

## Koleksiyonlar ( Collections) Nedir?

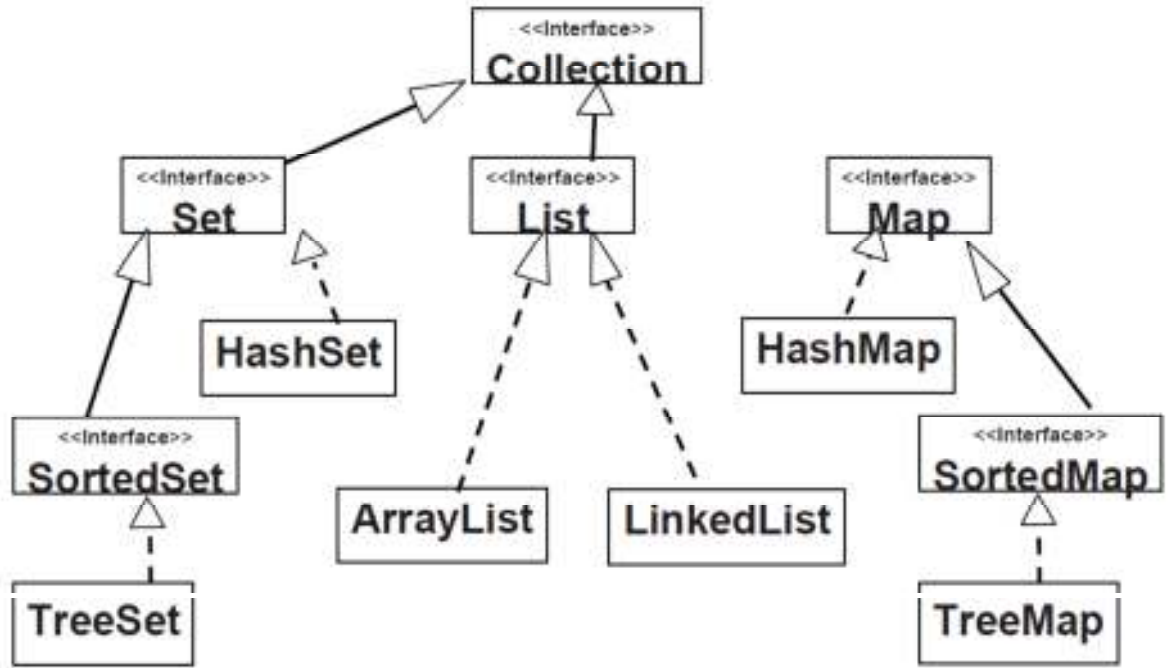
Adından da anlaşılacağı gibi bir koleksiyon (collection) içinde bir çok öğeyi barındıran bir nesnedir (object). Öğeler, veri gruplarından oluşur; sıralı ya da sırasız olabilirler. Bazı bazı koleksiyonlarda aynı öğe birden çok kez (dublikasyon) koleksiyonda yer alabilir, bazılarında yer alamaz. Veri koleksiyonu yapmaktaki amacımız verilere erişim sağlamak, veriler üzerinde işlem yapmak, verileri sıralamak, yeni veri eklemek ya da mevcut bir veriyi silmek, bir verinin koleksiyon içinde olup olmadığını aramak gibi eylemleri gerçekleştirebilmektir.

*Collections framework*'un özünü oluşturan şey '*Collection*' denen arayüzdür. Bu arayüz framework'un temeli olan metotları tanımlar. *List* ve *Set* arayüzleri *Collection* arayüzünde olmayan metotları tanımlayarak, *framework*'un uygulama alanını genişletirler.

Önemli bir başka arayüz *Map* adını alır. Ancak Map arayüzü Collection arayüzünün bir genişlemesi değildir. *Map* arayüzü *Collection* hiyerarşisine dahil değildir; ama *Collections framework*'un bir parçasıdır.

## Koleksiyonların Avantajları:

1. Verileri bir araya toplamamızı ve veriler üzerinde işlem yapmamızı kolaylaştırır.
2. Yazılımcı koleksiyon sınıfı içerisinde yer alan algoritmaları kullanarak yeni algoritma yazmak zorunda kalmaz.
3. Yazılımcıya uygulamanın performansını artıran kullanışlı algoritma ve veri yapıları sağlar.
4. Belli bir boyutları yoktur. İçerisine veri ekleyip çıkardıkça boyutları değişir.. Yani yazılımcının önceden koleksiyonu boyutlandırmasına gerek yoktur.



Java Collection Framework

- **Collection:** En genel grup

- **List:** nesnelerden oluşan topluluk. Topluluk içinde dublikasyon olabilir, topluluğun belirli bir sıralaması vardır.

- **Set:** Sırası olmayan ve duplikasyonu olmayan nesneler topluluğu.

- **SortedSet:** Artan sırada sıraya dizilmiş nesneler kümesi.

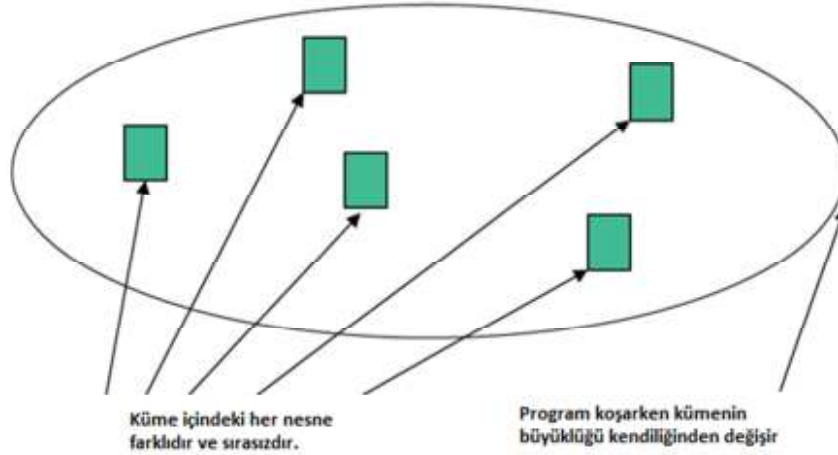
- **Map:** her ögesine bir anahtar atanmış nesneler topluluğu

- **SortedMap:** Anahtarlarına göre artan sırada dizilmiş nesneler topluluğu.

Şekilden görüldüğü gibi, **List** ve **Set** arayüzleri **Collection** arayüzünü genişletir. **SortedSet** arayüzü **Set** arayüzünü genişletir. **SortedMap** arayüzü **Map** arayüzünü genişletir.

## Set (Küme) :

Tekrarlı ya da kopya elemanları içerisinde barındırmayan bir koleksiyondur. Set sınıfında elemanlar sırasız bir şekilde tutulur. Basit ekleme ve silme yöntemleri vardır. **Tekrarlanan (“duplicate”) elemanlara izin verilmez.** Bir setten bir nesneyi silmeden önce, nesnenin orada var olduğunu bilmeniz gerekir. “Set” sınıflarından bazıları;



- **“HashSet”:** Hızlı bir kümeleme yapar. Kullanım Şekli;  
`HashSet <String> diziAdi = new HashSet <String>( );`
- **“TreeSet”:** Ağaç yapısını kullanarak kümeleme yapar. Kullanım şekli;  
`TreeSet <String> diziAdi= new TreeSet <String> ( );`

Kümeler için şu kısıtlar vardır:

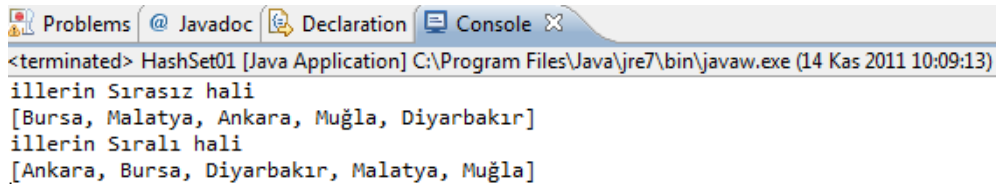
- Kümenin öğeleri sıralı olmadığı için, yeni gelen bir öğeyi kümede belirli bir konuma yerleştiremeyiz.
- Aynı nedenle, bir öğe yerine başka bir öğe koyamayız (replacement olamaz). Ama, istenen öğe kümeden silinebilir ve istenen öğe kümeye eklenebilir.
- Kümedeki öğelere erişmek (retrieving) mümkündür, ama erişim sırası belirsizdir.
- Kümede bir öğenin yeri belirsizdir.

## Örnek : HashSet ve TreeSet kullanımı için Örnek Program:

```
import java.util.*;
public class HashSet01 {
public static void main(String[] args) {
    HashSet <String> hs = new HashSet <String>();
    hs.add("Bursa");
    hs.add("Ankara");
    hs.add("Malatya");
    hs.add("Diyarbakır");
    hs.add("Muğla");
    hs.add("Ankara"); // Hashset çift veriye izin vermez
    System.out.println("illerin Sırasız hali");
    System.out.println(hs);
    System.out.println("illerin Sıralı hali");
    TreeSet <String> siraliKume= new TreeSet <String> (hs);
    System.out.println(siraliKume);
}
}
```

### Programın Ekran Çıktısı:

Burada **add()** komutu ile veriler kümeye eklenmektedir. HashSet'in tekrarlanan elemanlara izin vermediğine dikkat ediniz.



```
<terminated> HashSet01 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (14 Kas 2011 10:09:13)
illerin Sırasız hali
[Bursa, Malatya, Ankara, Muğla, Diyarbakır]
illerin Sıralı hali
[Ankara, Bursa, Diyarbakır, Malatya, Muğla]
```

## Iterator :

**Iterator**, elemanlar arası istenilen yönde ilerlemek için kullanılır. Üç temel yöntemi vardır:

- **next()** : Bir sonraki elemanı çağırır
- **hasNext()**: Bir sonraki eleman var mı kontrol eder . Varsa true döndürür
- **remove()**: **next()** yöntemi ile döndürülen son elemanı siler.

Kullanım şekli;

```
Iterator degisken = list_değişkeni.iterator();
```

### Örnek : Iterator kullanımı için Örnek Program:

```
import java.util.*;
public class Iterator {
    public static void main(String[] args)
    {
        HashSet hSet= new HashSet();
        hSet.add( "Ali");
        hSet.add( "Ahmet");
        hSet.add( "Mehmet");
        Iterator i=hSet.iterator();
        while(i.hasNext()){
            System.out.println("Küme elemanı:"+i.next());
            i.remove();
        }
        System.out.println("Elemanlar silindi");
        System.out.println(hSet);
    }
}
```

---

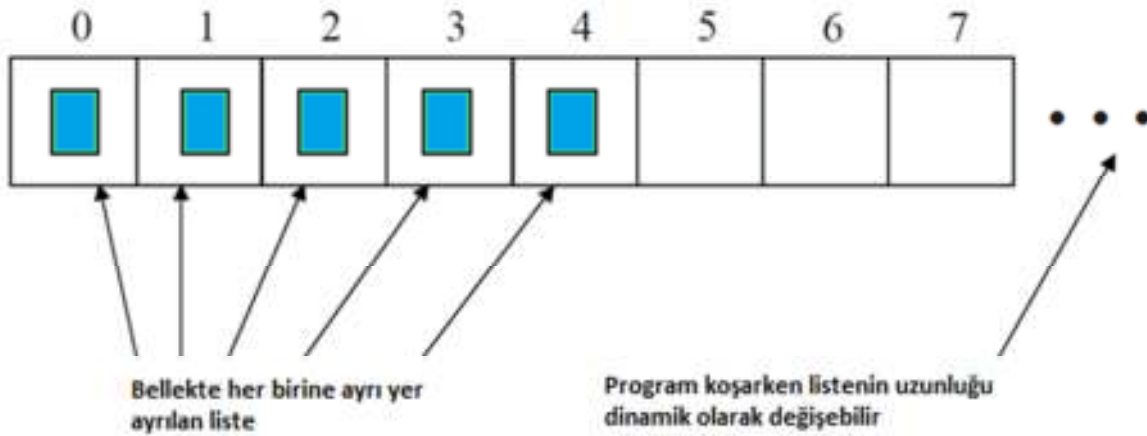
```
Küme elemanı:Ahmet
Küme elemanı:Mehmet
Küme elemanı:Ali
Elemanlar silindi
[]
```

## List (Liste) :

Sıralı elemanları içerisinde barındıran bir koleksiyondur. İçerisinde tekrarlı elemanlar olabilir. Java dilinde veri yapıları algoritmaları için özel sınıflar vardır. Yığın(stack), dinamik diziler(ArrayList, Vector) ve Bağlı listeler(LinkedList) gibi. Bu sınıfların tamamı List sınıfından türetilmiştir. List sınıfının temel yöntemleri, ekleme, silme ve ters çevirme gibi işlemlerdir.

*Listeler* koleksiyonların yaygın olarak kullanılan türüdür. Dizilerin kullanıldığı her yerde kullanılırlar. Ama veri işleme eyleminde, Dizilerin sağladığından daha çok şeye izin verirler. Listelerin her ögesi (terim) bellekte kendine özgü bir yer tutan veri yapılarıdır ve çok genel işlerin yapılmasına olanak sağlarlar. Listeler, bir çok bakımdan dizilere benzemekle birlikte, yeni öge eklendikçe uzunlukları kendiliğinden artar; dolayısıyla dizilere göre daha kullanışlıdır. Bunun yanında, veri işlemeye yarayan çok sayıda metot içerdikleri için, programcıya, dizilerin sağladığından daha büyük kolaylıklar sağlar.

Listeler, öğelerini bir dizi halinde depolar. Listede, aynı öğeler birden çok kez yer alabilir (duplicate).



## Linked List (Bağlı Liste)

Bağlı liste yapısında elemanlar eklenirken aralarına bir bağ konulur. Örneğin **bağlı liste** tanımı yapmak için aşağıdaki yapı kullanılır;

```
LinkedList <Tip> degisken_adi = new LinkedList <Tip>();
```

### List sınıfı ve alt sınıflara (ArrayList, LinkedList) ait bazı yöntemler;

<b>add</b>	Listeye eleman ekler
<b>addAll (Collection c)</b>	Parametrede verilen koleksiyonun bütün öğelerini listenin sonuna ekler
<b>clear()</b>	Listedeki tüm elemanları siler
<b>get (indis)</b>	İndisi belirtilen öğeyi listeden seçer
<b>set(i, b)</b>	b elemanını i indisli yere yerleştirir.
<b>indexOf()</b>	Aranan elemanın indisini döndürür, eleman yoksa -1değerini döndürür
<b>lastIndexOf()</b>	Aranan elemanın son indisini döndürür, eleman yoksa -1 değerini döndürür
<b>remove (int indis)</b>	İndis numarası verilen elemanı listeden kaldırır.
<b>size()</b>	Listedeki eleman sayısını verir
<b>toArray()</b>	Listedeki elemanları dizi elemanlarına dönüştürür.

## Örnek 2: Linked List örneği

```
import java.util.*;
public class BagliListe {
    public static void main(String[] args) {
        LinkedList<String> liste = new LinkedList<String>();
        liste.add("Portakal");
        liste.add("Limon");
        liste.add("Mandalina");
        liste.add("Armut");
        liste.add("Mandalina");
        liste.add("Elma");
        System.out.println("İlk liste = " + liste);
        liste.add(4, "Ayva");
        System.out.println("Değişen liste = " + liste);
        System.out.println("ilk öge = " + liste.getFirst());
        System.out.println("6.nci öge = " + liste.get(6));
        System.out.println("silinen = " + liste.removeFirst());
        System.out.println("silinen = " + liste.removeLast());
        System.out.println("Liste son hali = " + liste);
    }
}
```

## Örnek 2: Linked List örneği

### Programın ekran çıktısı:

```
<terminated> BagliListe [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (14 Kas 2011 11:13:20)
İlk liste = [Portakal, Limon, Mandalina, Armut, Mandalina, Elma]
Değişen liste = [Portakal, Limon, Mandalina, Armut, Ayva, Mandalina, Elma]
ilk öge = Portakal
6.nci öge = Elma
silinen = Portakal
silinen = Elma
Liste son hali = [Limon, Mandalina, Armut, Ayva, Mandalina]
```

## Örnek

```
import java.util.*;
public class Koleksiyon {
    public static void main(String[] args) {
        List<String> a = new ArrayList<String>();
        a.add("Çiğdem");
        a.add("Papatya");
        a.add("Kardelen");
        a.add("Lale");
        a.add("Sümbül");
        a.add("Gül");
        System.out.println(" a ArrayListi = " + a);
        a.add(3, "Diken");
        System.out.println(" a ArrayListi = " + a);
        a.set(3, "Çiğdem");
        System.out.println(" a ArrayListi = " + a);
        System.out.println("son index" + a.lastIndexOf("Çiğdem"));
        System.out.println("ilk index" + a.indexOf("Çiğdem"));
    }
}
```



```
a ArrayListi = [Çiğdem, Papatya, Kardelen, Lale, Sümbül, Gül]
a ArrayListi = [Çiğdem, Papatya, Kardelen, Diken, Lale, Sümbül, Gül]
a ArrayListi = [Çiğdem, Papatya, Kardelen, Çiğdem, Lale, Sümbül, Gül]
son index3
ilk index0
```

## Kuyruk (Queue) :

**Kuyruk** (queue), eleman ekleme işlemlerinin listenin sonundan (rear), çıkarma işlemlerinin ise listenin başından (front) yapıldığı özel bir yapıdır. Kuyruk, **İlk giren - İlk çıkar (First-In-First-Out (FIFO))** mantığı ile çalışır ve ara elemanlara erişim doğrudan yapılamaz. Kuyruğa ilk giren ilk çıkar’dan maksat, “ilk gelen ilk hizmet alır” anlamındadır.

Java 1.5 sürümü ile birlikte kuyruk (queue) işlemi için **Queue** isimli özel bir sınıf tanımlanmıştır. Örneğin String türde kuyruk isimli bir Queue sınıfı LinkedList sınıfından aşağıdaki gibi türetilebilir;

```
Queue <String> kyrk = new LinkedList <String>();
```

Bu sınıfın metotları şunlardır;

Queue Sınıfının Metotları	
<b>element ()</b>	Kuyruğun önündeki(başındaki) nesneyi döndürür (fakat silmez)
<b>offer ()</b>	Yeni bir elemanı kuyruk sonuna ekler
<b>peek ()</b>	Kuyruğun önündeki(başındaki) elemanı döndürür fakat silmez. Eğer kuyruk boş ise <b>null</b> değerini döndürür.
<b>poll ()</b>	Kuyruğun önündeki(başındaki) elemanı döndürür ve siler. Eğer kuyruk boş ise <b>null</b> değerini döndürür.
<b>remove ()</b>	Kuyruk önündeki nesneyi kuyruktan siler ve nesneyi döndürür
<b>size ()</b>	Kuyruktaki eleman sayısını döndürür. (Öncelik kuyruğu için)
<b>clear ()</b>	Kuyruktaki tüm elemanları siler. (Öncelik kuyruğu için)

**Örnek 5.** Kuyrukta bekleyen 4 kişiyi FIFO mantığı ile kuyruktan çıkaran programı yazınız.

**Çözüm:**

```

import java.util.*;
class Kuyruk {
    public static void main(String[] args) {
        Queue <String> kyrk = new LinkedList<String>();
        kyrk.offer("Bade");
        kyrk.offer("Zehra");
        kyrk.offer("Fatih");
        kyrk.offer("Hakan");
        System.out.println("Kuyruktan çıkartılıyor");
        while (!kyrk.isEmpty())
            System.out.print(kyrk.remove() + " ");
    }
}

```

4 kişiyi sırası ile kuyruğa ekledik ve bu kişiler FIFO (ilk giren ilk çıkar) mantığı ile kuyruktan çıkarıldı.

**Programın ekran çıktısı aşağıda gösterilmiştir.**

```
General Output
————Configuration: <Default>————
Kuyruktan çıkartılıyor
Bade Zehra Fatih Hakan
Process completed.
```

## Vector nesnesi

**Vector** sınıfı yalnızca nesneler içerebilir; temel/ilkel veri tiplerini içeremez. Temel veri tipleri kullanılacaksa, onların ilgili nesnesel veri tip karşılığı kullanılmalıdır. Örneğin int tipinden Vector sınıfı tanımlanamaz, ama Integer sınıfından tanımlanabilir.

Temel veri tipleri (int, long, char,...) Vector ve ArrayList'e doğrudan eklenemez ancak bunların nesnesel tipleri (wrapper sınıflar) eklenebilir. Temel veri tiplerinin nesnesel tip karşılıkları aşağıdaki tabloda listelenmiştir.

Temel Veri Tipleri	Nesnesel Veri Tip Karşılığı (Wrapper Class)
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean
	Object

Parametresiz Vector tanımlamasında dizi değişken boyutu 10 dur ve bu boyut dolarsa otomatik olarak dizi boyutu iki katına çıkar. Vektör tanımını farklı şekillerde yapabiliriz.

- 
- `Vector v = new Vector();` // 10 elemanlı bir v vektörü oluşturuldu,
  - `Vector v = new Vector(300);` // 300 elemanlı bir v vektörü oluşturuldu,
- 

Dizi değişken boyutu ve dizi de yer kalmaması durumunda arttırılacak miktar Vector tanımında belirtilebilir. Örneğin ' `Vector <String> A= new <String> Vector (20,5);` ' şeklindeki tanımda Vector, başlangıçta 20 elemanlık kapasiteye sahiptir, kapasite dolduğunda artım miktarı 5 olarak belirtilmiştir. Vector sınıfının genel ve yaygın kullanımı;

---

```
> Vector <tip> değişken_adi= new Vector <tip> ( );
```

---

şeklindedir.

## ArrayList Nesnesi

ArrayList, kullanım ve yapı olarak Vector'lere benzer. Kullanımı;

---

```
▪ ArrayList <tip> dizi_adi= new ArrayList <tip>();
```

---

Şeklindedir.

Dizi, ArrayList veya Vector sınıfı ile tanımlanırsa dizi boyutu istenildiği kadar arttırılıp, azaltılabilir. ArrayList ve Vector nesnelerini kullanabilmek için Java API deki **java.util.Vector** ya da **java.util.Arrays** paketlerinin import edilmesi / çağrılması gerekir. ArrayList ve Vector nesnelerinin bazı yöntemleri aşağıda verilmiştir.

Yöntem Adı	Açıklama	Kullanım Şekli
<code>add ( )</code>	Listesinin sonuna bir eleman eklemek için kullanılır.	<code>dizi.add (&lt;eklenecek değer&gt;)</code>

add(i, b)	i indisli b nesnesini listeye ekler (gerekirse elemanları kaydırır)	dizi.add(i, (<eklenecek değer>))
remove (indis)	İndis numarası verilen elemanı listeden çıkarmak için kullanılır.	dizi.remove (indis)
clear ( )	Dizideki bütün elemanları silmek için kullanılır.	dizi.clear( )
contains(b)	Eğer vektor veya liste b elemanını içeriyorsa true değerini döndürür	dizi.contains(<eleman>)
indexOf (aranan)	Dizideki elemanlar içerisinde arama yapar. Aranan değer bulunursa bulunduğu pozisyon, bulunamazsa, -1 sonucu döndürülecektir.	dizi.indexOf(<aranan>);
size( )	Listedeki elemanların sayısını verir	dizi.size( )
isEmpty( )	Listede hiç eleman yoksa True sonucunu üretir	
get(indis)	İndis numarasının gösterdiği string ifadeyi verir.	dizi.get.(indis)
set(i,b)	v elemanını i indisli yere yerleştirir.	dizi.set(indis,eleman)
toArray()	Vektörü diziye kopyalamak için kullanılır.	dizi.toArray()
capacity()	Vektörün kapasitesini/boyutunu verir.	Vektor.capacity( )
firstElement( )	İlk elemanı döndürür	Vektor.firsElement()
lastElement( )	En son elemanı döndürür.	Vektor.lastElement()

**Not:** Tablodaki son üç metot sadece Vector nesnesi ile kullanılmaktadır. Vector sınıfı ArrayList ‘ e göre daha işlevseldir.

### Örnek 3.

Bir dizi elemanlarının vektöre aktarımını ve vector nesnesinin yöntemlerini bir örnek üzerinde gösterelim.

---

```
import java.util.Vector;

public class Vektor {
    public static void main(String[] args) {
        Vector <String> isim = new Vector<String>();
        String[] ilkdizi = { "nuh", "yusuf", "ilyas",
                             "isa", "mehmet", "musa", "yunus", "adem" };
        // isimleri vektöre ekle
        System.out.println("Vektor boyutu..:" + isim.capacity());
        System.out.println("Vektor eleman sayısı..:" + isim.size());
    }
}
```

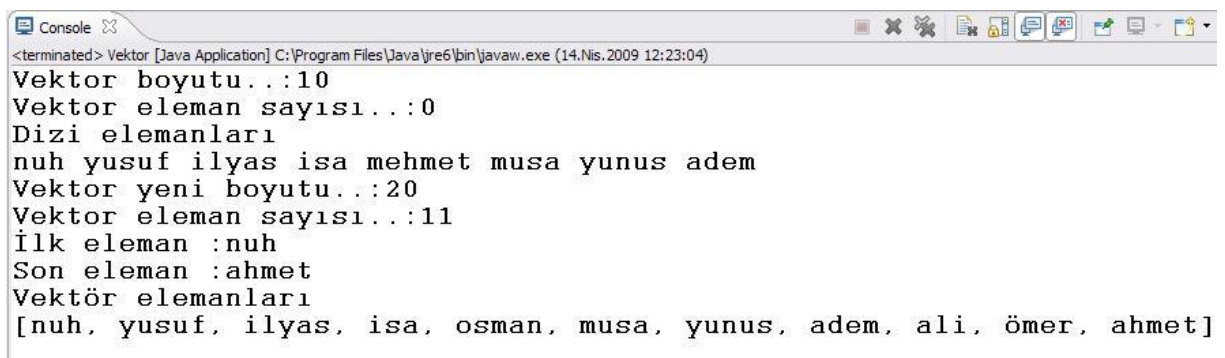
---

---

```
System.out.println("Dizi elemanları");
//Dizi elemanları vektöre aktarılıyor
for(String ad : ilkdizi) {
    isim.add(ad);
    System.out.print(ad + " ");
}
isim.set(4,"osman"); //4. eleman değişti
isim.add("ali");
isim.add("ömer");
isim.add("ahmet");
System.out.println("\nVektor yeni boyutu..:"+ isim.capacity());
System.out.println("Vektor eleman sayısı..:"+ isim.size());
System.out.println("İlk eleman :"+ isim.firstElement());
System.out.println("Son eleman :"+ isim.lastElement());
System.out.println("Vektör elemanları");
System.out.print(isim);
} }
```

---

### Programın ekran çıktısı;



```
<terminated> Vektor [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (14.Nis.2009 12:23:04)
Vektor boyutu..:10
Vektor eleman sayısı..:0
Dizi elemanları
nuh yusuf ilyas isa mehmet musa yunus adem
Vektor yeni boyutu..:20
Vektor eleman sayısı..:11
İlk eleman :nuh
Son eleman :ahmet
Vektör elemanları
[nuh, yusuf, ilyas, isa, osman, musa, yunus, adem, ali, ömer, ahmet]
```

### Örnek 2.

Tamsayılardan oluşan bir vektörü diziye aktaran ve bu dizinin en küçük elemanını ekranda gösteren programı yazalım.

### Çözüm:

Vektörü diziye aktarmak için dizi değişkeni Object tipinde seçildi ve ‘**v1.toArray()**’ komutu kullanıldı.

---

```
import java.util.Arrays;
import java.util.Vector;
public class Vektor1 {
    public static void main(String args[]) {
        Vector<Integer> v1 = new Vector<Integer>();
        v1.add(500);
        v1.add(45);
        v1.add(745);
        v1.add(842);
        v1.add(90);
        v1.add(230);
        Object[] dizi = v1.toArray(); // Vektörü diziye
        dönüştürdük
        for (int i = 0; i < dizi.length; i++) {
            System.out.println(dizi[i]);
        }
        Arrays.sort(dizi); // dizi elemanları sıralandı
        System.out.println("En küçük eleman..:" + dizi[0]);
    }
}
```

---

Programın ekran çıktısı aşağıdaki gibidir.

```
<terminated> Vektor1 [Java Application]
500
45
745
842
90
230
En küçük eleman.:45
```

### Örnek 3.

ArrayList yöntemlerini başka bir örnek üzerinde gösterdiğimiz aşağıdaki programın ekran çıktısı ne olur?

---

```
1. import java.util.* ;
2. class ArrayListornek
3. {
4. public static void main ( String[] args)    {
5. ArrayList<String> Ad = new ArrayList<String>();
6. Ad.add("Ahmet");
7. Ad.add("Ali");
8. Ad.add("Bade");
9. Ad.add("Ceyda");
10. Ad.add( 1, "Vedat");//Araya isim ekle
11. System.out.println("Dizinin yeni boyutu: " + Ad.size() );
12. for ( int j=0; j<Ad.size(); j++ )
13. System.out.println("İndis No " + j + ": " + Ad.get(j) );
14. Ad.remove(Ad.indexOf("Ali"));
15. Ad.remove(0);
16. System.out.println("Dizinin yeni eleman sayısı : " + Ad.size()
    );
17. for ( int j=0; j<Ad.size(); j++ )
18. System.out.println("İndis No " + j + ": " + Ad.get(j) );
19. }}
```

---

### Programın ekran çıktısı;

Programın 5. Satırında ArrayList nesnesi kullanılarak Ad isimli dinamik dizi tanımlandı ve bu diziye add() komutu (6. Ve 10. Satırlar arasında) ile 5 eleman yerleştirildi. Dizinin boyutu (size) ekrana yazdırıldı.

14 ve 15. Satırlarda dizi indis numarası esas alınarak iki eleman diziden silindi ve tekrar dizi boyutu ekrana yazdırıldı. Silinen elemanlardan sonraki dizinin hali indis numaraları ile ekranda gösterildi. Buna göre programı çalıştırdığımızda aşağıdaki ekran çıktısı elde edildi.



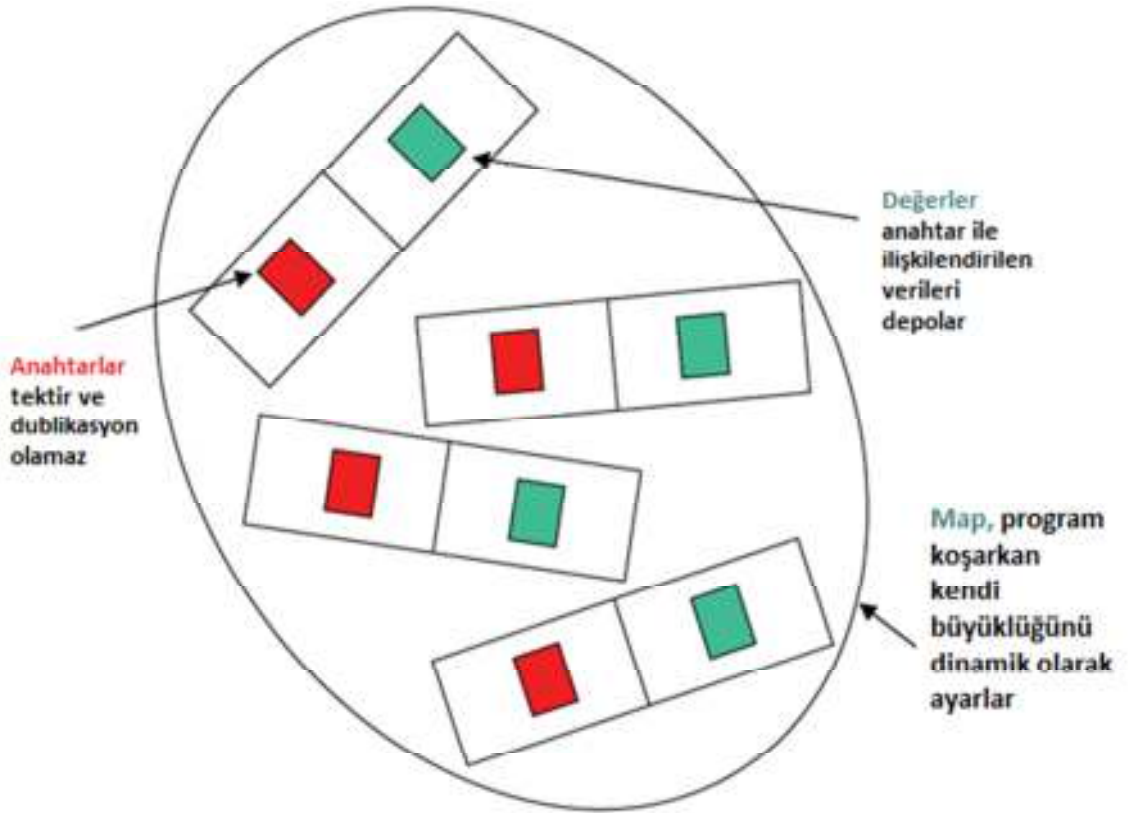
```
Console X
<terminated> ArrayListornek [Java Application] C:\Program Files\Java\jre6\bin
Dizinin yeni boyutu: 5
İndis No 0: Ahmet
İndis No 1: Vedat
İndis No 2: Ali
İndis No 3: Bade
İndis No 4: Ceyda
Dizinin yeni eleman sayısı :3
İndis No 0: Vedat
İndis No 1: Bade
İndis No 2: Ceyda
```

## Maps (Dönüşümler)

*Dönüşüm (Map)* 'ler, yapısal olarak, *Listeler* ve *Kümeler* 'den çok farklıdır. Öğeleri tek tek depolamak yerine nesne çiftlerini depo ederler. Depolanmak istenen her öğeye bir *anahtar* verilir; böylece bir öğe yerine bir öğe çifti oluşur ve bu çiftler depo edilirler. Depodaki her öğeye kendi anahtarıyla erişilir. Öğe çifti “*anahtar*” ve “*değer*” olmak üzere iki nesneden oluşur. Anahtar, oluşan çifti belirleyen işaretçidir; değer ise anahtara ilişkilendirilen bilgiyi içeren bir nesnedir.

*Map* içinde anahtarlar tektir; yani aynı anahtar birden çok değeri işaret edemez. Ancak, farklı iki anahtarın işaret ettiği değerlerde aynı veriler olabilir. Örneğin, adres defterinde farklı iki kişi aynı telefonu kullanabilir ya da aynı adreste kalabilirler. Örneğin, defterde kayıtlı herkese bir sıra numarası verilebilir.

Kümeler için olduğu gibi, bir *Map* öğeleri sıralamaz. Gerekirse, anahtara göre sıralama yapan *SortedMap* arayüzü kullanılır. Yeni öğeler geldikçe, *Map* kendi kendisinin boyunu (öge sayısını) artırabilir. Tersine olarak, öğe silinirse, *Map* kendi boyunu küçültür.



Map (dönüşüm)