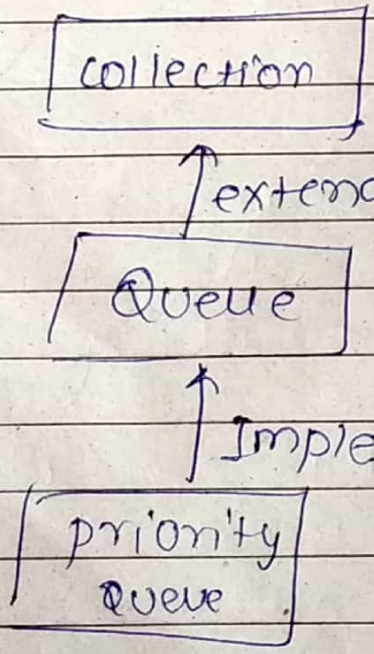
priority queue

A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority.
If elements with the same priority occur, they are served according to their order in the queue

In a queue, the first-in-first-out rule is implemented whereas, in a priority queue, the values are removed on the basis of priority. The element with the highest priority is removed first.



priority queue se element nikalte hai to sorted manner me nikalte hai

But jb hm insert krte hai to kisi v order me chla jaye,

Methods of priority Queue

- add()
- remove()
- elements()

Agr hme 10k elements diye hue hai and usme se hme top 5 elements nikalna hai to hme priority queue ko use krna hoga

n → elements में से top k elements nikalna hai toh sorting se nlogn में होगा and priority queue se nlogk me.

eg. 2, 1, 5, 3, 7, 4, 9, 1, 4, 5

Top 3 elements

p.q
2 (1) 5

smallest

so peek krne pe ye aega → 1 will be checked with 3, 3 is greater so 1 will be replaced with 3.

(2) 3 5	7 (3) 5	7 (4) 5	7 9 (5)	7 9 (5)

7 9 5

(n log 3)

ArrayDeque as a stack.

To implement a LIFO (last-In-First-Out) stacks in Java, it is recommended to use a deque over the stack class. The ArrayDeque class is likely to be faster than the stack class. ArrayDeque provides the following methods that can be used for implementing a stack.

push() → adds an element to the top of the stack.

peek() → returns an element from the top of the stack.

pop() → returns and removes an element from the top of the stack.

/* ArrayDeque are much fastes than stacks because stack are made from vector class & the funcns of vector class are thread safe.
And bcoz of arrayDeque does are not thread safe they are much faster than stack.
*/

Implementation of Deque & its func (AddTo Head & RemoveLast)

```
package deque;

public class MyDeque<E>{

    Node<E> head, tail;
    public void addToHead(E data){
        Node<E> toAdd = new Node<>(data);

        if(head == null){
            head = tail = toAdd;
            return;
        }

        head.next = toAdd;
        toAdd.pre = head;
        head = toAdd;
    }

public E removeLast(){
    if(head == null){
        return null;
    }
    Node(E) toRemove = tail;
    tail = tail.next;
    tail.pre = null;

    if(tail == null){
        head = null;
    }
    return toRemove.data;
}
```