Deque interface in Java with Example

The java.util.Deque interface is a subtype of the [java.util.Queue](https://www.geeksforgeeks.org/queue-interface-java/) interface. The Deque is related to the double-ended queue that supports addition or removal of elements from either end of the data structure, it can be used as a [queue (first-in-first-out/FIFO)](https://www.geeksforgeeks.org/queue/) or as a [stack (last-in-first-out/LIFO)](https://www.geeksforgeeks.org/stack/). These are faster than Stack and LinkedList.  
This is the hierarchy of Deque interface in Java:  
  
Few important features of Deque are:

* It provides the support of resizable array and helps in restriction-free capacity, so to grow the array according to the usage.
* Array deques prohibit the use of Null elements and do not accept any such elements.
* Any concurrent access by multiple threads is not supported.
* In the absence of external synchronization, Deque is not thread-safe.

**Methods of Deque:**

1. [**add(element)**](https://www.geeksforgeeks.org/deque-add-method-in-java/)**:**Adds an element to the tail.
2. [**addFirst(element)**](https://www.geeksforgeeks.org/deque-addfirst-method-in-java-with-examples/)**:**Adds an element to the head.
3. [**addLast(element)**](https://www.geeksforgeeks.org/deque-addlast-method-in-java/)**:** Adds an element to the tail.
4. [**offer(element)**](https://www.geeksforgeeks.org/deque-offer-method-in-java/)**:**Adds an element to the tail and returns a boolean to explain if the insertion was successful.
5. [**offerFirst(element)**](https://www.geeksforgeeks.org/deque-offerfirst-method-in-java/)**:**Adds an element to the head and returns a boolean to explain if the insertion was successful.
6. [**offerLast(element)**](https://www.geeksforgeeks.org/deque-offerlast-method-in-java/)**:**Adds an element to the tail and returns a boolean to explain if the insertion was successful.
7. [**iterator()**](https://www.geeksforgeeks.org/deque-iterator-method-in-java/)**:** Returna an iterator for this deque.
8. [**descendingIterator()**](https://www.geeksforgeeks.org/deque-descendingiterator-method-in-java/)**:** Returns an iterator that has the reverse order for this deque.
9. **push(element):**Adds an element to the head.
10. **pop(element):** Removes an element from the head and returns it.
11. **removeFirst():**Removes the element at the head.
12. **removeLast():** Removes the element at the tail.
13. **poll():** Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque), or returns null if this deque is empty.
14. **pollFirst():** Retrieves and removes the first element of this deque, or returns null if this deque is empty.
15. **pollLast():** Retrieves and removes the last element of this deque, or returns null if this deque is empty.
16. **peek():** Retrieves, but does not remove, the head of the queue represented by this deque (in other words, the first element of this deque), or returns null if this deque is empty.
17. **peekFirst():** Retrieves, but does not remove, the first element of this deque, or returns null if this deque is empty.
18. **peekLast():** Retrieves, but does not remove, the last element of this deque, or returns null if this deque is empty.

Summarizing the methods, we get:

|  |
| --- |
| // Java program to demonstrate working of  // Deque in Java  import java.util.\*;    public class DequeExample  {      public static void main(String[] args)      {          Deque<String> deque = new LinkedList<String>();            // We can add elements to the queue in various ways          deque.add("Element 1 (Tail)"); // add to tail          deque.addFirst("Element 2 (Head)");          deque.addLast("Element 3 (Tail)");          deque.push("Element 4 (Head)"); //add to head          deque.offer("Element 5 (Tail)");          deque.offerFirst("Element 6 (Head)");          deque.offerLast("Element 7 (Tail)");            System.out.println(deque + "\n");            // Iterate through the queue elements.          System.out.println("Standard Iterator");          Iterator iterator = deque.iterator();          while (iterator.hasNext())              System.out.println("\t" + iterator.next());              // Reverse order iterator          Iterator reverse = deque.descendingIterator();          System.out.println("Reverse Iterator");          while (reverse.hasNext())              System.out.println("\t" + reverse.next());            // Peek returns the head, without deleting          // it from the deque          System.out.println("Peek " + deque.peek());          System.out.println("After peek: " + deque);            // Pop returns the head, and removes it from          // the deque          System.out.println("Pop " + deque.pop());          System.out.println("After pop: " + deque);            // We can check if a specific element exists          // in the deque          System.out.println("Contains element 3: " +                          deque.contains("Element 3 (Tail)"));            // We can remove the first / last element.          deque.removeFirst();          deque.removeLast();          System.out.println("Deque after removing " +                              "first and last: " + deque);        }  } |