# LinkedList

**java.util**
## Class LinkedList
```
java.lang.Object
  └java.util.AbstractCollection
      └java.util.AbstractList
          └java.util.AbstractSequentialList
              └java.util.LinkedList
```

*LinkedList* sınıfı *List* arayüzünü kılgılar (implemantation). Dolayısıyla listelerle ilgili bütün metotları kılgılamış olur ve *null* dahil bütün öğelere izin verir. Özellikle *get(), remove()* ve *insert()* metotlarını listenin başına, sonuna veya ortasında bir yere uygulayabilir. Böylece bağlı listelerden, isteğe göre, *stack (yığın -LIFO), queue (kuyruk -FIFO), deque (çifte sonlanmış kuyruk)* yapıları elde edilebilir.

Java'da *LinkedList* sınıfı hem *tek-bağlı* hem *iki-bağlı* listeleri kılgılama yeteneğine sahiptir.

Aşağıdaki örnekler, LinkedList sınıfı ile yapılabilen işlerden bazılarını göstermektedir.

LİnkedLİst sınıfının iki tane kurucusu vardır:

**LinkedList**()
Boş bir bağlı-liste yaratır.

**LinkedList**(Collection c)
Parametrenin belirlediği koleksiyonu içeren bir bağlı-liste yaratır. Öğelerin sırası koleksiyonun *iteratör*'ünün (tekrarlayıcı) belirlediği sıradır.

### *LinkedList Sınıfının Metotları*

void **add**(int index, Object element)
Listede indisi belirtilen yere öğe sokuşturur (insert). O indisten sonraki öğelerin konumları birer geriye kayar.

boolean **add**(Object o)
Parametrede verilen nesneyi listenin sonuna ekler.

boolean **addAll**(Collection c)
Parametrede verilen koleksiyonun bütün öğelerini listenin sonuna ekler. Ekleme sırası koleksiyonun iteratörü'nün belirlediği sıradadır.

boolean **addAll**(int index, Collection c)
Belirtilen index'ten başlayarak verilen koleksiyonu listeye yerleştirir.

`void` **addFirst**`(Object o)`
Parametrede verilen nesneyi listenin başına ekler.

`void` **addLast**`(Object o)`
Parametrede verilen nesneyi listenin sonuna ekler.

`void` **clear**`()`
Listedeki bütün öğeleri siler; boş liste haline getirir.

`Object` **clone**`()`
Returns a shallow copy of this `LinkedList`.

`boolean` **contains**`(Object o)`
Parametrede belirtilen nesne listede varsa `true` değerini alır.

`Object` **get**`(int index)`
İndisi belirtilen öğeyi listeden seçer.

`Object` **getFirst**`()`
Listenin ilk öğesini verir.

`Object` **getLast**`()`
Listenin son öğesini verir.

`int` **indexOf**`(Object o)`
Parametrede verilen nesnenin listedeki indisini verir. Nesne listede yoksa -1 değerini verir.

`int` **lastIndexOf**`(Object o)`
Parametrede verilen nesnenin listedeki son indisini verir. Nesne listede yoksa -1 değerini verir.

`ListIterator` **listIterator**`(int index)`
Belirtilen indisten başlayarak öğelerin list-iteratörünü verir.

`Object` **remove**`(int index)`
İndisi verilen öğeyi listeden siler.

`boolean` **remove**`(Object o)`
Verilen nesneyi ilk karşılaştığı konumundan siler.

`Object` **removeFirst**`()`
Listenin ilk öğesini siler; değeri o öğedir.

`Object` **removeLast**`()`
Listenin son öğesini siler; metodun değeri silinen öğedir.

`Object` **set**`(int index, Object element)`
Verilen nesneyi istenen indisli öğenin yerine koyar.

`int` **size**`()`
Listedeki öğe sayısını verir.

`Object[]` **toArray**`()`

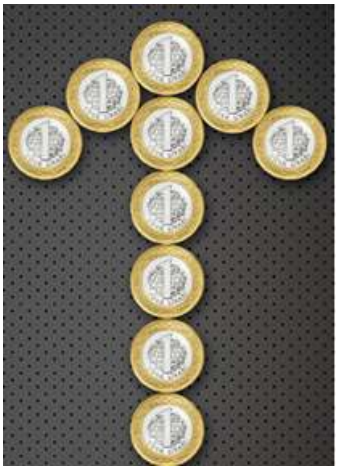
Listedeki öğeleri, aynı sırada bir arraye dönüştürür.

`Object[]` **toArray**`(Object[] a)`

Listedeki öğeleri, aynı sırada bir arraye dönüştürür. `Runtime` arrayi belirtilen arraydir.

`LinkedList` sınıfının metotları yardımıyla çok çeşitli listeler yapılabilir. Bu listeler arasında *yığıt (stack), kuyruk (queue), çembersel bağlı listeler, iki yönlü bağlı listeler* ve *çok bağlı listeler* vardır. `Yığıt` ve `kuyruk` tipi listeler, bilgisayar programcılığında özel öneme sahiptirler.Başka bir deyişle, `LinkedList` yapıları daha geneldir; `yığıt` ve `kuyruk` yapıları onun özel halleridir. Ancak, uygulamalarının çok olması nedeniyle, `yığıt` ve `kuyruk` listeleri için öğe ekleme ve öğe silme işlerini yapan özel metotlar vardır. O metotları ayrı altbölümlerde ele alacağız. Ancak, bu bölümde `add(), addFirst(), getFirst()` ve `getLast()` metotlarıyla `yığıt` ve `kuyruk` oluşturulabileceğini göreceğiz.

Örneklere başlamadan önce şu bilgiyi anımsamakta yarar vardır. Bir listede, yeni öğeler daima listenin sonuna ekleniyor ve silinen öğeler daima listenin sonundan siliniyorsa, bu liste bir `yığıt (stack)` olur. Bu tür listelere `FILO ( first-input-last-output, ilk giren son çıkar)` listesi denilir. Tabii, deyimi `LIFO (last-input-first-output , son giren ilk çıkar)` biçiminde de ifade edebiliriz. Bu liste, üst üste yığılmış nesneler gibidir. En üste konulan nesne ilk alınabilecek olandır.



Tersine olarak, bir listede, yeni öğeler daima listenin sonuna ekleniyor, ancak silinen öğeler daima listenin önünden siliniyorsa, bu liste bir kuyruk (queue) olur. Bu tür listelere `FIFO ( first-input-first-output, ilk giren ilk çıkar)` listesi denilir. Tabii, deyimi `LILO (last-input-last-output , son giren son çıkar)` biçiminde de ifade edebiliriz. Bu liste, bir gişe kuyruğuna benzer. Kuyruğa ilk giren gişedeki işini ilk bitirip ayrılan kişi olur.

Aşağıdaki program, listenin sonuna öğe ekleyerek özünde bir kuyruk yaratıyor. Ama bir `LinkedList` olduğu için, listenin önünden veya sonundan öğe silebiliyor.

*Örnek 1:*

Aşağıdaki program *list* adında bir LinkedList listesi yaratıyor. Liste *Portakal⇒Bergamot* sırasında bir kuyruktur (queue). İlk gelen kuyruğun önüne, son gelen kuyruğun sonuna konuşlanıyor.

```
package koleksiyon;

import java.util.*;

public class Koleksiyon {
	public static void main(String[] args) {
		LinkedList list = new LinkedList();
		list.add("Portakal");
		list.add("Limon");
		list.add("Mandalina");
		list.add("Turunç");
		list.add("Mandalina");
		list.add("Bergamot");
		System.out.println("list = " + list);

		list.add(3, "Greyfurt");
		System.out.println("list = " + list);
		System.out.println("ilk öğe = " + list.getFirst());
		System.out.println("son öğe = " + list.getLast());

		System.out.println("silinen = " + list.removeFirst());
		System.out.println("silinen = " + list.removeLast());
		System.out.println("list = " + list);
	}
}
/*
 Çıktı:
 list = [Portakal, Limon, Mandalina, Turunç, Mandalina, Bergamot]
 list = [Portakal, Limon, Mandalina, Greyfurt, Turunç, Mandalina, Bergamot]
 ilk öğe = Portakal
 son öğe = Bergamot
 silinen = Portakal
 silinen = Bergamot
 list = [Limon, Mandalina, Greyfurt, Turunç, Mandalina]
*/
```

Bu program *list* adlı bir *ArrayList* yaratıyor, *add()* metodu ile onun terimlerine Portakal, Limon, Mandalina, Turunç, Mandalina, Bergamot öğelerini atıyor ve o terimleri yazdırıyor.

*list.add(3, "Greyfurt")* metodu, listeye *"Greyfurt"* nesnesini sokuşturuyor ve bu nesneyi 3 indeksli öğe haline getiriyor. Listenin öteki öğeleri birer konum geriye kayıyor.

Bu adımda liste, yeni öğelerle birlikte tekrar yazılıyor.

*list.getFirst()* metodu, listenin ilk öğesi olan *Portakal* değerini veriyor.

*list.getLast()* metodu, listenin son öğesi olan *Bergamot* değerini veriyor.

*list.removeFirst()* metodu, listenin ilk öğesi olan *Portakal* nesnesini listeden siliyor. Silinen öğe metodun verdiği değerdir.

*list.removeLast()* metodu, listenin son öğesi olan *Bergamot* nesnesini listeden siliyor. Silinen öğe metodun verdiği değerdir.

Aşağıdaki program *liste* adında bir LinkedList listesi yaratıyor. Liste, özünde *Viyana⇒Londra* sırasında bir yığıt (stack) yapısıdır. İlk gelen kuyruğun önüne, son gelen kuyruğun sonuna konuşlanıyor. Önceki örneğin aksine, bu örnekte, ilk gelen listenin sonuna, son gelen listenin önüne konuşlanıyor. Ancak, LinkedList metotları bu listenin herhangi bir yerine öğe ekleme veya öğe silmeye olanak tanır.

*Örnek 2:*

```
package koleksiyon;

import java.util.*;

public class Koleksiyon {
	public static void main(String[] args) {
		LinkedList liste = new LinkedList();
		liste.addFirst("Londra");
		liste.addFirst("Moskova");
		liste.addFirst("Ankara");
		liste.addFirst("Paris");
		liste.addFirst("Viyana");
		System.out.println(liste);
		liste.removeLast();
		liste.removeLast();
		System.out.println(liste);

		System.out.println("liste = " + liste);

		liste.add(2, "Tahran");
		liste.addLast("Bağdat");
		System.out.println("liste = " + liste);
		System.out.println("ilk öğe = " + liste.getFirst());
		System.out.println("son öğe = " + liste.getLast());

		System.out.println("var mı? = " + liste.contains("Kahire"));
		liste.clear();
		System.out.println("liste = " + liste);
	}
}
	/*
	Çıktı:
	[Viyana, Paris, Ankara, Moskova, Londra]
	[Viyana, Paris, Ankara]
	liste = [Viyana, Paris, Ankara]
	liste = [Viyana, Paris, Tahran, Ankara, Bağdat]
	ilk öğe = Viyana
	son öğe = Bağdat
	var mı? = false
	liste = []
	*/
```

Bu program *liste* adlı bir *ArrayList* yaratıyor, *addFirst()* metodu ile onun terimlerine `Viyana, Paris, Ankara, Moskova, Londra` öğelerini atıyor ve o terimleri yazdırıyor.

*liste.removeLast()* metodu, listenin son öğesi olan *Londra* nesnesini listeden siliyor. Metot tekrar çağrılıyor, bu lez listenin son öğesi olan *Moskova* nesnesini listeden siliyor. Silinen öğe metodun verdiği değerdir.

Bu adımda, silinenler nedeniyle oluşan yeni tekrar yazdırılıyor.

`liste.add(2, "Tahran")` metodu, *Tahran* nesnesini indisi 2 olan konuma sokuşturuyor. Sonraki öğeler kuyruğun sonuna doğru birer konum kayıyor.

`liste.addLast("Bağdat")` metodu kuyruğun sonuna *Bağdat* nesnesini ekliyor.

`liste.getFirst()` metodu, listenin ilk öğesi olan *Viyana* değerini veriyor.

`liste.getLast()` metodu, listenin son öğesi olan *Bağdat* değerini veriyor.

`liste.contains("Kahire")` metodu false değeri ile Kahire nesnesinin listeta olmadığını söylüyor.

`liste.clear()` metodu, listenin tüm öğelerini silip boş bir liste haline getiriyor.

Aşağıdaki program `list` adında bir `LinkedList` listesi yaratıyor. Liste *"Orhan Kemal"⇒"Aziz Nesin"* sırasında bir kuyruk (queue) yapısıdır. Bu örnekte, ilk gelen kuyruğun önünde, son gelen kuyruğun soonunda konuşlanıyor.

*Örnek 3:*

```
import java.util.*;

public class LinkedList02 {

	public static void main(String[] args) {

		List list = new LinkedList();
		list.add("Orhan Kemal");
		list.add("Melih Cevdet Anday");
		list.add("Aziz Nesin");
		Iterator iter = list.iterator();
		for (int i = 0; i < 3; i++)
			System.out.println(iter.next());
	}
}
	/*
	Çıktı:
	Orhan Kemal
	Melih Cevdet Anday
	Aziz Nesin
	*/
```

Bu program `Iterator` yardımıyla Listeyi taramakta ve öğelerini yazdırmaktadır.

Aşağıdaki program `list` adında bir `LinkedList` listesi yaratıyor. Öğe eklemek için push() metodunu, öğe silmek için pop() metodunu tanımlıyor. For döngüsü ile listenin önüne *0, 1, 2, 3, 4, 5, 6, 7, 8, 9* öğelerini ekliyor. Liste *0⇒9* sırasında bir kuyruk (queue) yapısıdır.

*Örnek 4:*

```
import java.util.*;
```

```
// Making a stack from a LinkedList.

public class LinkedList01 {
      private LinkedList list = new LinkedList();

      public void push(Object v) {
            list.addFirst(v);
      }

      public Object top() {
            return list.getFirst();
      }

      public Object pop() {
            return list.removeFirst();
      }

      public static void main(String[] args) {
            LinkedList01 stack = new LinkedList01();
            for (int i = 0; i < 10; i++)
                  stack.push(new Integer(i));

            System.out.println(stack.top());
            System.out.println(stack.pop());
            System.out.println(stack.top());
      }
     }
         /*
         Çıktı:
         9
         9
         8
         */
```