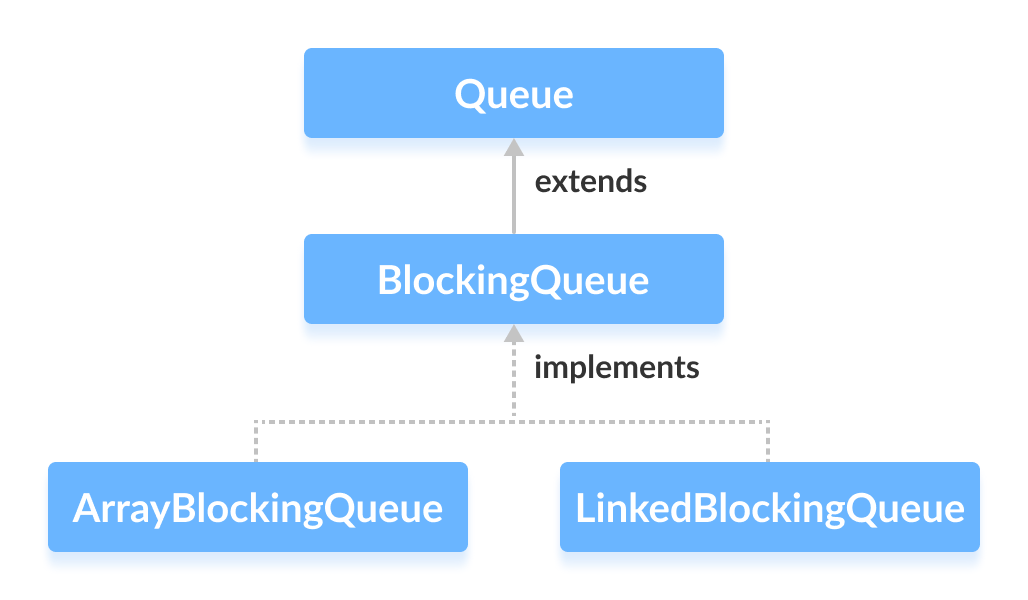
**Java LinkedBlockingQueue**

The LinkedBlockingQueue class of the Java Collections framework provides the blocking queue implementation using a linked list.

It implements the Java BlockingQueue interface.



## Creating LinkedBlockingQueue

In order to create a linked blocking queue, we must import the java.util.concurrent.LinkedBlockingQueue package.

Here is how we can create a linked blocking queue in Java:

**1. Without the initial capacity**

LinkedBlockingQueue<Type> animal = new LinkedBlockingQueue<>();

Here the default initial capacity will be 231-1.

**2. With the initial capacity**

LinkedBlockingQueue<Type> animal = new LinkedBlockingQueue<>(int capacity);

Here,

* Type - the type of the linked blocking queue
* capacity - the size of the linked blocking queue

For example,

// Creating String type LinkedBlockingQueue with size 5

LinkedBlockingQueue<String> animals = new LinkedBlockingQueue<>(5);

// Creating Integer type LinkedBlockingQueue with size 5

LinkedBlockingQueue<Integer> age = new LinkedBlockingQueue<>(5);

**Note:** **It is not compulsory to provide the size of the linked list.**

## Methods of LinkedBlockingQueue

The LinkedBlockingQueue class provides the implementation of all the methods in the BlockingQueue interface.

These methods are used to insert, access and delete elements from linked blocking queues.

Also, we will learn about two methods put() and take() that support the blocking operation in the linked blocking queue.

These two methods distinguish the linked blocking queue from other typical queues.

### Insert Elements

* add() - Inserts a specified element to the linked blocking queue. It throws an exception if the queue is full.
* offer() - Inserts a specified element to the linked blocking queue. It returns false if the queue is full.

For example,

import java.util.concurrent.LinkedBlockingQueue;

class Main {

public static void main(String[] args) {

LinkedBlockingQueue<String> animals = new LinkedBlockingQueue<>(5);

// Using add()

animals.add("Dog");

animals.add("Cat");

// Using offer()

animals.offer("Horse");

System.out.println("LinkedBlockingQueue: " + animals);

}

}

**Output**

LinkedBlockingQueue: [Dog, Cat, Horse]

### Access Elements

* peek() - Returns an element from the front of the linked blocking queue. It returns null if the queue is empty.
* iterator() - Returns an iterator object to sequentially access an element from the linked blocking queue. It throws an exception if the queue is empty. We must import the java.util.Iterator package to use it.

For example,

import java.util.concurrent.LinkedBlockingQueue;

import java.util.Iterator;

class Main {

public static void main(String[] args) {

LinkedBlockingQueue<String> animals = new LinkedBlockingQueue<>(5);

// Add elements

animals.add("Dog");

animals.add("Cat");

animals.add("Horse");

System.out.println("LinkedBlockingQueue: " + animals);

// Using peek()

String element = animals.peek();

System.out.println("Accessed Element: " + element);

// Using iterator()

Iterator<String> iterate = animals.iterator();

System.out.print("LinkedBlockingQueue Elements: ");

while(iterate.hasNext()) {

System.out.print(iterate.next());

System.out.print(", ");

}

}

}

**Output**

LinkedBlockingQueue: [Dog, Cat, Horse]

Accessed Element: Dog

LinkedBlockingQueue Elements: Dog, Cat, Horse,

### Remove Elements

* remove() - Returns and removes a specified element from the linked blocking queue. It throws an exception if the queue is empty.
* poll() - Returns and removes a specified element from the linked blocking queue. It returns null if the queue is empty.
* clear() - Removes all the elements from the linked blocking queue.

For example,

import java.util.concurrent.LinkedBlockingQueue;

class Main {

public static void main(String[] args) {

LinkedBlockingQueue<String> animals = new LinkedBlockingQueue<>(5);

animals.add("Dog");

animals.add("Cat");

animals.add("Horse");

System.out.println("LinkedBlockingQueue " + animals);

// Using remove()

String element1 = animals.remove();

System.out.println("Removed Element:");

System.out.println("Using remove(): " + element1);

// Using poll()

String element2 = animals.poll();

System.out.println("Using poll(): " + element2);

// Using clear()

animals.clear();

System.out.println("Updated LinkedBlockingQueue " + animals);

}

}

**Output**

LinkedBlockingQueue: [Dog, Cat, Horse]

Removed Elements:

Using remove(): Dog

Using poll(): Cat

Updated LinkedBlockingQueue: []

## put() and take() Methods

In multithreading processes, we can use put() and take() to block the operation of one thread to synchronize it with another thread. These methods will wait until they can be successfully executed.

### put() Method

To insert the specified element to the end of a linked blocking queue, we use the put() method.

If the linked blocking queue is full, it waits until there is space in the linked blocking queue to insert the element.

For example,

import java.util.concurrent.LinkedBlockingQueue;

class Main {

public static void main(String[] args) {

LinkedBlockingQueue<String> animals = new LinkedBlockingQueue<>(5);

try {

// Add elements to animals

animals.put("Dog");

animals.put("Cat");

System.out.println("LinkedBlockingQueue: " + animals);

}

catch(Exception e) {

System.out.println(e);

}

}

}

**Output**

LinkedBlockingQueue: [Dog, Cat]

Here, the put() method may throw an InterruptedException if it is interrupted while waiting. Hence, we must enclose it inside a try..catch block.

### take() Method

To return and remove an element from the front of the linked blocking queue, we can use the take() method.

If the linked blocking queue is empty, it waits until there are elements in the linked blocking queue to be deleted.

For example,

import java.util.concurrent.LinkedBlockingQueue;

class Main {

public static void main(String[] args) {

LinkedBlockingQueue<String> animals = new LinkedBlockingQueue<>(5);

try {

//Add elements to animals

animals.put("Dog");

animals.put("Cat");

System.out.println("LinkedBlockingQueue: " + animals);

// Remove an element

String element = animals.take();

System.out.println("Removed Element: " + element);

System.out.println("New LinkedBlockingQueue: " + animals);

}

catch(Exception e) {

System.out.println(e);

}

}

}

**Output**

LinkedBlockingQueue: [Dog, Cat]

Removed Element: Dog

New LinkedBlockingQueue: [Cat]

Here, the take() method will throw an InterrupedException if it is interrupted while waiting. Hence, we must enclose it inside a try...catch block.

## Other Methods

|  |  |
| --- | --- |
| Methods | Descriptions |
| contains(element) | Searches the linked blocking queue for the specified element. If the element is found, it returns true, if not it returns false. |
| size() | Returns the length of the linked blocking queue. |
| toArray() | Converts linked blocking queue to an array and return the array. |
| toString() | Converts the linked blocking queue to string |

## Why use LinkedBlockingQueue?

The LinkedBlockingQueue uses linked lists as its internal storage.

It is considered as a **thread-safe** collection. Hence, it is generally used in multi-threading applications.

Suppose, one thread is inserting elements to the queue and another thread is removing elements from the queue.

Now, if the first thread is slower than the second thread, then the linked blocking queue can make the second thread waits until the first thread completes its operations.