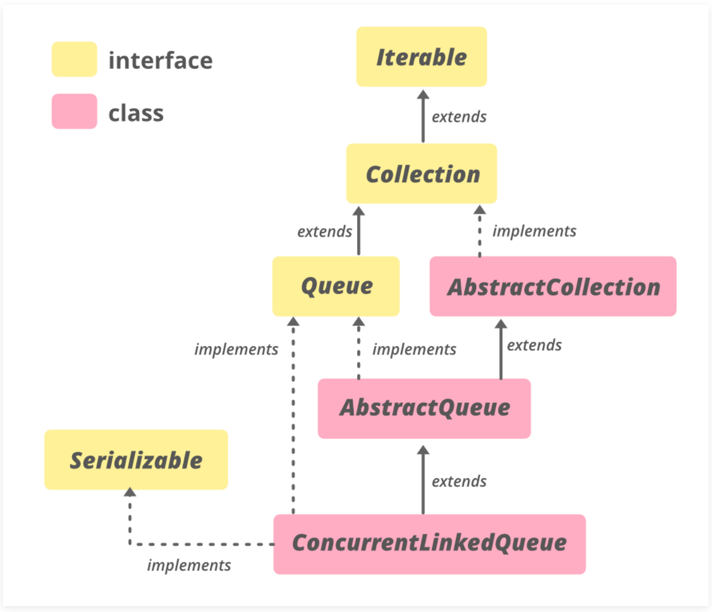
ConcurrentLinkedQueue in Java with Examples

Last Updated : 26 May, 2021

The **ConcurrentLinkedQueue** class in Java is a part of the Java Collection Framework. It belongs to **java.util.concurrent** package. It was introduced in JDK 1.5. It is used to implement Queue with the help of LinkedList concurrently. It is an **unbounded thread-safe** implementation of Queue which inserts elements at the tail of the Queue in a FIFO(first-in-first-out) fashion. It can be used when an unbounded Queue is shared among many threads. This class does not permit null elements. Iterators are weakly consistent. This class and its iterator implement all of the optional methods of the **Queue** and **Iterator** interfaces.

Class Hierarchy:

```
java.lang.Object
  ↳ java.util.AbstractCollection<E>
     ↳ java.util.AbstractQueue<E>
        ↳ Class ConcurrentLinkedQueue<E>
```



Declaration:

public class ConcurrentLinkedQueue<E> extends AbstractCollection<E> implements Queue<E>, Serializable

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

Constructors of ConcurrentLinkedQueue

To construct a ConcurrentLinkedQueue, we need to import it from **java.util.ConcurrentLinkedQueue**.

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

ConcurrentLinkedQueue<E> clq = new ConcurrentLinkedQueue<E>();

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

ConcurrentLinkedQueue<E> clq = new ConcurrentLinkedQueue<E>(Collection<E> c);

Below is a sample program to illustrate ConcurrentLinkedQueue in Java:

Example 1:

```java
// Java program to demonstrate ConcurrentLinkedQueue

import java.util.concurrent.*;

class ConcurrentLinkedQueueDemo {

    public static void main(String[] args)
    {
        // Create a ConcurrentLinkedQueue
        // using ConcurrentLinkedQueue() constructor
        ConcurrentLinkedQueue<Integer>
            clq = new ConcurrentLinkedQueue<Integer>();

        clq.add(12);
        clq.add(70);
        clq.add(1009);
        clq.add(475);

        // Displaying the existing LinkedQueue
        System.out.println("ConcurrentLinkedQueue: "
                           + clq);

        // Create a ConcurrentLinkedQueue
        // using ConcurrentLinkedQueue(Collection c)
        // constructor
        ConcurrentLinkedQueue<Integer>
            clq1 = new ConcurrentLinkedQueue<Integer>(clq);

        // Displaying the existing LinkedQueue
        System.out.println("ConcurrentLinkedQueue1: "
                           + clq1);
    }
}
```

Output:

```
ConcurrentLinkedQueue: [12, 70, 1009, 475]
ConcurrentLinkedQueue1: [12, 70, 1009, 475]
```

Example 2:

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

import java.util.concurrent.*;

class ConcurrentLinkedQueueDemo {

    public static void main(String[] args)
    {

        // Create a ConcurrentLinkedQueue
        // using ConcurrentLinkedQueue()
          // constructor
        ConcurrentLinkedQueue<Integer>
```

```
            clq = new ConcurrentLinkedQueue<Integer>();

        clq.add(12);
        clq.add(70);
        clq.add(1009);
        clq.add(475);

        // Displaying the existing ConcurrentLinkedQueue
        System.out.println("ConcurrentLinkedQueue: "
                           + clq);

        // Displaying the first element
        // using peek() method
        System.out.println("First Element is: "
                           + clq.peek());

        // Remove and display the first element
        // using poll() method
        System.out.println("Head Element is: "
                           + clq.poll());

        // Displaying the existing ConcurrentLinkedQueue
        System.out.println("ConcurrentLinkedQueue: "
                           + clq);

        // Get the size using size() method
        System.out.println("Size: "
                           + clq.size());
    }
}
```

Output:

```
ConcurrentLinkedQueue: [12, 70, 1009, 475]
First Element is: 12
Head Element is: 12
ConcurrentLinkedQueue: [70, 1009, 475]
Size: 3
```

Basic Operations

1. Adding Elements

To add elements ConcurrentLinkedQueue provides two methods.

- add() It inserts the element, passed as a parameter at the tail of this ConcurrentLinkedQueue. This method returns True if insertion is successful. ConcurrentLinkedQueue is unbounded, so this method will never throw IllegalStateException or return false.

- addAll() It inserts all the elements of the Collection, passed as a parameter at the end of a ConcurrentLinkedQueue. The insertion of the element is in the same order as returned by the collection's iterator.

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

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

public class AddingElementsExample {

    public static void main(String[] args)
    {

        // Create an instance of ConcurrentLinkedQueue
        ConcurrentLinkedQueue<String> queue = new ConcurrentLinkedQueue<String>

        // Add String to queue using add method
        queue.add("Kolkata");
        queue.add("Patna");
        queue.add("Delhi");
        queue.add("Jammu");

        // Displaying the existing ConcurrentLinkedQueue
        System.out.println("ConcurrentLinkedQueue: " + queue);

        // create a ArrayList of Strings
        ArrayList<String> arraylist = new ArrayList<String>();

        // add String to ArrayList
        arraylist.add("Sanjeet");
        arraylist.add("Rabi");
        arraylist.add("Debasis");
        arraylist.add("Raunak");
        arraylist.add("Mahesh");

        // Displaying the existing Collection
        System.out.println("Collection to be added: " + arraylist);

        // apply addAll() method and passed
        // the arraylist as parameter
        boolean response = queue.addAll(arraylist);

        // Displaying the existing ConcurrentLinkedQueue
        System.out.println("Collection added: " + response);

        // Displaying the existing ConcurrentLinkedQueue
        System.out.println("ConcurrentLinkedQueue: " + queue);
    }
}
```

Output

```
ConcurrentLinkedQueue: [Kolkata, Patna, Delhi, Jammu]
Collection to be added: [Sanjeet, Rabi, Debasis, Raunak, Mahesh]
Collection added: true
ConcurrentLinkedQueue: [Kolkata, Patna, Delhi, Jammu, Sanjeet, Rabi, Debasis,
```

2. Removing Elements

The remove(Object o) method of ConcurrentLinkedQueue is used to remove a single instance of the specified element if it is present. It removes an element **e** such that o.equals(e). It returns true if this ConcurrentLinkedQueue contained the specified element else it will return false.

```
// Java Program Demonstrate removing
// elements from ConcurrentLinkedQueue

import java.util.concurrent.*;
```

```
public class RemovingElementsExample {

    public static void main(String[] args)
    {

        // Create an instance of ConcurrentLinkedQueue
        ConcurrentLinkedQueue<Integer> queue = new ConcurrentLinkedQueue<Intege

        // Add Numbers to queue using add(e) method
        queue.add(4353);
        queue.add(7824);
        queue.add(78249);
        queue.add(8724);

        // Displaying the existing ConcurrentLinkedQueue
        System.out.println("ConcurrentLinkedQueue: " + queue);

        // apply remove() for Number 78249
        boolean response = queue.remove(78249);

        // print results
        System.out.println("Removing Number 78249 successful: " + response);

        // Displaying the existing ConcurrentLinkedQueue
        System.out.println("Updated ConcurrentLinkedQueue: " + queue);
    }
}
```

Output

```
ConcurrentLinkedQueue: [4353, 7824, 78249, 8724]
Removing Number 78249 successful: true
Updated ConcurrentLinkedQueue: [4353, 7824, 8724]
```

3. Iterating Elements

The iterator() method of ConcurrentLinkedQueue is used to return an iterator of the same elements as this ConcurrentLinkedQueue in a proper sequence. The elements returned from this method contains elements in order from first(head) to last(tail). The returned iterator is weakly consistent.

```java
// Java Program Demonstrate Iterating
// over ConcurrentLinkedQueue

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

public class TraversingExample {

    public static void main(String[] args)
    {
        // Create an instance of ConcurrentLinkedQueue
        ConcurrentLinkedQueue<String> queue = new ConcurrentLinkedQueue<String>

        // Add String to queue using add(e) method
        queue.add("Aman");
        queue.add("Amar");
        queue.add("Sanjeet");
        queue.add("Rabi");

        // Displaying the existing ConcurrentLinkedQueue
        System.out.println("ConcurrentLinkedQueue : " + queue);

        // Call iterator() method
        Iterator iterator = queue.iterator();

        // Print elements of iterator
        System.out.println("\nThe String Values of iterator are:");
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

Output

```
ConcurrentLinkedQueue : [Aman, Amar, Sanjeet, Rabi]

The String Values of iterator are:
Aman
Amar
Sanjeet
Rabi
```

4. Accessing Elements

peek() and element() methods provided by **Queue** are used to access the elements of ConcurrentLinkedQueue.

element() method differs from ***peek() method*** only in that it throws an exception if this queue is empty.

```java
// Java Program Demonstrate accessing
// elements of ConcurrentLinkedQueue

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

public class AccessingElementsExample {

    public static void main(String[] args) throws IllegalStateException
    {
        // Create an instance of ConcurrentLinkedQueue
        ConcurrentLinkedQueue<Integer> Q = new ConcurrentLinkedQueue<>();

        // Add numbers to end of Queue
        Q.add(7855642);
        Q.add(35658786);
        Q.add(5278367);
        Q.add(74381793);

        // print queue
        System.out.println("Queue: " + Q);

        // print head
        System.out.println("Queue's head: " + Q.element());

        // print head
        System.out.println("Queue's head: " + Q.peek());
    }
}
```

Output

```
Queue: [7855642, 35658786, 5278367, 74381793]
Queue's head: 7855642
Queue's head: 7855642
```

Methods of ConcurrentLinkedQueue

METHOD	DESCRIPTION
add(E e)	Inserts the specified element at the tail of this queue.
addAll (Collection<? extends E> c)	Appends all of the elements in the specified collection to the end of this queue, in the order that they are returned by the specified collection’s iterator.
contains(Object o)	Returns true if this queue contains the specified element.
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.

METHOD	DESCRIPTION
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.
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 the 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.AbstractQueue

METHOD	DESCRIPTION
clear()	Removes all 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.Collection

METHOD	DESCRIPTION
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.

METHOD	DESCRIPTION
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/ConcurrentLinkedQueue.html