**Queue Collections in Java**

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, the first element is removed first and the last element is removed at last. This interface is dedicated to storing all the elements where the order of the elements matter.

The Queue interface of the Java collections framework provides the functionality of the queue data structure. It extends the Collection interface. Since the Queue is an interface, we cannot provide the direct implementation of it.

**Classes that implement Queue Interface in Java:**

In order to use the functionalities of Queue, we need to use classes that implement it:

1. **Priority Queue**
2. **Dequeue**

**Methods of Queue Interface**

1. **add():**Inserts the specified element into the queue. If the task is successful, add() returns true, if not it throws an exception.
2. **offer():** Inserts the specified element into the queue. If the task is successful, offer() returns true, if not it returns false.
3. **element()**: Returns the head of the queue. Throws an exception if the queue is empty.
4. **peek():** Returns the head of the queue. Returns null if the queue is empty.
5. **remove()**: Returns and removes the head of the queue. Throws an exception if the queue is empty.
6. **poll()**: Returns and removes the head of the queue. Returns null if the queue is empty.

**Priority Queue Collection in Java:**

It implements the Queue interface. The PriorityQueue class provides the functionality of the heap data structure. The PriorityQueue class provides the facility of using a queue. But it does not order the elements in a FIFO manner. It is based on Priority Heap.

The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

**Creating Priority Queue**  
**Syntax : PriorityQueue<Integer> numbers = new PriorityQueue<Integer>();**  
Here, we have created a priority queue without any arguments. In this case, the head of the priority queue is the smallest element of the queue. And elements are removed in ascending order from the queue.

**Example to demonstrate Priority Queue in Java**

**import** *java.util.*\*;

**class** PriorityQueueDemo

**{**

**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 **()**;

**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 **())**

**{**

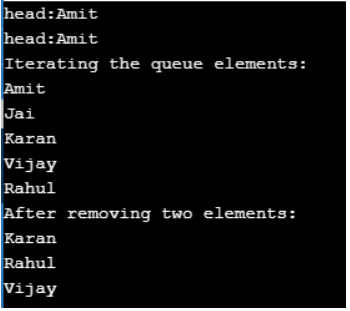
System.out.println **(**itr2.next **())**;

**}**

**}**

**}**

**Output:**



**PriorityQueue Example with Complex Data type in Java:**

**import** *java.util.Objects*;

**import** *java.util.PriorityQueue*;

**class** Employee **implements** Comparable **<** Employee **>**

**{**

**private** **String** name;

**private** **double** salary;

**public** Employee **(String** name, **double** salary**)**

**{**

this.name = name;

this.salary = salary;

**}**

**public** **String** getName **()**

**{**

**return** name;

**}**

**public** **void** setName **(String** name**)**

**{**

this.name = name;

**}**

**public** **double** getSalary **()**

**{**

**return** salary;

**}**

**public** **void** setSalary **(double** salary**)**

**{**

this.salary = salary;

**}**

@Override **public** **boolean** equals **(**Object o**)**

**{**

**if** **(**this == o**)**

**return** **true**;

**if** **(**o == **null** || getClass **()** != o.getClass **())**

**return** **false**;

Employee employee = **(**Employee**)** o;

**return** **Double**.compare **(**employee.salary, salary**)** == 0 && Objects.equals **(**name, employee.name**)**;

**}**

@Override **public** **int** hashCode **()**

**{**

**return** Objects.hash **(**name, salary**)**;

**}**

@Override **public** **String** toString **()**

**{**

**return** "Employee{" + "name='" + name + '\'' + ", salary=" + salary + '}';

**}**

// Compare two employee objects by their salary

@Override **public** **int** compareTo **(**Employee employee**)**

**{**

**if** **(**this.getSalary **()** **>** employee.getSalary **())**

**{**

**return** 1;

**}**

**else** **if** **(**this.getSalary **()** **<** employee.getSalary **())**

**{**

**return** -1;

**}**

**else**

**{**

**return** 0;

**}**

**}**

**}**

**public** **class** PriorityQueueDemo

**{**

**public** **static** **void** main **(String[]**args**)**

**{**

// Create a PriorityQueue

PriorityQueue **<** Employee **>** employeePriorityQueue = new PriorityQueue **<>** **()**;

// Add items to the Priority Queue

employeePriorityQueue.add **(**new Employee **(**"Rajeev", 100000.00**))**;

employeePriorityQueue.add **(**new Employee **(**"Chris", 145000.00**))**;

employeePriorityQueue.add **(**new Employee **(**"Andrea", 115000.00**))**;

employeePriorityQueue.add **(**new Employee **(**"Jack", 167000.00**))**;

/\*

The compareTo() method implemented in the Employee class is used to determine

in what order the objects should be dequeued.

\*/

**while** **(**!employeePriorityQueue.isEmpty **())**

**{**

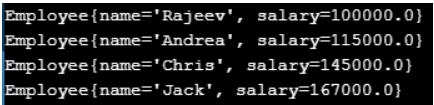
System.out.println **(**employeePriorityQueue.remove **())**;

**}**

**}**

**}**

**Output:**



**DeQueue Collection in Java**

Deque is an acronym for “double-ended queue”. Java Deque Interface is a linear collection that supports element insertion and removal at both ends. The class which implements this interface is ArrayDeque. It extends the Queue interface. Deque is an interface and has two implementations: LinkedList and ArrayDeque.

**Creating a Deque**  
**Syntax : Deque dq = new LinkedList();**  
**Deque dq = new ArrayDeque();**

**Methods of Deque**

1. **addFirst():**Adds the specified element at the beginning of the deque. Throws an Exception if the deque is full.
2. **addLast():**Adds the specified element at the end of the deque. Throws an exception if the deque is full.
3. **offerFirst():** Adds the specified element at the beginning of the deque. Returns false if the deque is full.
4. **offerLast():**Adds the specified element at the end of the deque. Returns false if the deque is full.
5. **getFirst():** Returns the first element of the deque. Throws an exception if the deque is empty.
6. **getLast():**Returns the last element of the deque. Throws an exception if the deque is empty.
7. **peekFirst():** Returns the first element of the deque. Returns null if the deque is empty.
8. **peekLast()**: Returns the last element of the deque. Returns null if the deque is empty.
9. **removeFirst():** Returns and removes the first element of the deque. Throws an exception if the deque is empty.
10. **removeLast():** Returns and removes the last element of the deque. Throws an exception if the deque is empty.
11. **pollFirst():**Returns and removes the first element of the deque. Returns null if the deque is empty.
12. **pollLast():** Returns and removes the last element of the deque. Returns null if the deque is empty.
13. **push():** Adds an element at the beginning of the deque.
14. **pop():** Removes an element from the beginning of the deque.
15. **peek():** Returns an element from the beginning of the deque.

**ArrayDeque Collection in Java:**

The ArrayDeque class provides the facility of using deque and resizable-array. It inherits the AbstractCollection class and implements the Deque interface. This is a special kind of array that grows and allows users to add or remove an element from both sides of the queue. Array deques have no capacity restrictions and they grow as necessary to support usage. Array Implementation of Deque

**Syntax : Deque<String> animal1 = new ArrayDeque<String>();**

**Example to demonstrate ArrayDeque Collection in Java**

**import** *java.util.Deque*;

**import** *java.util.ArrayDeque*;

**class** ArrayDequeDemo **{**

**public** **static** **void** main**(String[]** args**)** **{**

// Creating Deque using the ArrayDeque class

Deque**<Integer>** numbers = new ArrayDeque**<>()**;

// add elements to the Deque

numbers.offer**(**1**)**;

numbers.offerLast**(**2**)**;

numbers.offerFirst**(**3**)**;

System.out.println**(**"Deque: " + numbers**)**;

// Access elements of the Deque

**int** firstElement = numbers.peekFirst**()**;

System.out.println**(**"First Element: " + firstElement**)**;

**int** lastElement = numbers.peekLast**()**;

System.out.println**(**"Last Element: " + lastElement**)**;

// Remove elements from the Deque

**int** removedNumber1 = numbers.pollFirst**()**;

System.out.println**(**"Removed First Element: " + removedNumber1**)**;

**int** removedNumber2 = numbers.pollLast**()**;

System.out.println**(**"Removed Last Element: " + removedNumber2**)**;

System.out.println**(**"Updated Deque: " + numbers**)**;

**}**

**}**

**Output:**

