Queue [Interface]: Priority Queue

Definition

Priority Queue is an abstract data type, in the priority Queue every element has some priority. The priority of the elements in a priority Queue determines the order in which elements are removed from the priority Queue. All the elements are either arranged in an ascending or descending order.

A priority Queue is an extension of the queue with the properties:

- Every item has priority associate with it.
- An element with high priority is dequeued before an element with low priority.
- If two elements have the same priority, they are server according to their order in the queue.

A typical priority queue supports the following operations

1. Insertion: When a new element is inserted in a priority queue, it moves to the empty slot from top to bottom and left to right. If the element is not in the correct place, then it will be compared with the parent node and it will repeat the process until all elements are in the correct position.
2. Deletion: In the max heap, the maximum element is the root node. And it will remove the element which has maximum priority first. Thus, you can remove the root node from the queue.
3. Peek: This helps to return the maximum element from Max Heap or minimum element from Min Heap without deleting the node from the priority queue.

The elements of the priority queue are according to their natural ordering. A priority queue does not permit null elements

Types of Priority Queue

1. Ascending order: The element with a lower priority value is given a higher priority in the priority list.
2. Descending order: the root node is the maximum element in a max heap. It will remove the element with the highest priority first. As a result, the root node is removed from the queue.

Priority queue can be implemented with the following data structures

- Arrays
- LinkedList
- Heap data structure
- Binary Search Tree

This implementation is not synchronized. Multiple threads should not access a PriorityQueue instance concurrently if any of the threads modifies the queue.

This implementation provides O(log(n)) time for the enqueing and dequeing methods (offer, poll, remove() and add); linear time for the remove(Object) and contains(Object) methods; and constant time for the retrieval methods (peek, element, and size).

Constructor summary

PriorityQueue()	Creates a PriorityQueue with the default initial capacity (11) that orders its element according to their natural ordering.
PriorityQueue(**Collection** < ? extends **E** > c)	Creates a priorityQueu containing the elements in the specified collection
PriorityQueue(int initialCapacity)	Creates a PriorityQueu with the specified initial capacity that orders the elements according to their natural ordering.
PriorityQueue(int initialCapacity, **Comparator** < ? super **E** > comparator)	Creates a PriorityQueu with the specified initial capacity that order its elements according to specified comparator.
PriorityQueue (**PriorityQueue** < ? extends **E** > c)	Creates a PriorityQueue containing the elements in the specified priority queue.
PriorityQueue(**SortedSet** < ? extends **E** > c)	Creates PriorityQueue containing the elements in the specified sorted set.

Methods summary

Modifier and Type	**Method and Description**
boolean	**add**(E e) Inserts the specified element into this priority queue.
void	**clear**() Removes all of the elements from this priority queue
Comparator < ? super E >	**comparator**() returns comparator used in order the elements in this queue, or null if this queue is sorted according to the **natural ordering** of its elements.
boolean	**contains**(**Object** o) Returns true if this queue contains the specified element.
Iterator<E>	**addElement**(**E** obj) Returns an iterator over the elements in this queue
boolean	**offer**(**E** e**)**

	Inserts the specified element into this priority queue
E	**peek()** Retrieves but not removes the head of this queue or returns null if this queue is empty
E	**poll()** Retrieves and removes the head of this queue, or returns null if this queue is empty
boolean	**remove(Object** o) Removes a single instance of the specified element from this queue, if its present.
int	**size()** Returns the number of elements in this collection
Object[]	**toArray**() Returns an array containing all of the elements in this queue
<T>T[]	**toArray**(T[] a) Returns an array containing all of the elements in this queue ; the runtime type of the returned array is that of the specified array.

When is its use recommended?

- When we use a single thread process. Vector class is synchronized, which means that only one thread at a time can access the code.
- As the vector class is using list Interface. Vector may be used to store a list of products sold or list of supplements available at a local store.
- Vector can be used in local programs when the time of searching elements it's not important as when is a web service or similar.