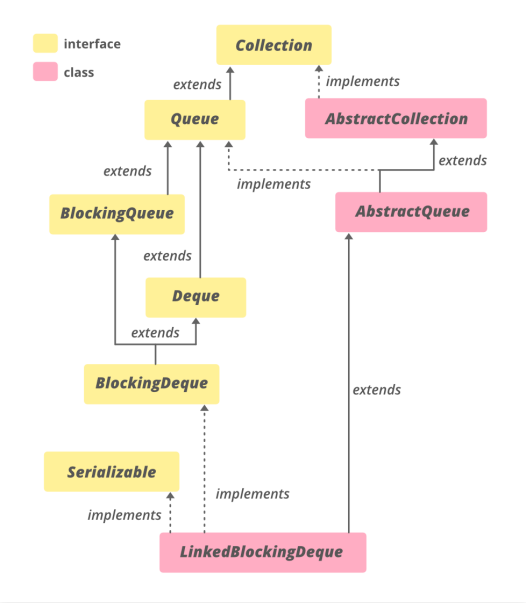
LinkedBlockingDeque in Java with Examples

Last Updated : 18 Jan, 2022

The **LinkedBlockingDeque** class in Java is a part of the Java Collection Framework. It was introduced in JDK 1.6 and it belongs to **java.util.concurrent** package. It is a Deque(Doubly Ended Queue) which blocks a thread if that thread tries to take elements out of it while the Deque is empty. It implements the BlockingDeque and provides an optionally-bounded functionality based on linked nodes. This optional boundedness is served by passing the required size in the constructor and helps in preventing memory wastage. When unspecified, the capacity is by default taken as **Integer.MAX_VALUE.** This class and its iterator implement all of the optional methods of the **Collection** and **Iterator** interfaces. The implementation provides by the LinkedBlockingDeque is **thread-safe**. All queuing methods in the class achieve their effects atomically using ReentrantLock internally.

The Hierarchy of LinkedBlockingDeque



It implements **Serializable**, **Iterable<E>**, **Collection<E>**, BlockingDeque<E>, BlockingQueue<E>, Deque<E>, Queue<E> interfaces and extends AbstractQueue<E> and AbstractCollection<E> classes.

Declaration:

```
public class LinkedBlockingDeque<E> extends AbstractQueue<E> implements Bloc
kingDeque<E>, Serializable
```

Here, **E** is the type of elements stored in the collection.

Constructors in Java LinkedBlockingDeque

In order to create an instance of LinkedBlockingDeque, we need to import it from **java.util.concurrent** package.

1. LinkedBlockingDeque(): This constructor is used to construct an empty deque. In this case the capacity is set to Integer.MAX_VALUE.

```
LinkedBlockingDeque<E> lbd = new LinkedBlockingDeque<E>();
```

2. LinkedBlockingDeque(int capacity): This constructor creates a LinkedBlockingDeque with the given (fixed) capacity.

```
LinkedBlockingDeque<E> lbd = new LinkedBlockingDeque<E>(int capacity);
```

3. LinkedBlockingDeque(Collection<E> c): This constructor is used to construct a deque with the elements of the Collection passed as the parameter.

```
LinkedBlockingDeque<E> lbd = new LinkedBlockingDeque<E>(Collection<? extend
s E> c);
```

Below is a sample program to illustrate LinkedBlockingDeque in Java:

Example 1:

```
// Java Program Demonstrate LinkedBlockingDeque

import java.util.concurrent.LinkedBlockingDeque;
import java.util.*;

public class LinkedBlockingDequeDemo {
    public static void main(String[] args)
        throws InterruptedException
    {

        // create object of LinkedBlockingDeque
        // using LinkedBlockingDeque() constructor
        LinkedBlockingDeque<Integer> LBD
            = new LinkedBlockingDeque<Integer>();

        // Add numbers to end of LinkedBlockingDeque
        LBD.add(7855642);
        LBD.add(35658786);
        LBD.add(5278367);
        LBD.add(74381793);

        // print Dequee
        System.out.println("Linked Blocking Deque1: "
                           + LBD);
        System.out.println("Size of Linked Blocking Deque1: "
                           + LBD.size());

        // create object of LinkedBlockingDeque
        // using LinkedBlockingDeque(int capacity) constructor
        LinkedBlockingDeque<Integer> LBD1
            = new LinkedBlockingDeque<Integer>(3);

        // Add numbers to end of LinkedBlockingDeque
        LBD1.add(7855642);
        LBD1.add(35658786);
        LBD1.add(5278367);

        try {
            // adding the 4th element
            // will throw exception for Deque full
            LBD1.add(74381793);
        }
        catch (Exception e) {
            System.out.println("Exception: " + e);
        }

        // print Dequee
        System.out.println("Linked Blocking Deque2: "
                           + LBD1);
        System.out.println("Size of Linked Blocking Deque2: "
```

```
                            + LBD1.size());

        // create object of LinkedBlockingDeque
        // using LinkedBlockingDeque(Collection c) constructor
        LinkedBlockingDeque<Integer> LBD2
            = new LinkedBlockingDeque<Integer>(LBD1);

        // print Dequee
        System.out.println("Linked Blocking Deque3: "
                            + LBD2);
    }
}
```

Output

```
Linked Blocking Deque1: [7855642, 35658786, 5278367, 74381793]
Size of Linked Blocking Deque1: 4
Exception: java.lang.IllegalStateException: Deque full
Linked Blocking Deque2: [7855642, 35658786, 5278367]
Size of Linked Blocking Deque2: 3
Linked Blocking Deque3: [7855642, 35658786, 5278367]
```

Example 2:

```
// Java code to illustrate methods of LinkedBlockingDeque

import java.util.concurrent.LinkedBlockingDeque;
import java.util.*;

public class LinkedBlockingDequeDemo {
    public static void main(String[] args)
        throws InterruptedException
    {

        // create object of LinkedBlockingDeque
        LinkedBlockingDeque<Integer> LBD
            = new LinkedBlockingDeque<Integer>();

        // Add numbers to end of LinkedBlockingDeque
        // using add() method
        LBD.add(7855642);
```

```
        LBD.add(35658786);
        LBD.add(5278367);
        LBD.add(74381793);

        // prints the Deque
        System.out.println("Linked Blocking Deque: "
                           + LBD);

        // prints the size of Deque after removal
        // using size() method
        System.out.println("Size of Linked Blocking Deque: "
                           + LBD.size());

        // removes the front element and prints it
        // using removeFirst() method
        System.out.println("First element: "
                           + LBD.removeFirst());

        // prints the Deque
        System.out.println("Linked Blocking Deque: "
                           + LBD);

        // prints the size of Deque after removal
        // using size() method
        System.out.println("Size of Linked Blocking Deque: "
                           + LBD.size());

        // Add numbers to end of LinkedBlockingDeque
        // using offer() method
        LBD.offer(20);

        // prints the Deque
        System.out.println("Linked Blocking Deque: "
                           + LBD);

        // prints the size of Deque after removal
        // using size() method
        System.out.println("Size of Linked Blocking Deque: "
                           + LBD.size());
    }
}
```

Output

```
Linked Blocking Deque: [7855642, 35658786, 5278367, 74381793]
Size of Linked Blocking Deque: 4
First element: 7855642
Linked Blocking Deque: [35658786, 5278367, 74381793]
Size of Linked Blocking Deque: 3
```

```
Linked Blocking Deque: [35658786, 5278367, 74381793, 20]
Size of Linked Blocking Deque: 4
```

Basic Operations

1. Adding Elements

There are various methods provided by LinkedBlockingDeque to add or insert elements at both ends. They are add(E e), addAll(Collection c), addFirst(E e), addLast(E e) etc.

```
// Java Program Demonstrate adding
// elements to LinkedBlockingDeque

import java.util.concurrent.LinkedBlockingDeque;
import java.util.*;

public class AddingElementsExample {
    public static void main(String[] args)
        throws IllegalStateException
    {

        // create object of LinkedBlockingDeque
        LinkedBlockingDeque<Integer> lbd
            = new LinkedBlockingDeque<Integer>();

        // Add number to end of LinkedBlockingDeque
        lbd.add(7855642);

        // Add integer at the head or front
        lbd.addFirst(35658786);

        // Add integer at the tail or end
        lbd.addLast(5278367);

        // print Deque
        System.out.println("Linked Blocking Deque: " + lbd);

          // Create object of ArrayList collection
        ArrayList<Integer> ArrLis
            = new ArrayList<Integer>();

        // Add number to ArrayList
        ArrLis.add(55);
        ArrLis.add(66);
        ArrLis.add(77);
        ArrLis.add(88);
```

```
        // Print ArrayList
        System.out.println("ArrayList: " + ArrLis);

        // Function addAll() adds all the elements of
        // ArrayList to Deque
        lbd.addAll(ArrLis);

        // Print deque
        System.out.println("Linked Blocking Deque: " + lbd);
    }
}
```

Output

```
Linked Blocking Deque: [35658786, 7855642, 5278367]
ArrayList: [55, 66, 77, 88]
Linked Blocking Deque: [35658786, 7855642, 5278367, 55, 66, 77, 88]
```

2. Removing Elements

There are various methods provided by LinkedBlockingDeque to delete or remove elements from either end. They are remove(), removeFirst(), removeLast() etc.

```
// Java Program Demonstrate removing
// elements of LinkedBlockingDeque

import java.util.concurrent.LinkedBlockingDeque;
import java.util.*;

public class RemovingElementsExample {
    public static void main(String[] args)
        throws InterruptedException
    {

        // create object of LinkedBlockingDeque
```

```
        LinkedBlockingDeque<Integer> lbd
            = new LinkedBlockingDeque<Integer>();

        // Add numbers to end of LinkedBlockingDeque
        lbd.add(7855642);
        lbd.add(35658786);
        lbd.add(5278367);
        lbd.add(74381793);
        lbd.add(12345566);

        // print Dequeue
        System.out.println("Linked Blocking Deque: " + lbd);

        // removes the front element
        lbd.remove();

        // print the modified deque
        System.out.println("Linked Blocking Deque: " + lbd);

        // removes the front element
        lbd.removeFirst();

        // print the modified deque
        System.out.println("Linked Blocking Deque: " + lbd);

        // removes the last element
        lbd.removeLast();

        // print the modified deque
        System.out.println("Linked Blocking Deque: " + lbd);
    }
}
```

Output

```
Linked Blocking Deque: [7855642, 35658786, 5278367, 74381793, 12345566]
Linked Blocking Deque: [35658786, 5278367, 74381793, 12345566]
Linked Blocking Deque: [5278367, 74381793, 12345566]
Linked Blocking Deque: [5278367, 74381793]
```

3. Iterating

The iterator() method of LinkedBlockingDeque returns an iterator over the elements in this deque in a proper sequence. The elements will be returned in order from first (head) to last (tail). The returned iterator is a “weakly consistent” iterator.

```
// Java Program Demonstrate iterating
// over LinkedBlockingDeque

import java.util.concurrent.LinkedBlockingDeque;
import java.util.*;

public class IteratingExample {
    public static void main(String[] args)

    {

        // create object of LinkedBlockingDeque
        LinkedBlockingDeque<Integer> LBD
            = new LinkedBlockingDeque<Integer>();

        // Add numbers to front of LinkedBlockingDeque
        LBD.addFirst(7855642);
        LBD.addFirst(35658786);
        LBD.addFirst(5278367);
        LBD.addFirst(74381793);

        // Call iterator() method of LinkedBlockingDeque
        Iterator iteratorVals = LBD.iterator();

        // Print elements of iterator
        // created from PriorityBlockingQueue
        System.out.println("The iterator values"
            + " of LinkedBlockingDeque are:");

        // prints the elements using an iterator
        while (iteratorVals.hasNext()) {
            System.out.println(iteratorVals.next());
        }
    }
}
```

Output

```
The iterator values of LinkedBlockingDeque are:
74381793
5278367
35658786
7855642
```

Methods of LinkedBlockingDeque

METHOD	DESCRIPTION
add(E e)	Inserts the specified element at the end of this deque unless it would violate capacity restrictions.
addAll(Collection<? extends E> c)	Appends all of the elements in the specified collection to the end of this deque, in the order that they are returned by the specified collection's iterator.
addFirst(E e)	Inserts the specified element at the front of this deque if it is possible to do so immediately without violating capacity restrictions, throwing an IllegalStateException if no space is currently available.
addLast(E e)	Inserts the specified element at the end of this deque if it is possible to do so immediately without violating capacity restrictions, throwing an IllegalStateException if no space is currently available.
clear()	Atomically removes all of the elements from this deque.
contains(Object o)	Returns true if this deque contains the specified element.
descendingIterator()	Returns an iterator over the elements in this deque in reverse sequential order.
drainTo (Collection<? super E> c)	Removes all available elements from this queue and adds them to the given collection.

METHOD	DESCRIPTION
drainTo (Collection<? super E> c, int maxElements)	Removes at most the given number of available elements from this queue and adds them to the given collection.
element()	Retrieves, but does not remove, the head of the queue represented by this deque.
forEach (Consumer<? super E> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
getFirst()	Retrieves, but does not remove, the first element of this deque.
getLast()	Retrieves, but does not remove, the last element of this deque.
iterator()	Returns an iterator over the elements in this deque in the proper sequence.
offer(E e)	Inserts the specified element into the queue represented by this deque (in other words, at the tail of this deque) if it is possible to do so immediately without violating capacity restrictions, returning true upon success and false if no space is currently available.
offer(E e, long timeout, TimeUnit unit)	Inserts the specified element into the queue represented by this deque (in other words, at the tail of this deque), waiting up to the specified wait time if necessary for space to become available.
offerFirst(E e)	Inserts the specified element at the front of this deque if it is possible to do so immediately without violating capacity restrictions, returning true upon success and false if no space is currently available.
offerFirst(E e, long timeout, TimeUnit unit)	Inserts the specified element at the front of this deque, waiting up to the specified wait time if necessary for space to become available.
offerLast(E e)	Inserts the specified element at the end of this deque if it is possible to do so immediately without violating capacity restrictions, returning true upon success and false if no space is currently available.

METHOD	DESCRIPTION
offerLast(E e, long timeout, TimeUnit unit)	Inserts the specified element at the end of this deque, waiting up to the specified wait time if necessary for space to become available.
pop()	Pops an element from the stack represented by this deque.
push(E e)	Pushes an element onto the stack represented by this deque (in other words, at the head of this deque) if it is possible to do so immediately without violating capacity restrictions, throwing an IllegalStateException if no space is currently available.
put(E e)	Inserts the specified element into the queue represented by this deque (in other words, at the tail of this deque), waiting if necessary for space to become available.
putFirst(E e)	Inserts the specified element at the front of this deque, waiting if necessary for space to become available.
putLast(E e)	Inserts the specified element at the end of this deque, waiting if necessary for space to become available.
remainingCapacity()	Returns the number of additional elements that this deque can ideally (in the absence of memory or resource constraints) accept without blocking.
remove()	Retrieves and removes the head of the queue represented by this deque.
remove(Object o)	Removes the first occurrence of the specified element from this deque.
removeAll (Collection<?> c)	Removes all of this collection’s elements that are also contained in the specified collection (optional operation).
removeFirst()	Retrieves and removes the first element of this deque.
removeIf (Predicate<? super E> filter)	Removes all of the elements of this collection that satisfy the given predicate.
removeLast()	Retrieves and removes the last element of this deque.

METHOD	DESCRIPTION
retainAll (Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection (optional operation).
size()	Returns the number of elements in this deque.
spliterator()	Returns a Spliterator over the elements in this deque.
toArray()	Returns an array containing all of the elements in this deque, in proper sequence (from first to last element).
toArray(T[] a)	Returns an array containing all of the elements in this deque, in proper sequence; the runtime type of the returned array is that of the specified array.

Methods declared in class java.util.AbstractCollection

METHOD	DESCRIPTION
containsAll (Collection<?> c)	Returns true if this collection contains all of the elements in the specified collection.
isEmpty()	Returns true if this collection contains no elements.
toString()	Returns a string representation of this collection.

Methods declared in interface java.util.concurrent.BlockingDeque

METHOD	DESCRIPTION
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.
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.

METHOD	DESCRIPTION
poll(long timeout, TimeUnit unit)	Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque), waiting up to the specified wait time if necessary for an element to become available.
pollFirst(long timeout, TimeUnit unit)	Retrieves and removes the first element of this deque, waiting up to the specified wait time if necessary for an element to become available.
pollLast(long timeout, TimeUnit unit)	Retrieves and removes the last element of this deque, waiting up to the specified wait time if necessary for an element to become available.
removeFirstOccurrence (Object o)	Removes the first occurrence of the specified element from this deque.
removeLastOccurrence (Object o)	Removes the last occurrence of the specified element from this deque.
take()	Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque), waiting if necessary until an element becomes available.
takeFirst()	Retrieves and removes the first element of this deque, waiting if necessary until an element becomes available.
takeLast()	Retrieves and removes the last element of this deque, waiting if necessary until an element becomes available.

Methods declared in interface java.util.Collection

METHOD	DESCRIPTION
containsAll (Collection<?> c)	Returns true if this collection contains all of the elements in the specified collection.
equals(Object o)	Compares the specified object with this collection for equality.
hashCode()	Returns the hash code value for this collection.

METHOD	DESCRIPTION
isEmpty()	Returns true if this collection contains no elements.
parallelStream()	Returns a possibly parallel Stream with this collection as its source.
stream()	Returns a sequential Stream with this collection as its source.
toArray (IntFunction<T[]> generator)	Returns an array containing all of the elements in this collection, using the provided generator function to allocate the returned array.

Methods declared in interface java.util.Deque

METHOD	DESCRIPTION
peekFirst()	Retrieves, but does not remove, the first element of this deque, or returns null if this deque is empty.
peekLast()	Retrieves, but does not remove, the last element of this deque, or returns null if this deque is empty.
pollFirst()	Retrieves and removes the first element of this deque, or returns null if this deque is empty.
pollLast()	Retrieves and removes the last element of this deque, or returns null if this deque is empty.

Reference: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/LinkedBlockingDeque.html