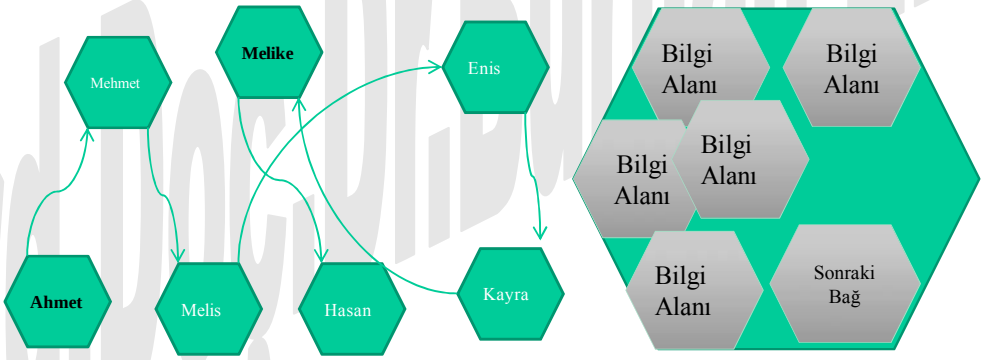
Tek Yönlü Bağlı Listeler, Yığıt ve Kuyruklar

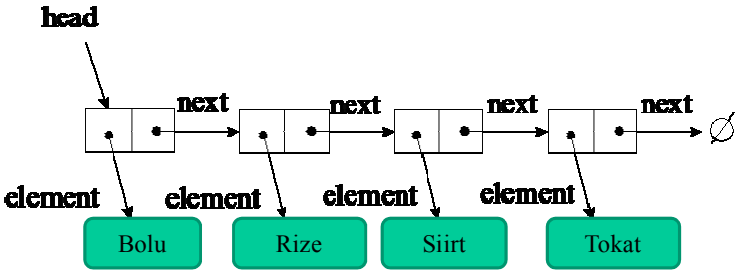
3.1 Bağlı Listeler

Bağlı listeler, bir elemanın kendinden sonra gelen verinin yerini göstermesi olarak tanımlanabilir. Dizi mantığı kullanılarak da her ne kadar bağlı liste oluşturmak mümkünse de genelde verinin hücre (düğüm) yapıları şeklinde tutulduğu liste oluşumu kullanılır. Çünkü düğüm yapısını kullanan bu mantıkta kaç elaman olduğunu bilmeniz ve sınırı önceden belirlemenizin gereği yoktur. Şekil 3.1'de bağlı liste mantığı şematik olarak verilmektedir. Liste başı 'Ahmet' verisi içeren düğüm ve liste sonu 'Hasan' verisi içeren düğümdür.



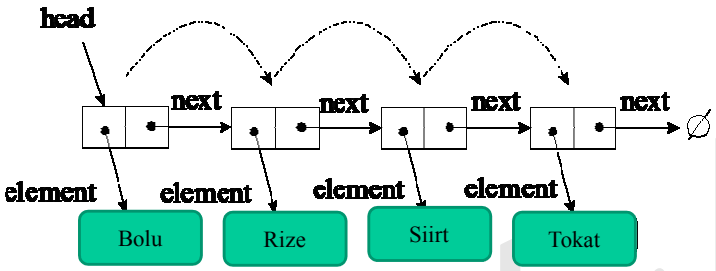
Şekil 3.1. Bağlı liste gösterimi; a) Bağlantılar, b) Bir düğüm yapısı.

Verilere doğru bir şekilde ulaşmak için ilk düğümden başlayarak düğümlerden zıplaya zıplaya ilerlemeniz gerekir. Her düğüm yapısı içerisinde mutlaka kendinden sonraki düğümü gösterebilecek bir bağ alanına ihtiyaç vardır. Düğümler basit bir tek veri barındırabilecekleri gibi çok sayıda ve karmaşık veriler de saklayabilirler. Şekil 3.1b'de böylesi bir yapı örnek olarak verilmiştir. Tanım olarak, bir bağlı liste, düğümlerin (node) bir düzen oluşturacak şekilde arda arda bağlanmış biçimidir. Düğüm (node)ise bilgi tutulacak alan ve bir sonraki düğümü gösteren referanstan (next) oluşur.



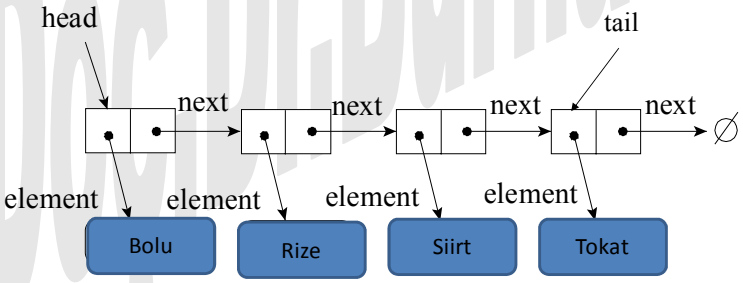
Şekil 3.2. Düğüm yapısı ile bir bağlı liste.

Şekil 3.2'de de görüleceği gibi bağ düğüm (node) içerisinde bir sonraki düğümü gösteren referanstır (next). Verilen bir düğüm (node) başlayarak, birinde bir diğerine geçilir. Burada liste başı 'head' referansı ile belirtilmiştir.



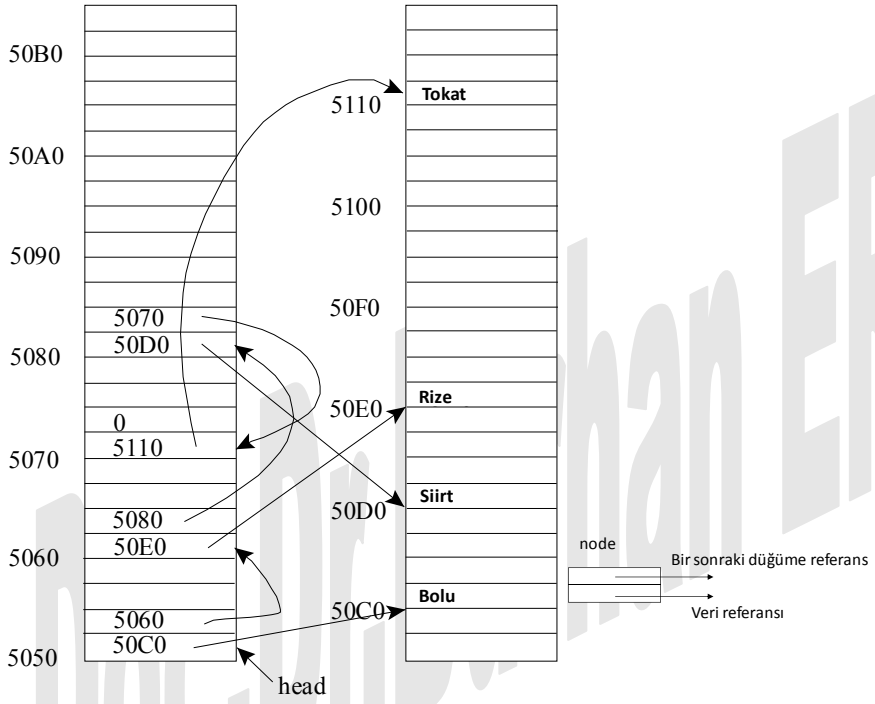
Şekil 3.3. Bağlı liste üzerinde ilerleme.

Bazı durumlarda listenin sonun veya sonundaki elemanın da bilinmesi gerekliği vardır. Kuyruk örneğinde olduğu gibi. Bu durumda en son düğümü gösteren yeni bir referansa ihtiyaç vardır. Şekil 3.4'de hem liste başı 'head' ve liste sonu 'tail' ile gösterilen bir bağlı liste yapısı verilmiştir.



Şekil 3.4. İki uçlu bağlı liste.

Bahsedilen bu bağlı liste yapısında bir bağ vardır ve sadece kendinden sonraki düğümü gösterir. Geriye yönelik bir bağ bulunmadığından sadece ileri yöne hareket söz konudur. Geriye doğru ilerlemek mümkün değildir. Bu nedenle bu tip bağlı listeleye tek yönlü bağlı liste denilir. Şekil 4'de gösterildiği gibi listedeki son düğüm 'tail' referenslı düğümdür ve bu düğümün 'next' referansı boş küme yani 'null'dır.
Bir bağlı listenin bilgisayar belleğinde yapısı ise şekil 3.5'teki gibi gösterilebilir.



Şekil 3.5. Düğümler ve içeriklerinin bellekteki yapısı.

Tek yönlü bağlı listeler ve diziler kabaca karşılaştırılırsa durum şöyledir;

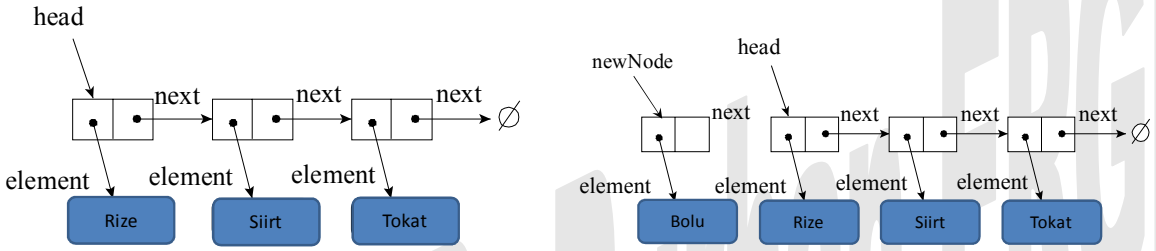
Tek Yön Bağlı Liste	Array
1. Elemanlar doğrusal düzende saklanır ve bağlar ile erişilebilir. 2. Sabit boyut yoktur. 3. Bir önceki elemana doğrudan erişim yapılamaz.	1. Elemanlar doğrusal düzende saklanır, indeks ile erişilebilir. 2. Sabit boyutludur. 3. Bir önceki elemana kolayca erişilebilir.

Bağlı liste oluşturmak için bir düğüm yapısı java programlama dilinde şöyle verilebilir. Burada Object sınıfından veri tutulması için referans tanımlanarak java'da her hangi bir veri yapısını gösterebilecek bir düğüm yapısı elde edilmiştir. Bilindiği üzere java'da object sınıfı en temel yapı olup bütün nesneler bu sınıftan türetilerek elde edilmiştir. Dolayısıyla Object sınıfından bir referans bütün nesneleri işaret edebilir.

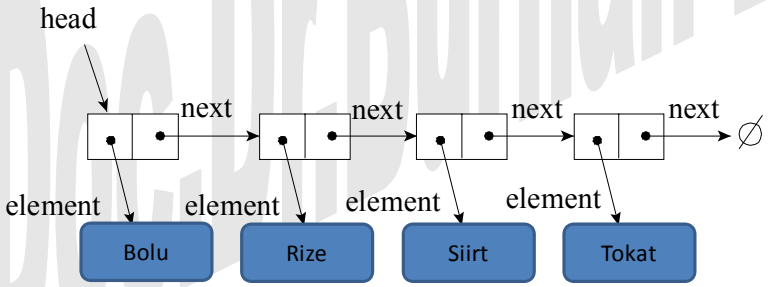
```
public class node {
        Object element;
        node next;
        public node() {
                this( null, null );
        }
        public node( Object e, Node n ) {
                element = e;
                next = n;
        }
```

}

Bağlı liste başına eleman eklenmesi durumu şekil 3.6'da verilmiştir. Şekil 3.7'de ekleme yapıldıktan sonra bağlı listenin durumu gösterilmiştir.



Şekil 3.6. Listeye yeni bir düğüm ekleme;a) Eklenmeden önceki durum,b)Ekleme sonrası.



Şekil 3.7. Yeni düğümün liste başına eklenmesi sonrası durum.

Yeni düğüm oluşturulması ve bilgi ilavesi yapılandırıcılar kullanılarak şöyle yapılabilir;

```
newNode x = new node();
newNode.element = new String("Bolu");
```

Ekleme açısından çok dikkatli olunmalıdır. Aksi halde bağlar kopabilir ve veriye erişim mümkün olmaz. Liste başına bağltıda şu kodlar sırasıyla kullanılmalıdır.

```
newNode.next = head;
head = newNode;
```

Listeden eleman çıkarılması ise tersi mantıkla şöyle kodlanabilir;

```
node oldNode;
oldNode=head;
head=head.next;
```

Eğer liste şekil 4'deki gibi hem liste başı ve hem liste sonu olan bir yapıda ise liste sonuna ekleme ve çıkarmalar yapılabilir. Ekleme kolayca yapılabilecek iken çıkarma biraz zordur. Çünkü bir önceki düğüm bilinememektedir. Bu nedenle 'tail'den önceki düğümü bulabilmek için liste başından hareketle bütün sonraki bağı 'tail' oluncaya kadar bütün düğümler gezilmelidir. Ayrıca listede tek bir düğüm olma ihtimali yani liste başı 'head' ve liste sonu 'tail'in aynı düğümü gösterme durumu da kontrol edilmelidir. Ekleme ve çıkarma kodları şöyle verilebilir.

Ekleme:

```
tail.next=newNode;
tail=newNode;
tail.nex=null;
```

Çıkarma:

```
node p=head;
node oldNode;
if (head==tail){
        oldNode=head;
        head=null;
        tail=null;
}else{
 while (p.next!==tail)
        p=p.next;
 oldNode=tail;
tail=p;
tail.next=null;
}
```

3.2 Bağlı Yığıt

Yığıt mantığında temel olan mantık eklenen sonra gelenin önce işlem görmesidir. Diğer bir deyişle son giren ilk çıkar mantığıdır. Bağlı liste yapısı üzerinde veri tutmak için ilk önce gelen verilerin bir bağlı düğüm yapısına büründürülmesi gerekir. Önce bir boş düğüm oluşturulur, gelen veri bu düğümün içerisine yerleştirilir. Listeye eklenecek şey bu oluşturulan düğümdür.

Bağlı liste mantığı ile yığıt mantığı beraber düşünüldüğünde bağlı listenin sonuna ekleme yapılması ve yine sonundan çıkarılması ile yığıt mantığı oluşturulması akla gelebilir. Yığıt mantığını gerçekleme açısından doğru olmakla beraber hız açısından oldukça yavaş kalır. Çünkü bağlı listelerde sadece listenin başı bilinmektedir. Bu nedenle ekleme veya çıkarmada yığıtın sonuna eklemek için bağlı liste baştan sona kadar bütün düğümlerin ziyaret edilmesi zorunlu olur. Asıl olanın, son gelenin ilk alınması olduğu hatırlanırsa liste başından itip yine liste başından çekmek daha akıllıca bir yaklaşım olacaktır. Gerek itme ve gerekse çekme aşamasında liste üzerinde her hangi bir gezintiye ihtiyaç duyulmaz. Liste başı zaten bilindiği için liste üzerinde ileri geri hareket gereksizdir. Aşağıda java programlami dili için bağlı yığıt gerçekleştirilişi verilmiştir. Burada her hangi bir türü tutabilecek Object sınıfı kullanılarak yığıt mantığı genelleştirilmiştir.

```
public node{                          //Genel tek yönlü düğüm yapısı
      object element;
      node next;
   }

public class LinkedStack {            //Bağlı yığıt sınıfı
     node top;
     int size;                        //itilen ve çekilenleri sayar
    public LinkedStack() {
        node top = null;
      size = 0;
   }
```

```
    public int size() {
       return size;
       }
    public boolean isEmpty() {
       if (top == null)
              return true;
       return false;
    }
    public void push(object elem) {          //Eleman itma
       node newNode = new Node();
       newNode.element=elem;
       newNode.next=top;
       top = newNode;
       size++;
       }
    public Object pop(){                     //Eleman çekme
       object temp=null;
       if (!isEmpty()){
              object temp = top.element;
                 top = top.next;
              size--;
       }
       return temp;                          //Eleman yoksa null döndürülür
     }
    public object top(){
       if (isEmpty())
              return null;
       return top.element;
     }
}
```

Sınıf tanımı yapılırken boyut öğrenilmesi için bütün bağlı yapı gezilmesini engelleyerek hız artırmak için yığıt yapısında fazladan bir boyut değişkeni kullanılmıştır. Eğer böyle yapılmasaydı boyut öğrenmek için bütün düğümleri gezerek sayan bir metod yazılması zorunluluğu doğardı. Aşağıda bağlı yığıttaki elemanları sayan method verilmiştir.

```
    public int size() {
       int size=0;
       node p=top;
       while (p!=null){
              size++;
              p=p.next;
              }
       return size;
     }
```

3.3 Bağlı Kuyruk

Kuyruk mantığı hatırlanacak olursa kuyruk mantığında ilk giren ilk çıkar mantığı amaçlanmaktaydı. İlk eklenen ve son eklenen bu mantık içerisinde bilinmeyi gerektirir. Bu nedenle, kuyruk mantığı bağlı liste üzerinde gerçekleştirilirken her iki uca da ihtiyaç duyulur. Ekleme liste sonuna yapılırken

çıkarılması liste başından yapılır. Bu mantık çerçevesinde bağlı liste kodlaması java programlama dili ile aşağıdaki gibi verilebilir.

Kuyruk boş iken yapılmak istenen ekleme ve çıkarmaları kodlarken dikkatli olunmalıdır. Kuyrukta başlangıçta hiçbir eleman bulunmadığından referanslar hiçbir yeri göstermemeli yani null özel değerinde olmalıdır. Bağlı kuyruk kullanımında uç şartlar oluştuğunda ekleme ve çıkarma farklılaşacaktır. Bağlı kuyruk için uç şart kuyruğun tamamen boş olması durumudur. O halde, "Kuyruk ne zaman tamamen boştur?" sorusunun cevabı aranmalıdır. Cevap; kuyruk ilk başlangıçta veya son eleman alındıktan sonra tamamen boştur. İlk başlangıçta front ve rear referansları null yapılarak bu durum işaretlenebilir. Çıkarma metodu yazılırken de her eleman çekiminden sonra son elemanın da kuyruktan alınıp alınmadığı kontrol edilmelidir. Eleman alımı front referansı ile yapıldığından bu referansın alımdan sonra null olunduğuna bakılmalıdır. Eğer front referansı null oluyorsa rear referansı da null yapılarak kuyruk üzerinde eleman kalmadığı belirtilmeli ve kuyruk en baştaki durumuna getirilmelidir.

Kuyruk boş ise çıkarma yapılamayacağı çok açıktır. Bu da kuyruğun başını veya sonunu tutan referanslardan her hangi biri ile kolayca kontrol edilebilir. front veya rear referanslarından biri null ise kuyruk boştur.

Benzer şekilde ekleme metodunun yazımında da kuyruğun boş olup olmadığına dikkat edilmelidir. Ekleme sondan rear referansı ile yapılmaktadır. Eğer kuyruk tamamen boşsa rear, null değerine sahiptir ve kuyrukta hiçbir düğüm yoktur. Dolayısıyla düğüm olmadığından ekleme bir düğümün sonrasına yapılmayacaktır. Kuyrukta tek bir eleman olacak front ve rear referansları bu düğümü gösterecektir.

```
public class LinkedQueue{                    //Genel tek yönlü kuyruk sınıfı
    private node front,rear;
    private int size;
    public LinkedQueue() {
        front = null;
        rear=null;
        size=0;
    }
    public int size() {
        return size;
    }

    public boolean isEmpty() {
        if (front == null)
            return true;
            return false;
    }
    public void enqueue(object obj) {
        node newNode = new node();
        node.element=obj;
        node.next=null;
        if (front == null){
                front = node;
            rear=node;
        else
```

```
                    rear.next=newNode;
                rear = newNode;
                    size++;
            }
    public object dequeue(){
        object obj=null;
        if (front != null){
            obj = front.element;
                front = front.next;
            size--;
            }
          if (front == null)
            rear = null;                        // Kuyruk boş!!!
          return obj;
    }
    public object front(){
        if (isEmpty())
            return null;
        return front.element;
    }
}
```

Eğer boyut değişkeni yoksa boyutu öğrenmek için aşağıdaki gibi düğüm sayan bir metoda ihtiyaç olacaktır.

```
    public int size() {
        int size=0;
        node p=front;
        while (p!=null){
            size++;
            p=p.next;
        }
        return size;
    }
```