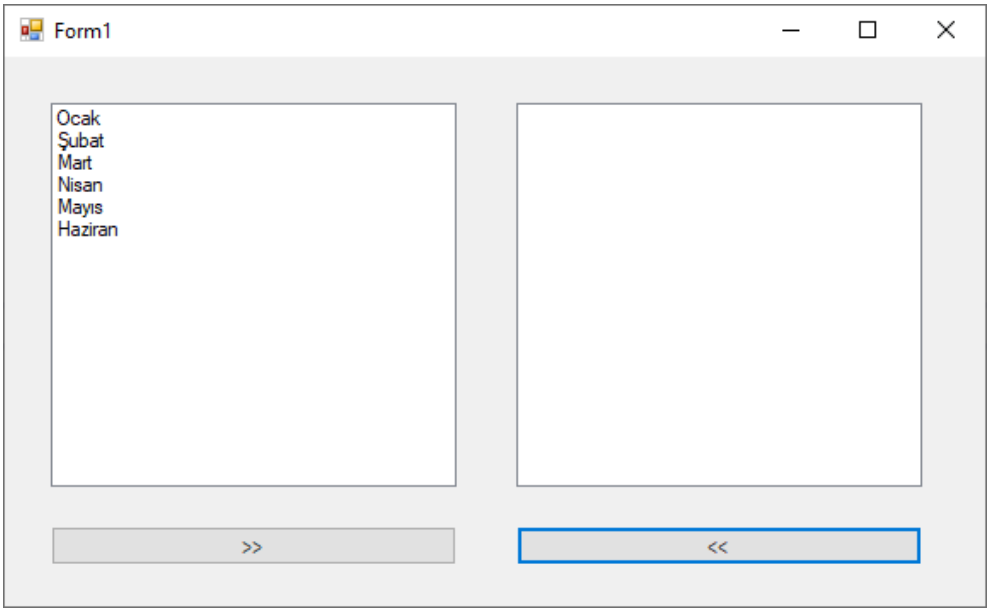
```
Queue kuyruk1 = new Queue();
Queue kuyruk2 = new Queue();
private void Form1_Load(object sender, EventArgs e)
    {
        kuyruk1.Enqueue("Ocak");
        kuyruk1.Enqueue("Şubat");
        kuyruk1.Enqueue("Mart");
        kuyruk1.Enqueue("Nisan");
        kuyruk1.Enqueue("Mayıs");
        kuyruk1.Enqueue("Haziran");
        IEnumerator sayac = kuyruk1.GetEnumerator();
        while (sayac.MoveNext())
            listBox1.Items.Add(sayac.Current);
    }
```

Yukarıdaki kod örneği incelendiğinde öncelikle *kuyruk1* ve *kuyruk2* adında iki tane *global* değişken tanımlanmıştır. Bu değişkenler *ListBox* nesnelerine veri ekleme ve tasarımda yer alan butonların *Click* olaylarında kullanılacağı için *global* olarak tanımlanmışlardır. *Form* nesnesinin *Load* olayında ilk olarak *Enqueue()* metodu kullanılarak *kuyruk1* koleksiyonuna *ay isimleri* eklenmiştir. Ardından bu elemanlar *ListBox1* nesnesine ekleneceği için *GetEnumerator()* metodu kullanılarak *IEnumerator* tipinde *sayac* isimli bir değişkene atanmıştır. Sonrasında *While* döngüsünde *MoveNext()* metodu kullanılarak döngü sağlanmış ve döngünün her turunda *Current()* metodu ile elemanlar *ListBox1* nesnesine atanmıştır.

Bu işlemin ardından Şekil 13.4'teki form tasarımında yer alan *>>* isimli butonun *Click* olayına girilerek aşağıdaki kod örneği yazılmalıdır. Böylece *kuyruk1* koleksiyonunda yer alan elemanların *ListBox1* nesnesine eklenmesi sağlanmıştır.

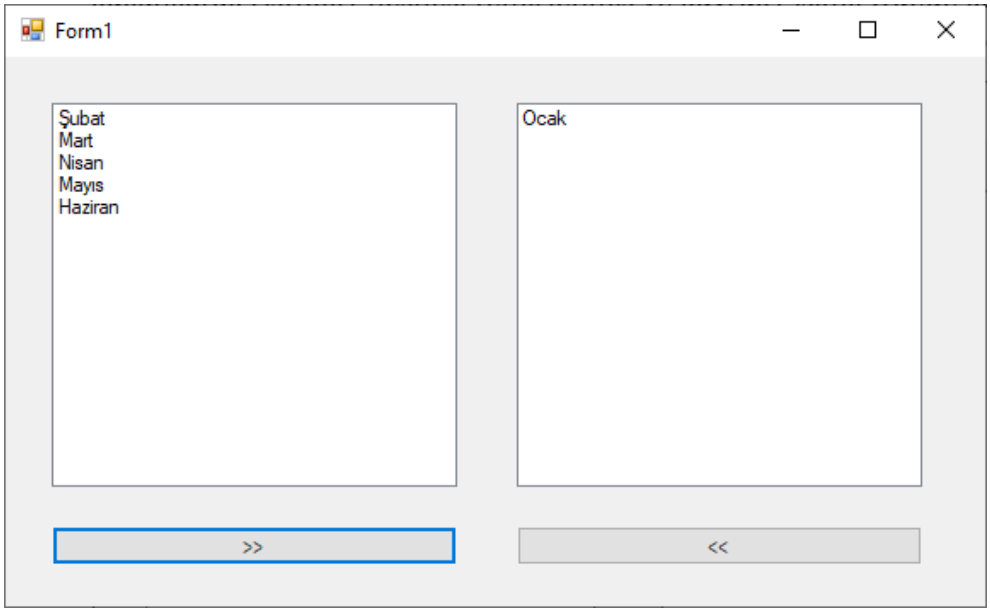
```
private void button1_Click(object sender, EventArgs e)
    {
      string aylar = kuyruk1.Dequeue().ToString();
      kuyruk2.Enqueue(aylar);
      listBox1.Items.Clear();
      IEnumerator sayac = kuyruk1.GetEnumerator();
      while (sayac.MoveNext())
          listBox1.Items.Add(sayac.Current);
          listBox2.Items.Add(aylar);
    }
```

Yukarıdaki kod örneği incelendiğinde ilk olarak *Dequeue()* metodu kullanılarak *kuyruk1* koleksiyonunun ilk elemanı aylar isimli değişkene atanmıştır. Ardından *Enqueue()* metodu kullanılarak aylar isimli değişken *ListBox2* nesnesinin veri kaynağı olan *kuyruk2* koleksiyonuna eklenmiştir. Daha sonra *Clear()* metodu kullanılarak *ListBox1* nesnesi temizlenmiş ve *kuyruk1* isimli *Queue* koleksiyonunun kalan elemanları yeniden *ListBox1* nesnesine eklenmiştir (Şekil 13.5).



Şekil 13.5. ListBox1'e eklenen elemanlar

Son olarak aylar isimli değişken *LisBox2* nesnesine eklenerek *ListBox2* nesnesi doldurulmuştur. Yukarıdaki kod örneği çalıştırılıp *>>* butonuna bir kez basıldığında aşağıdaki ekran görüntüsü oluşacaktır (Şekil 13.6).

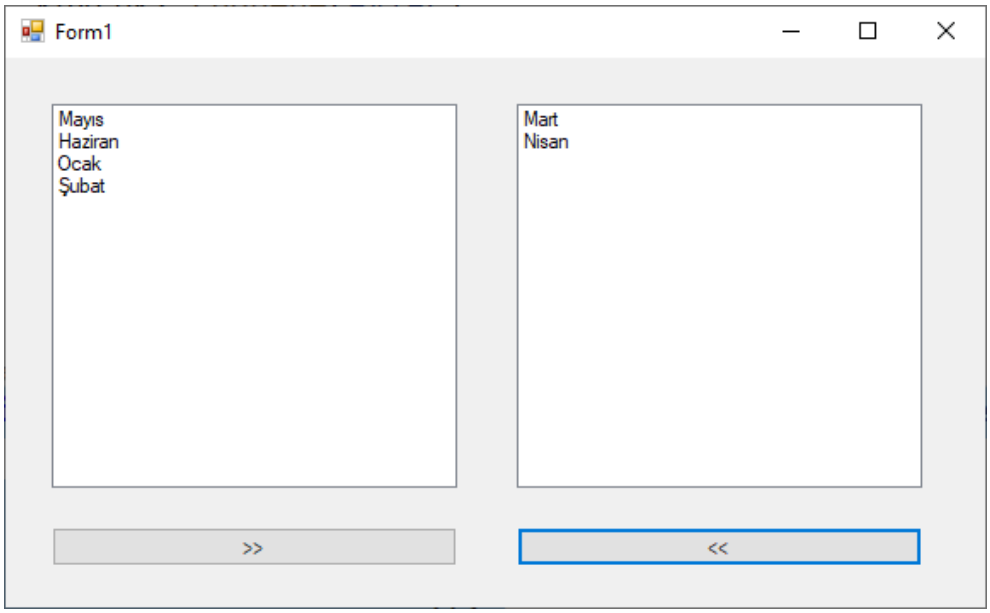


Şekil 13.6. "Ocak" isimli elemanın >> butonun ile ListBox1'den ListBox2'ye aktarılması

Bu işlemlerin ardından << isimli butonun *Click()* olayına aşağıdaki kod örneği yazılmalıdır. Yazılacak bu kod >> butonunun gerçekleştirdiği işlevin tam tersini yapacaktır.

```
private void button2_Click(object sender, EventArgs e)
        {
            string aylar = kuyruk2.Dequeue().ToString();
            kuyruk1.Enqueue(aylar);
            listBox2.Items.Clear();
            IEnumerator sayac = kuyruk2.GetEnumerator();
            while (sayac.MoveNext())
                listBox2.Items.Add(sayac.Current);
                listBox1.Items.Add(aylar);
        }
```

İki butona da gerekli kodlar eklendikten sonra program çalıştırıldığında Şekil 13.7'deki gibi bir görüntü elde edilecektir. Burada >> butonu ile ListBox1'in elemanları ListBox2'ye, << butonu ile de ListBox2'nin elemanları ListBox1'e aktarılacaktır.



Şekil 13.7. >> ve << butonları kullanılarak ListBox nesneleri arasında eleman aktarımı

Şekil 13.7'den de görüldüğü üzere ListBox'lar arasındaki eleman aktarımının temel mantığı *Queue* koleksiyonun *ilk giren ilk çıkar (FIFO)* prensibidir.

Bireysel Etkinlik

- Queue koleksiyonuyla ilgili verilen örneği kendi bilgisayarınızda yapınız. Yaptığınız programı çalıştırarak Queue koleksiyonunun çalışma mantığını daha iyi anlayabilirsiniz.

LinkedList Sınıfı

LinkedList (bağlı liste) sınıfı *System.Collections.Generic* isim uzayı altında yer alır ve elemanlarının birbirlerine bir link şeklinde bağlı olduğu koleksiyon türüdür.

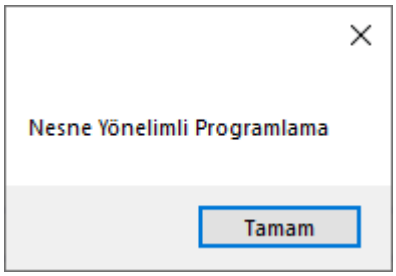
LinkedList sınıfı kullanılırken AddFirst() metodu yardımıyla listenin ilk alanına eleman eklenir. Benzer şekilde listenin son alanına eleman eklemek için AddLast() metodu kullanılır.

LinkedList sınıfı kullanılırken *AddFirst()* metodu yardımıyla listenin ilk alanına eleman eklenir. Benzer şekilde listenin son alanına eleman eklemek için *AddLast()* metodu kullanılır. Bu koleksiyonun elemanlarına erişirken *First* ve *Last* özellikleri kullanılır. *First* özelliği ile ilk elemana erişim sağlanırken *Last* ile son elemana erişilir.

Aşağıda *LinkedList* koleksiyonu oluşturma, koleksiyona eleman ekleme ve bu elemanlara erişme ile ilgili bir örnek verilmiştir.

```
LinkedList<string> dersler = new LinkedList<string>();
dersler.AddLast("Nesne Yönelimli Programlama");
dersler.AddLast("Temel Bilgi Teknolojileri");
dersler.AddLast("Web Tasarımı");
dersler.AddLast("İnternet Programcılığı");
MessageBox.Show(dersler.First.Value);
```

Yukarıdaki kod örneğinde ilk olarak *dersler* isminde bir *LinkedList* koleksiyonu oluşturulmuş ve *AddFirst()* metodu kullanılarak koleksiyona birkaç eleman eklenmiştir. Ardından *First* özelliği ile koleksiyonun ilk elemanı çağrılmış ve *MessageBox* ile görüntülenmiştir (Şekil 13.8). Örnek kodda yer alan *Value* ifadesi ise ilgili özelliğin değerinin görüntülenmesini sağlar.



Şekil 13.8. MessageBox görüntüsü

LinkedList sınıfından eleman silmek için *Remove()* metodu kullanılır. Bu metoda benzer şekilde *RemoveFirst()* metodu koleksiyon içerisindeki ilk elemanı silerken, *RemoveLast()* metodu koleksiyonun son elemanını silmektedir. *LinkedList* koleksiyonu içerisinde arama yapmak için *Find()* metodu kullanılmaktadır. Bu metotla birlikte genellikle *Next* ve *Previous* özellikleri kullanılmaktadır. *Next* özelliği bulunan değerden bir sonraki değeri bildirirken, *Previous* özelliği bulunan değerden bir önceki değeri geri döndürür. Ayrıca *AddAfter()* ve *AddBefore()* metotları aracılığıyla *Find()* metoduyla bulunan değerin önceki veya sonraki konumlarına ekleme yapılabilir.

Dictionary Sınıfı

Dictionary sınıfının elemanlarına benzersiz anahtarlar kullanılarak erişim sağlanır.

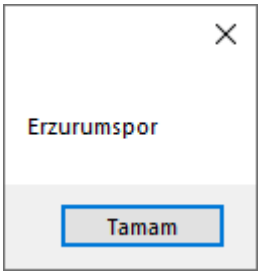
Dictionary (sözlük) sınıfı içerdiği elemanların bir anahtar yardımıyla saklandığı koleksiyon türü olup *System.Collections.Generic* isim uzayının altında yer alır. Tıpkı *HashTable* sınıfında olduğu gibi *Dictionary* sınıfının elemanlarına benzersiz anahtarlar kullanılarak erişim sağlanır. Bu koleksiyonda bildirilen tüm anahtaralar kesinlikle birbirinden farklı olmalıdır, anahtarlara atanan değerler ise birden fazla kullanılabilir. *Dictionary* sınıfını genel söz dizimi aşağıdaki gibidir.

Dictionary <Anahtar Değişken Tipi, Eleman Değişken Tipi> Koleksiyon Adı = new Dictionary <Anahtar Değişken Tipi, Eleman Değişken Tipi>()

Bu sözdizimi göz önünde bulundurularak Dictionary sınıfı tanımlama ve bu sınıfa eleman eklemeyle ilgili aşağıdaki gibi bir örnek verilebilir.

```
Dictionary<int, string> takimlar = new Dictionary<int,
string>();
takimlar.Add(1903, "Beşiktaş");
takimlar.Add(1905, "Galatasaray");
takimlar.Add(1907, "Fenerbahçe");
takimlar.Add(1968, "Erzurumspor");
MessageBox.Show(takimlar[1968]);
```

Yukarıdaki kod örneğinde ilk olarak *takimlar* isminde bir *Dictionary* koleksiyonu oluşturulmuş ve *Add()* metodu kullanılarak koleksiyona birkaç anahtar ve eleman eklenmiştir. Ardından koleksiyonun son elemanının anahtar değeri çağrılarak *MessageBox* ile görüntülenmiştir (Şekil 13.9).



Şekil 13.9. MessageBox görüntüsü

Bireysel Etkinlik

Stack ve Queue koleksiyon sınıfları birbirlerinin hangi durumlarda alternatifi olabilir, düşününüz.

Özet

- Generic koleksiyonlar, aynı türden verileri saklayacak şekilde tasarlanan ve bu sayede veri tipi güvenliğini sağlayan koleksiyon türleridir. Non generic koleksiyonlardan farklı olarak generic koleksiyonlar, tıpkı dizilerde olduğu gibi aynı veri tipinde elemanların tutulmasına izin veriler ve bu koleksiyonlar System.Collections.Generic isim uzayının altında yer alırlar. Generic koleksiyonlar, non generic koleksiyonlarda olduğu gibi boxing işlemi ile herhangi bir dönüşümü tabi tutulmazlar. Dolayısıyla bu da non generic koleksiyonara göre daha performanslı çalışmasını sağlar.
- *List* sınıfı, aynı türden verileri saklamak için kullanılır. Tıpkı dizi tanımlamada olduğu gibi veri tipi belirtilir ancak eleman sayısı ya da büyüklük belirtilmez. *List* sınıfı *non generic* bir koleksiyon olan *ArrayList* sınıfıyla birçok ortak metoda ve özelliği sahiptir. Ayrıca bu sınıf için *ArrayList* sınıfının *generic* karşılığı denilebilir. Ancak *List* tanımlanırken veri tipi belirtildiğinden dolayı veriler *boxing* kullanılarak herhangi bir dönüşüme tabi tutulmaz. Dolayısıyla *ArrayList* sınıfına göre daha performanslı çalışır.
- Koleksiyonlarla ilgili şimdiye kadar anlatılan sınıflarda basit veri ekleme, silme, sıralama ya da saklama gibi işlemler yapılmaktaydı. Koleksiyonlar sadece bu tür basit işlemleri gerçekleştirmek veya performansı artırmak amacıyla kullanılmamaktadır. *System.Collections.Generic* isim uzayının altında yer alan *Stack (Yığın)* sınıfı derleyici, sistem veya yapay zeka yazılımlarında sıklıkla kullanılan bir koleksiyon türüdür.
- *Stack* sınıfı, diğer *generic* koleksiyonların kullandığı birçok metodu ortak olarak kullanmaktadır. Bu metotlar daha önce anlatıldığı için burada değinilmeyecektir. Ancak *Stack* sınıfını kullanırken bu sınıfa ait olan *Push(), Pop()* ve *Peek()* metotlarının bilinmesi gerekmektedir.
- *Queue (kuyruk), System.Collections.Generic* isim uzayının altında yer alan bir koleksiyon türü olup genelde ağ trafiği yönetimi, simülasyonlar ve veri tabanı işlemlerinde kullanılmaktadır. *Stack* sınıfı, *son giren ilk çıkar (LIFO)* yöntemine göre çalışmaktayken, *Queue* sınıfı *ilk giren ilk çıkar (first in first out - FIFO)* çalışma yöntemini kullanır. Yani *Queue* koleksiyonuna eklenen ilk elemana ilk sırada erişim sağlanır.
- *LinkedList (bağlı liste)* sınıfı *System.Collections.Generic* isim uzayı altında yer alır ve elemanlarının birbirlerine bir link şeklinde bağlı olduğu koleksiyon türüdür.
- *LinkedList* sınıfı kullanılırken *AddFirst()* metodu yardımıyla listenin ilk alanına eleman eklenir. Benzer şekilde listenin son alanına eleman eklemek için *AddLast()* metodu kullanılır. Bu koleksiyonun elemanlarına erişirken *First* ve *Last* özellikleri kullanılır. *First* öze
- *Dictionary (sözlük)* sınıfı içerdiği elemanların bir anahtar yardımıyla saklandığı koleksiyon türü olup *System.Collections.Generic* isim uzayının altında yer alır. Tıpkı *HashTable* sınıfında olduğu gibi *Dictionary* sınıfının elemanlarına benzersiz anahtarlar kullanılarak erişim sağlanır. Bu koleksiyonda bildirilen tüm anahtaralar kesinlikle birbirinden farklı olmalıdır, anahtarlara atanan değerler ise birden fazla kullanılabilir.lliği ile ilk elemana erişim sağlanırken *Last* ile son elemana erişilir.

DEĞERLENDİRME SORULARI

1. List nesnesi içerisinde arama yapmak için kullanılan metot aşağıdakilerden hangisidir?
   a) Add()
   b) Clear()
   c) Contains()
   d) Remove()
   e) CopyTo()

2. List koleksiyonunun içeriğini terse çevirmek için aşağıdaki metotlardan hangisi kullanılır?
   a) Reverse()
   b) Remove()
   c) IndexOf()
   d) Add()
   e) Clear()

3. List koleksiyonundaki bütün elemanları silmek için aşağıdaki metotlardan hangisi kullanılır?
   a) Sort()
   b) Clear()
   c) Insert()
   d) Contains()
   e) Add()

4. Aşağıdakilerden hangisi Generic koleksiyonlar arasında yer almaz?
   a) Stack
   b) List
   c) Queue
   d) ArrayList
   e) LinkedList

5. Son giren ilk çıkar (LIFO) yöntemini kullanan koleksiyon aşağıdakilerden hangisidir?
   a) List
   b) Stack
   c) Queue
   d) LinkedList
   e) Dictionary

6. Stack koleksiyonuna eleman eklemek için kullanılan metot aşağıdakilerden hangisidir?
   a) Peek()
   b) Pop()
   c) Remove()
   d) AddFirst()
   e) Push()

7. Queue koleksiyonuna eleman eklemek için kullanılan metot aşağıdakilerden hangisidir?
   a) Peek()
   b) Pop()
   c) Remove()
   d) Enqueue()
   e) Dequeue()

8. Queue koleksiyonunda en üst sırada yer alan elemanı geri döndüren metot aşağıdakilerden hangisidir?
   a) Peek()
   b) Pop()
   c) Remove()
   d) Enqueue()
   e) Dequeue()

9. LinkedList koleksiyonunda arama yapmak için kullanılan metot aşağıdakilerden hangisidir?
   a) Contains()
   b) IndexOf()
   c) Find()
   d) AddAfter()
   e) AddBefore()

10. Elemanlarına erişim için benzersiz bir anahtar değeri kullanan koleksiyon aşağıdakilerden hangisidir?
    a) List
    b) Stack
    c) Queue
    d) LinkedList
    e) Dictionary

Cevap Anahtarı
1.c,2.a,3.b,4.d,5.b,6.e,7.d,8.a,9.c,10.e

YARARLANILAN KAYNAKLAR

Albaharı, Joseph & ALBAHARI, Ben, (2012), *C# 5.0 in a Nutshell Fifth Edition*, O'Reilly Media, California.

Aktaş, Volkan, (2021), *Her Yönüyle C# 9.0, KODLAB*, İstanbul.

Clark, Dan, (2013), *Beginning C# Object-Oriented Programming 2nd Edition*, Apress, New York.

Grıffıths, Ian, (2013), *Programming C# 5.0, O'Reilly Media*, California.

Skeet, Jon, (2014), *C# in Depth 3rd Edition*, Manning Publication Co, New York.