Queue (kuyruk) Arayüzü

java.util
Interface Queue<E>

Veri tipi parametresi:

`E` – parametresi koleksiyondaki veri tipini belirtir.

Üstarayüzleri:

Collection<E>, Iterable<E>

Altarayüzleri:

BlockingQueue<E>

Kılgılayan sınıflar:

AbstractQueue, ArrayBlockingQueue, ConcurrentLinkedQueue, DelayQueue, LinkedBlocking Queue, LinkedList, PriorityBlockingQueue, PriorityQueue, SynchronousQueue

---

```
public interface Queue<E>
extends Collection<E>
```

Queue arayüzü *Java Collections Framework* 'un bir üyesidir. İşlemlerden geçmeden önce öğeleri depolanmasını sağlar. *Collection* arayüzünün bir altarayüzü olduğundan, onun bütün metotlarını kullanır. Onlara ek olarak, kuyruk yapısındaki ekleme, silme gibi işlemleri kolaylaştıran metotlara sahiptir.

Ortaya çıkış nedeniyle, *Queue* bir *FIFO* (first-in-first-out, ilk giren ilk çıkar) yapısıdır. Ancak, *Queue* ve *Collection* arayüzündeki metotlar kullanılarak, *FIFO* yapısı *LIFO* (lsast-input-first-output, son giren ilk çıkar) yapısı gibi kullanılabilir. Fıfo ve *LIFO* yapılarına öncelik sıralamalı kuyruklar (priority queues) denilir. Bu yapılarda, öğeler ya doğal sıralarında ya da bu iş için oluşturulan comparator (mukayeseci) tarafından sıraya konulur. Tabii, *LinkedList* yapısı her ikisinden daha geneldir. Yapı ister *FIFO*, ister *LIFO* olsun, ilk çıkan öğe kuyruğun başındadır; `remove()` ya da `poll()` metodu ile alınır.

FIFO kuyruğunda yeni gelen her öğe kuyruğun sonuna *offer()* metodu ile eklenir. *LIFO* yapısında yeni gelen öğe kuyruğun başına *push()* metodu ile eklenir. *LinkedList* yapısında ise, yeni gelen öğeyi dizinin istenilen yerine ekleyen metotlar vardır.

offer() metodu *Collection.add()* methodunun yaptığı işi yapar; ancak hata işlemeleri birbirlerinden farklıdır. Birincide sığanın dolması gibi nedenlerle yeni öğe eklenemesi durumu normal karşılanır ve metot false değerini alır; program kesilmez. Ancak, ikincide, öğe eklenememesi

durumunda hata oluşur ve programın öngörülebilir hatalar için yapılacak işlemleri açıklıkla belirtmesi gerekir.

`remove()` ve `poll()` metotlarının her ikisi de kuyruğun başındaki öğeyi değer olarak verirler ve onu kuyruğun başından atarlar (silerler). Böylece, kuyruğun başı, sonraki öğeye geçer. Tabii, kuyruğun başı, sıralamanın *FIFO* ya da *LIFO* oluşuna bağlı olarak ilk giren ya da son giren öğe olabilir. İki metot arasındaki fark şudur: Kuyruk boş olduğunda `remove()` metodu hata verir, `poll()` metodu ise *null* verir.

Özel olarak, `element()` ve `peek()` metotları kuyruğun başındaki öğeyi değer olarak verirler, ama onları kuyruktan silmezler.

Queue arayüzü, programcılıkta çok kullanılan kuyruk bloklama metotlarına sahip değildir. O metotlar, Queue arayüzünün altarayüzü olan `BlockingQueue` arayüzünde tanımlıdırlar.

Queue arayüzü kuyruğa *null* öğe eklenmesine izin vermez; ama bu iş *LinkedList* yapısında mümkündür.

Queue yapılarında, genellikle, öğeler arasında eşitliği belirleyen *equals* ve *hashCode* metotları kullanılmaz; çünkü öğeler eşit olsalar bile sıralamadaki yerleriyle birbirlerinden farklı niteliklere sahiptirler.

Kuyruk yapıları için ayrıca şunlara bakınız:

`Collection`, `LinkedList`, `PriorityQueue`, `LinkedBlockingQueue`, `BlockingQueue`, `ArrayBlockingQueue`, `LinkedBlockingQueue`, `PriorityBlockingQueue`

Metotlar	
`E`	**`element`**`()` Kuyruğun başındaki öğeyi verir, ama onu kuyruktan atmaz.
`boolean`	**`offer`**`(E o)` Mümkünse, verilen öğeyi kuyruğa ekler.
`E`	**`peek`**`()` Kuyruğun başındaki öğeyi verir, ama onu kuyruktan atmaz. Kuyruk boşsa, *null* verir.
`E`	**`poll`**`()` Kuyruğun başındaki öğeyi verir ve onu kuyruktan atar. Kuyruk boşsa, *null* verir.
`E`	**`remove`**`()` Kuyruğun başındaki öğeyi verir ve onu kuyruktan atar.

java.util.Collection arayüzünden kalıtım yoluyla alınan metotlar:
`add`, `addAll`, `clear`, `contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `remove`, `removeAll`, `retainAll`, `size`, `toArray`, `toArray`

Örnek 1:

Aşağıdaki program *kuyruk* adında bir `LinkedList` listesi yaratıyor. Liste *Merve⇒Tuğçe* sırasında bir kuyruktur (queue). İlk gelen kuyruğun önüne, son gelen kuyruğun sonuna konuşlanıyor.

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Queue;

public class Kuyruk {

	public static void main(String[] args) {

		Queue<String> kuyruk = new LinkedList<String>();

		kuyruk.offer("Merve");
		kuyruk.offer("Damla");
		kuyruk.offer("Aykut");
		kuyruk.offer("Fırat");
		kuyruk.offer("Ceyda");
		kuyruk.offer("Görkem");
		kuyruk.offer("Tuğçe");

		Iterator it = kuyruk.iterator();

		System.out.println("Initial Size of Queue :" + kuyruk.size());

		while (it.hasNext()) {
 String iteratorValue = (String) it.next();
 System.out.println("Kuyrukta sonraki öğe :" + iteratorValue);
		}

		// get value and does not remove element from queue
		System.out.println("Queue.peek()  :" + kuyruk.peek());

		// get first value and remove that object from queue
		System.out.println("Queue.poll()  :" + kuyruk.poll());

 System.out.println("Kuyruğun kalan öğe sayısı :" + kuyruk.size());
	}
}
	/*
	Çıktı:
	Kuyrukta sonraki öğe :Fırat
	Kuyrukta sonraki öğe :Ceyda
	Kuyrukta sonraki öğe :Görkem
	Kuyrukta sonraki öğe :Tuğçe
	Queue.peek()  :Merve
	Queue.poll()  :Merve
	Kuyruğun kalan öğe sayısı :6
	*/
```

Örnek 2:

Aşağıdaki program kuyruğu bir metotla bir `LinkedList` listesi olarak yaratıyor. Liste *Merve⇒Tuğçe* sırasında bir kuyruktur (queue). İlk gelen kuyruğun önüne, son gelen kuyruğun sonuna konuşlanıyor.

```
import java.util.LinkedList;
import java.util.Queue;
```

```
public class Kuyruk {

	public void queueExample() {

		Queue queue = new LinkedList();

		// add() metodu ile kuyruğa öğe ekleme
		// ekleme yapılamazsa hata verir
		queue.add("Deniz");
		queue.add("Berna");

		// offer() metodu ile kuyruğa öğe ekleme
		// ekleme yapılamazsa false verir
		queue.offer("Volkan");
		queue.offer("Çağlar");

		// remove() metodu kuyruğun başındaki değeri verir ve onu
kuyruktan atar
		// Kuyruk boş ise java.util.NoSuchElementException hatasını
verir.
		System.out.println("remove()   : " + queue.remove());

		// element() metodu kuyruğun başındaki öğeyi verir; onu
kuyruktan atmaz
		// Kuyruk boş ise java.util.NoSuchElementException hatasını
verir.
		System.out.println("element() : " + queue.element());

		// poll() metodu kuyruğun başındaki öğeyi verir ve onu
kuyruktan atar
		// Kuyruk boş ise false değerini verir
		System.out.println("poll()     : " + queue.poll());

		// peek() metodu kuyruğun başındaki öğeyi verir; onu kuyruktan
atmaz
		// Kuyruk boş ise false değerini verir
		System.out.println("peek()     : " + queue.peek());

	}

	public static void main(String[] args) {
		new Kuyruk().queueExample();
	}
}
	/*
	Çıktı:
	remove()   : Deniz
	element() : Berna
	poll()     : Berna
	peek()     : Volkan
	*/
```

Örnek 3:

```
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 4:

```
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.

Örnek 5:

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

```
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
          */
```

Örnek 6:

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.

```
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
	*/
```