Collections Framework

Java' da çeşitliliğin olduğu yapılardan biri de Collections (Yığın - Koleksiyon) Framework' tür. Aslında bu başlı başına bir konu olmakla birlikte sadece Collections Framework' ü içeren kitaplar da yer almaktadır.

Collections Framework aslında arka planda dizi yapısını kullanan ancak dizilere göre avantajlar içeren sınıf ve arayüzlerden oluşan bir yapıdır. Peki standart dizi kullanmak yerine bu mimariyi neden kullanalım ? Bunu uzunluk sınırı olmaması, tip kısıtlaması veya genişlemesi , kuyruk yapıları gibi bir çok avantajla örnekleyebiliriz. Kaldıki yeni nesil bir çok mimari sıralı yapılarını bu framework üzerine kurmaktadır.

Collections Framework sınıfları List, Set, Map interface lerine göre planlanmıştır. Bunun haricinde Queue ve java 1.6 ile dile dahil edilen Deque interface leri yer almaktadır.

Not : Java 1.5 itibari ile dile dahil edilen Generic kavramı sayesinde Collections yapısında önemli değişiklikler olmuştur. Bunun getirdiği en temel özellik tip kısıtlaması ve otomatik çevrim diyebiliriz.

List Arayüzü ve Sınıfları

List aslında bir interface yani arayüzdür ve Collection interface inden türetilmiştir. Bu yüzden içeriğinde iş yapan herhangi bir yapı yer almamaktadır. Ancak bu interface i kullanan ArrayList, LinkedList gibi gerçekleştirim sınıfları bulunmaktadır. List kullanımındaki amaç standart tek boyutlu diziler gibi tek sıralı kayıtlar tutmaktır.

List metodları

List interface' i için ArrayList ile birlikte temel metodları inceleyelim.

```
List<String> liste = new ArrayList<String>();
liste.add("Ankara");
liste.add("Istanbul");
liste.add("Izmir");
liste.add("Bursa");

List<String> yeniListe = new ArrayList<String>();
yeniListe.add("Istanbul");
yeniListe.add("Ankara");

liste.add("Balıkesir"); //Kayıt Ekleme

liste.add(0, "Çanakkale"); //İndeks numarasına kayıt ekleme

liste.get(0); //Sıra numaralı kaydı getirme

liste.remove("Çanakkale"); //Kaydı silme
liste.remove(0); //Sıra numarasına göre kaydı silme
liste.removeAll(yeniListe); //Liste içerisinde alt listedeki kayıtları siler

liste.addAll(yeniListe); //Başka bir listeyi ekleme
```

```
liste.contains("Melih"); //Liste içerisinde varsa true döner
liste.containsAll(yeniListe); //Liste içerisinde var olan listeyi arama

liste.size(); //Listenin uzunluğunu döner

liste.clear(); //Listedeki tüm kayıtları silme
String[] dizi = liste.toArray(new String[0]); //Listeyi diziye çevirir

liste.set(1, "Isparta"); // Sıra numaralı kaydın değerini değiştirir
liste.isEmpty(); //Liste boş mu diye kontrol eder
liste.indexOf("Ankara"); //Liste içerisinde arama yapıp kaydın sıra
numarasını getirir. Kayıt yoksa -1 döner

liste.lastIndexOf("Ankara"); //Kaydın en son hangi sırada geçtiğini döner

List<String> altListe = liste.subList(2, 4); //Liste icerisinde sıraya göre
alt liste alma

liste.retainAll(altListe); //Alt liste dışındakileri liste den çıkarır
```

Not : List bir interface olduğunu için new ile birlikte yeni nesne oluşturamayacaktı bu yüzden örneğimizi ArrayList ile birlikte gerçekleştirdik.

ArrayList

List interface ini kullanan bir gerçekleştirim sınıfıdır. Standart olarak liste şeklinde veri saklamak için kullanılır ve Collections Framework' ün en çok kullanılan sınıfıdır diyebiliriz.

Temel haliyle aşağıdaki gibi kullanılabilir.

```
import java.util.ArrayList;
import java.util.List;

public class Ornek {
	public static void main(String[] args) {
		ArrayList<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		System.out.println(liste.get(2));

	}
}
```

Eğer liste üzerindeki elemanları listelemek istiyorsak bu bölümdeki listeleme seçeneklerini inceleyebiliriz. Ancak biz basit liste örneklerinde aşağıdaki gibi for-each döngülerini kullanıyor olacağız.

```
import java.util.ArrayList;
import java.util.List;

public class Ornek {

	public static void main(String[] args) {
		ArrayList<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		for (String eleman : liste) {
			System.out.println(eleman);
		}

	}

}
```

LinkedList

LinkedList yine ArrayList gibi List interface' i için bir gerçekleştirim olduğu gibi Java 1.6 ile gelen Deque interface' i içinde bir gerçekleştirim uygular. Bu bize bir kuyruk yapısı yeteneği kazandıracaktır.

Örnek verecek olursak aşağıdaki kod bloğunda başa ve sona eklemeler yapılıyor.

```
import java.util.LinkedList;

public class Ornek {

	public static void main(String[] args) {
		LinkedList<String> liste = new LinkedList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		for (String eleman : liste) {
			System.out.println(eleman);
		}
		System.out.println("******* " + liste.size());

		liste.addFirst("Balıkesir"); //En başa atama
		liste.addLast("Adana"); //En sona atama

		liste.offerFirst("Eskişehir"); //En başa atama
		liste.offerLast("Edirne"); //En sona atama

		for (String eleman : liste) {
			System.out.println(eleman);
		}
		System.out.println("******* " + liste.size());

		System.out.println(liste.peekLast()); //Son kaydı getirir
```

```
            System.out.println(liste.getLast()); //Son kaydı getirir

            System.out.println(liste.removeLast()); //Son kaydı siler
            System.out.println(liste.pollLast()); //Son kaydı siler

      }
}
```

Buna göre çıktımız aşağıdaki gibi olacaktır.

```
Ankara
Istanbul
Izmir
Bursa
******* 4
Eskişehir
Balıkesir
Ankara
Istanbul
Izmir
Bursa
Adana
Edirne
******* 8
Edirne
Edirne
Edirne
Adana
```

Burada dikkat ederseniz başa ve sona eklemelerde add ve offer la başlayan metodlarımız bulunmaktadır. Aralarındaki fark eğer ekleme yapılamazsa add metodu hata üretirken offer metodu false dönecektir. Yine aynı şekilde getFirst() metodu ile ilk kayıt çağırılırsa liste boş iken hata üretilir peekFirst() metodu ile nesne yoksa null dönecektir.

Set Arayüzü ve Sınıfları

Set interface' i List türevleri ile aynı işleri yapacaktır ancak temel farkı Set arayüzünün tekillik sağlamasıdır.

HashSet

Set için temel gerçekleştirim HashSet sınıfıdır. HashSet ile amaç yine tekillik ifade etmesidir. Bunu örnekleyecek olursak aşağıdaki kod bloğunda ArrayList ve HashSet' e 3 kere Ankara kaydı ekleniyor.

```
import java.util.ArrayList;
import java.util.HashSet;

public class Ornek {

      public static void main(String[] args) {
            ArrayList<String> liste1 = new ArrayList<String>();
            liste1.add("Ankara");
            liste1.add("Ankara");
            liste1.add("Ankara");
            System.out.println("liste1.size() : " + liste1.size());
```

```
            HashSet<String> liste2 = new HashSet<String>();
            liste2.add("Ankara");
            liste2.add("Ankara");
            liste2.add("Ankara");
            System.out.println("liste2.size() : " + liste2.size());

      }
}
```

Örneğe göre çıktımız aşağıdaki gibidir.

```
liste1.size() : 3
liste2.size() : 1
```

Sonuç itibari ile Set türevi sınıflarda aynı kayıt varsa tekrar ekleme yapılmayacak ve add metodu false dönecektir. Bu durumda istediğimiz kadar Ankara kaydını eklesekte listemizde sadece bir tane olacaktır.

Not : Set içerisindeki kayıt karşılaştırmaları hasCode üzerinden yapılır.

Dikkat : Set h ekleme ile geriye dönük kayıt sorgulaması yapacağından List' e göre performans dezavantajı yaratabilir.

LinkedHashSet

HashSet ile eklenen bilgilerin sırası alınan hashCode a göre verilir yani eklediğimiz sırada bilgiye ulaşamayız. Eğer ekleme sırası bizim için önemliyse LinkedHashSet kullanabiliriz. Bu durumda liste sırası ekleme sıramıza göre olacaktır.

Bunu aşağıdaki kod ile örneklendirelim.

```
import java.util.HashSet;
import java.util.LinkedHashSet;

public class Ornek {

      public static void main(String[] args) {

            HashSet<String> liste1 = new HashSet<String>();
            liste1.add("Ankara");
            liste1.add("Istanbul");
            liste1.add("Izmir");
            liste1.add("Adana");

            for (String str : liste1) {
                  System.out.println(str);
            }

            System.out.println("**************");

            LinkedHashSet<String> liste2 = new LinkedHashSet<String>();
            liste2.add("Ankara");
```

```
            liste2.add("Istanbul");
            liste2.add("Izmir");
            liste2.add("Adana");

            for (String str : liste2) {
                  System.out.println(str);
            }
      }
}
```

Örneğimize göre çıktı aşağıdaki gibi olacaktır.

```
Ankara
Istanbul
Adana
Izmir
*************
Ankara
Istanbul
Izmir
Adana
```

Dikkat edersek ilk çıktı eklediğimiz sıraya göre olmazken ikincisi yani LinkedHashSet kullandığımız örnek doğru sırada gelmiştir.

TreeSet

Bilgiyi ağaç yapısında tutan bir standartı vardır. Bu performan avantajı sağlar ve bilgi sıralı olarak elde edilir.

Bunu örneklendirelim.

```
import java.util.TreeSet;

public class Ornek {

      public static void main(String[] args) {

            TreeSet<String> liste = new TreeSet<String>();
            liste.add("Ankara");
            liste.add("Istanbul");
            liste.add("Izmir");
            liste.add("Adana");

            for (String str : liste) {
                  System.out.println(str);
            }

      }
}
```

Buna göre çıktımız aşağıdaki gibi sıralı olacaktır.

```
Adana
Ankara
Istanbul
Izmir
```

Map Arayüzü ve Sınıfları

Şimdiye kadarki Collection örneklerinde bilgiye sıralı veya indeks numarasına göre ulaştık. Map interface' ini kullanan sınıflar ise bilgiyi anahtar – değer (key-value) ile saklar veya erişir. Bu durumda indeks yani sıra numarası kullanılmayacaktır.

HashMap

Map için temel gerçekleştirimdir. Amaç anahtar bilgisine göre değeri saklamaktır.

Bunu aşağıdaki gibi örneklendirelim.

```
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

public class Ornek {

	public static void main(String[] args) {

		HashMap<Integer, String> liste = new HashMap<Integer, String>();
		//Veri ekleme
		liste.put(10, "Balıkesir");
		liste.put(34, "İstanbul");
		liste.put(35, "İzmir");

		//Veriye erişim
		System.out.println(liste.get(10));

		//Veriyi silme dönüş olarak silinen kayıt alınıyor
		System.out.println(liste.remove(34));

		//Anahtar listesi alınıyor
		Set<Integer> anahtarListesi = liste.keySet();
		//Anahtarlar listeleniyor
		for (Integer anahtar : anahtarListesi) {
			System.out.println(anahtar);
		}

		System.out.println("**********");

		//Değer listesi alınıyor
		Collection<String> degerler =  liste.values();
		//Değerler listeleniyor
		for (String deger : degerler) {
			System.out.println(deger);
		}

		System.out.println("**********");
```

```
            //Kayıt anahtar ve değerle alınıyor
            Set<Map.Entry<Integer, String>> kayitlar = liste.entrySet();
            //Anahtar ve değerler listeleniyor
            for (Map.Entry<Integer, String> kayit : kayitlar) {
                  System.out.println(kayit.getKey()
                                    + " - " + kayit.getValue());
            }
      }
}
```

Örneğimize göre çıktı aşağıdaki gibi olacaktır.

```
Balıkesir
İstanbul
35
10
**********
İzmir
Balıkesir
**********
35 - İzmir
10 - Balıkesir
```

Dikkat edersek eklediğimiz her veriyi bir anahtar ile atadık. Yine silme ve getirme işlemlerinde de bu anahtarı kullandık. Eğer olmayan bir anahtar ile kayıt getirmeye çalışırsak geriye null değeri dönecektir.

Listelemede üç farklı erişimimiz bulunuyor. Bunlar birincisi keySet() metodu ile sadece anahtarları getirmek. Bu durumda Set tipinde anahtar listesini alıyor oluruz. values() metodu ise Collection tipinde bize değerleri getirecektir. Eğer her iki bilgiyide getirmek istersek entrySet() metodunu kullanabiliriz. Bu durumda Set tipinde Map.Entry interface' i ile hem anahtar hemde değer bilgisine ulaşmış oluruz.

Not : Anahtar alanımız Set tipinde olduğu için tekillik gerektirir ve aynı anahtar ile ekleme yapıldığında eski bilginin üzerine yazılmış olur.

Dikkat : HashMap listelerini yazdırırken ekleme sırasına uymadığını farkedebilirsiniz. Bu durumda LinkedHashMap veya sıralı getirmek için TreeMap kullanılabilir.

TreeMap

HashMap ten farklı olarak bilgiyi key alanına göre sıralı getirir. Bunu aşağıdaki gibi örnekleyelim.

```
import java.util.HashMap;
import java.util.TreeMap;
import java.util.Map.Entry;

public class Ornek {

      public static void main(String[] args) {

            //HashMap listesi
```

```
            HashMap<Integer, String> liste1 = new HashMap<Integer, String>();
            liste1.put(34, "İstanbul");
            liste1.put(10, "Balıkesir");
            liste1.put(35, "İzmir");
            liste1.put(16, "Bursa");

            System.out.println("HashMap listesi");
            for (Entry<Integer, String> kayit : liste1.entrySet()) {
                  System.out.println(kayit.getKey() +
                                    " - " + kayit.getValue());
            }

            System.out.println("************");

            //TreeMap listesi
            TreeMap<Integer, String> liste2 = new TreeMap<Integer, String>();
            liste2.put(34, "İstanbul");
            liste2.put(10, "Balıkesir");
            liste2.put(35, "İzmir");
            liste2.put(16, "Bursa");

            System.out.println("TreeMap listesi");
            for (Entry<Integer, String> kayit : liste2.entrySet()) {
                  System.out.println(kayit.getKey() +
                                    " - " + kayit.getValue());
            }

      }
}
```

Örneğe göre çıktımız aşağıdaki gibi olacaktır.

```
HashMap listesi
34 - İstanbul
16 - Bursa
35 - İzmir
10 - Balıkesir
************
TreeMap listesi
10 - Balıkesir
16 - Bursa
34 - İstanbul
35 - İzmir
```

Dikkat ederseniz ilk listede yani HashMap' te bilgi herhangi bir sıraya göre getirilmedi. Ancak ikinci TreeMap listesinde anahtar sırasına göre listemizi görebilirsiniz.

LinkedHashMap

HashMap herhangi bir sıraya uymazken TreeMap anahtar sırasına uyuyordu. Peki listeyi eklediğim sıraya göre almak istersem ne yapmalıyım ? Bu durumda LinkedHashMap kullanabiliriz. Adından da anlaşılacağı gibi eklenen kayıtlar birbirlerine linklenip çağırma durumunda yine eklenen sırada getiriliyor.

Bunun HashMap ile farkını aşağıdaki gibi örneklendirelim.

```
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Map.Entry;

public class Ornek {

	public static void main(String[] args) {

		//HashMap listesi
		HashMap<Integer, String> liste1 = new HashMap<Integer, String>();
		liste1.put(34, "İstanbul");
		liste1.put(10, "Balıkesir");
		liste1.put(35, "İzmir");
		liste1.put(16, "Bursa");

		System.out.println("HashMap listesi");
		for (Entry<Integer, String> kayit : liste1.entrySet()) {
			System.out.println(kayit.getKey() +
					" - " + kayit.getValue());
		}

		System.out.println("************");

		//LinkedHashMap listesi
		LinkedHashMap<Integer, String> liste2 =
					new LinkedHashMap<Integer, String>();
		liste2.put(34, "İstanbul");
		liste2.put(10, "Balıkesir");
		liste2.put(35, "İzmir");
		liste2.put(16, "Bursa");

		System.out.println("LinkedHashMap listesi");
		for (Entry<Integer, String> kayit : liste2.entrySet()) {
			System.out.println(kayit.getKey() +
					" - " + kayit.getValue());
		}

	}
}
```

Örneğe göre çıktımız aşağıdaki gibi olacaktır.

```
HashMap listesi
34 - İstanbul
16 - Bursa
35 - İzmir
10 - Balıkesir
************
LinkedHashMap listesi
34 - İstanbul
10 - Balıkesir
35 - İzmir
16 - Bursa
```

Buna göre HashMap listesi belirgin bir sırada gelmezken LinkedHashMap ile eklediğimiz sıraya göre bilgilere ulaşabiliriz.

Liste Üzerinde Gezinmek

Veri listeleri üzerindeki bilgiyi çekmek için metodları kullanırız örneğin List sınıfları için get() metodu ve indeks numarası ile ilgili veriye ulaşım sağlanabilir. Ancak dizilerde gördüğümüz gibi döngüler ile bu bilgileri yazdırmak için birden fazla seçeneğimiz vardir.

Sıra numarasına göre listelemek

Birinci seçeneğimiz son indeks numarasına kadar döngü içerisinde bir int değişken ile bilgiyi almaktır.

```
import java.util.ArrayList;
import java.util.List;

public class Ornek {
	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		for (int i = 0; i < liste.size(); i++) {
			System.out.println(i +  " - " + liste.get(i));
		}
	}
}
```

Buna göre int sayısı listenin uzunluğu kadar artırılmakta ve her seferinde liste.get(indeks_numarası) metodu ile ilgili siradaki nesne alınmaktadır.

Buna göre çıktı aşağıdaki gibi olur.

```
0 - Ankara
1 - Istanbul
2 - Izmir
3 - Bursa
```

Eğer buradaki nesne String tipinde değilde Ogrenci gibi bir sınıf tipinde olsaydı aynı işlem aşağıdaki gibi yapılabilirdi.

```
import java.util.ArrayList;
import java.util.List;

public class Ornek {
	public static void main(String[] args) {
		List<Ogrenci> liste = new ArrayList<Ogrenci>();
```

```
            liste.add(new Ogrenci("Melih"));
            liste.add(new Ogrenci("Ahmet"));
            liste.add(new Ogrenci("Mehmet"));

            for (int i = 0; i < liste.size(); i++) {
                  Ogrenci siradakiOgrenci = liste.get(i);
                  System.out.println(i + " - " + siradakiOgrenci.getAd());
            }
      }
}
```

Buna göre çıktımız aşağıdaki gibidir.

```
0 - Melih
1 - Ahmet
2 - Mehmet
```

For-each döngüsü ile listelemek

For-each döngüsü daha önce gördüğümüz gibi Java 1.5 ile dile dahil olmuş ve iterasyon şeklindeki gösterimler için getirilmiştir. Benzer örnekleri daha önce diziler kısmında yapmıştık yine aynı şekilde kullanımları Collections Framework' e ait sınıflarda gerçekleştirebiliriz.

Dikkat : Daha önceki konulardan da hatırlayacağımız gibi for-each kullanımda each keyworkd ü bulunmamaktadır. Sadece aşağıdaki örnekteki gibi iterasyon şeklinde for kullanımlarına for-each denmektedir.

Az önce verdiğimiz örneği for-each döngüsü ile gerçekleştirelim.

```
import java.util.ArrayList;
import java.util.List;

public class Ornek {
      public static void main(String[] args) {
            List<String> liste = new ArrayList<String>();
            liste.add("Ankara");
            liste.add("Istanbul");
            liste.add("Izmir");
            liste.add("Bursa");

            for (String s : liste) {
                  System.out.println(s);
            }
      }
}
```

Buna göre bilgiye sıra numarası ile erişmek yerine artan sıralı şekilde erişmiş olduk. Bu durum Java 1.5 versiyonundan önce kullanılan Iterator interface ine benzemektedir.

Yine benzer durumu Ogrenci sınıfı ile gerçekleştirecek olursak Generic kullanımı sayesinde tip çevrimine ihtiyac duymadan aşağıdaki yöntem ile erişim sağlayabiliriz.

```
import java.util.ArrayList;
import java.util.List;

public class Ornek {
	public static void main(String[] args) {
		List<Ogrenci> liste = new ArrayList<Ogrenci>();
		liste.add(new Ogrenci("Melih"));
		liste.add(new Ogrenci("Ahmet"));
		liste.add(new Ogrenci("Mehmet"));

		for (Ogrenci ogr : liste) {
			System.out.println(ogr.getAd());
		}
	}
}
```

Not : For-each döngüsü sadece Iterable interface ini kullanan sınıflar için geçerlidir. Tüm Collections sınıfları buna dahildir.

Iterator ile listelemek

Iterator interface' i özellikle Java 1.5 ten önce sıralı erişimlerde tek alternatifimizdi diyebiliriz. Java 1.5 sonrasında ise for-each döngüleri sayesinde bu yöntem çok tercih edilmeyebiliyor. List üzerinden Iterator tipinde bir nesne elde etmek istiyorsanız liste.iterator() metodunu kullanmanız gerekir. Bu durumda size listeye ait Generic tipte bir iterasyon dönecektir.

Aşağıdaki örnekte listemiz üzerinde Iterator interface' i ile verilerin listelendiğini görebiliriz.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Ornek {
	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		Iterator<String> sira = liste.iterator();
		while(sira.hasNext()){
			System.out.println(sira.next());
		}
	}
}
```

Buna göre liste.iterator() ile Iterator tipinde bir nesne yaratılır. Ardından while döngüsü içerisinde hasNext() ile hareket edip etmediği kontrol edilir. Eğer hareket sağlanıyorsa ki bu durum bir sonraki verinin olduğunu ifade eder while içerisine girilip next() metodu ile ilgili veriye erişilir. İlgili kod bloğumuzun çıktısıda aşağıdaki gibi olacaktır.

```
Ankara
Istanbul
Izmir
Bursa
```

Eğer Iterator nesnesi üzerinde remove() metodunu kullanırsanız ilgili kaydın silinmesi gerçekleşecektir. Aşağıdaki örnekte eğer Izmir kaydına denk gelirsek yazdırma işlemi yerine kaydı siliyoruz. Bu durumda kayıt List içerisinden çıkarılıyor.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Ornek {
	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		System.out.println(liste.size());

		Iterator<String> sira = liste.iterator();
		while(sira.hasNext()){
			String eleman = sira.next();
			if(eleman.equals("Izmir")){
				sira.remove();
			}
			else{
				System.out.println(eleman);
			}
		}
		System.out.println(liste.size());
	}
}
```

Kodun çıktısı aşağıdaki gibi olacaktır.

```
4
Ankara
Istanbul
Bursa
3
```

Burada dikkat edersek ilk başta liste uzunluğu 4 iken işlem sonrasında 3 oluyor. Yeni remove() metodu sadece iterator değil liste üzerinde de değişikliğe neden oluyor.

ListIterator ile listelemek

ListIterator arayüzü Iterator arayüzünden türemiştir ancak bazı ekstra özellikler getirir. Bunlardan başlıcaları istenen sıra numarasından başlayabilme, geriye doğru hareket, kayıt üzerinde değişiklik yapabilme diyebiliriz.

Peki istenen sıra numarasından kastettiğimiz nedir ? Örneğin elimizde 100.000 adet kayıt var ve bu kayıtlar içerisinde 50.000 sonrasının listesini istiyoruz. Bu durumda Iterator sırasının 50.000 numaralı indeks ile başalatabiliriz. Bunun küçük bir örneği aşağıda bulunmaktadır.

```
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;

public class Ornek {
	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		ListIterator<String> sira = liste.listIterator(2);
		while (sira.hasNext()) {
			System.out.println(sira.next());
		}

	}
}
```

Buna göre çıktımız aşağıdaki gibi olacaktır. Yani kayıtlar 2. sıra itibari ile başlayacaktır.

```
Izmir
Bursa
```

Veri üzerinde değişiklik içinse aşağıdaki örneği verebiliriz.

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;

public class Ornek {
	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
```

```
            liste.add("Bursa");

            ListIterator<String> sira = liste.listIterator();
            while(sira.hasNext()){
                  String eleman = sira.next();
                  sira.set(eleman + " - ek");
            }

            for (String s : liste) {
                  System.out.println(s);
            }
      }
}
```

Örneğe göre öncelikli olarak veriye erişim sonrasında set metodu ile ilgili veri üzerindeki elemana ekleme yapıyoruz. Bu durumda liste üzerinde de gerçekleşen değişiklik sonrasında çıktımız aşağıdaki gibi olacaktır.

```
Ankara - ek
Istanbul - ek
Izmir - ek
Bursa – ek
```

Bir diğer özellik olan geriye ilerleme Iterator' ün bir önceki değerine eriştirmeyi sağlar. Aşağıdaki örnekte önceki ve sonraki sıra numaraları ile birlikte önce ileriye sonra geriye doğru hareketle bilgiler listelenmektedir.

```
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;

public class Ornek {
      public static void main(String[] args) {
            List<String> liste = new ArrayList<String>();
            liste.add("Ankara");
            liste.add("Istanbul");
            liste.add("Izmir");
            liste.add("Bursa");

            ListIterator<String> sira = liste.listIterator(0);
            while (sira.hasNext()) {
                  System.out.println(sira.next());
                  System.out.println("Sonraki sıra: " + sira.nextIndex());
            }

            while (sira.hasPrevious()) {
                  System.out.println(sira.previous());
                  System.out.println("Önceki sıra: " + sira.previousIndex());
            }


      }
}
```

Buna göre çıktımız aşağıdaki gibidir.

```
Ankara
Bir sonraki sıra : 1
Istanbul
Bir sonraki sıra : 2
Izmir
Bir sonraki sıra : 3
Bursa
Bir sonraki sıra : 4
Bursa
Bir önceki sıra : 2
Izmir
Bir önceki sıra : 1
Istanbul
Bir önceki sıra : 0
Ankara
Bir önceki sıra : -1
```

Dikkat : Iterator yerine for-each döngüsünü kullanabilirsiniz. Ancak bu size ListIterator' deki gibi bilgi üzerinde değişiklik veya geriye doğru ilerleme imkanı vermez.

Listelerde Sıralama İşlemleri

Comparable Arayüzü

Java içerisinde TreeSet veya TreeMap gibi sıralı bir liste kullandığımızda bu sırayı neye göre yapacaktır ?

Bunu örnekleyecek olursak aşağıdaki gibi TreeSet tipinde bir liste oluşturalım.

```
import java.util.TreeSet;

public class Ornek {

	public static void main(String[] args) {
		TreeSet<String> liste = new TreeSet<String>();
		liste.add("Ankara");
		liste.add("İstanbul");
		liste.add("İzmir");
		liste.add("Adana");

		for (String str : liste) {
			System.out.println(str);
		}

	}
}
```

Buna göre çıktımız aşağıdaki gibi harfe göre sıralı olacaktır.

```
Adana
Ankara
İstanbul
İzmir
```

Peki bunun harfe göre sıralı olmasının nedeni nedir yani buna nasıl karar verilmiştir ? Aslında olayın temelinde Comparable interface i yatmaktadır ve Integer ve String gibi sınıflarda Comparable interface' inin gerçekleştirimi bulunmaktadır.

Comprable interface i aşağıdaki gibidir.

```
package java.lang;
import java.util.*;

public interface Comparable<T> {
    public int compareTo(T o);
}
```

Buna göre compareTo metodunu ezmemizi beklenmektedir ve geriye bir int tipinde değer dönmelidir. Bu int değer compareTo metoduna gelen ilgili nesne parametresi için karşılaştırma yapmanızı eğer değer küçükse 0 dan küçük, büyükse 0 dan büyük ve eğer eşitse 0 dönmenizi bekleyecektir. Karşılaştırmalarda buna göre yapılacaktır.

Örneğin String sınıfının java içerisinde tanımı aşağıdaki gibidir yani Comparable interface ini kullanmaktadır. İşte bu yüzden String işlemleri otomatik olarak sıralanabilmektedir.

```
public final class String implements java.io.Serializable,
Comparable<String>, CharSequence
{
}
```

Şimdi az önce String ile yaptığımız TreeSet örneğini Sehir sınıfı için yapalım.

Sehir sınıfım aşağıdaki gibi olsun.

```
public class Sehir {

	public Sehir(Integer palaka, String sehirAdi) {
		this.palaka = palaka;
		this.sehirAdi = sehirAdi;
	}

	private Integer palaka;
	private String sehirAdi;

	public void setPalaka(Integer palaka) {
		this.palaka = palaka;
	}
```

```
	public Integer getPalaka() {
		return palaka;
	}

	public void setSehirAdi(String sehirAdi) {
		this.sehirAdi = sehirAdi;
	}

	public String getSehirAdi() {
		return sehirAdi;
	}
}
```

Şimdi bu sınıftan aşağıdaki gibi bir TreeSet listesi oluşturalım.

```
import java.util.TreeSet;

public class Ornek {

	public static void main(String[] args) {
		TreeSet<Sehir> liste = new TreeSet<Sehir>();
		liste.add(new Sehir(6, "Ankara"));
		liste.add(new Sehir(34, "İstanbul"));
		liste.add(new Sehir(35, "İzmir"));
		liste.add(new Sehir(1, "Adana"));

		for (Sehir sehir : liste) {
			System.out.println(sehir.getPalaka() +
					" - " + sehir.getSehirAdi());
		}
	}
}
```

Buna göre çıktımız aşağıdaki gibi olacaktır ! Yani hata alıyor olacağız.

```
Exception in thread "main" java.lang.ClassCastException: Sehir cannot be cast
to java.lang.Comparable
	at java.util.TreeMap.put(Unknown Source)
	at java.util.TreeSet.add(Unknown Source)
	at Ornek.main(Ornek.java:8)
```

Aslında hatanın nedeni oldukça net. Sehir sınıfının bir Comparable olmadığı daha doğrusu bu interface i kullanmadığı belirtiliyor. Sebepse TreeSet gibi sıralama işlemlerinin karşılaştırma için bir Comparable interface ini dolayısıyla da compareTo metodunu kullanmak istemeleridir. Aksi halde zaten bu sınıf karşılaştırmayı plaka koduna göre mi yoksa şehir adına göremi yapacağını bilemeyecektir.

Buna göre Sehir sınıfım aşağıdaki gibi olmalıdır.

```
public class Sehir implements Comparable<Sehir>{
```

```
	public Sehir(Integer palaka, String sehirAdi) {
		this.palaka = palaka;
		this.sehirAdi = sehirAdi;
	}

	private Integer palaka;
	private String sehirAdi;

	public void setPalaka(Integer palaka) {
		this.palaka = palaka;
	}

	public Integer getPalaka() {
		return palaka;
	}

	public void setSehirAdi(String sehirAdi) {
		this.sehirAdi = sehirAdi;
	}

	public String getSehirAdi() {
		return sehirAdi;
	}

	@Override
	public int compareTo(Sehir o) {

		return this.getPalaka().compareTo(o.getPalaka());
	}
}
```

En sonda yazdığımız compareTo metoduna dikkat edelim. Burada plaka kodlarına göre karşılaştırma yapıp dönüş sağlıyoruz. Buna göre şimdi Ornek sınıfını çalıştırdığımızda çıktımız aşağıdaki gibi plaka koduna göre sıralı olarak gelecektir.

```
1 - Adana
6 - Ankara
34 - İstanbul
35 - İzmir
```

Dikkat : Sıralama işlemi yaptığınız sınıflarda Comparable arayüzü kullanılmalıdır.

Comparator Arayüzü

Az önceki Sehir sınıfı için Comparable arayüzünün gerçekleştirimini sağladığımda bir tane compareTo metodu yazabildim. Bu durumda sadece plaka numarasına göre sıralama yapabildik. Peki ben alternatif olarak istediğim durumlarda listenin isim sırasına göre olmasını istersem ne yapmalıyım ? Bunun için Comparator arayüzü için gerçekleştirim saglamış bir sınıf kullanılabilir.

Comparator arayüzü aşağıdaki gibidir.

```
package java.util;

public interface Comparator<T> {
    int compare(T o1, T o2);

    boolean equals(Object obj);
}
```

Buradaki compare metodu iki nesne için karşılaştırma yapmalı ve küçükse 0 dan küçük, büyükse 0 dan büyük, eğer eşitsede 0 değeri döndürmelidir.

Not : equals metodu karşılaştırma için kullanımaktadır ve gerçekleştirimi yani ezilmesi zorunlu değildir. Gerçekleştirim yoksa Object sınıfındakini kullanır.

Örnek verecek olursak aşağıda Sehir listesinin ada göre sıralanmasını sağlayan bir Comparator sınıfı yer almaktadır.

```
import java.util.Comparator;

public class SehirAdaGoreSiralama implements Comparator<Sehir> {

      @Override
      public int compare(Sehir s1, Sehir s2) {
            return s1.getSehirAdi().compareToIgnoreCase(s2.getSehirAdi());
      }

}
```

Şimdi bu sınıfı TreeSet' te sıralama amaçlı kullanalım.

```
import java.util.TreeSet;

public class Ornek {

      public static void main(String[] args) {
            SehirAdaGoreSiralama adaGoreSiralama =
                                    new SehirAdaGoreSiralama();
            TreeSet<Sehir> liste = new TreeSet<Sehir>(adaGoreSiralama);
            liste.add(new Sehir(6, "Ankara"));
            liste.add(new Sehir(34, "İstanbul"));
            liste.add(new Sehir(35, "İzmir"));
            liste.add(new Sehir(1, "Adana"));
            liste.add(new Sehir(75, "Ardahan"));

            for (Sehir sehir : liste) {
                  System.out.println(sehir.getPalaka() +
                                    " - " + sehir.getSehirAdi());
            }
      }
}
```

Dikkat ederseniz Comporator parametresini constucter' da gönderdik. Buna göre çıktımız aşağıdaki gibi şehir adına göre sıralı gelecektir.

```
1 - Adana
6 - Ankara
75 - Ardahan
34 - İstanbul
35 - İzmir
```

Yardımcı Sınıflar

Java içerisinde dizi ve Collection Framework üzerinde işlem yapabileceğimiz Arrays ve Collections gibi yardımcı sınıflar bulunmaktadır.

Collections Sınıfı

Collections sınıfı Collection Framework elemanları için kullanılan bir yardımcı sınıftır. Bize bir çok kolaylaştırılmış özellik getirmektedir. Collections sınıfı kurucu metodu private olduğundan bir nesne oluşturulamaz ve bunun yerine static olan metodları kullanılır.

Collections özelliklerini aşağıdaki gibi inceleyelim.

Sıralama

Sıralamayı aşağıdaki örnekteki gibi düz ve ters olmak üzere yapabiliriz. Bu durumda compareTo metodundaki pozitif ve negatif dönüşleri dikkate alacaktır.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Ornek {

	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		Collections.sort(liste);

		for (String str : liste) {
			System.out.println(str);
		}
		System.out.println("**********");

		Collections.reverse(liste);
		for (String str : liste) {
			System.out.println(str);
		}
	}
}
```

Örneğe göre çıktımız aşağıdaki gibi olacaktır.

```
Ankara
Bursa
Istanbul
Izmir
**********
Izmir
Istanbul
Bursa
Ankara
```

Dikkat : Yinede isteğimiz duruma göre sıralama yapmak istiyorsak sort metoduna ekstradan bir Comparator gönderebiliriz.

Arama

Eğer kayıtlar üzerinde arama yapmak istersek aşağıdaki gibi binarySearch metodu kullanabilabilir.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Ornek {

	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		int sonuc = Collections.binarySearch(liste, "Izmir");
		System.out.println("Aranan kayıt sırası : " + sonuc);
	}
}
```

Arama sonucunda çıktımız aşağıdaki gibi olacaktır.

```
Aranan kayıt sırası : 2
```

Alternatif olarak liste içerisinde bir alt liste aramasıda yapabiliriz. Bunun için indexOf veya lastIndexOf metodları kullanılabilir. Aşağıdaki örnekte liste içerisinde bir liste araması yapılıp başlangıç indeksi döndürülüyor.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
```

```
public class Ornek {

	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		List<String> liste2 = new ArrayList<String>();
		liste2.add("Izmir");
		liste2.add("Bursa");



		int sonuc = Collections.indexOfSubList(liste, liste2);
		System.out.println("Aranan kayıt sırası : " + sonuc);
	}
}
```

Buna göre çıktı aşağıdaki gibidir.

```
Aranan kayıt sırası : 2
```

Not : Arama sonucunda listede kayıt bulunamazsa geriye -1 değeri dönecektir.

Arama alternatiflerinden biride listedeki minumum ve maksimum değerleri bulmaktır. Bu aşağıdaki gibi örneklendirelim.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Ornek {

	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");
		liste.add("Bursa");

		String enBuyuk = Collections.max(liste);
		String enKucuk = Collections.min(liste);
		System.out.println("en buyuk : " + enBuyuk);
		System.out.println("en kucuk : " + enKucuk);
	}
}
```

Yukarıdaki kodun çıktısı aşağıdaki gibi olacaktır.

```
en buyuk : Izmir
en kucuk : Ankara
```

Burada harf sırasına göre en büyük ve en küçük değerler bulundu. Eğer sıralamayı istediğimiz gibi yapmak istersek yine bir Comparator parametresi gönderebiliriz.

Diğer Metodlar

Alternatif metodları aşağıdaki gibi iki ayrı örnekte inceleyelim.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Ornek {

	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");

		System.out.println("listeye ekleme yapar");
		Collections.addAll(liste, "Izmir", "Edirne", "Balıkesir");
		for (String str : liste) {
			System.out.println(str);
		}

		System.out.println("****************");

		System.out.println("listede kac tane kayit gectigini verir");
		int kayitSayisi = Collections.frequency(liste, "Izmir");
		System.out.println("Izmir sayisi : " + kayitSayisi);

		System.out.println("****************");

		System.out.println("liste yi liste2 ye kopyalar");
		List<String> liste2 = new ArrayList<String>();
		Collections.copy(liste, liste2);
		for (String str : liste2) {
			System.out.println(str);
		}

		System.out.println("****************");

		System.out.println("elemanların yerlerini karıştırır");
		Collections.shuffle(liste);
		for (String str : liste) {
			System.out.println(str);
		}

		System.out.println("****************");

		System.out.println("listedeki tum elemanlari Izmir yapar");
		Collections.fill(liste, "Izmir");
```

```
            for (String str : liste) {
                  System.out.println(str);
            }

            System.out.println("****************");

            System.out.println("listedeki değerleri değiştirir");
            Collections.replaceAll(liste, "Izmir", "Ankara");
            for (String str : liste) {
                  System.out.println(str);
            }
      }
}
```

Buna göre birinci örneğimizin çıktısı aşağıdaki gibidir.

```
listeye ekleme yapar
Ankara
Istanbul
Izmir
Izmir
Edirne
Balıkesir
****************
listede kac tane kayit gectigini verir
Izmir sayisi : 2
****************
liste yi liste2 ye kopyalar
****************
liste deki elemanların yerlerini karıştırır
Balıkesir
Ankara
Izmir
Istanbul
Edirne
Izmir
****************
listedeki tum elemanlari Izmir yapar
Izmir
Izmir
Izmir
Izmir
Izmir
Izmir
****************
listedeki değerleri değiştirir
Ankara
Ankara
Ankara
Ankara
Ankara
Ankara
```

Bunun dışında Collections sınıfı içerisinde aşağıdaki gibi özellikler bulunmaktadır.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class Ornek {

	public static void main(String[] args) {
		List<String> liste = new ArrayList<String>();
		liste.add("Ankara");
		liste.add("Istanbul");
		liste.add("Izmir");

		// Değiştirilemez List yaratıldı
		List<String> liste2 = Collections.emptyList();
		// liste2 değiştirilemez bir litedir kod hata verir.
		// liste2.add("Ankara");

		// Değiştirilemez Set yaratıldı
		Set<String> liste3 = Collections.emptySet();
		// Değiştirilemez Map yaratıldı
		Map<Integer, String> liste4 = Collections.emptyMap();

		// 10 Tane Ankara elemanı olan bir liste oluşturdu.
		List<String> liste5 = Collections.nCopies(10, "Ankara");

		// Hata verecektir çünkü liste değiştirilemez
		// liste5.add("Izmir");

	}
}
```

Arrays Sınıfı

Collections sınıfının bir benzeri olarak Java içerisinde standart diziler için Arrays sınıfı bulunmaktadır. Bu sınıfla birlikte sıralama, arama veya atama gibi işlemler yapılabilir.

Arrays sınıfı içerisindeki metodların kullanımı aşağıdaki gibidir.

```
import java.util.Arrays;
import java.util.List;

public class Ornek {

	public static void main(String[] args) {
		String[] dizi = { "Ankara", "İstanbul", "İzmir", "Balıkesir" };

		// Dizi içerisinde arama yapıp sıra numarasını döndürür
		int sonuc = Arrays.binarySearch(dizi, "İzmir");
		System.out.println("Arama sonucu : " + sonuc);

		// Diziyi List tipine çevirir
		List<String> listTipindeDizi = Arrays.asList(dizi);
```

```
            // Dizi uzunluğu yetersizse yeni bir dizi oluşturur
            String[] yeniDizi = Arrays.copyOf(dizi, 20);
            System.out.println("Yeni dizi : " + yeniDizi.length);

            String[] altDizi = Arrays.copyOfRange(dizi, 2, 4);
            System.out.println("Alt dizi : " + altDizi.length);

            // Elemanları String olarak yazdırmak için kullanılır
            System.out.println(Arrays.deepToString(dizi));

            String[] ornekDizi = new String[4];
            // Dizinin tüm elemanlarına atama yapar
            Arrays.fill(ornekDizi, "İzmir");
            System.out.println(Arrays.deepToString(ornekDizi));

            // Diziyi sıralar (Comparator verilebilir)
            Arrays.sort(dizi);
        }
}
```