

JAVA

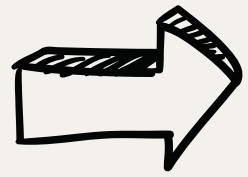
NECLA OZDEMİR

JAVA

ARRAY NEDİR?

- Aynı türdeki birden fazla variable depolamak için kullanılan bir **Object**'tir.
- Array oluşturmak için :
 - 1) Data tipi
 - 2) Data sayısı söylenmelidir
- Boyutları sabit olup baştan belirtildiği için hızlıdır
- Java'da arrayler dinamik değildir, yani oluşturulduktan sonra boyutları değiştirilemez.
- Diğer collectionlardan daha az memory kullanırlar (ekstra işlevselliği yoktur)
- Hem primitive veri tiplerini hem de object referanslarını(String, Integer , Boolean ...referanslarını)
- Stack memory'de depolanır

Array nasıl tanımlanır ve initialize edilir?

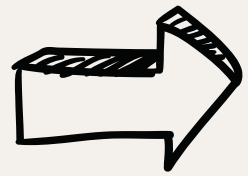


int türünde bir array tanımlama:

- `int[] notlar = new int[3];` (3 elemanlı bir array)

Array'i initialize etme :

- `notlar[0] = 85;`
- `notlar[1] = 90;`
- `notlar[2] = 75;`



Tek hamlede tanımla ve initialize etme :

- `int[] yaslar = { 18, 22, 30, 25, 27 };` (5 elemanlı bir array)

Arrays nedir?

- Arrays javada Array'lerle çalışmak için kullanılan yardımcı bir **class**'tır.
- Bu classta bulunan `toString()` methodu array'i yazdırmak için kullanılır.

Array için default değerler nedir?



Eleman eklemekten arrayi yazdırmak istersek default değerlerini görürüz.

- int[] notlar = new int[3];

System.out.println(Arrays.toString(notlar)); ==> [0, 0, 0]

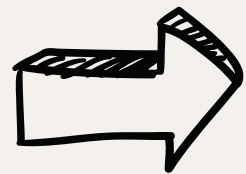
- String[] isimler = new String[3];

System.out.println(Arrays.toString(isimler)); ==> [null, null, null]

- char[] basHarf = new char[3];

System.out.println(Arrays.toString(basHarf)); ==> [, ,]

- Sayısal değerler için default '0' , String için 'null' , char için ise hiçliktir!!



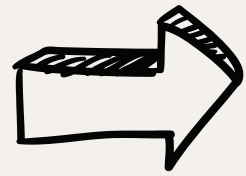
int[] notlar= { 85, 90, 75 };

System.out.println(Arrays.toString(notlar)); ==> [85, 90, 75]

System.out.println(notlar[1]); ==> 90

System.out.println(notlar); ==> [I@5b480cf9 dizinin bellekteki referansını temsil eden bir dize yazdırır.

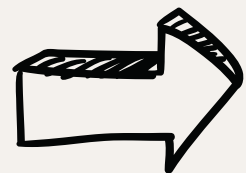
İki array'in eşit olup olmadığını nasıl kontrol edersiniz?



```
int[] array1 = {1, 2, 3, 4, 5};  
int[] array2 = {1, 2, 3, 4, 5};  
***Arrays.equals() metodunu kullanarak  
boolean isEqual = Arrays.equals(array1, array2);  
System.out.println(isEqual); ==> true
```

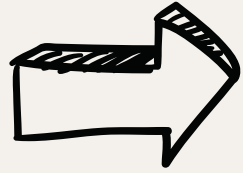


Arraylerin eşit olması için ;
aynı index numarasında, aynı
elaman olması gerekir



```
int[] array1 = {1, 2, 3, 4, 5};  
int[] array3 = {5, 4, 3, 2, 1};  
****Arrays.equals() metodunu kullanarak  
boolean isEqual = Arrays.equals(array1, array3);  
System.out.println(isEqual); ==> false
```

2 adet int tipinde dizi bulunmaktadır. Biri 50 element içeriyor, diğeri 30 element içeriyor. 50 elemanlı diziyi 30 elemanlı bir diziye atayabilir miyiz?



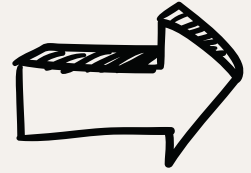
- Hayır, doğrudan atayamazsınız çünkü iki dizinin boyutları farklıdır. Java'da dizilerin boyutları sabittir ve bir diziyi başka bir diziye atarken boyutları aynı olmalı veya hedef array daha fazla eleman içeriyor olmalı.

EXAMPLE

- `int[] array3 = {6 ,7 ,8 };` `int[] array5 ={1 ,2 ,3 ,4 ,5 };`
- array5'in ilk 3 elemanı array3'a kopyala :
`System.arraycopy (array5 , 0 , array3 , 0 , 3);` ==> 1,2,3
- array5'in son iki elemanını array3'ün son iki elemanına kopyala :
`System.arraycopy (array5 , 3 , array3 , 1 , 2);` ==>6,4,5
- array3'u yazdır :

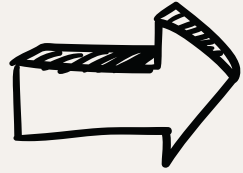
```
for (int w : array3) {  
    System.out.print(w + " ");  
}
```

EXAMPLE 10 kişilik bir sınıfı ilk 5 kişiyi bir grup ,son 5 kişiyi bir grup olacak şekilde gruplara ayır.



- `String[] sınıf10 = {"Ali", "Ayşe", "Mehmet", "Fatma", "Ahmet", "Elif", "Murat", "Zeynep", "Hasan", "Canan"};`
- `String[] grupIlk5 = {null, null, null, null, null};` (5 elemanlı dizi)
- `String[] grupSon5 = {null, null, null, null, null};` (5 elemanlı dizi)
- `System.arraycopy(sınıf10, 0, grupIlk5 , 0, 5);`
- `System.out.println(Arrays.toString(grupIlk5));` ==> [Ali, Ayşe, Mehmet, Fatma, Ahmet]
- `System.arraycopy(sınıf10, 5, grupSon5, 0, 5);`
- `System.out.println(Arrays.toString(grupSon5));` ==>[Elif, Murat, Zeynep, Hasan, Canan]

Bana Java'daki bir dizinin sınıf adını söyleyebilir misiniz?



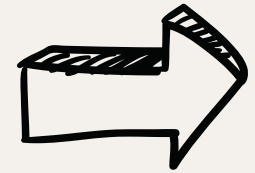
Dizi bir nesnedir. Sınıf adını öğrenmek için nesne üzerinde getClass(). getName() yöntemini kullanabiliriz.

EXAMPLE

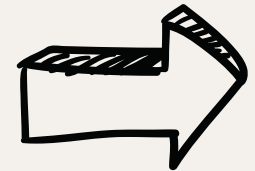
- `int[]` intArray = new int [5];
- `char[]` charArray = new char [5];
- `String[]` stringArray = new String [5];
- System.out.println("intArray sınıf adı: " + intArray.getClass().getName());
intArray sınıf adı ==> `[I`
- System.out.println("charArray sınıf adı: " + charArray.getClass().getName());
charArray sınıf adı ==> `[C`
- System.out.println("stringArray sınıf adı: " + stringArray.getClass().getName());
stringArray sınıf adı ==> `[Ljava.lang.String;`

EXAMPLE

Bir dizideki belirli bir öğeyi nasıl ararız?



- `Arrays.binarySearch()` yöntemini kullanabiliriz.
- `int[] arr = {2, 1, 7, 6};`
- `Arrays.sort(arr);`
- `System.out.println(Arrays.toString(arr));` ==> [1, 2, 6, 7]
- `System.out.println(Arrays.binarySearch(arr, 2));` ==> 1 (index)
- `System.out.println(Arrays.binarySearch(arr, 7));` ==> 3 (index)
- `System.out.println(Arrays.binarySearch(arr, 3));` ==> -3 - olmadığı anlamına gelir
[1, 2, 3, 6, 7] 3 de olsaydı bulunacağı sıra



- `String[] str = {"A", "C", "B", "D"};`
- `Arrays.sort(str);` ==> [A, B, C, D]
- `System.out.println(Arrays.binarySearch(str, "C"));` ==> 2 (index)
- `System.out.println(Arrays.binarySearch(str, "G"));` ==> -5 (sıra)

EXAMPLE 1'den 7'ye kadar olan tamsayı dizisinde eksik olan eleman nasıl bulunur?

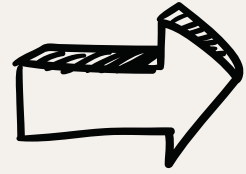
- `int ar [] = new int [] {1 ,2 ,3 ,5 ,6 ,7 };`
- `int n = ar.length+1;`
- `int total = n * (n + 1) / 2 ;` $\Rightarrow 28$ (1'den 7 ye kadar rakamların toplamı)
- `for (int i =0 ; i < ar.length ; i ++) {
 total -= ar [i] ; }
 System.out.println (total);
}`

EXAMPLE

[0, 2, 3, 0, 12, 0] arrayinde tum sifirlari sona yerlestiriniz.

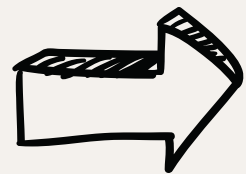
- `int[] arr = {0, 2, 3, 0, 12, 0};` `int[] yeniArr = new int [arr.length];`
 `System.out.println(Arrays.toString(arr));` \Rightarrow `[0, 2, 3, 0, 12, 0]`
 `System.out.println(Arrays.toString(yeniArr));` \Rightarrow `[0, 0, 0, 0, 0, 0]`
- `int ilkIndex = 0;`
- `[0, 2, 3, 0, 12, 0]`
- `for (int w : arr) {`
 `if (w != 0) {`
 `yeniArr [ilkIndex] = w;`
 `ilkIndex ++ ;`
 `}`
 `}`
- `System.out.println(Arrays.toString(yeniArr));` \Rightarrow `[2 , 3 , 12 , 0 , 0 , 0]`

ArrayList nedir?



- ArrayList, Java'da dinamik olarak büyüyeabilen bir array yapısıdır (class) ve boyutları önceden belirlenmiş array'lere göre daha esnek ve kullanışlıdır.
- Bir listenin eleman sayısını önceden tahmin edemiyorsanız veya dinamik olarak değişebileceğini biliyorsanız, ArrayList kullanmak daha uygundur.
- Direkt arraylist ismiyle yazdırılabilir , toString() methoduna ihtiyaç yoktur (class olduğu için)

ArrayList nasıl tanımlanır ve initialize edilir?



- `ArrayList<Integer> newAges = new ArrayList<>();`
`newAges.add (15);`
`newAges.add (21);`
`newAges.add (30);` `==> [15, 1, 30]`
- Araya eleman eklemek için `add (index , eklenecek data)`
`newAges.add (1 , 42);` `==> [15, 42, 1, 30]`

Array ve ArrayList arasındaki fark nedir?

Array

- **Primitive** data type ve **referanslarını** depolayabilir
- Baştan eleman sayısı belirtilir
- Daha hızlıdır , memory'de az yer kaplar
- Eleman sayısı belli olan dataları depolamak için kullanılır
- Eleman sayısından bahsedilirken lenght kullanılır
- Elemanları natural ordara göre sıralamak için : Arrays.sort(**array ismi**);

ArrayList

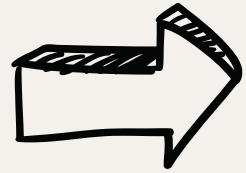
- **Non-primitive** data type ve **wrapper classları** depolayabilir
- Eleman sayısı esnek
- Memory'de arraylerden fazla yer kaplar
- Eleman sayısı değişken dataları depolamak için kullanılır
- Eleman sayısından bahsedilirken size() kullanılır
- **ArrayList ismi.sort(null);**
- Collections.sort(**arraylist ismi**);
-

EXAMPLE

Tekrarli elemanlari olan bir listten, tekrarsiz elemanlari olan bir list elde ediniz.

- `List<Character> a = new ArrayList<>();`
- `a.add('J'); a.add('a'); a.add('v'); a.add('a'); a.add('v');` ==> `[J, a, v, a, v]`
- `List<Character> b = new ArrayList<>();`
- `for (Character w : a) {`
 `if (!b.contains (w)) {`
 `b.add(w);`
 `}`
`}`
- `System.out.println (b);` ==> `[J, a, v]`

MultiDimensional Array

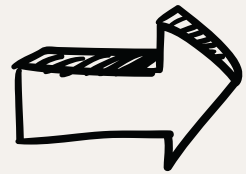


- İki boyutlu bir dizi, satır ve sütunlardan oluşur, tıpkı bir Excel tablosu gibi.

- `int[][] a = new int [3] [2];` ==> 3 dıştaki , 2 içteki eleman sayısı

- `a[0][0]=5;` `a[1][0]=81;` `a[2][0]=123;`

- `a[0][1]=12;` `a[1][1]=45;` `a[2][1]=5;`



- Kısa yol:

- `int[][] a = { {5,12} , {81,45} , {123,5} } ;`

- MultiDimensional Array'leri yazdırırken `toString()` değil, `deepToString()` method'u kullanırız.

`System.out.println (Arrays.deepToString (a));` ==> `[[5, 12], [81, 45], [123, 5]]`

- MultiDimensional Array'den spesifik bir eleman nasıl yazdırılır?

`System.out.println (a [1] [1]);` ==> `45`

- MultiDimensional Array içindeki bir Array nasıl yazdırılır?

`System.out.println(Arrays.toString (a[2]));` ==> `[123 , 5]`

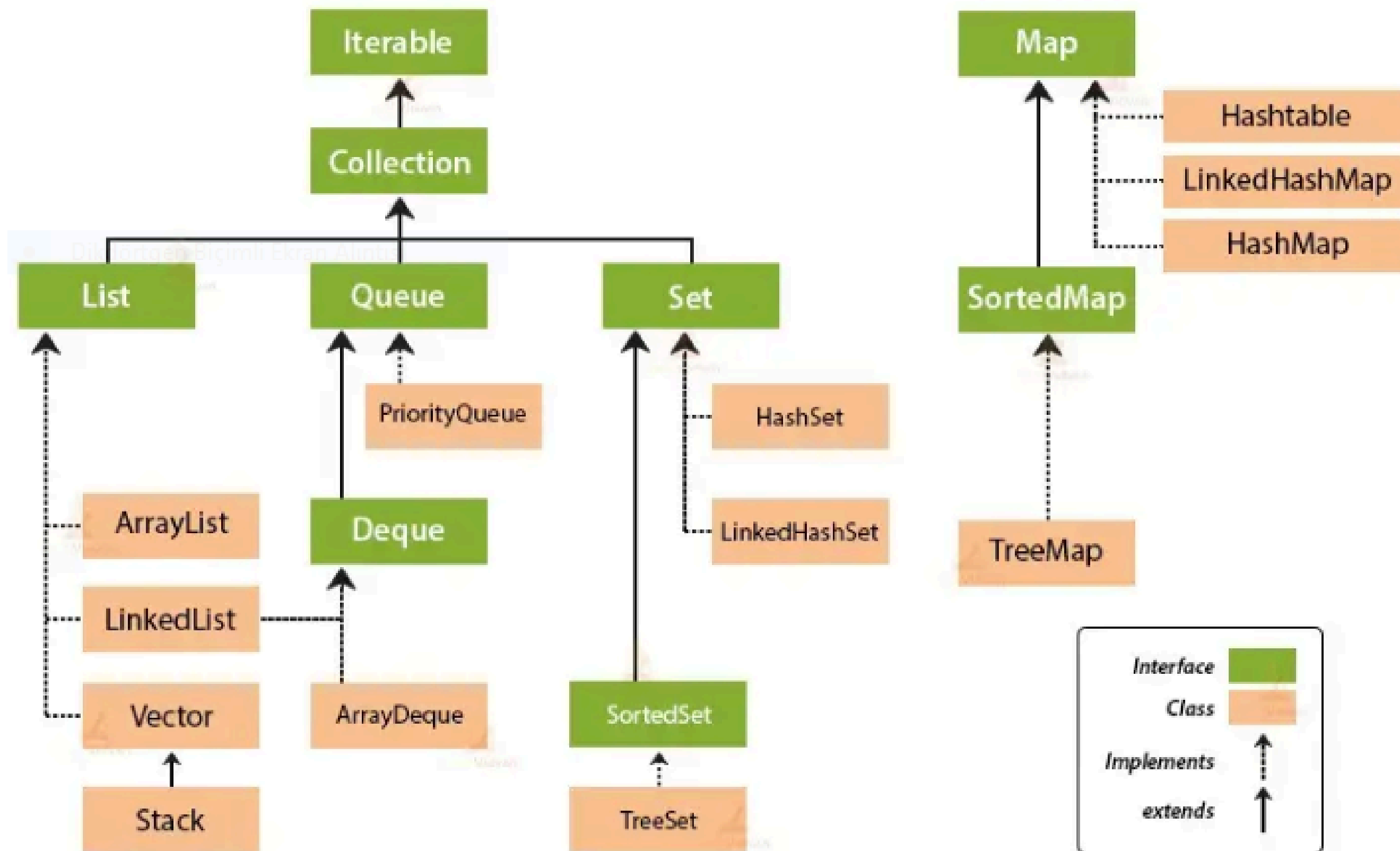
EXAMPLE

int turunde bir Multidimensional Array'deki en kucuk ve en buyuk elemanin toplamini bulunuz.

- `int[][] ages = { { 15 , 4 } , { 12 , 43 , 21 } }; ==> 4+43=47`
- `int enKucuk= ages [0][0]; ==> 15` `int enBuyuk= ages [0][0]; ==> 15`
- ```
for (int[] w : ages) {
 for (int k : w){
 enKucuk = Math.min(enKucuk , k);
 enBuyuk = Math.max(enBuyuk , k);
 }
}
```
- `System.out.println(enKucuk); ==> 4`
- `System.out.println(enBuyuk); ==> 43`
- `System.out.println("En büyük ve en küçük toplamı : " + (enBuyuk+enKucuk) );`



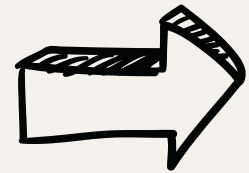
## ***Collection Framework Hierarchy in Java***

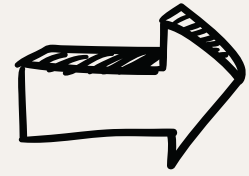


# Collections

---

- Collections, Java'da verileri depolamak, sıralamak, aramak ve manipüle etmek için çeşitli sınıflar ve arayüzler sunar.
- Java'da Collections çerçevesinde en yaygın kullanılan türler şunlardır:
  - \*\*\* **List** : Sıralı bir veri grubudur ve elemanlar indekslerle erişilebilir.  
Örnek: **ArrayList**, **LinkedList**.
  - \*\*\* **Set** : Benzersiz elemanları tutar, yani aynı elemandan sadece bir tane olabilir.  
Örnek: **HashSet**, **TreeSet**.
  - \*\*\* **Map** : Anahtar-değer çiftleri şeklinde verileri tutar. Her anahtar benzersizdir ve bir değere eşlenir. Örnek: **HashMap**, **TreeMap**.
  - \*\*\* **Queue** : İlk giren ilk çıkar (FIFO) mantığıyla çalışır. Örnek: **PriorityQueue**, **LinkedList**.



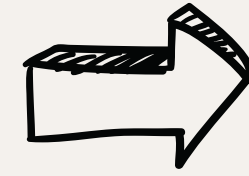


# List

- Elemanları sıralı ve indekslenmiş bir şekilde tutar. Aynı elemandan birden fazla olabilir.

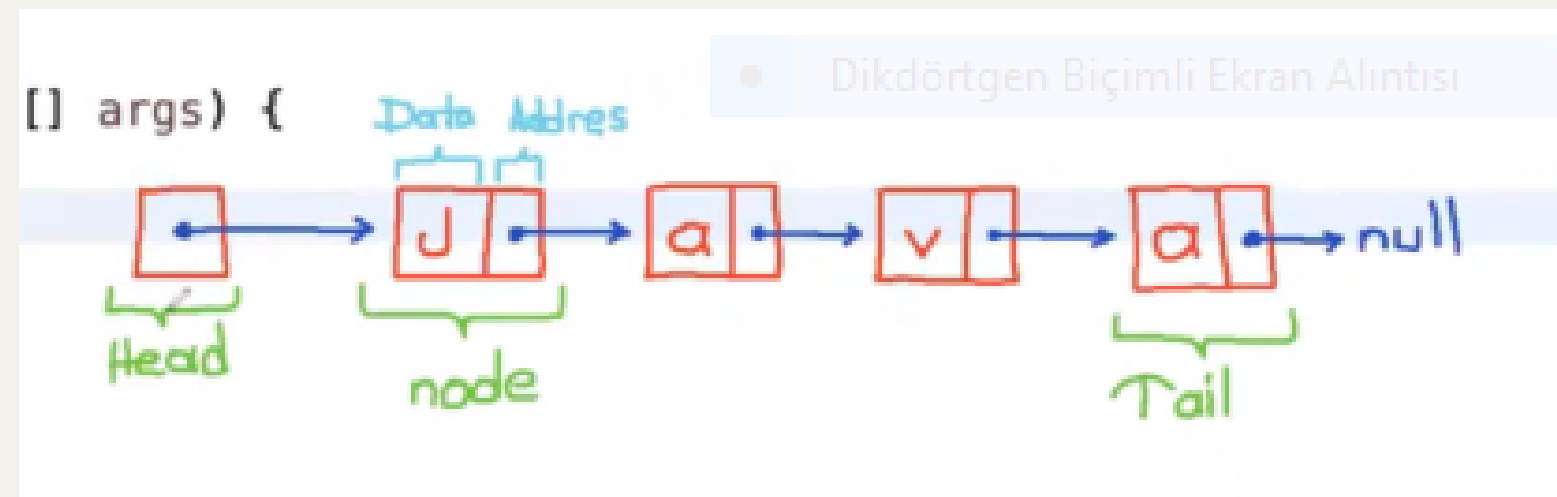
\*\*ArrayList ==> Elemana erişme hız istersek arraylist (index var çünkü)

\*\*LinkedList ==> Ekleme çıkarmada hız istersek linklist



## LinkedList

- LinkedList, **hem List hem de Queue** özelliklerini destekler.
- İçerdiği elemanlar sıralı olarak saklanır ve aynı elemandan birden fazla bulunabilir.
- Elemanlar 'data' ve 'address' olarak iki kısımdan oluşur. Elemanların yapısı farklı olduğundan 'node' olarak adlandırılır. Head sadece address kısmından oluşur
- Ekleme ve silme işlemlerinde hızlıdır, çünkü düğümleri yeniden düzenlemek için dizideki gibi elemanların yerini değiştirmek gerekmez.



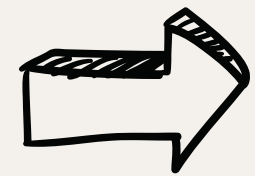
# LinkedList

---

- `LinkedList<String> linkedList = new LinkedList<>();`
- Eleman ekleme :  
`linkedList.add("Apple");`  
`linkedList.add("Banana");`  
`linkedList.add("Cherry");`
- İlk ve son eleman ekleme :  
`linkedList.addFirst("First Fruit");`  
`linkedList.addLast("Last Fruit");`
- İlk ve son elemanı silme :  
`linkedList.removeFirst();`  
`linkedList.removeLast();`
- `pop()` ve `poll()` metodları bir koleksiyonun ilk elemanını alır ve koleksiyondan kaldırır.
  - \*\* `pop()` koleksiyon boşsa `NoSuchElementException` istisnası atar.
  - \*\* `poll()` koleksiyon boşsa `null` döndürür.
- `element()` ve `peek()` metodları bir koleksiyonun ilk elemanına erişmek için kullanılır.
  - \*\* `element()` metodu, koleksiyon boşsa `NoSuchElementException` istisnası atar.
  - \*\* `peek()` metodu, koleksiyon boşsa `null` döndürür.

# Set

- Java Collections Framework'ünde yer alan bir interface'tir ve elemanların **benzersiz** şekilde saklanmasını sağlar. Bir Set içinde aynı elemandan birden fazla bulunamaz ( **unique** ).
- Örneğin; e-mail, tc kimlik no, il plaka no

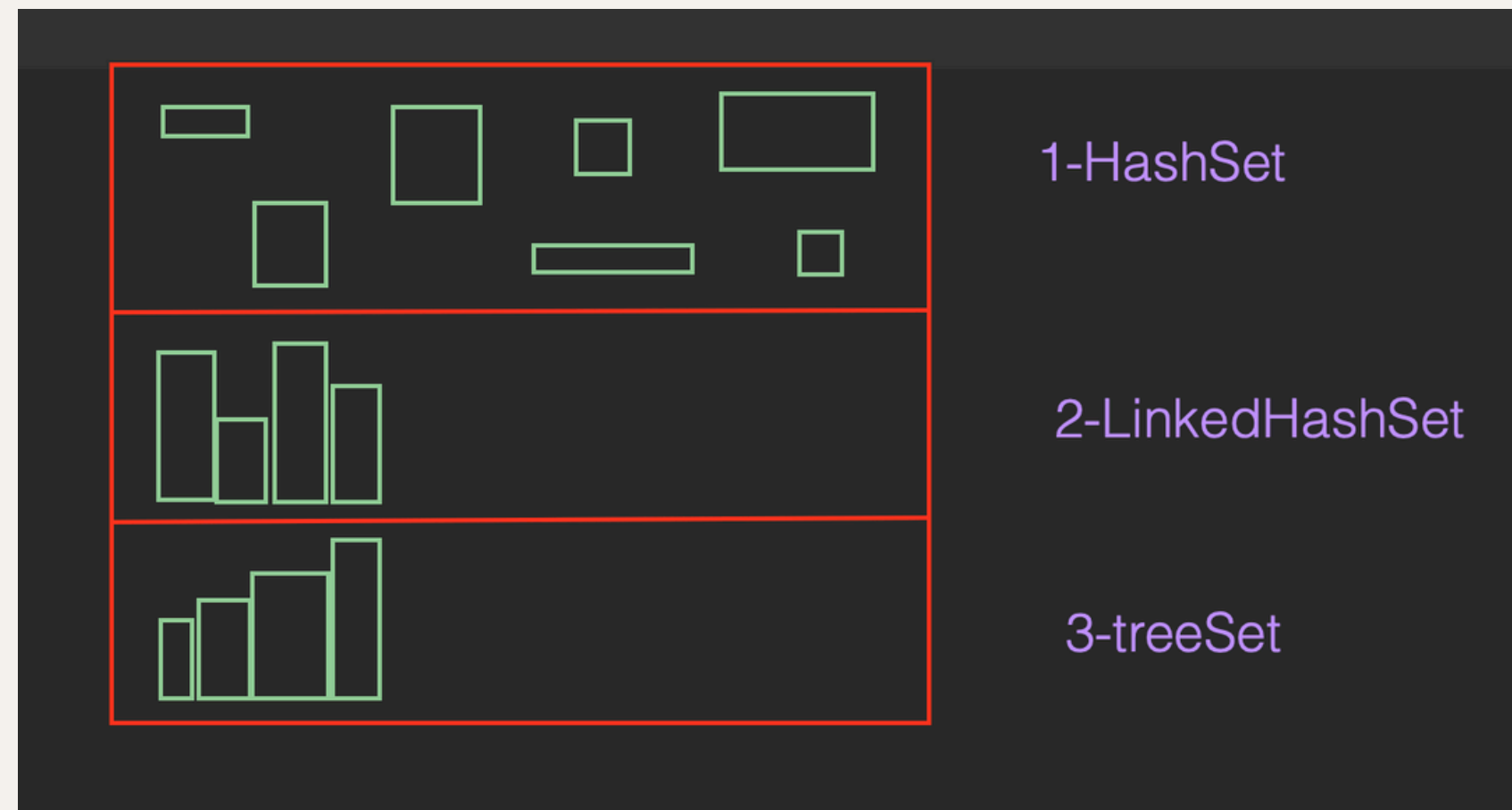


Java'da Set interface'ini uygulayan üç temel Class vardır:

\*\* HashSet

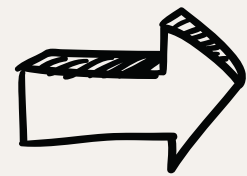
\*\* LinkedHashSet

\*\* TreeSet



# HashSet

- HashSet, elemanları benzersiz ve **sirasız** olarak saklayan bir Set türüdür.
- Elemanların sırası ekleme sırasına göre korunmaz. Elemanların her biri benzersiz bir hash kodu (kimlik numarası) ile depolanır.
- HashSet'ler, **en hızlı** performansı sunar , çünkü elemanları sıralamakla uğraşmaz.
- null'i eleman olarak kabul ederler



```
HashSet<String> hs = new HashSet<>();
```

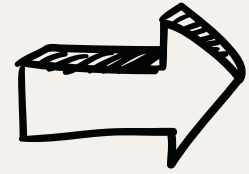
```
System.out.println (hs); ==> [] (teknik olarak içinde null yoktur , hiçlik var)
```

- hs.add("Necla");
- hs.add("Uğur");
- System.out.println(hs); ==> [ Uğur ,Necla ]
- hs.add("Necla"); ==> tekrarlı elemanı **üzerine yazar**
- System.out.println(hs); ==> [ Uğur ,Necla ]
- hs.add (null) ; ==> hashSet'e **null ekleyebiliriz**
- System.out.println(hs); ==> [ **null** ,Uğur ,Necla ]

# HashSet'e eleman eklemenin kısa yolu

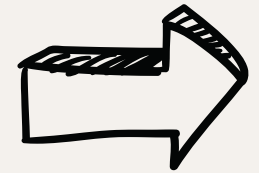
---

- `HashSet<String> hs2 = new HashSet<>( Arrays.asList("Ali","Veli") );`
- Burada new ile oluşturulduğu için kısıtlama olmaz
- `System.out.println(hs2);` ==> [Veli, Ali]



# LinkedHashSet

- Insertion order'a göre sıralar ( ekleme sırası )



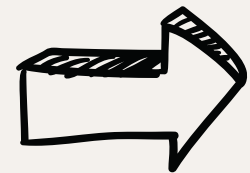
```
LinkedHashSet<Integer> lhs = new LinkedHashSet<>();
```

```
lhs.add (14) ;
```

```
lhs.add (19) ;
```

```
lhs.add (7) ;
```

```
lhs.add (null) ; ==> System.out.println (lhs) ; ==> [14, 19, 7, null]
```




```
LinkedHashSet<Integer> ls = new LinkedHashSet<>();
```

```
ls.add (14) ;
```

```
ls.add (19) ;
```

```
ls.add (11) ; ==> System.out.println(ls); ==> [14, 19, 11]
```

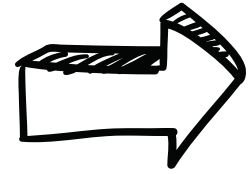
- `retainAll()` metodu, her iki koleksiyonda da bulunan öğeleri(**ortak**) bulmak için kullanılır

 `lhs.retainAll( ls );` ==> değişikliğe uğrayan methodun sol tarafı



# TreeSet

- **Natural order**'a göre sıralar
- TreeSet içinde null bir eleman eklemeye çalıştığınızda NullPointerException fırlatır. Çünkü null değeri olan bir elemanın, diğer elemanlarla nasıl karşılaştırılacağı belirsizdir. ( **x null** )



```
TreeSet<Character> ts = new TreeSet<>();
```

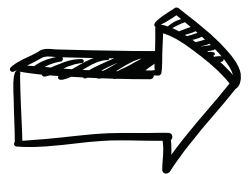
```
ts.add('G');
```

```
ts.add('A');
```

```
ts.add('Z');
```

```
ts.add('R');
```

```
ts.add('U'); ==> System.out.println(ts); ==> [A, G, R, U, Z]
```



**subset()** belirli bir aralıktaki elemanları içeren yeni bir küme oluşturmak için kullanılır.

subSet() yöntemini kullanmak için, SortedSet türünden bir değişken oluşturmanız gerekir.

```
SortedSet<Character> ss = ts.subSet('G','U');
```

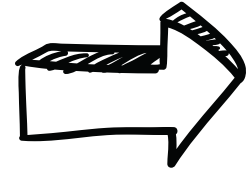
==> G-U arasındaki elemanları al

G dahil , U dahil değil [G,U)

```
System.out.println(ss); ==> [G, R]
```

# Bir array'de tekrar eden elemanları bulun ve bu tekrar eden elemanları ekrana yazdırın.

---



- `int[] array = {1, 2, 3, 4, 5, 2, 3, 6, 7, 8, 9, 1};`
- `HashSet<Integer> seen = new HashSet<>( );`
- `HashSet<Integer> duplicates = new HashSet<>( );`
- ```
for (int w : array) {  
    if ( !seen.add(w) ) {  
        duplicates.add(w);  
    }  
}
```
- Tekrar eden elemanları yazdırma
- `System.out.println(duplicates);` ==> `[1, 2, 3]`
- `System.out.println(seen);` ==> `[1, 2, 3, 4, 5, 6, 7, 8, 9]`

Queue

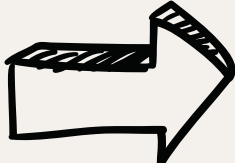
- İlk giren ilk çıkar (FIFO) mantığıyla çalışır
- Elemanlar kuyruğun sonuna eklenir ve kuyruğun başından çıkarılır. Araya eleman eklemek mümkün değildir.
- İşlem sıralarını yönetmek için kullanılır, örneğin yazdırma işlemleri, müşteri hizmetleri çağrı sıraları.

** Queue ==> Queue<String> depo = new LinkedList<>();

** PriorityQueue ==> PriorityQueue<String> acilSirasi = new PriorityQueue<>();

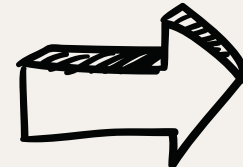
** Deque ==> Deque<String> d = new LinkedList<>();
Deque<String> d = new ArrayDeque<>();

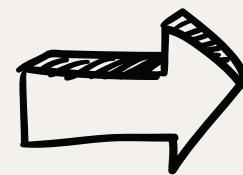
** ArrayDeque ==> ArrayDeque<String> ad = new ArrayDeque<>();

 Queue ve Deque interface olduğu için obje üretemez. Bu yüzden adresi bu kısımdan alır ve onun methodlarını kullanabilirler , üreten kısımda ise LinkedList<>() veya ArrayDeque<>(); bulunur.

Queue

- Bu tip kuyruk, ilk giren elemanın ilk çıktığı (**FIFO**) prensibiyle çalışır. Kuyruğa eklenen ilk eleman, kuyruğun başında olur ve ilk çıkarılır.

 Bir dondurma sırası düşünün. İlk sıraya giren kişi, dondurmasını ilk alır. Yeni gelenler sıranın sonuna eklenir ve sırayla dondurmalarını alır.



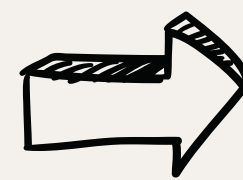
```
Queue<String> queue = new LinkedList<>();  
queue.add("Ali");  
queue.add("Ayşe");  
queue.add("Mehmet"); [ Ali , Ayşe , Mehmet ]
```

- **remove()** : Kuyruğun başındaki öğeyi kuyruktan **kaldırır ve döndürür**. Kuyruk **boşsa** bir **istisna** fırlatır.
- **poll()** : Kuyruğun başındaki öğeyi kuyruktan **kaldırır ve döndürür**. Kuyruk **boşsa null** döner.
- **element()** : Kuyruğun başındaki öğeyi **döndürür**, ancak kuyruk **boşsa** bir **istisna** fırlatır.
- **peek()** : Kuyruğun başındaki öğeyi **döndürür**, kuyruk **boşsa null** döner.
- **clear()** : Kuyruğun içindeki **tüm öğeleri kaldırır**, böylece kuyruk boş bir duruma gelir. []

PriorityQueue

- Bu tip kuyrukta, elemanlar önceliklerine göre sıralanır. Yüksek öncelikli elemanlar önce işlenir. FIFO sırası takip edilmez. Bu öncelik sırasını belirlemek için doğal sıralama veya özel bir karşılaştırıcı (Comparator) kullanabilirsiniz.

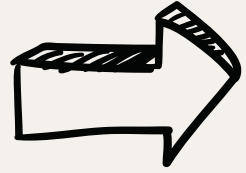
 Acil serviste hastalar düşünün. Daha acil olan hastalar önce tedavi edilir, daha az acil olanlar bekleyebilir.



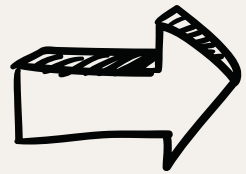
```
PriorityQueue<String> acilSirasi = new PriorityQueue<>();  
acilSirasi.add("Mehmet");  
acilSirasi.add("Mahmut");  
acilSirasi.add("Ekrem");  
acilSirasi.add("Oya");  
acilSirasi.add("Tansu");
```

```
System.out.println (acilSirasi) ; ==> [Ekrem, Mehmet, Mahmut, Oya, Tansu]
```

Natural order'a göre sıralar

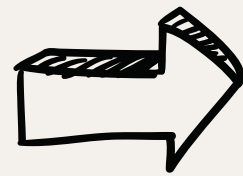


- `PriorityQueue<Integer> queue = new PriorityQueue<>();`
- `queue.add(4);`
- `queue.add(2);`
- `queue.add(5);`
- `queue.add(1);`
- `System.out.println (queue) ;` ==> `[1, 2, 5, 4]`



- `PriorityQueue<Integer> pq = new PriorityQueue<>(Comparator.reverseOrder());`
- `pq.add(4);`
- `pq.add(2);`
- `pq.add(5);`
- `pq.add(1);`
- `System.out.println (pq) ;` ==> `[5, 2, 4, 1]`

-
- Comparator arabirimini kullanarak kendi özel karşılaştırıcılarınızı oluşturabilirsiniz. Bu, elemanları istediğiniz herhangi bir kriter gere sıralamanıza olanak tanır.



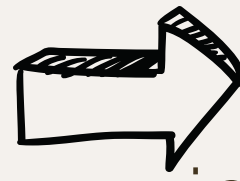
String elemanlarını uzunluklarına göre sıra :

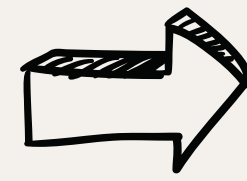
- `Comparator<String> lengthComparator = new Comparator<String>() {
 public int compare(String s1, String s2) {
 return Integer.compare(s1.length(), s2.length());
 }
};`

- `PriorityQueue<String> x = new PriorityQueue<>(lengthComparator);
x.add("apple");
x.add("banana");
x.add("kiwi");
System.out.println(x); ==> [kiwi, banana, apple]`

Deque

- Bu tip kuyruk, hem baştan hem de sondan eleman ekleyip çıkarabilme özelliğine sahiptir.

 İki uçlu bir otobüs düşünün. Yolcular hem ön kapıdan hem de arka kapıdan binip inebilirler.

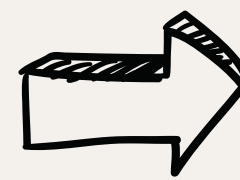


```
Deque<String> d = new LinkedList<>( );  
d.add ("Mehmet");  
d.add ("Mahmut");  
d.add ("Ekrem");
```

```
System.out.println (d);
```

 ==> [Mehmet, Mahmut, Ekrem]

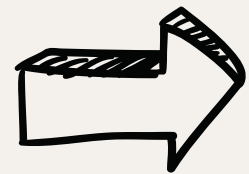
LinkedList olduğundan insertion order'a göre sıraladı



```
d.addFirst ("Ön kapıdan binen yolcu");  
d.addLast ("Arka kapıdan binen yolcu");  
System.out.println ( "İlk inen yolcu: " + deque.removeFirst( ) );  
System.out.println ( "Son inen yolcu: " + deque.removeLast( ) );
```


Map

- Anahtar-değer çiftleri şeklinde verileri tutar. Her anahtar benzersizdir ve bir değere eşlenir.
- Map'ler collections değildir. Ordan miras almaz



key:

Tekrarsız (unique)

Set kullanılır

value:

Tekrarlı olabilir

Genellikle Collection yapısı kullanılır

****HashMap** : Sıralamayı garanti etmez, Verilere hızlı erişimin gerektiği durumlarda kullanılabilir

****LinkedHashMap** : Verilerin ekleme sırasına göre saklanmasının önemli olduğu durumlarda kullanılabilir

****TreeMap** : Key'leri natural order'a göre (alfabetik veya sayısal) sıralar.

Map'ler key-value çiftleri ile çalışırken kullanılan methodlar :

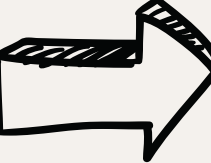
- `put(key, value)` : Map'e eleman ekler
- `putIfAbsent()` : Belirtilen key ve value çiftini, key Map'te yoksa Map'e ekler.
Key Map'te zaten varsa, metodun hiçbir etkisi olmaz.
- `get(key)` : Belirli bir key'in value değerini return eder, key bulunmuyorsa null return eder
- `getOrDefault(key, value)` : Belirli bir key'in value değerini return eder. Map'te o eleman yok ise verdiğimiz default değeri return eder, key bulunuyorsa etkisi olmaz
- `replace(key ,newValue)` : value'lari key'leri kullanarak update etmeye yarar
`replace(key , oldValue ,newValue)`
- `containsKey(key)` : Map'te bir key olup olmadığını kontrol eder (Boolean return eder)
- `containsValue(value)` : Map'te bir value olup olmadığını kontrol eder (Boolean return eder)
- `remove(key)` : Key kullanarak entry silmeye yarar (`mapName.remove("keyName");`)
- `keySet()` : Tüm anahtarları döner
- `values()` : Tüm değerleri döner.

Bir metin içindeki her karakterin kaç kez geçtiğini bulan bir Java programı yazın. Hello world!

- `String text = "Hello world!";`
`text = text .toLowerCase() .replaceAll (" [^a-z]" , "");` ==> `helloworld`
- `char[] harfArray = text.toCharArray();`
`System.out.println (Arrays.toString (harfArray));` ==> `[h, e, l, l, o, w, o, r, l, d]`
- `HashMap<Character, Integer> harfMap = new HashMap<>();`
`for (Character w : harfArray) {`
 `if (harfMap .containsKey (w)) {`
 `harfMap .put(w, harfMap .get(w) + 1);`
 `} else {`
 `harfMap .put (w, 1);`
 `}`
`}`
- `System.out.println(harfMap);` ==> `{r=1, d=1, e=1, w=1, h=1, l=3, o=2}`

LinkedHashMap

- Bir kafede siparişleri takip ettiğinizi düşünün. Siparişler hem müşteri numarası hem de sipariş sırasına göre takip edilmelidir. Bu durumda, siparişleri sıralı bir şekilde saklamak için LinkedHashMap kullanabiliriz.

 `LinkedHashMap<Integer, String> orders = new LinkedHashMap<>();`
`orders.put(101, "Latte");`
`orders.put(102, "Cappuccino");`
`orders.put(103, "Espresso");`

- Tüm siparişleri ekleme sırasına göre yazdır

`System.out.println(orders);` `==>` {101=Latte, 102=Cappuccino, 103=Espresso}

- Bir siparişi tamamla ve çıkar

`int completedOrder = 101;`

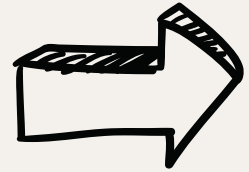
`orders.remove(completedOrder);`

`System.out.println("Tamamlanan Sipariş: Müşteri Numarası = " + completedOrder);`

`==>` Tamamlanan Sipariş: Müşteri Numarası = 101

TreeMap

- Treemap'ler entry'leri natural order'a gore siraya koyar, bu yuzden yavastirlar
- Treemap'lerin key'lerinde null kullanilamaz ama value kisimlerinde istediginiz kadar kullanabilirsiniz
- Treemapler key'e bakarak siralama yapar



```
TreeMap<String, Integer> countryPoulation = new TreeMap<>();
```

```
countryPoulation .put ("Germany",830000000);
```

```
countryPoulation .put ("USA",4000000000 );
```

```
countryPoulation .put ("Turkey",830000000);
```

```
countryPoulation .put (null,830000000 ); ==> HATA
```

```
countryPoulation .put ("Italy",null);
```

```
countryPoulation .put ("France",null);
```

```
System.out.println(countryPoulation); ==> {France=null, Germany=830000000,  
Italy=null, Turkey=830000000, USA=4000000000}
```

Thank You
FOR YOUR PARTICIPATION