BlockingQueue Interface in Java

**BlockingQueue interface** in Java is added in Java 1.5 along with various other concurrent Utility classes like ConcurrentHashMap, Counting Semaphore, CopyOnWriteArrrayList etc. BlockingQueue interface supports flow control (in addition to queue) by introducing blocking if either BlockingQueue is full or empty. A thread trying to enqueue an element in a full queue is blocked until some other thread makes space in the queue, either by dequeuing one or more element or clearing the queue completely. Similarly it blocks a thread trying to delete from an empty queue until some other treads inserts an item. BlockingQueue does not accept null value. If we try to enqueue null item, then it throws NullPointerException.

Java provides several BlockingQueue implementations such as LinkedBlockingQueue, ArrayBlockingQueue, PriorityBlockingQueue, SynchronousQueue etc. Java BlockingQueue interface implementations are thread-safe. All methods of BlockingQueue are atomic in nature and use internal locks or other forms of concurrency control. Java 5 comes with BlockingQueue implementations in the **java.util.concurrent package**.

**BlockingQueue Types**

The BlockingQueue are two types-

* **Unbounded Queue:** The Capacity of blocking queue will be set to Integer.MAX\_VALUE. In case of unbounded blocking queue, queue will never block because it could grow to a very large size. when you add elements it’s size grow.

**Syntax:**

BlockingQueue blockingQueue = new LinkedBlockingDeque();

* **Bounded Queue:** The second type of queue is the bounded queue. In case of bounded queue you can create a queue by passing the capacity of queue in queues constructor:

**Syntax:**

// Creates a Blocking Queue with capacity 5

BlockingQueue blockingQueue = new LinkedBlockingDeque(5);

**Methods in Blocking Queue Interface**

| **MODIFIER AND TYPE** | **METHOD SYNTAX** | **USED FOR** | **DESCRIPTION** |
| --- | --- | --- | --- |
| boolean | add(E e) | Insertion | 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. |
| boolean | contains(Object o) | Examine | Returns true if this queue contains the specified element. |
| int | drainTo(Collection c) | Retrieving or Removal | Removes all available elements from this queue and adds them to the given collection. |
| int | drainTo(Collection c, int maxElements) | Retrieving or Removal | Removes at most the given number of available elements from this queue and adds them to the given collection. |
| boolean | offer(E e) | Insertion | Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and false if no space is currently available. |
| boolean | offer(E e, long timeout, TimeUnit unit) | Insertion | Inserts the specified element into this queue, waiting up to the specified wait time if necessary for space to become available. |
| E | poll(long timeout, TimeUnit unit) | Retrieving or Removal | Retrieves and removes the head of this queue, waiting up to the specified wait time if necessary for an element to become available. |
| void | put(E e) | Insertion | Inserts the specified element into this queue, waiting if necessary for space to become available. |
| int | remainingCapacity() | Examine | 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. |
| boolean | remove(Object o)+ | Retrieving or Removal | Removes a single instance of the specified element from this queue, if it is present. |
| E | take() | Retrieving or Removal | Retrieves and removes the head of this queue, waiting if necessary until an element becomes available. |