Implementing Queue in C++ and Java Using built-in- class

Queue in C++ STL

The Standard template Library in C++ offers a built-in implementation of the Queue data structure for simpler and easy use. The STL implementation of queue data structure implements all basic operations on queue such as enque(), deque(), clear() etc.

Syntax:

```
queue< data_type > queue_name;

where,
data_type is the type of element to be stored
in the queue.
queue_name is the name of the queue data structure.
```

The functions supported by std::queue are :

- **empty()** – Returns whether the queue is empty.
- **size()** – Returns the size of the queue.
- **swap()**: Exchange the contents of two queues but the queues must be of same type, although sizes may differ.
- **emplace()**: Insert a new element into the queue container, the new element is added to the end of the queue.
- **front() and back()**: front() function returns a reference to the first element of the queue. back() function returns a reference to the last element of the queue.
- **push(g) and pop()**: The push() function adds the element 'g' at the end of the queue. The pop() function deletes the first element of the queue.

```cpp
// CPP code to illustrate Queue in Standard Template Library (STL)
#include <iostream>
#include <queue>
using namespace std;
void showq(queue <int> gq)
{
    queue <int> g = gq;
    while (!g.empty())
    {
        cout << '\t' << g.front();
        g.pop();
    }
    cout << '\n';
}
int main()
{
    queue <int> gquiz;
    gquiz.push(10);
    gquiz.push(20);
    gquiz.push(30);
    cout << "The queue gquiz is : ";
    showq(gquiz);
    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();
    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showq(gquiz);
    return 0;
}
```

Output:

```
The queue gquiz is : 10 20 30

gquiz.size() : 3
gquiz.front() : 10
gquiz.back() : 30
gquiz.pop() : 20 30
```

Queue interface in Java

The Queue interface is available in java.util package and extends the Collection interface. The queue collection is used to hold the elements about to be processed and provides various operations like the insertion, removal etc. It is an ordered list of objects with its use limited to insert elements at the end of the list and deleting elements from the start of list i.e. it follows the FIFO or the First-In-First-Out principle. Being an interface the queue needs a concrete class for the declaration and the most commonly used classes are the PriorityQueue and LinkedList in Java. It is to be noted that both the implementations are not thread safe. PriorityBlockingQueue is one alternative implementation if thread safe implementation is needed.

Methods in Queue:

- **add()**- This method is used to add elements at the tail of the queue. More specifically, at the last of linkedlist if it is used, or according to the priority in case of priority queue implementation.
- **peek()**- This method is used to view the head of a queue without removing it. It returns Null if the queue is empty.
- **element()**- This method is similar to peek(). It throws NoSuchElementException when the queue is empty.
- **remove()**- This method removes and returns the head of the queue. It throws NoSuchElementException when the queue is empty.
- **poll()**- This method removes and returns the head of the queue. It returns null if the queue is empty.
- **size()**- This method returns the no. of elements in the queue.

Below is a simple Java program to demonstrate these methods:

```java
// Java orogram to demonstrate working of Queue interface in Java
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 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);
    }
}
```

Output:

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