QUEUE INTERFACE IN JAVA

The Queue Interface is present in java.util package and extends the Collection interface. It stores and processes the data in FIFO(First In First Out) order. It is an ordered list of objects limited to inserting elements at the end of the list and deleting elements from the start of the list.

- No Null Elements: Most implementations like PriorityQueue do not allow null elements.
- Implementation Classes: LinkedList , PriorityQueue, ArrayDeque, ConcurrentLinkedQueue (for thread-safe operations).
- Use Cases: Commonly used for Task scheduling, Message passing, and Buffer management in applications.
- Iteration: Supports iterating through elements. The order of iteration depends on the implementation.

```
// Java Program Implementing Queue Interface

import java.util.LinkedList;

import java.util.Queue;


public class QueueCreation {


    public static void main(String args[])
   {


      // Create a Queue of Integers using LinkedList

      Queue<Integer> q = new LinkedList<>();


      // Displaying the Queue

      System.out.println("Queue elements: " + q);
```

```
    }
}
```

The Queue interface provides several methods for adding, removing, and inspecting elements in the queue. Here are some of the most commonly used methods:

- add(element): Adds an element to the rear of the queue. If the queue is full, it throws an exception.
- offer(element): Adds an element to the rear of the queue. If the queue is full, it returns false.
- remove(): Removes and returns the element at the front of the queue. If the queue is empty, it throws an exception.
- poll(): Removes and returns the element at the front of the queue. If the queue is empty, it returns null.
- element(): Returns the element at the front of the queue without removing it. If the queue is empty, it throws an exception.
- peek(): Returns the element at the front of the queue without removing it. If the queue is empty, it returns null.

```
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();

        // add elements to the queue
        queue.add("apple");
```

```
        queue.add("banana");
        queue.add("cherry");

        // print the queue
        System.out.println("Queue: " + queue);

        // remove the element at the front of the queue
        String front = queue.remove();
        System.out.println("Removed element: " + front);

        // print the updated queue
        System.out.println("Queue after removal: " + queue);

        // add another element to the queue
        queue.add("date");

        // peek at the element at the front of the queue
        String peeked = queue.peek();
        System.out.println("Peeked element: " + peeked);

        // print the updated queue
        System.out.println("Queue after peek: " + queue);
    }
}
```

Queue: [apple, banana, cherry]

Removed element: apple

Queue after removal: [banana, cherry]

Peeked element: banana

Queue after peek: [banana, cherry, date]

```
// Java program to demonstrate a Queue

import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {

    public static void main(String[] args)
    {
        Queue<Integer> q
            = new LinkedList<>();

        // Adds elements {0, 1, 2, 3, 4} to
        // the queue
        for (int i = 0; i < 5; i++)
            q.add(i);

        // Display contents of the queue.
        System.out.println("Elements of queue " + q);
```

```
        // To remove the head of queue.
        int removedele = q.remove();
        System.out.println("removed element-"
                           + removedele);

        System.out.println(q);

        // To view the head of queue
        int head = q.peek();
        System.out.println("head of queue-"
                           + head);

        // Rest all methods of collection
        // interface like size and contains
        // can be used with this
        // implementation.
        int size = q.size();
        System.out.println("Size of queue-"
                           + size);
    }
}
```

```
Elements of queue [0, 1, 2, 3, 4]
removed element-0
[1, 2, 3, 4]
```

head of queue-1

Size of queue-4

Operations on Queue Interface

Let’s see how to perform a few frequently used operations on the queue using the Priority Queue class.

1. Adding Elements:

In order to add an element in a queue, we can use the add() method. The insertion order is not retained in the PriorityQueue. The elements are stored based on the priority order which is ascending by default.

Example: Java program to add elements

// to a Queue

```
import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        Queue<String> pq = new PriorityQueue<>();

        pq.add("harman");
        pq.add("samsung");
        pq.add("company");
```

```
        System.out.println(pq);
    }
}
```

Output

```
[company, harman, samsung]
```

2. Removing Elements:

In order to remove an element from a queue, we can use the remove() method. If there are multiple such objects, then the first occurrence of the object is removed. Apart from that, poll() method is also used to remove the head and return it.

Example Java program to remove elements

```
// from a Queue

import java.util.*;

public class GFG {

    public static void main(String args[])
    {
        Queue<String> pq = new PriorityQueue<>();

        pq.add("harman");
```

```
        pq.add("samsung");
        pq.add("comapny");

        System.out.println("Initial Queue " + pq);

        pq.remove("harman");

        System.out.println("After Remove " + pq);

        System.out.println("Poll Method " + pq.poll());

        System.out.println("Final Queue " + pq);
    }
}
```

Output

```
Initial Queue [harman, samsung, company]
After Remove [samsung, company]
Poll Method samsung
Final Queue [company]
```

3. Iterating the Queue:

There are multiple ways to iterate through the Queue. The most famous way is converting the queue to the array and traversing using the for loop. However, the queue also has an inbuilt iterator which can be used to iterate through the queue.

Example

```
// Java program to iterate elements

// to a Queue

import java.util.*;

public class GFG {


    public static void main(String args[])

    {

        Queue<String> pq = new PriorityQueue<>();


        pq.add("harman");

        pq.add("samsung");

        pq.add("company");


        Iterator iterator = pq.iterator();


        while (iterator.hasNext()) {

            System.out.print(iterator.next() + " ");

        }

    }

}
```

Characteristics of a Queue

The following are the characteristics of the queue:

- The Queue is used to insert elements at the end of the queue and removes from the beginning of the queue. It follows FIFO concept.
- The Java Queue supports all methods of Collection interface including insertion, deletion, etc.
- LinkedList, ArrayBlockingQueue and PriorityQueue are the most frequently used implementations.
- If any null operation is performed on BlockingQueues, NullPointerException is thrown.
- The Queues which are available in java.util package are Unbounded Queues.
- The Queues which are available in java.util.concurrent package are the Bounded Queues.
- All Queues except the Deques supports insertion and removal at the tail and head of the queue respectively. The Deques support element insertion and removal at both ends.