LinkedTransferQueue in Java with Examples

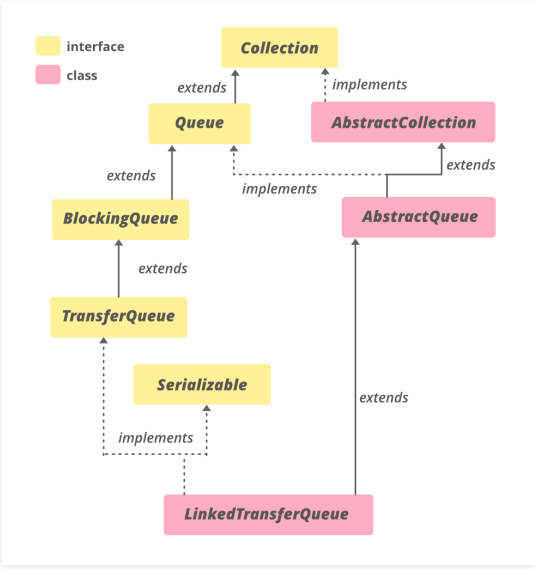
Last Updated : 06 Nov, 2020

The **LinkedTransferQueue** class in Java is a part of the Java Collection Framework. It was introduced in JDK 1.7 and it belongs to **java.util.concurrent** package. It implements the **TransferQueue** and provides an unbounded functionality based on linked nodes. The elements in the LinkedTransferQueue are ordered in FIFO order, with the head pointing to the element that has been on the Queue for the longest time and the tail pointing to the element that has been on the queue for the shortest time. Because of its asynchronous nature, size() traverses the entire collection, so it is not an O(1) time operation. It may also give inaccurate size if this collection is modified during the traversal. Bulk operations like addAll, removeAll, retainAll, containsAll, equals, and toArray are not guaranteed to be performed atomically. For example, an iterator operating concurrently with an addAll operation might observe only some of the added elements.

LinkedTransferQueue has used message-passing applications. There are two aspects in which the message will be passed from Producer thread to Consumer thread.

1. put(E e): This method is used if the producer wants to enqueue elements without waiting for a consumer. However, it waits till the space becomes available if the queue is full.
2. transfer(E e): This method is generally used to transfer an element to a thread that is waiting to receive it, if there is no thread waiting then it will wait till a thread comes to waiting for state as soon as the waiting thread arrives element will be transferred into it.

The Hierarchy of LinkedTransferQueue



It

implements **Serializable**, **Iterable<E>**, **Collection<E>**, BlockingQueue<E>, **TransferQueue<E>**,

ueue<E> interfaces and extends AbstractQueue<E> and AbstractCollection<E> classes.

Declaration:

public class LinkedTransferQueue<E> extends AbstractQueue<E> implements TransferQueue<E>, Serializable

Here, **E** is the type of elements maintained by this collection.

Constructors of LinkedTransferQueue

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

1. LinkedTransferQueue(): This constructor is used to construct an empty queue.

LinkedTransferQueue<E> ltq = new LinkedTransferQueue<E>();

2. LinkedTransferQueue(Collection<E> c): This constructor is used to construct a queue with the elements of the Collection passed as the parameter.

LinkedTransferQueue<E> ltq = new LinkedTransferQueue<E>(Collection<E> c);

Example 1: Sample program to illustrate LinkedTransferQueue in Java

```
// Java Program Demonstrate LinkedTransferQueue

import java.util.concurrent.LinkedTransferQueue;
import java.util.*;

public class LinkedTransferQueueDemo {
    public static void main(String[] args)
        throws InterruptedException
    {
        // create object of LinkedTransferQueue
        // using LinkedTransferQueue() constructor
```

```
        LinkedTransferQueue<Integer> LTQ
            = new LinkedTransferQueue<Integer>();

        // Add numbers to end of LinkedTransferQueue
        LTQ.add(7855642);
        LTQ.add(35658786);
        LTQ.add(5278367);
        LTQ.add(74381793);

        // print Queue
        System.out.println("Linked Transfer Queue1: " + LTQ);

        // create object of LinkedTransferQueue
        // using LinkedTransferQueue(Collection c)
        // constructor
        LinkedTransferQueue<Integer> LTQ2
            = new LinkedTransferQueue<Integer>(LTQ);

        // print Queue
        System.out.println("Linked Transfer Queue2: " + LTQ2);
    }
}
```

Output

```
Linked Transfer Queue1: [7855642, 35658786, 5278367, 74381793]
Linked Transfer Queue2: [7855642, 35658786, 5278367, 74381793]
```

Example 2:

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

import java.util.concurrent.LinkedTransferQueue;
import java.util.*;

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

        // create object of LinkedTransferQueue
        LinkedTransferQueue<Integer> LTQ
            = new LinkedTransferQueue<Integer>();

        // Add numbers to end of LinkedTransferQueue
```

```
        // using add() method
        LTQ.add(7855642);
        LTQ.add(35658786);
        LTQ.add(5278367);
        LTQ.add(74381793);

        // prints the Queue
        System.out.println("Linked Transfer Queue: " + LTQ);

        // prints the size of Queue after removal
        // using size() method
        System.out.println("Size of Linked Transfer Queue: "
                           + LTQ.size());

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

        // prints the Queue
        System.out.println("Linked Transfer Queue: " + LTQ);

        // prints the size of Queue after removal
        // using size() method
        System.out.println("Size of Linked Transfer Queue: "
                           + LTQ.size());

        // Add numbers to end of LinkedTransferQueue
        // using offer() method
        LTQ.offer(20);

        // prints the Queue
        System.out.println("Linked Transfer Queue: " + LTQ);

        // prints the size of Queue after removal
        // using size() method
        System.out.println("Size of Linked Transfer Queue: "
                           + LTQ.size());
    }
}
```

Output

```
Linked Transfer Queue: [7855642, 35658786, 5278367, 74381793]
Size of Linked Transfer Queue: 4
First element: 7855642
Linked Transfer Queue: [35658786, 5278367, 74381793]
Size of Linked Transfer Queue: 3
Linked Transfer Queue: [35658786, 5278367, 74381793, 20]
Size of Linked Transfer Queue: 4
```

Basic Operations

1. Adding Elements

There are various methods provided by LinkedTransferQueue to add or insert elements. They are add(E e), put(E e), offer(E e), transfer(E e). add, put, and offer methods do not care about other threads accessing the queue or not while a transfer() waits for one or more recipient threads.

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

import java.util.concurrent.*;

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

        // Initializing the queue
        LinkedTransferQueue<Integer> queue
            = new LinkedTransferQueue<Integer>();

        // Adding elements to this queue
        for (int i = 10; i <= 14; i++)
            queue.add(i);

        // Add the element using offer() method
        System.out.println("adding 15 "
            + queue.offer(15, 5, TimeUnit.SECONDS));

        // Adding elements to this queue
        for (int i = 16; i <= 20; i++)
            queue.put(i);

        // Printing the elements of the queue
        System.out.println(
            "The elements in the queue are:");
        for (Integer i : queue)
            System.out.print(i + " ");

        System.out.println();

        // create another queue to demonstrate transfer
        // method
        LinkedTransferQueue<String> g
            = new LinkedTransferQueue<String>();

        new Thread(new Runnable() {
            public void run()
            {
                try {
                    System.out.println("Transferring"
                                       + " an element");
```

```
                    // Transfer a String element
                    // using transfer() method
                    g.transfer("is a computer"
                               + " science portal.");
                    System.out.println(
                        "Element "
                        + "transfer is complete");
                }
                catch (InterruptedException e1) {
                    System.out.println(e1);
                }
                catch (NullPointerException e2) {
                    System.out.println(e2);
                }
            }
        })
            .start();

        try {

            // Get the transferred element
            System.out.println("Geeks for Geeks "
                               + g.take());
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Output

```
adding 15 true
The elements in the queue are:
10 11 12 13 14 15 16 17 18 19 20
Transferring an element
Geeks for Geeks is a computer science portal.
Element transfer is complete
```

2. Removing Elements

The remove() method provided by LinkedTransferQueue is used to remove an element if it is present in this queue.

```
// Java Program Demonstrate removing
```

```
// elements of LinkedTransferQueue

import java.util.concurrent.LinkedTransferQueue;

class RemoveElementsExample {
    public static void main(String[] args)
    {
        // Initializing the queue
        LinkedTransferQueue<Integer> queue
            = new LinkedTransferQueue<Integer>();

        // Adding elements to this queue
        for (int i = 1; i <= 5; i++)
            queue.add(i);

        // Printing the elements of the queue
        System.out.println(
            "The elements in the queue are:");
        for (Integer i : queue)
            System.out.print(i + " ");

        // remove() method will remove the specified
        // element from the queue
        queue.remove(1);
        queue.remove(5);

        // Printing the elements of the queue
        System.out.println("\nRemaining elements in queue : ");
        for (Integer i : queue)
            System.out.print(i + " ");
    }
}
```

Output

```
The elements in the queue are:
1 2 3 4 5
Remaining elements in queue :
2 3 4
```

3. Iterating

The iterator() method of LinkedTransferQueue is used to return an iterator over the elements in this queue in the proper sequence.

```
// Java Program Demonstrate iterating
// over LinkedTransferQueue

import java.util.Iterator;
import java.util.concurrent.LinkedTransferQueue;

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

        // Initializing the queue
        LinkedTransferQueue<String> queue
            = new LinkedTransferQueue<String>();

        // Adding elements to this queue
        queue.add("Gfg");
        queue.add("is");
        queue.add("fun!!");

        // Returns an iterator over the elements
        Iterator<String> iterator = queue.iterator();

        // Printing the elements of the queue
        while (iterator.hasNext())
            System.out.print(iterator.next() + " ");
    }
}
```

Output

```
Gfg is fun!!
```

Methods of LinkedTransferQueue

METHOD	DESCRIPTION
add(E e)	Inserts the specified element at the tail of 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.

METHOD	DESCRIPTION
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.
isEmpty()	Returns true if this queue contains no elements.
iterator()	Returns an iterator over the elements in this queue in proper sequence.
offer(E e)	Inserts the specified element at the tail of this queue.
offer(E e, long timeout, TimeUnit unit)	Inserts the specified element at the tail of this queue.
put(E e)	Inserts the specified element at the tail of this queue.
remainingCapacity()	Always returns Integer.MAX_VALUE because a LinkedTransferQueue is not capacity constrained.
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.
transfer(E e)	Transfers the element to a consumer, waiting if necessary to do so.
tryTransfer(E e)	Transfers the element to a waiting consumer immediately, if possible.

METHOD	DESCRIPTION
tryTransfer(E e, long timeout, TimeUnit unit)	Transfers the element to a consumer if it is possible to do so before the timeout elapses.

Methods declared in class java.util.AbstractQueue

METHOD	DESCRIPTION
addAll(Collection<? extends E> c)	Adds all of the elements in the specified collection to this queue.
clear()	Removes all of the elements from 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 class java.util.AbstractCollection

METHOD	DESCRIPTION
containsAll (Collection<?> c)	Returns true if this collection contains all of the elements in the specified collection.
toString()	Returns a string representation of this collection.

Methods declared in interface java.util.concurrent.BlockingQueue

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

Methods declared in interface java.util.concurrent.TransferQueue

METHOD	DESCRIPTION
getWaitingConsumerCount()	Returns an estimate of the number of consumers waiting to receive elements via BlockingQueue.take() or timed poll.

METHOD	DESCRIPTION
hasWaitingConsumer()	Returns true if there is at least one consumer waiting to receive an element via BlockingQueue.take() or timed poll.

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