**Queue:**

**1.Queue Interface:**

**public interface Queue<E> extends Collection<E>**

(See, an interface can extend other interface. It cannot implement it)

**boolean add(object):** It is used to insert the specified element into this queue and return true upon success.

**boolean offer(object):** It is used to insert the specified element into this queue.

Object remove() It is used to retrieves and removes the head of this queue.

**Object poll():** It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.

**Object element():** It is used to retrieves, but does not remove, the head of this queue.

**Object peek():** It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

**Classes With Implements Queue Interface:**

There is another interface named DeQue (for common Queue) and there is a class named PriorityQueue.

**PriorityQueue:**

**Example:**

import java.util.\*;

class TestCollection12

{

public static void main(String args[])

{

PriorityQueue<String> queue=new PriorityQueue<String>();

queue.add("Amit");

queue.add("Vijay");

queue.add("Karan");

queue.add("Jai");

queue.add("Rahul");

System.out.println("head:"+queue.element());

System.out.println("head:"+queue.peek());

System.out.println("iterating the queue elements:");

Iterator itr=queue.iterator();

//This is offered by Iterable

while(itr.hasNext())

{

System.out.println(itr.next());

}

queue.remove();

queue.poll();

System.out.println("after removing two elements:");

Iterator<String> itr2=queue.iterator();

while(itr2.hasNext())

{

//hasNext function is offered by Iterable interface

System.out.println(itr2.next());

}

}

}

**Some Important Note About The Functions PriorityQueue Class Offers:**

**The Difference Between add and offer:**

Now, what’s the difference between these two?

The two functions come from two different interfaces that PriorityQueue implements: (as PriorityQueue implements both. A collection,

add() comes from Collection.

offer() comes from Queue.

For a capacity-constrained queue, the difference is that add() always returns true and throws an exception if it can't add the element, whereas offer() is allowed to return false if it can't add the element.

An unbounded priority queue based on a priority heap. The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used. A priority queue does not permit null elements. A priority queue relying on natural ordering also does not permit insertion of non-comparable objects (doing so may result in ClassCastException).

PriorityQueue Example with Comparable Interface (Otherwise it chooses natural ordering. This ordering is not sorting. Remember that)

**An Example Of PriorityQueue With Comparable Interface: (So, the ordering is no more natural ordering, rather a user defined order)**

import java.util.Comparator;

import java.util.PriorityQueue;

import java.util.Queue;

import java.util.Random;

public class PriorityQueueExample {

public static void main(String[] args) {

//natural ordering example of priority queue

Queue<Integer> integerPriorityQueue = new PriorityQueue<>(7);

Random rand = new Random();

for(int i=0;i<7;i++){

integerPriorityQueue.add(new Integer(rand.nextInt(100)));

}

for(int i=0;i<7;i++){

Integer in = integerPriorityQueue.poll();

System.out.println("Processing Integer:"+in);

}

//PriorityQueue example with Comparator

Queue<Customer> customerPriorityQueue = new PriorityQueue<>(7, idComparator);

addDataToQueue(customerPriorityQueue);

pollDataFromQueue(customerPriorityQueue);

}

//Comparator anonymous class implementation

public static Comparator<Customer> idComparator = new Comparator<Customer>(){

@Override

public int compare(Customer c1, Customer c2) {

return (int) (c1.getId() - c2.getId());

}

};

//utility method to add random data to Queue

private static void addDataToQueue(Queue<Customer> customerPriorityQueue) {

Random rand = new Random();

for(int i=0; i<7; i++){

int id = rand.nextInt(100);

customerPriorityQueue.add(new Customer(id, "Pankaj "+id));

}

}

//utility method to poll data from queue

private static void pollDataFromQueue(Queue<Customer> customerPriorityQueue) {

while(true){

Customer cust = customerPriorityQueue.poll();

if(cust == null) break;

System.out.println("Processing Customer with ID="+cust.getId());

}

}

}

Here , you will see the comparable interface is passed as second argument to constructor of PriorityQueue

**2.Dequeue Interface:**

**Interface Deque<E>**

**First Element(Head)** **Last Element (Tail)**

**Throws exception Special value Throws exception Special value**

Insert addFirst(e) offerFirst(e) addLast(e) offerLast(e)

Remove removeFirst() pollFirst() removeLast() pollLast()

Examine getFirst() peekFirst() getLast() peekLast()

**Other Important Functions:**

**iterator():** Returns a an iterator for this deque.

**descendingIterator():** Returns an iterator that has the reverse order for this deque.

**Known Classes Which Implements DeQue Interface:**

ArrayDeque, ConcurrentLinkedDeque