ConcurrentLinkedDeque in Java with Examples

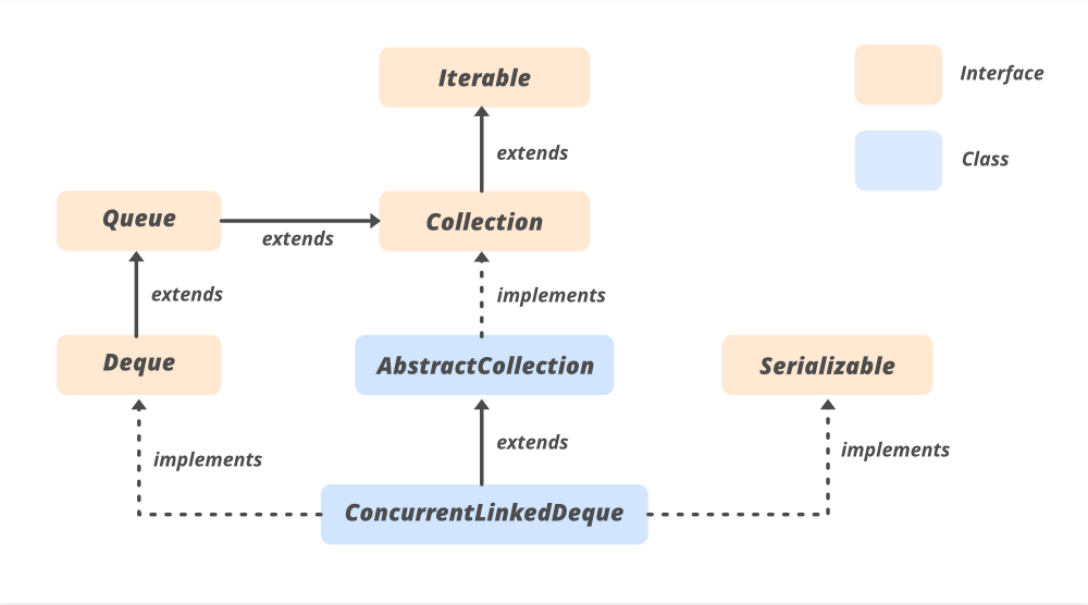
Last Updated : 16 Sep, 2020

The **ConcurrentLinkedDeque** class in Java is a part of the Java Collection Framework and implements the *Collection interface* and the *AbstractCollection class*. It belongs to **java.util.concurrent** package. It is used to implement Deque with the help of LinkedList concurrently.

Features of ConcurrentLinkedDeque

- Iterators and spliterators are weakly consistent.
- Concurrent insertion, removal, and access operations execute safely across multiple threads.
- It does not permit null elements.
- size() method is not implemented in constant time. Because of the asynchronous nature of these deques, determining the current number of elements requires a traversal of the elements.

Class Hierarchy:



Declaration:

```
public abstract class ConcurrentLinkedDeque<E>
   extends AbstractCollection<E>
      implements Deque<E>, Serializable
```

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

It implements Serializable, Iterable<E>, Collection<E>, Deque<E>, Queue<E> interfaces.

Constructors of ConcurrentLinkedDeque:

1. ConcurrentLinkedDeque(): This constructor is used to construct an empty deque.

ConcurrentLinkedDeque<E> cld = new ConcurrentLinkedDeque<E>();

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

ConcurrentLinkedDeque<E> cld = new ConcurrentLinkedDeque<E>(Collection<E> c);

Below is the sample program to illustrate ConcurrentLinkedDeque in Java:

```
// Java Program to demonstrate ConcurrentLinkedDeque

import java.util.concurrent.*;

class ConcurrentLinkedDequeDemo {

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

          // add element to the front
```

```
            // using addFirst() method
        cld.addFirst(12);
        cld.addFirst(70);
        cld.addFirst(1009);
        cld.addFirst(475);

        // Displaying the existing ConcurrentLinkedDeque
        System.out.println("ConcurrentLinkedDeque: "
                           + cld);

        // Create a ConcurrentLinkedDeque
        // using ConcurrentLinkedDeque(Collection c)
        // constructor
        ConcurrentLinkedDeque<Integer>
            cld1 = new ConcurrentLinkedDeque<Integer>(cld);

        // Displaying the existing ConcurrentLinkedDeque
        System.out.println("ConcurrentLinkedDeque1: "
                           + cld1);
    }
}
```

Output:

```
ConcurrentLinkedDeque: [475, 1009, 70, 12]
ConcurrentLinkedDeque1: [475, 1009, 70, 12]
```

Example:

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

import java.util.concurrent.*;

class ConcurrentLinkedDequeDemo {

    public static void main(String[] args)
    {

        // Create a ConcurrentLinkedDeque
        // using ConcurrentLinkedDeque() constructor
```

```
        ConcurrentLinkedDeque<Integer>
            cld = new ConcurrentLinkedDeque<Integer>();

          // add element to the front
          // using addFirst() method
        cld.addFirst(12);
        cld.addFirst(70);
        cld.addFirst(1009);
        cld.addFirst(475);

        // Displaying the existing ConcurrentLinkedDeque
        System.out.println("ConcurrentLinkedDeque: "
                           + cld);

        // Displaying the Last element
        // using getLast() method
        System.out.println("The Last element is: "
                           + cld.getLast());

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

        // Remove the Last element
        // using removeLast() method
        cld.removeLast();

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

Output:

```
ConcurrentLinkedDeque: [475, 1009, 70, 12]
The Last element is: 12
First Element is: 475
ConcurrentLinkedDeque: [475, 1009, 70]
```

Basic Operations of ConcurrentLinkedDeque

1. Adding Elements

To add an element or Collection of elements, ConcurrentLinkedDeque provides methods like add(E e), addAll(Collection<? extends E> c), addFirst(E e), addLast(E e) methods. The example below explains these methods.

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

import java.util.concurrent.*;

class AddingElements {

    public static void main(String[] args)
    {
        // create instance using ConcurrentLinkedDeque
        ConcurrentLinkedDeque<Integer> cld1
            = new ConcurrentLinkedDeque<Integer>();

        // Add element to the tail using
        // add or addLast methods
        cld1.add(12);
        cld1.add(110);

        // Add element to the head
        // using addFirst method
        cld1.addFirst(55);

        // Displaying the existing ConcurrentLinkedDeque
        System.out.println("Initial Elements in "
                           + "the LinkedDeque cld : "
                           + cld1);

        // create instance using ConcurrentLinkedDeque
        ConcurrentLinkedDeque<Integer> cld2
            = new ConcurrentLinkedDeque<Integer>();

        // Add elements of cld1 to the
        // cld2 using addAll method
        cld2.addAll(cld1);

        // Displaying the modified ConcurrentLinkedDeque
        System.out.println("Initial Elements in "
                           + "the LinkedDeque cld2: "
                           + cld2);
    }
}
```

Output:

```
Initial Elements in the LinkedDeque cld : [55, 12, 110]
Initial Elements in the LinkedDeque cld2: [55, 12, 110]
```

2. Remove Elements

To remove an element, ConcurrentLinkedDeque provides methods like remove(), remove(Object o), removeFirst(), removeLast() etc. These methods are explained in the below example.

```
// Java Program to demonstrate removing
// elements of ConcurrentLinkedDeque

import java.util.concurrent.*;

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

        // Create a ConcurrentLinkedDeque
        // using ConcurrentLinkedDeque() constructor
        ConcurrentLinkedDeque<Integer> cld
            = new ConcurrentLinkedDeque<Integer>();

        // Add elements using add() method
        cld.add(40);
        cld.add(50);
        cld.add(60);
        cld.add(70);
        cld.add(80);

        // Displaying the existing LinkedDeque
        System.out.println(
            "Existing ConcurrentLinkedDeque: " + cld);

        // remove method removes the first
        // element of ConcurrentLinkedDeque
        // using remove() method
        System.out.println("Element removed: "
                           + cld.remove());

        // Remove 60 using remove(Object)
        System.out.println("60 removed: " + cld.remove(60));
```

```
        // Displaying the existing ConcurrentLinkedDeque
        System.out.println(
            "Modified ConcurrentLinkedDeque: " + cld);

        // Remove the first element
        cld.removeFirst();

        // Remove the Last element
        cld.removeLast();

        // Displaying the existing ConcurrentLinkedDeque
        System.out.println(
            "Modified ConcurrentLinkedDeque: " + cld);
    }
}
```

Output

```
Existing ConcurrentLinkedDeque: [40, 50, 60, 70, 80]
Element removed: 40
60 removed: true
Modified ConcurrentLinkedDeque: [50, 70, 80]
Modified ConcurrentLinkedDeque: [70]
```

3. Iterating Elements

We can iterate the ConcurrentLinkedDeque using iterator() or descendingIterator() methods. The below code explains both these methods.

```
// Java code to illustrate iterating
// elements of ConcurrentLinkedDeque

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

public class IteratingConcurrentLinkedDeque {

    public static void main(String args[])
    {
        // Creating an empty ConcurrentLinkedDeque
        ConcurrentLinkedDeque<String> deque
            = new ConcurrentLinkedDeque<String>();
```

```
        // Use add() method to add elements
        // into the ConcurrentLinkedDeque
        deque.add("Welcome");
        deque.add("To");
        deque.add("Geeks");
        deque.add("4");
        deque.add("Geeks");

        // Displaying the ConcurrentLinkedDeque
        System.out.println("ConcurrentLinkedDeque: "
                           + deque);

        // Creating an iterator
        Iterator fitr = deque.iterator();

        // Displaying the values
        // after iterating through the ConcurrentLinkedDeque
        System.out.println("The iterator values are: ");
        while (fitr.hasNext()) {
            System.out.println(fitr.next());
        }

        // Creating a desc_iterator
        Iterator ditr = deque.descendingIterator();

        // Displaying the values after iterating
        // through the ConcurrentLinkedDeque
        // in reverse order
        System.out.println("The iterator values are: ");
        while (ditr.hasNext()) {
            System.out.println(ditr.next());
        }
    }
}
```

Output

```
ConcurrentLinkedDeque: [Welcome, To, Geeks, 4, Geeks]
The iterator values are:
Welcome
To
Geeks
4
Geeks
The iterator values are:
Geeks
4
```

```
Geeks
To
Welcome
```

4. Accessing Elements

To access the elements of ConcurrentLinkedDeque, it provides methods like getFirst(), getLast(), element() methods. The below example explains these methods.

```
// Java Program to Demonstrate accessing
// elements of ConcurrentLinkedDeque

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

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

        // Creating an empty ConcurrentLinkedDeque
        ConcurrentLinkedDeque<String> cld
            = new ConcurrentLinkedDeque<String>();

        // Add elements into the ConcurrentLinkedDeque
        cld.add("Welcome");
        cld.add("To");
        cld.add("Geeks");
        cld.add("4");
        cld.add("Geeks");

        // Displaying the ConcurrentLinkedDeque
        System.out.println("Elements in the ConcurrentLinkedDeque: " + cld);

        // Displaying the first element
        System.out.println("The first element is: "
                           + cld.getFirst());

        // Displaying the Last element
        System.out.println("The Last element is: "
                           + cld.getLast());

        // Displaying the head of ConcurrentLinkedDeque
        System.out.println("The Head of ConcurrentLinkedDeque is: "
                           + cld.element());
    }
}
```

Output:

```
Elements in the ConcurrentLinkedDeque: [Welcome, To, Geeks, 4, Geeks]
The first element is: Welcome
The Last element is: Geeks
The Head of ConcurrentLinkedDeque is: Welcome
```

Methods of ConcurrentLinkedDeque

Here, **E** is the type of element.

METHOD	DESCRIPTION
add(E e)	Inserts the specified element at the tail of this deque.
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.
addLast(E e)	Inserts the specified element at the end of this deque.
clear()	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.
element()	Retrieves, but does not remove, the head of the queue represented by this deque (in other words, the first element of 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.

METHOD	DESCRIPTION
getFirst()	Retrieves, but does not remove, the first element of this deque.
getLast()	Retrieves, but does not remove, the last element of this deque.
isEmpty()	Returns true if this collection contains no elements.
iterator()	Returns an iterator over the elements in this deque in the proper sequence.
offer(E e)	Inserts the specified element at the tail of this deque.
offerFirst(E e)	Inserts the specified element at the front of this deque.
offerLast(E e)	Inserts the specified element at the end of this deque.
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.
remove()	Retrieves and removes the head of the queue represented by this deque (in other words, the first element of 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.
removeFirstOccurrence (Object o)	Removes the first occurrence of the specified element from 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
removeLastOccurrence (Object o)	Removes the last occurrence of the specified element from this deque.
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 (from first to last element); 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.
toString()	Returns a string representation of this collection.

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
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
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.
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.
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.
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/ConcurrentLinkedDeque.html