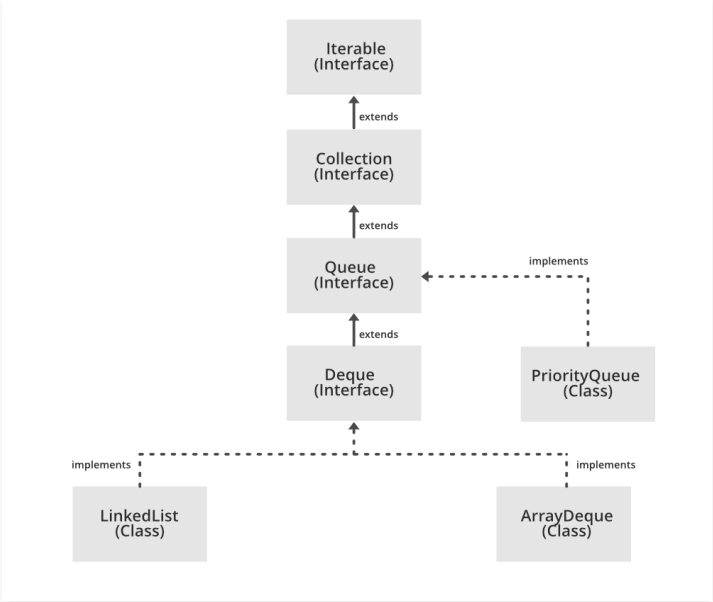
Deque interface in Java with Example

Difficulty Level : Easy Last Updated : 03 Sep, 2020

The **Deque** interface present in java.util package is a subtype of the queue 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 either be used as a queue(first-in-first-out/FIFO) or as a stack(last-in-first-out/LIFO). Deque is the acronym for double ended queue.



Declaration: The deque interface is declared as:

```
public interface Deque extends Queue
```

Creating Deque Objects

Since Deque is an interface, objects cannot be created of the type deque. We always need a class which extends this list in order to create an object. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the Deque. This type-safe queue can be defined as:

```
// Obj is the type of the object to be stored in Deque
Deque<Obj> deque = new ArrayDeque<Obj> ();
```

Example of a Deque:

```
// Java program to demonstrate the working
// of a 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

        // Add at the last
        deque.add("Element 1 (Tail)");

        // Add at the first
        deque.addFirst("Element 2 (Head)");

        // Add at the last
        deque.addLast("Element 3 (Tail)");

        // Add at the first
        deque.push("Element 4 (Head)");

        // Add at the last
        deque.offer("Element 5 (Tail)");

        // Add at the first
        deque.offerFirst("Element 6 (Head)");

        System.out.println(deque + "\n");

        // We can remove the first element
        // or the last element.
        deque.removeFirst();
        deque.removeLast();
        System.out.println("Deque after removing "
                           + "first and last: "
                           + deque);
    }
}
```

Output:

[Element 6 (Head), Element 4 (Head), Element 2 (Head), Element 1 (Tail), Element 3 (Tail), Element 5 (Tail)]

Deque after removing first and last: [Element 4 (Head), Element 2 (Head), Element 1 (Tail), Element 3 (Tail)]

Operations using the Deque Interface and the ArrayDeque class

Let's see how to perform a few frequently used operations on the deque using the ArrayDeque class.

1. Adding Elements: In order to add an element in a deque, we can use the add() method. The difference between a queue and a deque is that in deque, the addition is possible from any direction. Therefore, there are other two methods available named addFirst() and addLast() which are used to add the elements at either end.

```
// Java program to demonstrate the
// addition of elements in deque

import java.util.*;
public class ArrayDequeDemo {
    public static void main(String[] args)
    {
        // Initializing an deque
        Deque<String> dq
            = new ArrayDeque<String>();

        // add() method to insert
        dq.add("For");
        dq.addFirst("Geeks");
        dq.addLast("Geeks");

        System.out.println(dq);
    }
}
```

Output:

```
[Geeks, For, Geeks]
```

2. Removing Elements: In order to remove an element from a deque, there are various methods available. Since we can also remove from both the ends, the deque interface provides us with *removeFirst()*, *removeLast()* methods. Apart from that, this interface also provides us with the poll(), pop(), pollFirst(), pollLast() methods where pop() is used to remove and return the head of the deque. However, poll() is used because this offers the same functionality as pop() and doesnt return an exception when the deque is empty.

```
// Java program to demonstrate the
// removal of elements in deque

import java.util.*;
public class ArrayDequeDemo {
    public static void main(String[] args)
    {
        // Initializing an deque
        Deque<String> dq
            = new ArrayDeque<String>();

        // add() method to insert
        dq.add("For");
        dq.addFirst("Geeks");
        dq.addLast("Geeks");

        System.out.println(dq);

        System.out.println(dq.pop());

        System.out.println(dq.poll());

        System.out.println(dq.pollFirst());

        System.out.println(dq.pollLast());
    }
}
```

Output:

```
[Geeks, For, Geeks]
Geeks
For
Geeks
null
```

3. Iterating through the Deque: Since a deque can be iterated from both the directions, the iterator method of the deque interface provides us two ways to iterate. One from the first and the other from the back.

```
// Java program to demonstrate the
// iteration of elements in deque

import java.util.*;
public class ArrayDequeDemo {
    public static void main(String[] args)
    {
        // Initializing an deque
        Deque<String> dq
            = new ArrayDeque<String>();

        // add() method to insert
        dq.add("For");
        dq.addFirst("Geeks");
        dq.addLast("Geeks");
        dq.add("is so good");

        for (Iterator itr = dq.iterator();
             itr.hasNext();) {
            System.out.print(itr.next() + " ");
        }

        System.out.println();

        for (Iterator itr = dq.descendingIterator();
             itr.hasNext();) {
            System.out.print(itr.next() + " ");
        }
    }
}
```

Output:

```
Geeks For Geeks is so good
is so good Geeks For Geeks
```

The class which implements the Deque interface is ArrayDeque.

ArrayDeque**:** ArrayDeque class which is implemented in the collection framework provides us with a way to apply resizable-array. This is a special kind of array that grows and allows users to add or remove an element from both sides of the queue. Array deques

have no capacity restrictions and they grow as necessary to support usage. They are not thread-safe which means that in the absence of external synchronization, ArrayDeque does not support concurrent access by multiple threads. ArrayDeque class is likely to be faster than Stack when used as a stack. ArrayDeque class is likely to be faster than LinkedList when used as a queue. Let's see how to create a queue object using this class.

```
// Java program to demonstrate the
// creation of deque object using the
// ArrayDeque class in Java

import java.util.*;
public class ArrayDequeDemo {
    public static void main(String[] args)
    {
        // Initializing an deque
        Deque<Integer> de_que
            = new ArrayDeque<Integer>(10);

        // add() method to insert
        de_que.add(10);
        de_que.add(20);
        de_que.add(30);
        de_que.add(40);
        de_que.add(50);

        System.out.println(de_que);

        // clear() method
        de_que.clear();

        // addFirst() method to insert the
        // elements at the head
        de_que.addFirst(564);
        de_que.addFirst(291);

        // addLast() method to insert the
        // elements at the tail
        de_que.addLast(24);
        de_que.addLast(14);

        System.out.println(de_que);
    }
}
```

Output:

```
[10, 20, 30, 40, 50]
[291, 564, 24, 14]
```

Methods of Deque Interface

The following are the methods present in the deque interface:

Method	Description
add(element)	This method is used to add an element at the tail of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion.
addFirst(element)	This method is used to add an element at the head of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion.
addLast(element)	This method is used to add an element at the tail of the queue. If the Deque is capacity restricted and no space is left for insertion, it returns an IllegalStateException. The function returns true on successful insertion.
contains()	This method is used to check whether the queue contains the given object or not.
descendingIterator()	This method returns an iterator for the deque. The elements will be returned in order from last(tail) to first(head).
element()	This method is used to retrieve, but not remove, the head of the queue represented by this deque.
getFirst()	This method is used to retrieve, but not remove, the first element of this deque.
getLast()	This method is used to retrieve, but not remove, the last element of this deque.
iterator()	This method returns an iterator for the deque. The elements will be returned in order from first (head) to last (tail).
offer(element)	This method is used to add an element at the tail of the queue. This method is preferable to add() method since this

	method does not throws an exception when the capacity of the container is full since it returns false.
offerFirst(element)	This method is used to add an element at the head of the queue. This method is preferable to addFirst() method since this method does not throws an exception when the capacity of the container is full since it returns false.
offerLast(element)	This method is used to add an element at the tail of the queue. This method is preferable to add() method since this method does not throws an exception when the capacity of the container is full since it returns false.
peek()	This method is used to retrieve the element at the head of the deque but doesn't remove the element from the deque. This method returns null if the deque is empty.
peekFirst()	This method is used to retrieve the element at the head of the deque but doesn't remove the element from the deque. This method returns null if the deque is empty.
peekLast()	This method is used to retrieve the element at the tail of the deque but doesn't remove the element from the deque. This method returns null if the deque is empty.
poll()	This method is used to retrieve and remove the element at the head of the deque. This method returns null if the deque is empty.
pollFirst()	This method is used to retrieve and remove the element at the head of the deque. This method returns null if the deque is empty.
pollLast()	This method is used to retrieve and remove the element at the tail of the deque. This method returns null if the deque is empty.
pop()	This method is used to remove an element from the head and return it.
push(element)	This method is used to add an element at the head of the queue.
removeFirst()	This method is used to remove an element from the head of the queue.