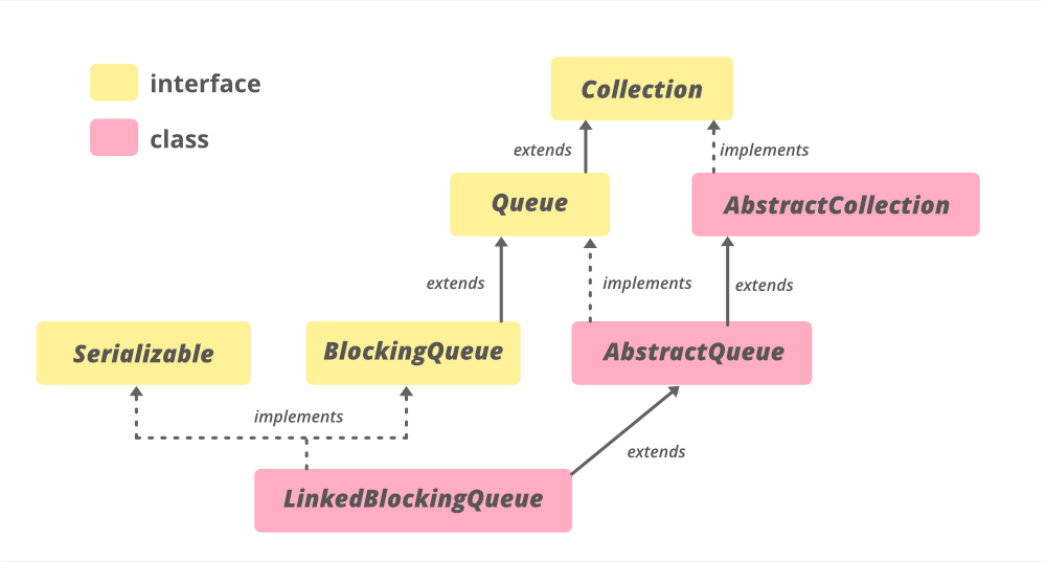
LinkedBlockingQueue Class in Java

Last Updated : 21 Oct, 2020

The **LinkedBlockingQueue** is an *optionally-bounded* blocking queue based on linked nodes. It means that the LinkedBlockingQueue can be bounded, if its capacity is given, else the LinkedBlockingQueue will be unbounded. The capacity can be given as a parameter to the constructor of LinkedBlockingQueue. This queue orders elements **FIFO (first-in-first-out)**. It means that the head of this queue is the oldest element of the elements present in this queue. The tail of this queue is the newest element of the elements of this queue. The newly inserted elements are always inserted at the tail of the queue, and the queue retrieval operations obtain elements at the head of the queue. Linked queues typically have higher throughput than array-based queues but less predictable performance in most concurrent applications.

The capacity, if unspecified, is equal to **Integer.MAX_VALUE**. Linked nodes are dynamically created upon each insertion, till the capacity of the queue is not filled. This class and its iterator implement all of the optional methods of the Collection and Iterator interfaces. It is a member of the Java Collections Framework.

The Hierarchy of LinkedBlockingQueue



LinkedBlockingQueue<E> extends AbstractQueue<E> and implements **Serializable**, **Iterable<E>**, **Collection<E>**, BlockingQueue<E>, Queue<E> interfaces.

Declaration:

*public class **LinkedBlockingQueue<E>** extends AbstractQueue<E> implements Blocking Queue<E>, Serializable*

E – type of elements held in this collection.

Constructors of LinkedBlockingQueue:

To construct a LinkedBlockingQueue, we need to import it from **java.util.concurrent.LinkedBlockingQueue**. Here, **capacity** is the size of the linked blocking queue.

1. LinkedBlockingQueue() : Creates a LinkedBlockingQueue with a capacity of Integer.MAX_VALUE.

LinkedBlockingQueue<E> lbq = new LinkedBlockingQueue<E>();

Example:

```
// Java program to demonstrate
// LinkedBlockingQueue() constructor

import java.util.concurrent.LinkedBlockingQueue;

public class LinkedBlockingQueueDemo {

    public static void main(String[] args)
    {

        // create object of LinkedBlockingQueue
        // using LinkedBlockingQueue() constructor
        LinkedBlockingQueue<Integer> lbq
            = new LinkedBlockingQueue<Integer>();

        // add  numbers
        lbq.add(1);
        lbq.add(2);
        lbq.add(3);
        lbq.add(4);
        lbq.add(5);

        // print queue
        System.out.println("LinkedBlockingQueue:" + lbq);
    }
}
```

Output

```
LinkedBlockingQueue:[1, 2, 3, 4, 5]
```

2. LinkedBlockingQueue(int capacity): Creates a LinkedBlockingQueue with the given (fixed) capacity.

LinkedBlockingQueue<E> lbq = new LinkedBlockingQueue(int capacity);

Example:

```
// Java program to demonstrate
// LinkedBlockingQueue(int initialCapacity) constructor

import java.util.concurrent.LinkedBlockingQueue;

public class GFG {

    public static void main(String[] args)
    {
        // define capacity of LinkedBlockingQueue
        int capacity = 15;

        // create object of LinkedBlockingQueue
        // using LinkedBlockingQueue(int initialCapacity)
        // constructor
        LinkedBlockingQueue<Integer> lbq
            = new LinkedBlockingQueue<Integer>(capacity);

        // add  numbers
        lbq.add(1);
        lbq.add(2);
        lbq.add(3);

        // print queue
        System.out.println("LinkedBlockingQueue:" + lbq);
    }
}
```

Output

```
LinkedBlockingQueue:[1, 2, 3]
```

3. LinkedBlockingQueue(Collection<? extends E> c): Creates a LinkedBlockingQueue with a capacity of Integer.MAX_VALUE, initially containing the elements of the given collection, added in traversal order of the collection's iterator.

LinkedBlockingQueue<E> lbq = new LinkedBlockingQueue(Collection<? extends E> c);

Example:

```
// Java program to demonstrate
// LinkedBlockingQueue(Collection c) constructor

import java.util.concurrent.LinkedBlockingQueue;
import java.util.*;

public class GFG {

    public static void main(String[] args)
    {

        // Creating a Collection
        Vector<Integer> v = new Vector<Integer>();
        v.addElement(1);
        v.addElement(2);
        v.addElement(3);
        v.addElement(4);
        v.addElement(5);

        // create object of LinkedBlockingQueue
        // using LinkedBlockingQueue(Collection c)
        // constructor
        LinkedBlockingQueue<Integer> lbq
            = new LinkedBlockingQueue<Integer>(v);

        // print queue
        System.out.println("LinkedBlockingQueue:" + lbq);
    }
}
```

Output

```
LinkedBlockingQueue:[1, 2, 3, 4, 5]
```

Basic Operations

1. Adding Elements

The add(E e) method of LinkedBlockingQueue inserts element passed as a parameter to method at the tail of this LinkedBlockingQueue, if the queue is not full. If the queue is full, then this method will wait for space to become available and after space is available, it inserts the element to LinkedBlockingQueue.

```
// Java Program to Demonstrate adding
// elements to the LinkedBlockingQueue

import java.util.concurrent.LinkedBlockingQueue;

public class AddingElementsExample {

    public static void main(String[] args)
    {
        // define capacity of LinkedBlockingQueue
        int capacity = 15;

        // create object of LinkedBlockingQueue
        LinkedBlockingQueue<Integer> lbq
            = new LinkedBlockingQueue<Integer>(capacity);

        // add  numbers
        lbq.add(1);
        lbq.add(2);
        lbq.add(3);

        // print queue
        System.out.println("LinkedBlockingQueue:" + lbq);
    }
}
```

Output:

```
LinkedBlockingQueue:[1, 2, 3]
```

2. Removing Elements

The remove(Object obj) method of LinkedBlockingQueue removes only one instance of the given Object, passed as a parameter, from this LinkedBlockingQueue if it is present. It removes an element e such that obj.equals(e) and if this queue contains one or more instances of element e. This method returns true if this queue contained the element which is now removed from LinkedBlockingQueue.

```
// Java Program to Demonstrate removing
// elements from the LinkedBlockingQueue

import java.util.concurrent.LinkedBlockingQueue;

public class RemovingElementsExample {

    public static void main(String[] args)
    {
        // define capacity of LinkedBlockingQueue
        int capacity = 15;

        // create object of LinkedBlockingQueue
        LinkedBlockingQueue<Integer> lbq
            = new LinkedBlockingQueue<Integer>(capacity);

        // add  numbers
        lbq.add(1);
        lbq.add(2);
        lbq.add(3);

        // print queue
        System.out.println("LinkedBlockingQueue:" + lbq);

        // remove all the elements
        lbq.clear();

        // print queue
        System.out.println("LinkedBlockingQueue:" + lbq);
    }
}
```

Output:

```
LinkedBlockingQueue:[1, 2, 3]
LinkedBlockingQueue:[]
```

3. Iterating

The iterator() method of LinkedBlockingQueue returns an iterator of the same elements, as this LinkedBlockingQueue, in a proper sequence. The elements returned from this method contains all the elements in order from **first(head)** to **last(tail)** of LinkedBlockingQueue. The returned iterator is weakly consistent.

```
// Java Program Demonstrate iterating
// over LinkedBlockingQueue

import java.util.concurrent.LinkedBlockingQueue;
import java.util.Iterator;

public class IteratingExample {

    public static void main(String[] args)
    {
        // define capacity of LinkedBlockingQueue
        int capacityOfQueue = 7;

        // create object of LinkedBlockingQueue
        LinkedBlockingQueue<String> linkedQueue
            = new LinkedBlockingQueue<String>(capacityOfQueue);

        // Add element to LinkedBlockingQueue
        linkedQueue.add("John");
        linkedQueue.add("Tom");
        linkedQueue.add("Clark");
        linkedQueue.add("Kat");

        // create Iterator of linkedQueue using iterator() method
        Iterator<String> listOfNames = linkedQueue.iterator();

        // print result
        System.out.println("list of names:");
        while (listOfNames.hasNext())
            System.out.println(listOfNames.next());
    }
}
```

Output

```
list of names:
John
```

```
Tom
Clark
Kat
```

4. Accessing Elements

The peek() method of LinkedBlockingQueue returns the head of the LinkedBlockingQueue. It retrieves the value of the head of LinkedBlockingQueue but does not remove it. If the LinkedBlockingQueue is empty then this method returns null.

```
// Java Program Demonstrate accessing
// elements of LinkedBlockingQueue

import java.util.concurrent.LinkedBlockingQueue;
public class AccessingElementsExample {

    public static void main(String[] args)
    {
        // define capacity of LinkedBlockingQueue
        int capacityOfQueue = 7;

        // create object of LinkedBlockingQueue
        LinkedBlockingQueue<String> linkedQueue
            = new LinkedBlockingQueue<String>(capacityOfQueue);

        // Add element to LinkedBlockingQueue
        linkedQueue.add("John");
        linkedQueue.add("Tom");
        linkedQueue.add("Clark");
        linkedQueue.add("Kat");

        // find head of linkedQueue using peek() method
        String head = linkedQueue.peek();

        // print result
        System.out.println("Queue is " + linkedQueue);

        // print head of queue
        System.out.println("Head of Queue is " + head);

        // removing one element
        linkedQueue.remove();

        // again get head of queue
        head = linkedQueue.peek();

        // print result
        System.out.println("\nRemoving one element from Queue\n");
        System.out.println("Queue is " + linkedQueue);

        // print head of queue
```

```
            System.out.println("Head of Queue is " + head);
        }
    }
```

Output

```
Queue is [John, Tom, Clark, Kat]
Head of Queue is John

Removing one element from Queue

Queue is [Tom, Clark, Kat]
Head of Queue is Tom
```

Methods of LinkedBlockingQueue

METHOD	DESCRIPTION
clear()	Atomically removes all of the elements from this queue.
contains(Object o)	Returns true if this queue contains the specified element.
drainTo (Collection<? super E> c)	Removes all available elements from this queue and adds them to the given collection.
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.
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.
iterator()	Returns an iterator over the elements in this queue in the proper sequence.
offer(E e)	Inserts the specified element at the tail of this queue if it is possible to do so immediately without exceeding the queue's capacity, returning true upon success and false if this queue is full.

METHOD	DESCRIPTION
offer(E e, long timeout, TimeUnit unit)	Inserts the specified element at the tail of this queue, waiting if necessary up to the specified wait time for space to become available.
put(E e)	Inserts the specified element at the tail of this queue, waiting if necessary for space to become available.
remainingCapacity()	Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking.
remove(Object o)	Removes a single instance of the specified element from this queue, if it is present.
removeAll (Collection<?> c)	Removes all of this collection's elements that are also contained in the specified collection (optional operation).
removeIf (Predicate<? super E> filter)	Removes all of the elements of this collection that satisfy the given predicate.
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 queue.
spliterator()	Returns a Spliterator over the elements in this queue.
toArray()	Returns an array containing all of the elements in this queue, in proper sequence.
toArray(T[] a)	Returns an array containing all of the elements in this queue, 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.

METHOD	DESCRIPTION
toString()	Returns a string representation of this collection.

Methods declared in class java.util.AbstractQueue

METHOD	DESCRIPTION
add(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success, and throwing an IllegalStateException if no space is currently available.
addAll (Collection<? extends E> c)	Adds all of the elements in the specified collection to this queue.
element()	Retrieves, but does not remove, the head of this queue.
remove()	Retrieves and removes the head of this queue.

Methods declared in interface java.util.concurrent.BlockingQueue

METHOD	DESCRIPTION
add(E e)	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success, and throwing an IllegalStateException if no space is currently available.
poll(long timeout, TimeUnit unit)	Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available.
take()	Retrieves and removes the head of this queue, waiting if necessary until an element becomes available.

Methods declared in interface java.util.Collection

METHOD	DESCRIPTION

METHOD	DESCRIPTION
addAll(Collection<? extends E> c)	Adds all of the elements in the specified collection to this collection (optional operation).
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.
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.Queue

METHOD	DESCRIPTION
element()	Retrieves, but does not remove, the head of this queue.
peek()	Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
poll()	Retrieves and removes the head of this queue, or returns null if this queue is empty.
remove()	Retrieves and removes the head of this queue.

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