# Class PriorityQueue

**java.util**

**Class PriorityQueue<E>**

```
java.lang.Object
  └ java.util.AbstractCollection<E>
      └ java.util.AbstractQueue<E>
          └ java.util.PriorityQueue<E>
```

**Parametre tipleri:**

`E` - the type of elements held in this collection

**Kılgıladığı Arayüzler:**

Serializable, Iterable<E>, Collection<E>, Queue<E>

**Bildirimi:**

```
public class PriorityQueue<E>
extends AbstractQueue<E>
implements Serializable
```

`PriorityQueue` sınıfı Java Collections Framework 'un bir üyesidir.

`PriorityQueue`, terim anlamıyla gündelik yaşamda sık karşılaştığımız bir olguyu belirler. Bazı durumlarda bir işi öteki işlerin hepsinden önce yapmak zorunda kalabiliriz. Örneğin, Bir banka veznesinde kuyruğa girenler arasında öncelik sırası önde olanındır. Ancak, bir kavşakta geçiş önceliği cankurtaranındır. Bir hava meydanına iniş sırası bekleyen uçaklar arasında, öncelik sırası acil iniş isteyen uçağındır. Bir hastanede muayene önceliği durumu en acil olan hastanındır.

Görüldüğü gibi, bir koleksiyon içinde öncelik sırasını farklı amaçlarla belirleyebiliriz. En basiti, ilk gelen ilk çıkar dediğimiz FIFO yapısıdır. Ama bu yapı karşılaşılacak bütün olasılıklara çözüm getiremez. Dolayısıyla, koleksiyonun öğelerini istenen önceliğe göre sıralayan bir yapıya gereksinim vardır.

Biraz genellemeyle, `PriorityQueue` yapısına da kuyruk diyeceğiz; ama burada yüklenen anlamı FIFO yapısından farklı olabilir. Sıralama bazen öğelerin doğal sırasıdır, bazen ilgili Comparator tarafından belirlenen sıradır.

Bir `PriorityQueue` kuyruğunda öğeler ya doğal sıralarındadır ya da kuruluş anında kullanılan Comparator'un yaptığı sıradadırlar. Tabii, böyle oluşu, hangi kurucunun kullanıldığına bağlıdır. Bir `PriorityQueue` kuyruğuna `null` öğe konamaz. Doğal sırasında olan bir `PriorityQueue` kuyruğuna, kendi öğeleriyle bağdaşmayan (mukayese edilemeyen) bir nesne konulamaz. Böyle yapılırsa derleyiciden `ClassCastException` hatası alınır.

`PriorityQueue` kuyruğunun başı (head), kullanılan sıralamaya göre en küçük olan öğedir. Eğer en küçük olan birden çok öğe varsa, kuyruğun başı onlardan birisidir. `Poll()`, `remove()`, `peek()` ve `element()` metotları kuyruğun başına erişir.

`PriorityQueue` kuyruğunun uzunluğu için bir sınır yoktur; ama yeni öğeler eklendikçe kendi kendine uzunluğunu değiştirebilir; yani kuyruğun sığası otomatik olarak artar.

`PriorityQueue` kuyruğu *Collection* ve *Iterator* arayüzlerinin seçimlik bütün metotlarını kılgılar.

**İlgili Konular:**

Serialized Form

---

| `PriorityQueue` **Sınıfının Kurucuları** |
|---|
| **`PriorityQueue`**`()`<br>Öğeleri doğal sırada sıralanacak olan ve sığası 11 olan boş bir `PriorityQueue` yaratır. |
| **`PriorityQueue`**`(Collection<? extends E> c)`<br>Verilen koleksiyonun öğelerini içeren bir `PriorityQueue` yaratır. |
| **`PriorityQueue`**`(int initialCapacity)`<br>Öğeleri doğal sırada sıralanacak olan ve sığası parametrede belirtilen sayıda olan boş bir `PriorityQueue` yaratır. |
| **`PriorityQueue`**`(int initialCapacity,`<br>`Comparator<? super E> comparator)`<br>Öğeleri verilen comparator ile sıralanacak olan ve sığası parametrede belirtilen sayıda olan boş bir `PriorityQueue` yaratır. |
| **`PriorityQueue`**`(PriorityQueue<? extends E> c)`<br>Verilen `PriorityQueue` 'nun öğelerini içeren bir `PriorityQueue` yaratır. |
| **`PriorityQueue`**`(SortedSet<? extends E> c)`<br>Verilen SortedSet kümesinin öğelerini içeren bir `PriorityQueue` yaratır. |

| `PriorityQueue` **Sınıfının Metotları** | |
|---|---|
| `boolean` | **`add`**`(E e)`<br>Parametrede verilen öğeyi `PriorityQueue` yapısına katar. |
| `void` | **`clear`**`()`<br>`PriorityQueue` yapısındaki bütün öğeleri siler. |
| `Comparator<? super E>` | **`comparator`**`()`<br>`PriorityQueue` yapısının öğelerini sıralayacak comparatoru verir. Eğer |

| | |
|---|---|
| | yapı doğal sıralı ise null değer verir. |
| boolean | **contains**(Object o)<br>Parametrede verilen nesne PriorityQueue içinde ise true verir. |
| Iterator<E> | **iterator**()<br>PriorityQueue yapısını tarayacak bir iterator verir. |
| boolean | **offer**(E e)<br>Parametrede verilen öğeyi PriorityQueue yapısına koyar. |
| E | **peek**()<br>Kuyruğun başındaki öğeyi verir, ama onu kuyruktan silmez. Kuyruk boş ise, null verir. |
| E | **poll**()<br>Kuyruğun başındaki öğeyi verir ve onu kuyruktan siler. Kuyruk boş ise, null verir. |
| boolean | **remove**(Object o)<br>Parametrede belirtilen nesne kuyrukta ise onu siler. |
| int | **size**()<br>Kuyruğun uzunluğunu (kuyrukta bekleyen öğe sayısını) verir. Uzunlık kavramının sığa (capacity) kavramından farklı olduğuna dikkat ediniz. |
| Object[] | **toArray**()<br>Kuyruktaki öğeleri içeren bir array yaratır. |
| <T> T[] | **toArray**(T[] a)<br>Kuyruktaki öğeleri içeren bir array yaratır. Yaratılan array'in runtime tipi verilen arraydir. |

| **java.util.AbstractQueue Sınıfından Kalıtsal Gelen Metotlar** |
|---|
| addAll, element, remove |

| **java.util.AbstractCollection Sınıfından Kalıtsal Gelen Metotlar** |
|---|

| containsAll, isEmpty, removeAll, retainAll, toString |
|---|

| java.lang.Object Sınıfından Kalıtsal Gelen Metotlar |
|---|
| clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait |

| Methods inherited from interface java.util.Collection Arayüzünden Gelen Metotlar |
|---|
| containsAll, equals, hashCode, isEmpty, removeAll, retainAll |

**Örnek:**

```
import java.util.*;

public class PriorityQueueDemo {

	public static void main(String[] args) {
		// bir PriorityQueue yarat
		PriorityQueue p = new PriorityQueue();
		// kuyruğa öğe ekle
		p.add("1");
		p.add("2");
		p.add("3");
		p.add("4");
		p.add("5");
		p.add("6");
		Iterator i = p.iterator();
		while (i.hasNext()) {
			System.out.print(i.next() + "\t");
		}
		System.out.println();
		// kuyruğun başındaki öğeyi al
		System.out.print("Kuyruğun başı : " + p.peek());
	}
}

	/*
	Çıktı:
	 1	2	3	4	5	6
	 Kuyruğun başı : 1
	*/
```

**Örnek:**

```
import java.util.*;

public class PriorityQueueDemo {
	PriorityQueue<String> stringQueue;

	public static void main(String[] args) {
		PriorityQueueDemo pqd = new PriorityQueueDemo();

		pqd.stringQueue = new PriorityQueue<String>();

		pqd.stringQueue.add("ab");
		pqd.stringQueue.add("abcd");
		pqd.stringQueue.add("abc");
		pqd.stringQueue.add("a");

		while (pqd.stringQueue.size() > 0)
			System.out.println(pqd.stringQueue.remove());
	}
}
	/*
	Çıktı:
	 a
	 ab
	 abc
	 abcd
	*/
```

**Örnek:**

```
import java.util.Comparator;
import java.util.PriorityQueue;
import java.util.Queue;

public class PriorityQueueDemo {

	public static void main(String[] args) {
		// Comparator çift sayılara öncelik veriyor
		PriorityQueue<Integer> pq = new PriorityQueue<Integer>(20,
				new Comparator<Integer>() {
					public int compare(Integer i, Integer j) {
						int result = i % 2 - j % 2;
						if (result == 0)
							result = i - j;
						return result;
					}
				});
		// ters sırada sayıları kuyruğa ekle
		for (int i = 0; i < 20; i++) {
			pq.offer(20 - i);
		}
		// öğeleri Comparator'un sırayla yazdır
		for (int i = 0; i < 20; i++) {
			System.out.print("\t" + pq.poll());
		}
```

```
        }
}
    /*
    Çıktı:
      2    4     6     8     10    12    14    16    18    20    1     3
      5    7     9     11    13    15    17    19
    */
```