

BIL207 VERİ YAPILARI

5. Hafta

Kuyruk (Queues) Veri Yapısı

Doç. Dr. Sercan YALÇIN

Kuyruk (Queue)



- ❑ Kuyruklar, eleman eklemelerinin **sondan** (rear) ve eleman çıkarmalarının **baştan** (front) yapıldığı doğrusal veri yapılarıdır.
- ❑ Bir eleman ekleneceği zaman kuyruğun sonuna eklenir.
- ❑ Bir eleman çıkarılacağı zaman kuyrukta bulunan ilk eleman çıkarılır.
- ❑ Bu nedenle kuyruklara **FIFO** (First In First Out-İlk giren ilk çıkar) veya **LIFO** (Last-in-Last-out-Son giren son çıkar) listeleri de denilmektedir.

Kuyruk (Queue)

- ❑ Gerçek yaşamda banklarda, duraklarda, otoyollarda kuyruklar oluşmaktadır. Kuyruğu ilk olarak girenler işlemlerini ilk olarak tamamlayıp kuyruktan çıkarlar.
- ❑ İşletim sistemleri bünyesinde öncelik kuyruğu, yazıcı kuyruğu gibi birçok uygulama alanı vardır.
- ❑ Kuyruk modeli, program tasarımında birçok yerde kullanılır. Örneğin, iki birimi arasında iletişim kanalı, ara bellek/tampon bellek oluşturmada bu modele başvurulur.

Kuyruk (queue)

- ❑ Ardışıl nesneleri saklarlar
- ❑ Ekleme ve silme FIFO'ya göre gerçekleşir.
- ❑ Ekleme işlemi kuyruk arkasına yapılırken, çıkarma işlemi ise kuyruk önünden yapılır.
- ❑ **Ana Kuyruk İşlemleri:**
- ❑ İki tane temel işlem yapılabilir ;
 - ❑ **enqueue (object)**, bir nesneyi kuyruğun en sonuna ekler (**insert**).
 - ❑ **object dequeue ()**,
Kuyruk başındaki nesneyi getirir ve kuyruktan çıkarır (**remove** veya **delete**).

Kuyruk Veri Yapısı Bileşenleri

? Yardımcı kuyruk işlemleri:

- ? **object front()** (getHead/getFront): kuyruk başındaki nesneyi kuyruktan çıkarmadan geri döndürür.
- ? **integer size()**: kuyrukta saklanan nesne sayısını geri döner
- ? **boolean isEmpty()**: Kuyrukta nesne olup olmadığını kontrol eder.

? İstisnai Durumlar (Exceptions)

- ? Boş bir kuyruktan çıkarma işlemi yapılmak istendiğinde veya ilk nesne geri döndürülmek istendiğinde **EmptyQueueException** oluşur.

[illegible]

```
enqueue(1);
```

```
enqueue(5);
```

```
dequeue();
```

```
dequeue();
```

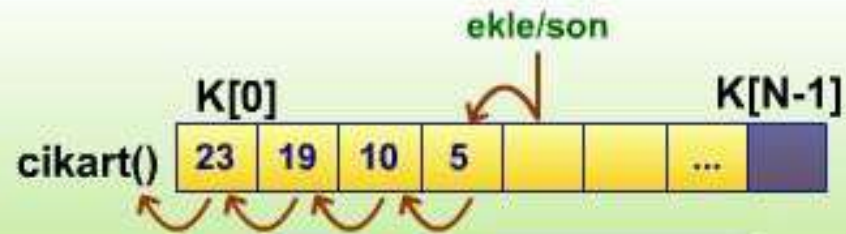
```
dequeue();
```

	5	1			
--	---	---	--	--	--

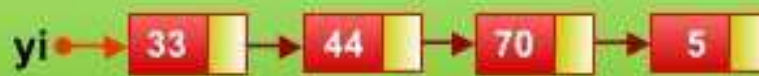
Kuyruk Tasarımı

- ❑ Kuyruk tasarımı çeşitli şekillerde gerçekleştirilebilir. Biri, belki de en yalın olanı, bir dizi ve bir indis değişkeni kullanılmasıdır.
- ❑ Dizi gözlerinde yığına atılan veriler tutulurken indis değişkeni kuyruğa eklenen son veriyi işaret eder.
- ❑ Kuyruktan alma/çıkartma işlemi her zaman için dizinin başı olan 0 indisli gözden yapılır.
- ❑ Kuyruk tasarımı için, genel olarak, üç değişik çözüm şekli vardır:
 - ❑ **Dizi Üzerinde Kaydırmalı (QueueAsArray) (Bir indis Değişkenli)**
 - ❑ **Dizi Üzerinde Çevrimsel (QueueAsCircularArray) (İki indis Değişkenli)**
 - ❑ **Bağlantılı Liste (QueueAsLinkedList) ile**

Kuyruk Tasarımı



a) Kaydırmalı kuyruk



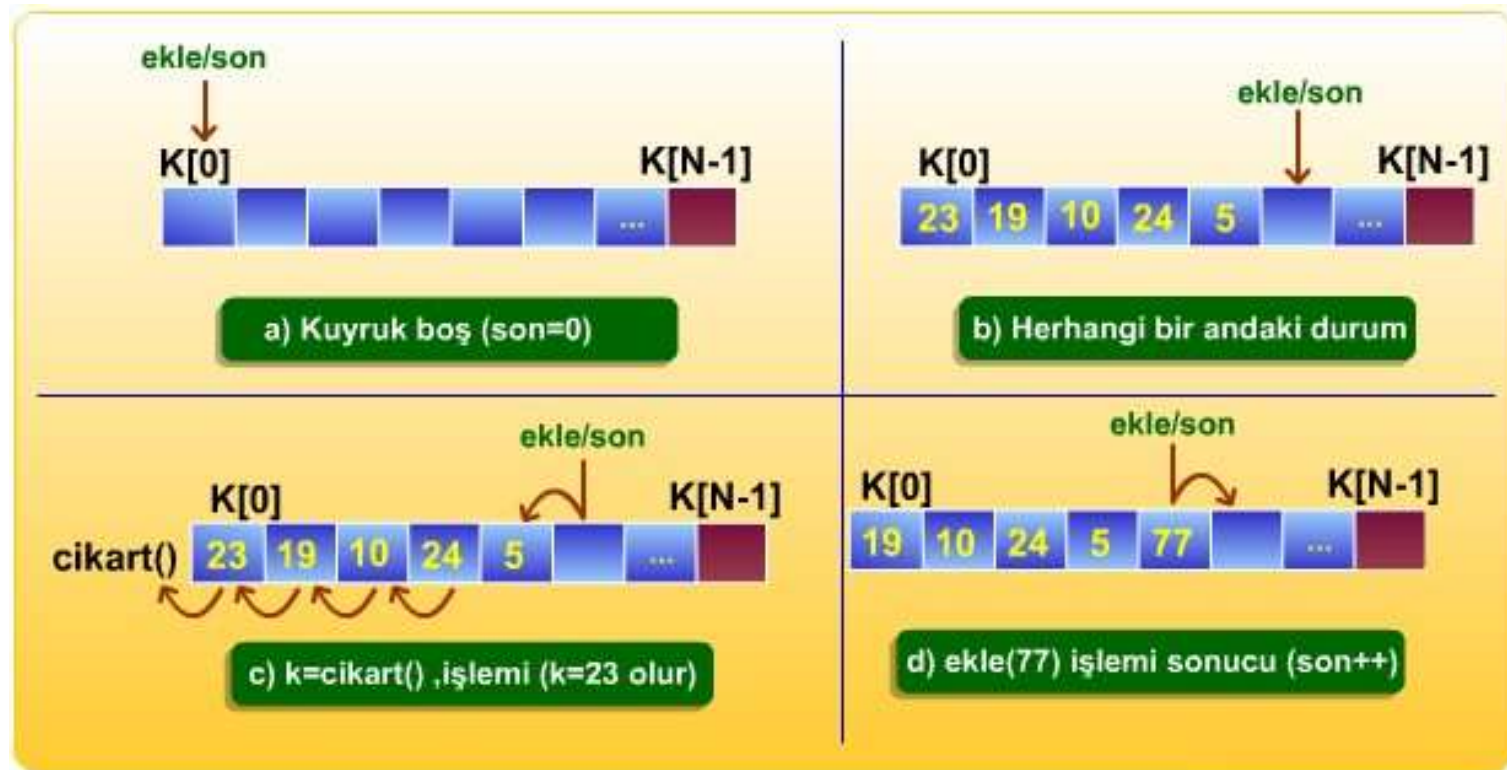
c) Bağlantılı listeli kuyruk



b) Çevrimsel kuyruk

Dizi Üzerinde Kaydırmalı Kuyruk

- ❓ N uzunluktaki bir dizi üzerinde kaydırmalı kuyruk yapısının davranışı şekilde gösterilmiştir;



Dizi Üzerinde Kaydırmalı Kuyruk

- ❑ **a)**'da kuyruğun boş hali,
- ❑ **b)**'de ise herhangi bir andaki tipik hali görülmektedir. Kuyruktan çıkartma işlemi,
- ❑ **c)**'de görüldüğü gibi dizinin ilk gözünden, yani $K[0]$ elemanı üzerinden yapılır; eğer kuyrukta birden fazla veri varsa, geride olanlarda bir öne kaydırılır.
- ❑ Kuyruğa ekleme işlemi çıkartma işleminden daha az maliyetle yapılır; eklenecek veri doğrudan ekle/son adlı indis değişkenin gösterdiği bir sonraki göze yapılır.

Dizi Üzerinde Kaydırmalı Kuyruk Kaba Kodu

Dizi üzerinde kaydırmalı kuyruk-ekleme işlemi kaba-kodu

```
if(Kuyrukta yer yoksa)  
    Kuyruk dolu mesajını yaz ve EKES değerini gönder;  
else  
    son'u bir sonraki göz için artır;  
    Veriyi K kuyruğunda son ile gösterilen göze ekle;  
}
```

Dizi üzerinde kaydırmalı kuyruk-çıkartma işlemi kaba-kodu

```
if(Kuyrukta veri yoksa)  
    Kuyruk boş mesajını yaz ve EKES değerini gönder;  
else  
    K kuyruğundaki K[0] verisi al;  
    Kuyruktaki verileri öne kaydır, K[0]'i K[1], K[1]'i K[2]...K[son-2]'i K[son-1].  
    son'u bir önceki göz için azalt;  
    Veriyi gönder;  
}
```

Kuyruk Tasarımı ve Kullanımı -Java

```
? class Kuyruk
? { private int boyut; private int[ ] kuyrukDizi;
? private int bas; private int son; private int elemanSayisi;
?
? public Kuyruk(int s)
? { boyut = s; kuyrukDizi = new int[boyut];
? bas = 0; son = -1; elemanSayisi = 0;
? }
? public void ekle(int j) // Kuyrugun sonuna eleman ekler
? {
? if (son==boyut-1) son = -1;
? kuyrukDizi[++son] = j; elemanSayisi++;
? }
```

Kuyruk Tasarımı ve Kullanımı -Java

```
? public int cikar()
? {
?     int temp = kuyrukDizi[bas++];
?     if(bas==boyut) bas=0;
?     elemanSayisi--;    return temp;
? }
? public boolean bosMu() {    return(elemanSayisi==0); }
? }

? public class KuyrukTest {
?     public static void main(String args[])
?     {    Kuyruk k = new Kuyruk(25);    k.ekle(1);    k.ekle(2);
?         System.out.println(k.cikar());    k.ekle(3);
?         for(int i=4; i<10; ++i)        k.ekle(i);
?         while(!k.bosMu())    System.out.println(k.cikar());
?     }
? }
```

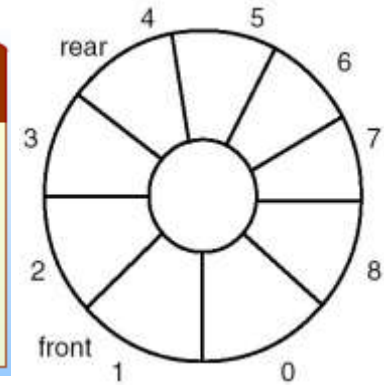
Dizi Üzerinde Çevrimsel Kuyruk

- ❑ Bu çözüm şeklinde dizi elemanlarına erişim doğrusal değil de çevrimsel yapılır; yani dizinin son elemanına ulaşıldığında bir sonraki göz dizinin ilk elemanı olacak şekilde indis değeri hesaplanır.
- ❑ Böylece kuyruk için tanımlanan dizi üzerinde sanki bir halka bellek varmış gibi dolaşılır.
- ❑ Kuyruktaki sıradaki eleman alındığında bu işaretçinin değeri bir sonraki gözü gösterecek biçimde arttırılır.
- ❑ Arttırma işleminin de, dizinin sonuna gelindiğinde başına gidecek biçimde çevrimsel olması sağlanır.
- ❑ Aşağıda dizi üzerinde çevrimsel kuyruk modeli için ekleme ve çıkartma işlemleri kaba-kodları verilmiştir:

Dizi Üzerinde Çevrimsel Kuyruk

Dizi üzerinde çevrimsel kuyruk - ekleme işlemi kaba-kodu

```
if(Kuyrukta yer yoksa)
    Kuyruk dolu mesajını yaz ve EKES değerini gönder;
else
    son'u çevrimsel şekilde bir sonraki göz için ayarla;
    Veriyi K kuyruğuna ekle;
}
```



- Yukarıda görüldüğü gibi ilk önce ekleme yapılması için kuyrukta yer olup olmadığına bakılmakta, yer varsa son adlı ekleme sayacı çevrimsel erişime imkan verecek şekilde arttırılmakta ve işaret ettiği yere veri koyulmaktadır.

Dizi Üzerinde Çevrimsel Kuyruk

- ❓ Aşağıda ise alma işlemi kaba kodu verilmiştir. Görüleceği gibi ilk önce kuyrukta veri olup olmadığı sınanmakta ve veri varsa ilk adlı işaretçinin gösterdiği gözdeki veri alınıyor, ilk adlı işaretçi çevrimsel olarak sıradaki veriyi gösterecek şekilde ayarlanıyor ve veri gönderiliyor.

Dizi üzerinde çevrimsel kuyruk - çıkartma işlemi kaba-kodu

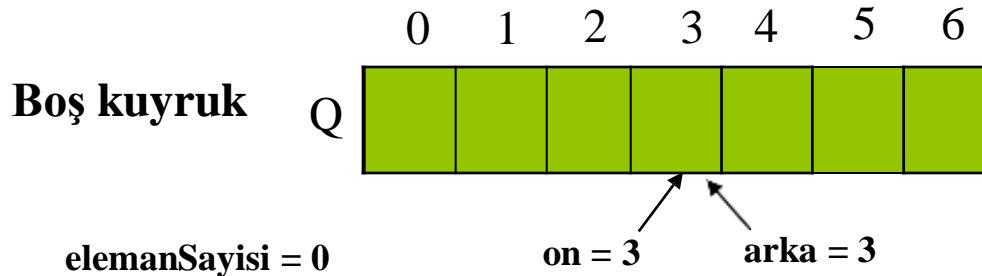
```
if(Kuyrukta yer yoksa)
    Kuyruk boş mesajını yaz ve EKES değerini gönder;
else
    K kuyruğundan ilk indisli yerdeki veriyi al;
    ilk'i çevrimsel şekilde bir sonraki göz için ayarla;
    Kuyruktan alınan veriyi gönder;
}
```


Dizi Kullanılarak Gerçekleştirim

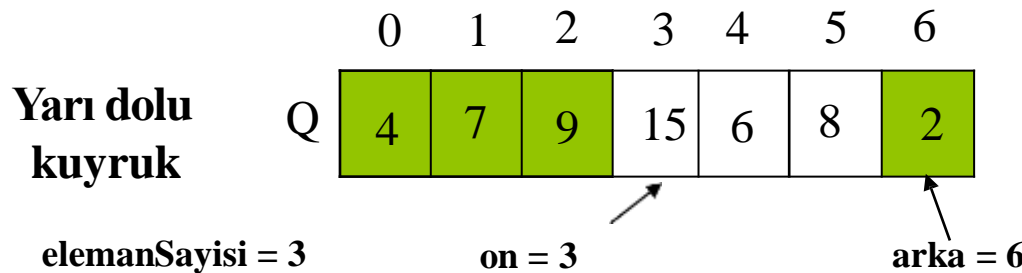
- ❑ **N** boyutlu bir dizi kullanılır.
- ❑ **ön** kuyruğun ilk elemanını tutar. Dizide ilk elemanın kaçınıcı indisten başlayacağını belirtir.
- ❑ **arka** kuyrukta son elemandan sonraki ilk boşluğu tutar.
- ❑ **elemanSayisi** kuyruktaki eleman sayısını tutar.
- ❑ **Boş kuyruk** eleman sayısının sıfır olduğu durumdur.
- ❑ **Dolu kuyruk** eleman sayısının N'ye eşit olduğu durumdur.

Dizi Kullanarak Gerçekleştirim

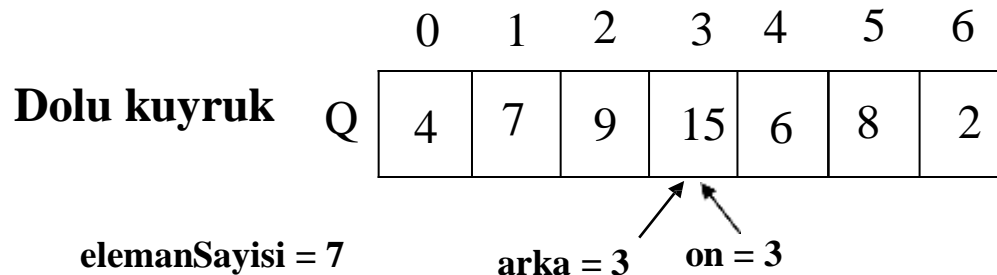
❓ Kuyruğu N boyutlu bir dizi (`int K[N]`) ve 3 değişken (`int on`, `int arka`, `int elemanSayisi`) ile gerçekleştirebiliriz.



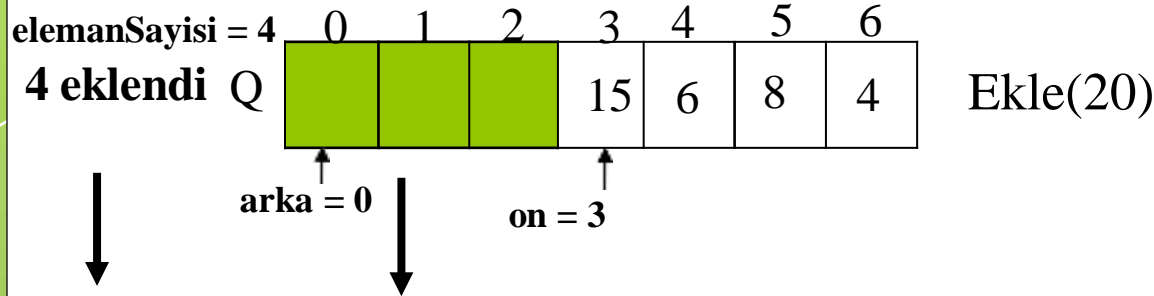
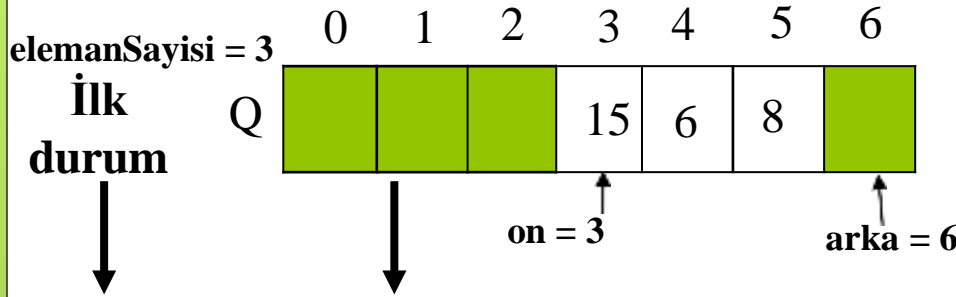
- on ve arka birbirlerine eşit ve elemanSayisi = 0 → Boş kuyruk



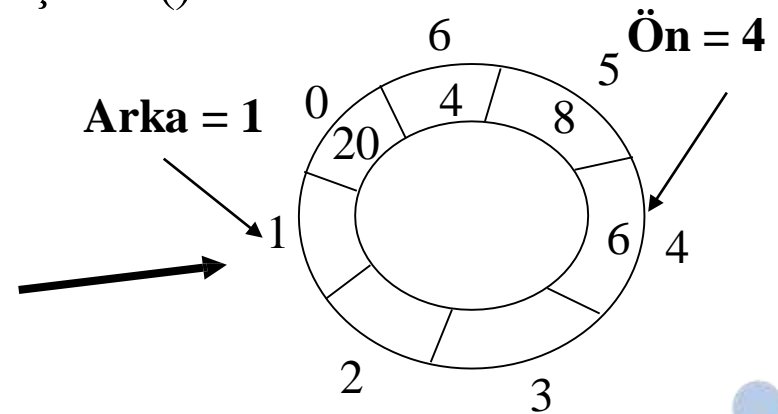
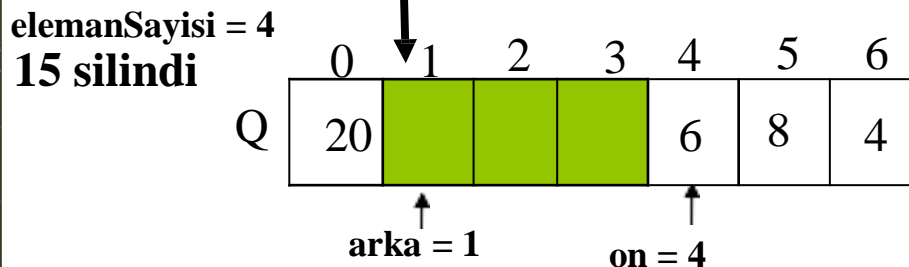
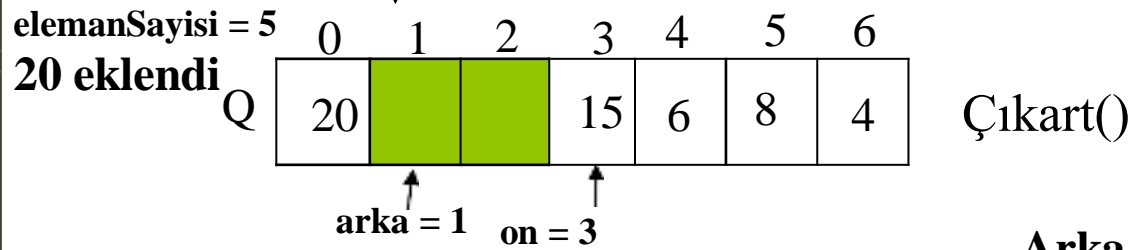
- on kuyruktaki ilk elemanı tutar
- arka son elemandan sonraki ilk boş yeri tutar.



- on ve arka birbirlerine eşit ve elemanSayisi=7 → Dolu kuyruk.



Kuyruğun kavramsal görünümü:
Döngüsel dizi



Dizi Üzerinde Çevrimsel Kuyruk- Java

```
public class kuyruk {  
  
    private int K[N];        // kuyruk elemanlarını tutan dizi  
    private int on;          // kuyruğun başı  
    private int arka;        // kuyruğun sonu  
    private int elemanSayisi; // kuyruktaki eleman sayısı  
  
    public kuyruk();  
  
    public boolean bosmu();  
    public boolean dolumu();  
    public int ekle(int item);  
    public int cikart();  
};
```

Dizi Üzerinde Çevrimsel Kuyruk- Java

```
// yapıcı yordam
public Kuyruk(){
    on = arka = elemanSayisi = 0;
}

// Kuyruk boşsa true döndür
public boolean bosmu(){
    return elemanSayisi == 0;
}

// Kuyruk doluysa true döndür
public boolean dolumu(){
    return elemanSayisi == N;
}
```

Dizi Üzerinde Çevrimsel Kuyruk- Java

```
// Kuyruğa yeni bir eleman ekle
// Başarılı olursa 0 başarısız olursa -1 döndür
public int ekle(int yeni){
    if (dolumu()){
        System.out.println("Kuyruk dolu.");
        return -1;
    }

    K[arka] = yeni;    // Yeni elemanı sona koy
    arka++; if (arka == N) arka = 0;
    elemanSayisi++;

    return 0;
}
```

Dizi Üzerinde Çevrimsel Kuyruk- Java

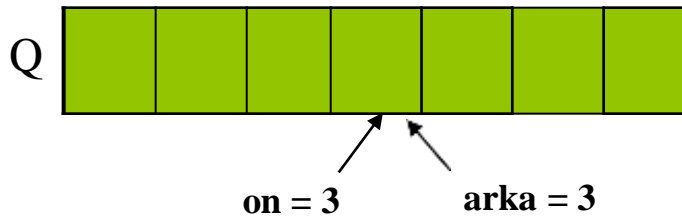
```
// Kuyruğun önündeki elemanı çıkart ve döndür.  
// Kuyruk boşsa -1 döndür  
public int cikart(){  
    int id = -1;  
  
    if (bosmu()){  
        System.out.println("Kuyruk boş");  
        return -1;  
    }  
  
    id = on;    // ilk elemanın olduğu yer  
    on++; if (on == N) on = 0;  
    elemanSayisi--;  
  
    return K[id];    // elemanı döndür  
}
```

```
public static void main(String[] args) {  
    Kuyruk k;  
  
    if (k.bosmu())  
        System.out.println("Kuyruk boş");  
  
    k.ekle(49);  
    k.ekle(23);  
  
    System.out.println("Kuyruğun önü: "+ k.cikart());  
    k.ekle(44);  
    k.ekle(22);  
  
    System.out.println("Kuyruğun ilk elemanı: "+ k.cikart());  
    System.out.println("Kuyruğun ilk elemanı: "+ k.cikart());  
    System.out.println("Kuyruğun ilk elemanı: "+ k.cikart());  
    System.out.println("Kuyruğun ilk elemanı: "+ k.cikart());  
  
    if (k.bosmu())  
        System.out.println("Kuyruk boş");  
}
```


Dizi Gerçekleştirimi: Son Söz

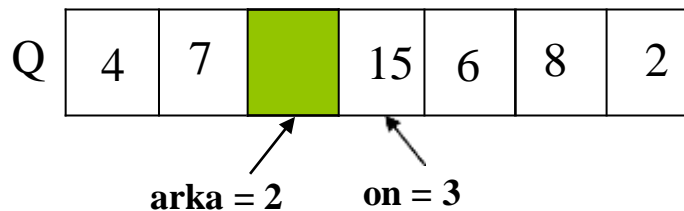
- ❑ Kuyruğu N-1 elemanlı bir dizi (`int K[N]`) ve 2 tane değişken (`int on`, `int arka`) ile gerçekleştirebiliriz.
- ❑ Aşağıdakileri nasıl tanımlayabileceğimizi düşünün.
 - ❑ Boş kuyruk
 - ❑ Dolu kuyruk

Boş kuyruk



- On ve arka birbirine eşit ise boş kuyruk

Dolu kuyruk



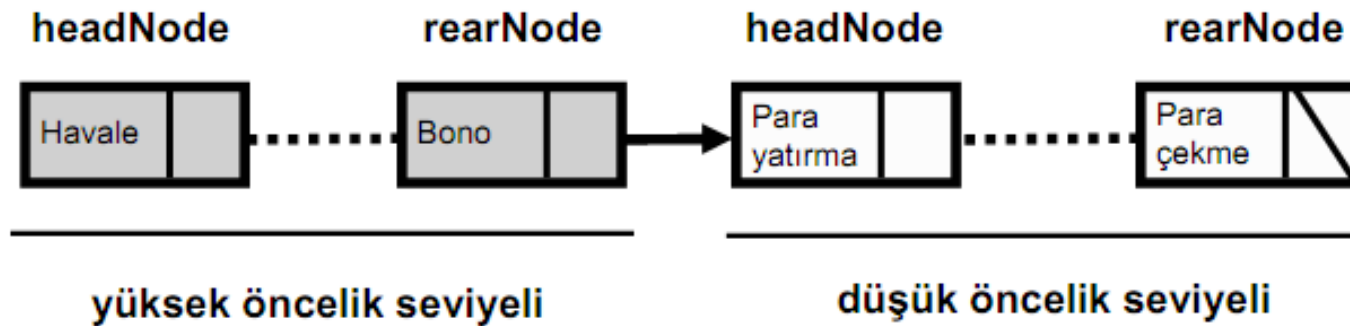
- On ve arka arasında 1 tane boşluk varsa dolu kuyruk

Öncelikli Kuyruklar (Priority Queues)

- ❑ Öncelikli kuyruk uygulamasında kuyruğa girecek verilerin veya nesnelerin birer öncelik değeri vardır ve ekleme bu öncelik değerine bakılarak yapılır.
- ❑ Eğer eklenecek verinin önceliği en küçük ise, yani en önceliksiz ise, doğrudan kuyruğun sonuna eklenir; diğer durumlarda kuyruk üzerinde arama yapılarak kendi önceliği içerisinde sona eklenir.
- ❑ Örneğin bazı çok prosesli işletim sistemlerinde bir proses işleme kuyruğu vardır ve yürütölme için hazır olan proseslerin listesi bir kuyrukta tutulur. Eğer proseslerin birbirlerine göre herhangi bir önceliği yoksa, prosesler kuyruğa ekleniş sırasına göre işlemci üzerinde yürütölürler; ilk gelen proses ilk yürütölür. Ancak, proseslerin birbirlerine göre bir önceliği varsa, yani aynı anda beklemekte olan proseslerden bazıları daha sonra kuyruğa eklenmiş olsalar dahi diğerlerine göre ivedi olarak yürütölmesi gerekebilir.
- ❑ Öncelikli kuyruk oluşturmak için bağlantılı listeyle, kümeleme ağacına dayalı ve araya sokma sıralaması yöntemiyle sağlanan çözümler olmaktadır.

Öncelikli Kuyruklar (Priority Queues)

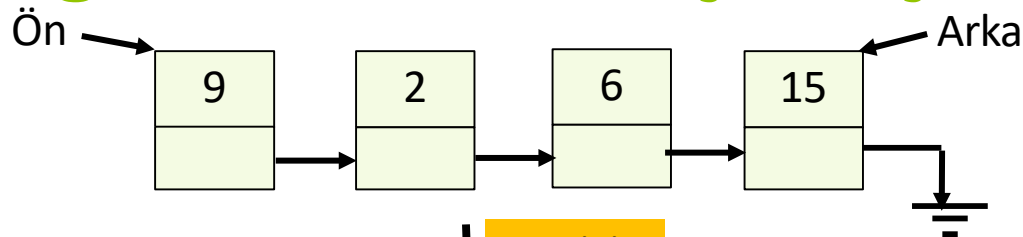
- ❑ Kuyruktaki işler kendi arasında önceliklerine göre seviyelendirilebilir.
- ❑ Her öncelik seviyesinin headNode ve rearNode'u vardır.
- ❑ Elaman alınırken en yüksek seviyeye ait elemanların ilk geleni öncelikli alınır.
- ❑ Yüksek öncelik seviyeli grubun son elemanı düşük öncelik seviyeli grubun ilk elemanından daha önceliklidir.



Bağlantılı Liste ile Kuyruk Tasarımı

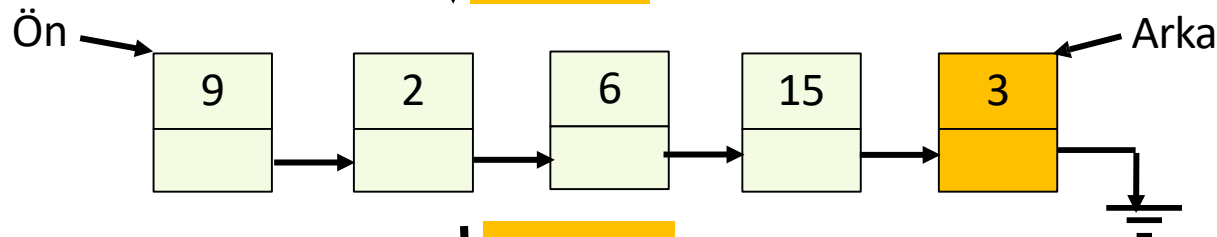
- ❑ ekleme(add) işlemi, son adresindeki verinin arkasına eklenir.
- ❑ çıkarma (get) işlemi, ilk verinin alınması ve kuyruktan çıkarılması ile yapılır. bellekte yer olduğu sürece kuyruk uzayabilir.

Bağlantılı Liste Gerçekleştirimi



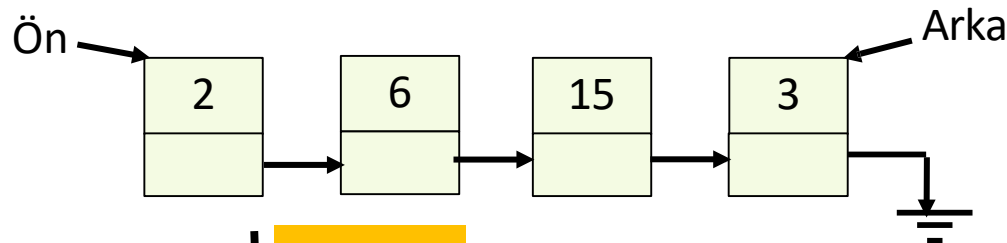
**Başlangıç
durumu**

ekle(3)



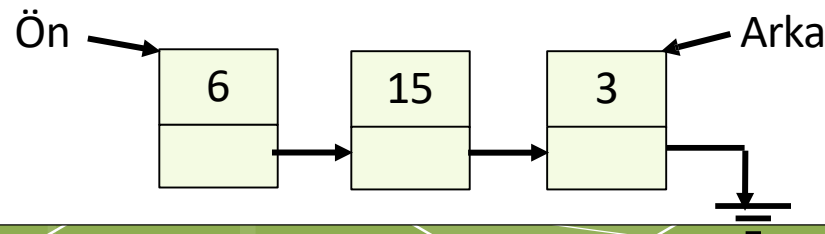
**3 eklendikten
sonra**

cikart()



**9 çıkartıldıktan
sonra**

cikart()



**2 çıkartıldıktan
sonra**

```
public class KuyrukDugumu {  
    public int eleman;  
    public KuyrukDugumu sonraki;  
  
    public KuyrukDugumu(int e){  
        eleman = e; sonraki = null;  
    }  
}  
  
public class Kuyruk{  
    private KuyrukDugumu on; // Kuyruğun önü  
    private KuyrukDugumu arka; // Kuyruğun arkası  
  
    public Kuyruk() {  
        on = arka = null;  
    }  
    public boolean bosmu() { ... }  
    public void ekle(int eleman) { ... }  
    public int cikart() { ... }  
}
```

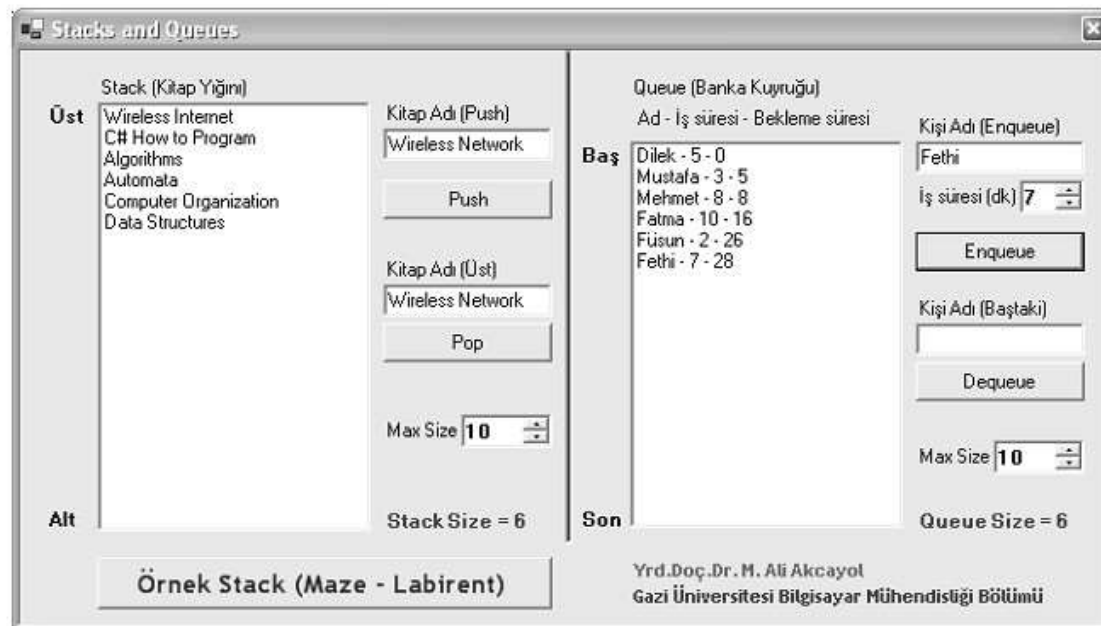
Haftalık Ödev

- 1- Banka kuyruğu örneğini ele alıp Banka işlemlerini kendi içerisinde üç öncelik grubuna ayırınız. Her yeni gelen kişiyi iş seçimine göre otomatik olarak ait olduğu grubun en sonuna ekleyiniz (enqueue). Her eleman alımında (dequeue) ise kuyruktaki en yüksek seviyeli grubun ilk elemanını alınız.

Haftalık Ödev

- 2- Aşağıda verilen uygulamayı C# veya Javada yapınız

Uygulama programı (Stacks ve Queues):



Stack (Kitap Yığını)

Üst

Wireless Internet
C# How to Program
Algorithms
Automata
Computer Organization
Data Structures

Kitap Adı (Push)
Wireless Network

Push

Kitap Adı (Üst)
Wireless Network

Pop

Max Size 10

Alt

Stack Size = 6

Queue (Banka Kuyruğu)

Baş

Dilek - 5 - 0
Mustafa - 3 - 5
Mehmet - 8 - 8
Fatma - 10 - 16
Fusun - 2 - 26
Fethi - 7 - 28

Ad - İş süresi - Bekleme süresi

Kişi Adı (Enqueue)
Fethi

İş süresi (dk) 7

Enqueue

Kişi Adı (Baştaki)

Dequeue

Max Size 10

Son

Queue Size = 6

Örnek Stack (Maze - Labirent)

Yrd.Doç.Dr. M. Ali Akcayol
Gazi Üniversitesi Bilgisayar Mühendisliği Bölümü