Queue Interface Methods:-

Shambhu Kumar

@Kumarsam07

@javac_java

http://linkedin.com/in/kumarsam07

1. Queue Interface Methods:

The Queue interface in Java, part of the java.util package, is used to represent a collection of elements that follows the **FIFO (First-In-First-Out)** principle, though some implementations may follow other orders (e.g., PriorityQueue).

Below is a list of all methods in the Queue interface along with descriptions and their uses:

Primary Methods

1. **boolean add(E e)**
    - **Description**: Inserts the specified element into the queue. Throws an exception if the element cannot be added (e.g., due to capacity restrictions).
    - **Use**: Use when you are certain the queue has enough capacity and want to handle failures explicitly.
    - **Exception**: IllegalStateException if no space is available.
2. **boolean offer(E e)**
    - **Description**: Inserts the specified element into the queue. Returns true if successful; false if the queue is full or unable to add.

- **Use**: Use when you prefer to handle capacity issues without exceptions.

3. **E remove()**
    - **Description**: Retrieves and removes the head of the queue. Throws an exception if the queue is empty.
    - **Use**: Use when you know the queue is not empty and want explicit error handling for an empty queue.
    - **Exception**: NoSuchElementException if the queue is empty.
4. **E poll()**
    - **Description**: Retrieves and removes the head of the queue, or returns null if the queue is empty.
    - **Use**: Use when you want to handle empty queues gracefully without exceptions.
5. **E element()**
    - **Description**: Retrieves, but does not remove, the head of the queue. Throws an exception if the queue is empty.
    - **Use**: Use when you need to peek at the head element and are certain the queue is not empty.
    - **Exception**: NoSuchElementException if the queue is empty.
6. **E peek()**
    - **Description**: Retrieves, but does not remove, the head of the queue, or returns null if the queue is empty.
    - **Use**: Use when you want to safely check the head element without removing it or throwing exceptions for an empty queue.

Key Differences Between Methods

Method	Behavior if Empty	Purpose
`add(E e)`	Throws `IllegalStateException`	Insert; use when certain of capacity availability.
`offer(E e)`	Returns `false`	Insert; handles full queues without exceptions.
`remove()`	Throws `NoSuchElementException`	Remove; use when certain queue is not empty.
`poll()`	Returns `null`	Remove; handles empty queues gracefully.
`element()`	Throws `NoSuchElementException`	Peek; use when certain queue is not empty.
`peek()`	Returns `null`	Peek; handles empty queues gracefully.

Common Uses

1. **add(E e) vs offer(E e)**
    - Use add() if you want to strictly enforce capacity rules (e.g., when working with a BlockingQueue).
    - Use offer() for non-blocking insertion that allows you to check whether the operation succeeded.
2. **remove() vs poll()**
    - Use remove() when the queue should never be empty (e.g., when working with a controlled data flow).
    - Use poll() when you want to handle an empty queue without exceptions.
3. **element() vs peek()**
    - Use element() when the queue should not be empty.
    - Use peek() for safe access when the queue might be empty.

Summary

- Queue is primarily used in applications requiring **FIFO behavior**, such as task scheduling or message passing.
- Always choose methods (add vs offer, remove vs poll, element vs peek) based on whether you want strict behavior or graceful handling for special cases like full or empty queues.

2. PriorityQueue Methods:

The **PriorityQueue** class in Java is part of the **java.util** package and implements the **Queue** interface. It is designed to handle elements according to their natural ordering (if they implement **Comparable**) or using a custom **Comparator** provided during the queue's construction. The priority queue does not permit null elements and does not guarantee the order of elements with the same priority.

Here is a detailed list of all the methods of the **PriorityQueue** class, along with their descriptions and details:

Constructor:

1. **PriorityQueue()**
    - **Description**: Creates an empty priority queue with the default initial capacity (11) and orders elements according to their natural ordering.
2. **PriorityQueue(int initialCapacity)**
    - **Description**: Creates a priority queue with the specified initial capacity and orders elements according to their natural ordering.

3. **PriorityQueue(Collection<? extends E> c)**
    - **Description**: Creates a priority queue containing the elements of the specified collection, ordered according to their natural ordering.

Methods in PriorityQueue:

Adding Elements:

1. **boolean add(E e)**
    - **Description**: Inserts the specified element into the priority queue.
    - **Throws**: ClassCastException if the element is not compatible with the queue's ordering.
    - **Returns**: true if the element was successfully added.
2. **boolean offer(E e)**
    - **Description**: Inserts the specified element into the priority queue. This method is functionally the same as add(E e) but is preferred for queues.
    - **Returns**: true if the element was successfully added.

Removing Elements

3. **E remove()**
    - **Description**: Retrieves and removes the **head of the queue**.
    - **Throws**: NoSuchElementException if the queue is empty.
    - **Returns**: The head of the queue.

4. **E poll()**
    - **Description**: Retrieves and removes the **head of the queue**, or returns null if the queue is empty.
    - **Returns**: The head of the queue, or null.

Retrieving Elements

5. **E element()**
    - **Description**: Retrieves, but does not remove, the **head of the queue**.
    - **Throws**: NoSuchElementException if the queue is empty.
    - **Returns**: The head of the queue.
6. **E peek()**
    - **Description**: Retrieves, but does not remove, the **head of the queue**, or returns null if the queue is empty.
    - **Returns**: The head of the queue, or null.

Inspecting the Queue

7. **int size()**
    - **Description**: Returns the number of elements in the queue.
8. **boolean isEmpty()**
    - **Description**: Checks if the queue is empty.
    - **Returns**: true if the queue contains no elements.

Bulk Operations

9. **boolean addAll(Collection<? extends E> c)**
    - **Description**: Adds all elements in the specified collection to the queue.
    - **Throws**: ClassCastException if the collection contains elements incompatible with the queue's comparator.
    - **Returns**: true if the queue was modified.
10. **boolean removeAll(Collection<?> c)**
    - **Description**: Removes all elements in the queue that are also contained in the specified collection.
    - **Returns**: true if any elements were removed.
11. **boolean retainAll(Collection<?> c)**
    - **Description**: Retains only the elements in the queue that are contained in the specified collection.
    - **Returns**: true if the queue was modified.
12. **void clear()**
    - **Description**: Removes all elements from the queue.

Search and Comparison

13. **boolean contains(Object o)**
    - **Description**: Checks if the queue contains the specified element.
    - **Returns**: true if the queue contains the element.

14. **boolean containsAll(Collection<?> c)**
    - **Description**: Checks if the queue contains all elements in the specified collection.
    - **Returns**: true if all elements are present.

Iteration and Array Conversion

15. **Iterator<E> iterator()**
    - **Description**: Returns an iterator over the elements in the queue in no particular order.
    - **Returns**: An iterator for the queue.
16. **Object[] toArray()**
    - **Description**: Converts the queue into an array containing all elements in no particular order.
    - **Returns**: An array of the elements in the queue.
17. **<T> T[] toArray(T[] a)**
    - **Description**: Converts the queue into an array, using the specified array type, containing all elements in no particular order.
    - **Returns**: An array containing all elements.
18. **Spliterator<E> spliterator()**
    - **Description**: Creates a spliterator over the elements in the queue.
    - **Returns**: A Spliterator for traversing elements.

---

4. Key Characteristics

- **Ordering**: Elements are stored in a heap structure, ensuring that the head of the queue is the element with the highest priority (based on natural ordering or the comparator).
- **Null Handling**: PriorityQueue does not allow null elements.
- **Thread-Safety**: Not synchronized; for thread-safe usage.

3. DeQueue Methods:

The **Deque** interface in Java is part of the **java.util** package and extends the **Queue** interface. It represents a **double-ended queue**, meaning elements can be added to or removed from both ends of the queue.

Here is a detailed list of all methods provided by the **Deque** interface, categorized by functionality, with descriptions.

---

1. Methods for Adding Elements

1. **void addFirst(E e)**
    - **Description**: Inserts the specified element at the **front** of the deque.
    - **Throws**: IllegalStateException if the deque is full.
2. **void addLast(E e)**
    - **Description**: Inserts the specified element at the **end** of the deque.
    - **Throws**: IllegalStateException if the deque is full.

3. **boolean offerFirst(E e)**
    - **Description**: Inserts the specified element at the **front** of the deque.
    - **Returns**: true if the element was added, or false if the deque is full.
4. **boolean offerLast(E e)**
    - **Description**: Inserts the specified element at the **end** of the deque.
    - **Returns**: true if the element was added, or false if the deque is full.
5. **boolean add(E e)**
    - **Description**: Adds the specified element at the **end** of the deque. Equivalent to addLast(E e).
    - **Throws**: IllegalStateException if the deque is full.
6. **boolean offer(E e)**
    - **Description**: Adds the specified element at the **end** of the deque. Equivalent to offerLast(E e).
    - **Returns**: true if the element was added, or false if the deque is full.

---

2. Methods for Removing Elements

7. **E removeFirst()**
    - **Description**: Removes and returns the **first** element of the deque.
    - **Throws**: NoSuchElementException if the deque is empty.

8. **E removeLast()**
    - **Description**: Removes and returns the **last** element of the deque.
    - **Throws**: NoSuchElementException if the deque is empty.
9. **E pollFirst()**
    - **Description**: Removes and returns the **first** element of the deque, or returns null if the deque is empty.
10. **E pollLast()**
    - **Description**: Removes and returns the **last** element of the deque, or returns null if the deque is empty.
11. **boolean remove(Object o)**
    - **Description**: Removes the **first occurrence** of the specified element from the deque.
    - **Returns**: true if the deque was modified, or false if the element was not found.
12. **void clear()**
    - **Description**: Removes all elements from the deque.

---

3. Methods for Retrieving Elements

13. **E getFirst()**
    - **Description**: Retrieves, but does not remove, the **first** element of the deque.
    - **Throws**: NoSuchElementException if the deque is empty.
14. **E getLast()**
    - **Description**: Retrieves, but does not remove, the **last** element of the deque.

- **Throws**: NoSuchElementException if the deque is empty.

15. **E peekFirst()**
    - **Description**: Retrieves, but does not remove, the **first** element of the deque, or returns null if the deque is empty.
16. **E peekLast()**
    - **Description**: Retrieves, but does not remove, the **last** element of the deque, or returns null if the deque is empty.
17. **E element()**
    - **Description**: Retrieves, but does not remove, the **head** of the deque. Equivalent to getFirst().
    - **Throws**: NoSuchElementException if the deque is empty.
18. **E peek()**
    - **Description**: Retrieves, but does not remove, the **head** of the deque. Equivalent to peekFirst().
    - **Returns**: The head of the deque, or null if the deque is empty.

---

4. Methods for Iteration

19. **Iterator<E> iterator()**
    - **Description**: Returns an iterator over the elements in the deque in **proper sequence** (from first to last).
20. **Iterator<E> descendingIterator()**
    - **Description**: Returns an iterator over the elements in the deque in **reverse order** (from last to first).

---

5. Methods for Stack-like Operations

21. **void push(E e)**
    - **Description**: Pushes an element onto the **stack represented by the deque**. Equivalent to addFirst(E e).
22. **E pop()**
    - **Description**: Pops an element from the **stack represented by the deque**. Equivalent to removeFirst().
    - **Throws**: NoSuchElementException if the deque is empty.

---

6. Bulk Operations

23. **boolean contains(Object o)**
    - **Description**: Checks if the deque contains the specified element.
    - **Returns**: true if the deque contains the element.
24. **int size()**
    - **Description**: Returns the number of elements in the deque.
25. **boolean isEmpty()**
    - **Description**: Checks if the deque is empty.
    - **Returns**: true if the deque has no elements.
26. **boolean removeAll(Collection<?> c)**
    - **Description**: Removes all elements in the deque that are also contained in the specified collection.
    - **Returns**: true if the deque was modified.

27. **boolean retainAll(Collection<?> c)**
    - **Description**: Retains only the elements in the deque that are contained in the specified collection.
    - **Returns**: true if the deque was modified.
28. **Object[] toArray()**
    - **Description**: Returns an array containing all elements in the deque.
29. **<T> T[] toArray(T[] a)**
    - **Description**: Returns an array containing all elements in the deque, using the specified array type.

---

8. Key Points

- **Deque** can function as both a **FIFO queue** and a **LIFO stack**.
- **ArrayDeque** and **LinkedList** are common implementations of the Deque interface.
- Adding/removing elements from either end is efficient, as **Deque** is optimized for these operations.

4. ArrayDeque Methods:

The **ArrayDeque** class in Java is part of the **java.util** package and implements the **Deque** interface. It is a resizable array implementation of a double-ended queue, allowing efficient addition and removal of elements from both ends.

Here's a comprehensive list of all **ArrayDeque** methods along with their descriptions:

Constructor:

1. **ArrayDeque()**
    - **Description**: Creates an empty deque with an initial capacity of 16.
2. **ArrayDeque(int initialCapacity)**
    - **Description**: Creates an empty deque with the specified initial capacity.
3. **ArrayDeque(Collection<? extends E> c)**
    - **Description**: Creates a deque containing the elements of the specified collection, in the order they are returned by the collection's iterator.

---

2. Methods for Adding Elements

1. **void addFirst(E e)**
    - **Description**: Inserts the specified element at the **front** of the deque.
    - **Throws**: NullPointerException if the specified element is null.
2. **void addLast(E e)**
    - **Description**: Inserts the specified element at the **end** of the deque.
    - **Throws**: NullPointerException if the specified element is null.
3. **boolean offerFirst(E e)**
    - **Description**: Inserts the specified element at the **front** of the deque.

- **Returns**: true if the element was successfully added, or false if the deque is full.

4. **boolean offerLast(E e)**
    - **Description**: Inserts the specified element at the **end** of the deque.
    - **Returns**: true if the element was successfully added, or false if the deque is full.
5. **boolean add(E e)**
    - **Description**: Adds the specified element to the **end** of the deque. Equivalent to addLast(E e).
    - **Throws**: NullPointerException if the specified element is null.
6. **boolean offer(E e)**
    - **Description**: Adds the specified element to the **end** of the deque. Equivalent to offerLast(E e).
    - **Returns**: true if the element was successfully added, or false if the deque is full.

---

3. Methods for Removing Elements

1. **E removeFirst()**
    - **Description**: Removes and returns the **first** element of the deque.
    - **Throws**: NoSuchElementException if the deque is empty.
2. **E removeLast()**
    - **Description**: Removes and returns the **last** element of the deque.

- **Throws**: NoSuchElementException if the deque is empty.

3. **E pollFirst()**
    - **Description**: Removes and returns the **first** element of the deque, or returns null if the deque is empty.
4. **E pollLast()**
    - **Description**: Removes and returns the **last** element of the deque, or returns null if the deque is empty.
5. **boolean remove(Object o)**
    - **Description**: Removes the **first occurrence** of the specified element from the deque.
    - **Returns**: true if the element was found and removed, or false otherwise.
6. **E remove()**
    - **Description**: Removes and returns the **head** of the deque. Equivalent to removeFirst().
    - **Throws**: NoSuchElementException if the deque is empty.
7. **E poll()**
    - **Description**: Removes and returns the **head** of the deque, or returns null if the deque is empty. Equivalent to pollFirst().
8. **void clear()**
    - **Description**: Removes all elements from the deque.

---

4. Methods for Retrieving Elements

1. **E getFirst()**
    - **Description**: Retrieves, but does not remove, the **first** element of the deque.
    - **Throws**: NoSuchElementException if the deque is empty.
2. **E getLast()**
    - **Description**: Retrieves, but does not remove, the **last** element of the deque.
    - **Throws**: NoSuchElementException if the deque is empty.
3. **E peekFirst()**
    - **Description**: Retrieves, but does not remove, the **first** element of the deque, or returns null if the deque is empty.
4. **E peekLast()**
    - **Description**: Retrieves, but does not remove, the **last** element of the deque, or returns null if the deque is empty.
5. **E element()**
    - **Description**: Retrieves, but does not remove, the **head** of the deque. Equivalent to getFirst().
    - **Throws**: NoSuchElementException if the deque is empty.
6. **E peek()**
    - **Description**: Retrieves, but does not remove, the **head** of the deque. Equivalent to peekFirst().
    - **Returns**: null if the deque is empty.

---

5. Iteration Methods

1. **Iterator<E> iterator()**
    - **Description**: Returns an iterator over the elements in the deque in **proper sequence** (from first to last).
2. **Iterator<E> descendingIterator()**
    - **Description**: Returns an iterator over the elements in the deque in **reverse order** (from last to first).

---

6. Stack-Like Methods

1. **void push(E e)**
    - **Description**: Pushes an element onto the stack represented by the deque. Equivalent to addFirst(E e).
2. **E pop()**
    - **Description**: Pops an element from the stack represented by the deque. Equivalent to removeFirst().
    - **Throws**: NoSuchElementException if the deque is empty.

---

7. Bulk and Utility Methods

1. **boolean contains(Object o)**
    - **Description**: Checks if the deque contains the specified element.
    - **Returns**: true if the element is present, or false otherwise.
2. **boolean removeAll(Collection<?> c)**
    - **Description**: Removes all elements in the deque that are also contained in the specified collection.

- **Returns**: true if the deque was modified.

3. **boolean retainAll(Collection<?> c)**
   - **Description**: Retains only the elements in the deque that are contained in the specified collection.
   - **Returns**: true if the deque was modified.
4. **boolean isEmpty()**
   - **Description**: Checks if the deque is empty.
   - **Returns**: true if the deque contains no elements.
5. **int size()**
   - **Description**: Returns the number of elements in the deque.
6. **Object[] toArray()**
   - **Description**: Returns an array containing all elements in the deque.
7. **<T> T[] toArray(T[] a)**
   - **Description**: Returns an array containing all elements in the deque, using the specified array type.
8. **Spliterator<E> spliterator()**
   - **Description**: Returns a Spliterator over the elements in the deque.

==========END==========