Java TransferQueue – Java LinkedTransferQueue class

Last Updated: December 26, 2020 By: Lokesh Gupta Java Collections Java Collections, Java Queue

Java TransferQueue is a concurrent blocking queue implementation in which producers may wait for receipt of messages by consumers. **LinkedTransferQueue** class is an implementation of `TransferQueue` in Java.

TransferQueue may be useful for example in message passing applications in which producers sometimes (using method **transfer()**) await receipt of elements by consumers invoking take or poll, while at other times enqueue elements (via method **put()**) without waiting for receipt.

> When a producer reaches to TransferQueue to transfer a message and there are consumers waiting to take message, then producer directly transfers the message to consumer.
> If there is no consumer waiting, then producer will not directly put the message and returned, rather it will wait for any consumer to be available to consume the message.

1. LinkedTransferQueue Features

Let's note down few important points on the LinkedTransferQueue in Java.

- LinkedTransferQueue is an **unbounded** queue on linked nodes.
- This queue orders elements FIFO (first-in-first-out) with respect to any given producer.
- Elements are inserted at the tail, and retrieved from the head of the queue.
- It supplies **blocking insertion and retrieval operations**.
- It does not allow NULL objects.
- LinkedTransferQueue is **thread safe**.
- The size() method is NOT a constant-time operation because of the asynchronous nature, so may report inaccurate results if this collection is modified during traversal.
- The bulk operations 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 view only some of the added elements.

2. Java LinkedTransferQueue Example

2.1. LinkedTransferQueue example

A very simple example to add and poll messages from LinkedTransferQueue.

```
LinkedTransferQueueExample.java

LinkedTransferQueue<Integer> linkedTransferQueue = new LinkedTransferQueue<>();

linkedTransferQueue.put(1);

System.out.println("Added Message = 1");

Integer message = linkedTransferQueue.poll();

System.out.println("Recieved Message = " + message);
```

Program Output.

```
Console

Added Message = 1
Recieved Message = 1
```

2.2. LinkedTransferQueue blocking insertion and retrieval example

Java example to put and take elements from LinkedTransferQueue using blocking insertions and retrieval.

- Producer thread will wait until there is consumer ready to take the item from the queue.
- Consumer thread will wait if queue is empty. As soon as, there is a single element in queue, it take out the element. Only after consumer has taken the message, producer can another message.

```
LinkedTransferQueueExample.java

import java.util.Random;
import java.util.concurrent.LinkedTransferQueue;
import java.util.concurrent.TimeUnit;

public class LinkedTransferQueueExample
{
    public static void main(String[] args) throws InterruptedException
    {
        LinkedTransferQueue<Integer> linkedTransferQueue = new LinkedTransferQueue<>();

        new Thread(() ->
        {
            Random random = new Random(1);
            try
            {
                while (true)
                {
                    System.out.println("Producer is waiting to transfer message...");

                    Integer message = random.nextInt();
                    boolean added = linkedTransferQueue.tryTransfer(message);
                    if(added) {
                        System.out.println("Producer added the message - " + message);
                    }
                    Thread.sleep(TimeUnit.SECONDS.toMillis(3));
                }

            } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }

        }).start();

        new Thread(() ->
        {
            try
            {
                while (true)
                {
                    System.out.println("Consumer is waiting to take message...");

                    Integer message = linkedTransferQueue.take();

                    System.out.println("Consumer recieved the message - " + message);

                    Thread.sleep(TimeUnit.SECONDS.toMillis(3));
                }

            } catch (InterruptedException e) {
                e.printStackTrace();
            }

        }).start();
    }
}
```

Program Output.

```
Console

Producer is waiting to transfer message...
Consumer is waiting to take message...
Producer is waiting to transfer message...
Producer added the message - 431529176
Consumer recieved the message - 431529176
Consumer is waiting to take message...
Producer is waiting to transfer message...
Producer added the message - 1761283695
Consumer recieved the message - 1761283695
Consumer is waiting to take message...
Producer is waiting to transfer message...
Producer added the message - 1749940626
Consumer recieved the message - 1749940626
Consumer is waiting to take message...
Producer is waiting to transfer message...
Producer added the message - 892128508
Consumer recieved the message - 892128508
Consumer is waiting to take message...
Producer is waiting to transfer message...
Producer added the message - 155629808
Consumer recieved the message - 155629808
```

Please note that there may be some print statements in console where it seems that consumer consumed the message even before producer produced the message. Do not be confused, it is because of concurrent nature of example. In real, it works as expected.

3. Java LinkedTransferQueue Constructors

LinkedTransferQueue class provides 3 different ways to construct a queue in Java.

- **LinkedTransferQueue()** : constructs an initially empty LinkedTransferQueue.
- **LinkedTransferQueue(Collection c)** : constructs a LinkedTransferQueue initially containing the elements of the given collection, added in traversal order of the collection's iterator.

4. Java LinkedTransferQueue Methods

LinkedTransferQueue class has below given important methods, you should know.

- **Object take()** : Retrieves and removes the head of this queue, waiting if necessary until an element becomes available.
- **void transfer(Object o)** : Transfers the element to a consumer, waiting if necessary to do so.
- **boolean tryTransfer(Object o)** : Transfers the element to a waiting consumer immediately, if possible.
- **boolean tryTransfer(Object o, long timeout, TimeUnit unit)** : Transfers the element to a consumer if it is possible to do so before the timeout elapses.
- **int getWaitingConsumerCount()** : Returns an estimate of the number of consumers waiting to receive elements via BlockingQueue.take() or timed poll.
- **boolean hasWaitingConsumer()** : Returns true if there is at least one consumer waiting to receive an element via BlockingQueue.take() or timed poll.
- **void put(Object o)** : Inserts the specified element at the tail of this queue.**boolean add(object)** : Inserts the specified element at the tail of this queue.
- **boolean offer(object)** : Inserts the specified element at the tail of this queue.
- **boolean remove(object)** : Removes a single instance of the specified element from this queue, if it is present.
- **Object peek()** : Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
- **Object poll()** : Retrieves and removes the head of this queue, or returns null if this queue is empty.
- **Object poll(timeout, timeUnit)** : Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available.
- **void clear()** : Removes all of the elements from this the queue.
- **boolean contains(Object o)** : Returns true if this queue contains the specified element.
- **Iterator iterator()** : Returns an iterator over the elements in this queue in proper sequence.
- **int size()** : Returns the number of elements in this queue.
- **int drainTo(Collection c)** : Removes all available elements from this queue and adds them to the given collection.
- **int drainTo(Collection c, int maxElements)** : Removes at most the given number of available elements from this queue and adds them to the given collection.
- **int remainingCapacity()** : Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking.
- **Object[] toArray()** : Returns an array containing all of the elements in this queue, in proper sequence.

5. Java TransferQueue Conclusion

In this **Java LinkedTransferQueue tutorial**, we learned to use **LinkedTransferQueue class** which is a concurrent blocking queue implementation in which producers may wait for receipt of messages by consumers.

We also learned few important methods and constructors of LinkedTransferQueue class.

Drop me your questions in comments section.

Happy Learning !!

References:

TransferQueue interface Java Docs
LinkedTransferQueue Class Java Docs

Recommended Reading:

1. [Solved]: javax.xml.bind.JAXBException: class java.util.ArrayList nor any of its super class is known to this context
2. Java TreeMap class
3. Java PriorityBlockingQueue class
4. Java ArrayBlockingQueue class
5. Java CopyOnWriteArrayList class
6. Java CopyOnWriteArraySet class
7. Java LinkedHashMap class
8. Java LinkedHashSet class
9. Java TreeSet class
0. Java LinkedList class

Leave a Comment

Name *

Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search …