**Blocking Queue in Java**

java.util.concurrent.BlockingQueue is a java Queue that support operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element.

BlockingQueue interface is part of java collections framework and it’s primarily used for implementing producer consumer problem.

Java provides several BlockingQueue implementations such as ArrayBlockingQueue, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue etc. While implementing producer consumer problem in BlockingQueue, we will use ArrayBlockingQueue implementation. Following are some important methods you should know.

* put(E e): This method is used to insert elements to the queue. If the queue is full, it waits for the space to be available.
* E take(): This method retrieves and remove the element from the head of the queue. If queue is empty it waits for the element to be available.
* **BlockingQueue Types**

There are two types of BlockingQueue:

**1. Unbounded Queue:** Unbounded blocking queue is the queue that never blocks because its size could be grown to a very large size. The capacity of the BlockingQueue will be set to Integer.MAX\_VALUE. When the elements are added, the size of the Unbounded queue grows.

**Syntax :**

1. BlockingQueue blockingQueue = **new** LinkedBlockingDeque( ) ;

**2. Bounded Queue:** Another type of the blocking queue is the bounded queue. It can be created by passing the capacity of the queue to the constructor of the queue.

**Syntax :**

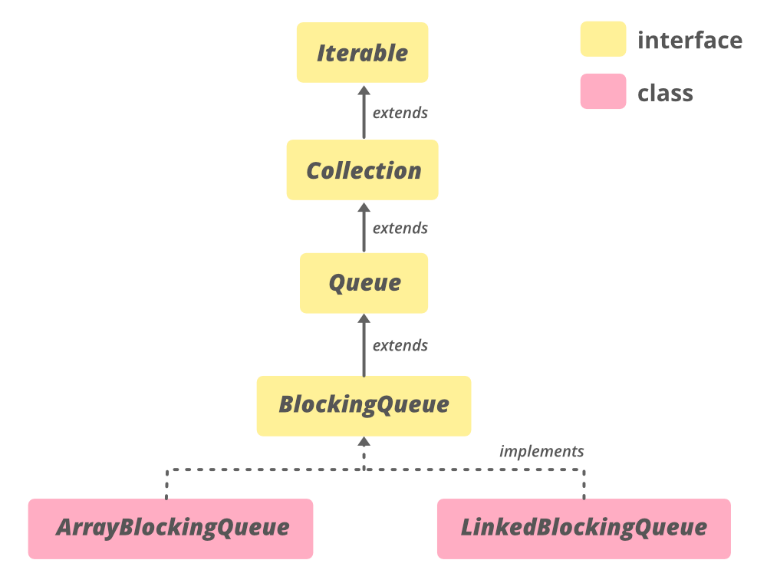
// Creating a Blocking Queue of the bounded nature with capacity 10

1. BlockingQueue blockingQueue = **new** LinkedBlockingDeque( 10 ) ;

* **Basic Operations**

1. Adding elements
2. Accessing elements
3. Deleting elements
4. Iterating through the elements

* **The Hierarchy of BlockingQueue**



| METHOD | DESCRIPTION |
| --- | --- |
| [add​(E e)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/blockingqueue-add-in-java-with-examples/&sa=U&ved=2ahUKEwiH8r7zjO3rAhXxheYKHZGWBxAQFjAJegQIBRAB&usg=AOvVaw0lalEAVVD0ljY1gR1Vw5BM) | Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success, and throwing an IllegalStateException if no space is currently available. |
| [contains​(Object o)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/blockingqueue-contains-method-in-java-with-examples/&sa=U&ved=2ahUKEwiH8r7zjO3rAhXxheYKHZGWBxAQFjAGegQIAxAC&usg=AOvVaw07Ey-6hKgq0RsLhXU-tCKg) | Returns true if this queue contains the specified element. |
| [poll​(long timeout, TimeUnit unit)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/blockingqueue-poll-method-in-java-with-examples/&sa=U&ved=2ahUKEwiH8r7zjO3rAhXxheYKHZGWBxAQFjACegQICBAB&usg=AOvVaw3kWMXpoSA1qoZXtY-YsbdZ) | Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available. |
| [put​(E e)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/blockingqueue-put-method-in-java-with-examples/&sa=U&ved=2ahUKEwiH8r7zjO3rAhXxheYKHZGWBxAQFjAHegQIBBAC&usg=AOvVaw2_F6-qEEVdUHAiH3LAlA1O) | Inserts the specified element into this queue, waiting if necessary for space to become available. |
| [remainingCapacity()](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/blockingqueue-remainingcapacity-method-in-java-with-examples/&sa=U&ved=2ahUKEwiH8r7zjO3rAhXxheYKHZGWBxAQFjAFegQIARAC&usg=AOvVaw0jguEYga8Zpg594VJIsmZD) | Returns the number of additional elements that this queue can ideally (in the absence of memory or resource constraints) accept without blocking, or Integer.MAX\_VALUE if there is no intrinsic limit. |
| [remove​(Object o)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/blockingqueue-remove-method-in-java-with-examples/&sa=U&ved=2ahUKEwiH8r7zjO3rAhXxheYKHZGWBxAQFjAIegQIBhAC&usg=AOvVaw1i9WpGRhfri71kFldYLhdQ) | Removes a single instance of the specified element from this queue, if it is present. |
| [take()](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/blockingqueue-take-method-in-java-with-examples/&sa=U&ved=2ahUKEwiH8r7zjO3rAhXxheYKHZGWBxAQFjABegQICRAC&usg=AOvVaw1wR1N67HxtNJXmKOHn73uF) | 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)](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collection-addall-method-in-java-with-examples/&sa=U&ved=2ahUKEwitvNPJje3rAhWW73MBHa1BC9I4ChAWMAJ6BAgIEAI&usg=AOvVaw2xR9z67AO1NmXiMixsYCRH) | Adds all of the elements in the specified collection to this collection (optional operation). |
| [clear()](https://www.google.com/url?client=internal-element-cse&cx=009682134359037907028:tj6eafkv_be&q=https://www.geeksforgeeks.org/collection-clear-method-in-java-with-examples/&sa=U&ved=2ahUKEwitvNPJje3rAhWW73MBHa1BC9I4ChAWMAN6BAgHEAI&usg=AOvVaw3b20olhneSrvsfRp4FNmxn) | 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. |
| isEmpty() | Returns true if this collection contains no elements. |
| iterator() | Returns an iterator over the elements in this collection. |
| parallelStream() | Returns a possibly parallel Stream with this collection as its source. |
| 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 collection. |
| spliterator() | Creates a [Spliterator](https://www.geeksforgeeks.org/java-util-interface-spliterator-java8/) over the elements in this collection. |
| stream() | Returns a sequential Stream with this collection as its source. |

Methods declared in interface java.lang.Iterable

| METHOD | DESCRIPTION |
| --- | --- |
| [forEach​(Consumer<? super T> action)](https://www.geeksforgeeks.org/iterable-foreach-method-in-java-with-examples/#:~:text=One%20of%20them%20is%20forEach,the%20elements%20inside%20the%20collection.) | Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. |

Methods declared in interface java.util.Queue

| METHOD | DESCRIPTION |
| --- | --- |
| [element()](https://www.geeksforgeeks.org/queue-element-method-in-java/#:~:text=The%20element()%20method%20of,if%20this%20queue%20is%20empty.) | Retrieves, but does not remove, the head of this queue. |
| [peek()](https://www.geeksforgeeks.org/queue-peek-method-in-java/#:~:text=The%20peek()%20method%20of,empty%2C%20it%20returns%20null%20instead.) | Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |
| [poll()](https://www.geeksforgeeks.org/queue-poll-method-in-java/) | Retrieves and removes the head of this queue, or returns null if this queue is empty. |
| [remove()](https://www.geeksforgeeks.org/blockingqueue-interface-in-java/) | Retrieves and removes the head of this queue. |

**Example: producer consumer problem using java BlockingQueue**

* **queue message:**

public class Message

{

private String msg;

public Message(String str)

{

this.msg=str;

}

public String getMsg()

{

return msg;

}

}

* **Producer class that will create messages and put it in the queue:**

public class Producer implements Runnable

{

private BlockingQueue<Message> queue;

public Producer(BlockingQueue<Message> q)

{

this.queue=q;

}

@Override

public void run()

{

//produce messages

for(int i=0; i<100; i++)

{

Message msg = new Message(""+i);

try

{

Thread.sleep(i);

queue.put(msg);

System.out.println("Produced "+msg.getMsg());

} catch (InterruptedException e)

{

e.printStackTrace();

}

}

//adding exit message

Message msg = new Message("exit");

try {

queue.put(msg);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

* **Consumer class that will process on the messages from the queue and terminates when exit message is received**:

import java.util.concurrent.BlockingQueue;

public class Consumer implements Runnable

{

private BlockingQueue<Message> queue;

public Consumer(BlockingQueue<Message> q)

{

this.queue=q;

}

@Override

public void run()

{

try{

Message msg;

//consuming messages until exit message is received

while((msg = queue.take()).getMsg() !="exit"){

Thread.sleep(10);

System.out.println("Consumed "+msg.getMsg());

}

}catch(InterruptedException e) {

e.printStackTrace();

}

}

}

* **BlockingQueue service for producer and consumer:**

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.BlockingQueue;

public class ProducerConsumerService

{

public static void main(String[] args)

{

//Creating BlockingQueue of size 10

BlockingQueue<Message> queue = new ArrayBlockingQueue<>(10);

Producer producer = new Producer(queue);

Consumer consumer = new Consumer(queue);

//starting producer to produce messages in queue

new Thread(producer).start();

//starting consumer to consume messages from queue

new Thread(consumer).start();

System.out.println("Producer and Consumer has been started");

}

}

**Output:**

Producer and Consumer has been started

Produced 0

Produced 1

Produced 2

Produced 3

Produced 4

Consumed 0

Produced 5

Consumed 1

Produced 6

Produced 7

Consumed 2

Produced 8

...