Deque Interface

Deque Interface’s definition is as follows.

**public** **interface** Deque<E> **extends** Queue<E>

Deque<E> interface comes under collections framework and accepts E as element.

Deque<E> interface extends Queue<E> & Queue<E> extends Collection<E> making Deque<E> a collection.

Deque is a collection that supports insertion and deletion at both the ends. So Deque name, **Double ended Queue.**

Let us understand the methods provided in this interface.

As there are tons of methods in this interface we will understand each of them by performing **Deque** (double ended queue) **operations** on LinkList<E>.

|  |  |
| --- | --- |
| **Method** | **Description.** |
| **boolean** add(E e); | adds the element at the end of queue without violating capacity restrictions and return true upon success and throwing IllegalStateException if no space is available. |
| **void** addFirst(E e); | Inserts element in front of queue. This method is push(e) method equivalent in Stack<E>. |
| **void** addLast(E e); | adds the element E at the end queue. |
| **boolean** contains(Object o); | Checks whether the Deque has specified element. Returns true if it exists. |
| Iterator<E> descendingIterator(); | Returns an iterator that can traverse Deque in reverse sequential manner. |
| E element(); | Retrieves but does not remove the element from the head of the queue represented by the Deque. |
| E getFirst(); | Retrieves but does not remove the first element of this Deque. |
| E getLast(); | Retrieves but does not remove the last element of this Deque. |
| Iterator<E> iterator(); | Returns the iterator on this Deque. |
| **boolean** offer(E e); | Inserts element into queue represented by Deque without violating capacity restrictions and returns true upon success and false if no space available. |
| **boolean** offerFirst(E e); | Inserts specified element at the front of Deque unless it would violate capacity restrictions. |
| **boolean** offerLast(E e); | Inserts specified element at the end of Deque unless it would violate capacity restrictions. |
| E peek() | Retrieves but does not remove the head of the queue or null if Deque is empty. |
| E peekFirst() | Retrieves but does not remove the first element of the queue or null if Deque is empty. |
| E peekLast() | Retrieves but does not remove the last element of the queue or null if Deque is empty. |
| E poll() | Retrieves and remove the head of the queue or null if Deque is empty. |
| E pollFirst() | Retrieves and remove the first element of the queue or null if Deque is empty. |
| E pollLast() | Retrieves and remove the last element of the queue or null if Deque is empty. |
| E pop() | Pops the element from the stack as represented by the Deque. |
| **void** push(E e) | Pushed the element at the head of the Deque. |
| E remove() | Retrieves and removes the head of queue. |
| E remove(Object o) | Removes the Object in the Deque. It removes the first occurrence only. |
| E removeFirst() | Removes the first element in the Deque. |
| **boolean** removeFirstOccurrence(Object o) | Removes the first occurrence of the object in the Deque. |
| E removeLast() | Removes the last element in Deque. |
| **boolean** removeLastOccurrence(Object o) | Removes the last occurrence of the specified element from this Deque. |
| int size() | Returns number of elements in Deque. |