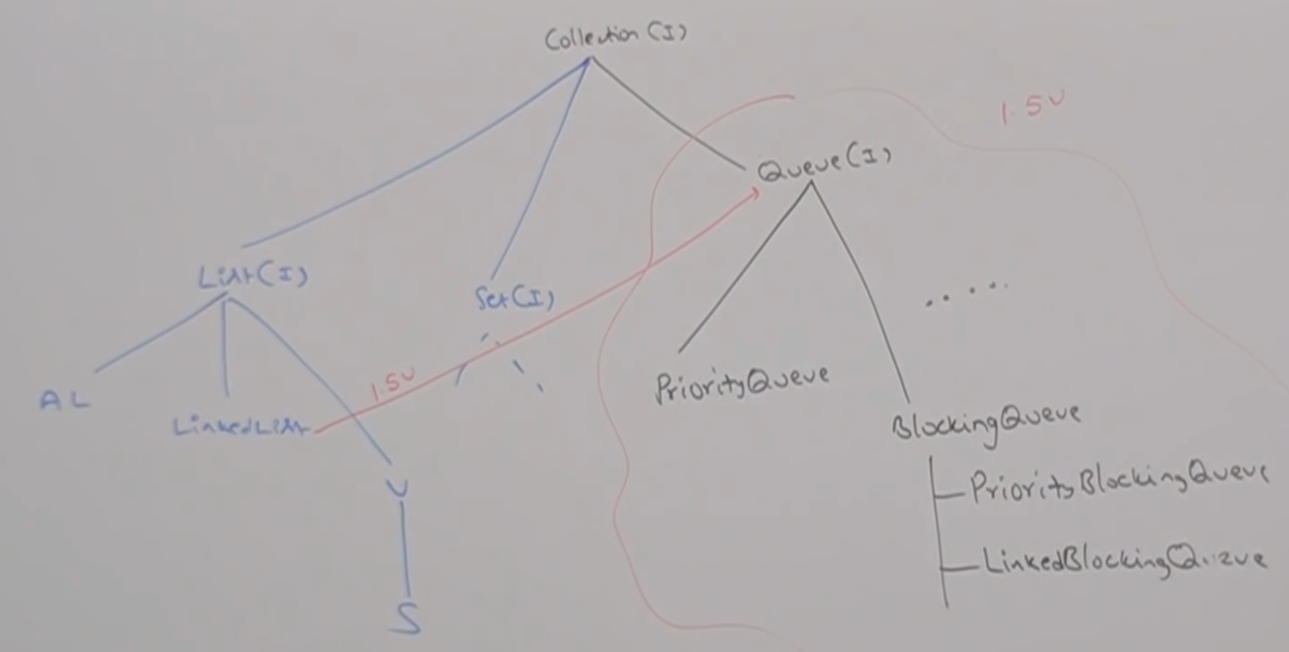
**Collection Framework – Part\_15**

* **1.5 version enhancement (Queue interface):**

It is the child interface of Collection.



If we want to represent a group of individual objects prior to processing then we should go for Queue.

For example, before sending SMS message all mobile numbers we have to store all mobile numbers in some data strucutures. In which order we added mobile numbers in the same order only message should be sent/delivered. For this First In First Out (FIFO) requirement, Queue is the best choice.

Usually, Queue follows FIFO order but based on our requirement we can implement our own priority order also (PriorityQueue).

From 1.5 vesion onwards LinkedList class also implements Queue interface.

LinkedList based implementation of Queue always following FIFO order.

* **Queue Interface specific methods:**

boolean offer(Object o);

To add an object into the queue.

Object peek()

To return head element of the queue. If queue is empty then this method returns null.

Object element()

To return head element of the queue. If queue is empty then this method raises RE: NoSuchElementException

Object poll()

To remove and return head element of the queue. If queue is empty then this method returns null.

Object remove()

To remove and return head of the queue. If the queue is empty then this method raises RE: NoSuchElementException

* **PriorityQueue:**

1. If we want to represent a group of individual objects prior to processing according to some priority then we should go for PriorityQueue.
2. The priority can be either default natural sorting order or customized sorting order defined by Comparator.
3. Insertion order is not preserved and it is based on some priority.
4. Duplicate objects are not allowed.
5. If we are depending on default natural sorting order, compulsory the objects should be homogeneous and comparable, otherwise we will get runtime exception saying:ClassCastException.

If we are defining our own sorting by Comparator, then objects need not be homogeneous and comparable.

1. Null is not allowed even as the first element also.

* **Constructors:**

PriorityQueue q = new PriorityQueue();

Creates an empty PriorityQueue with default initial capacity of 11 and all objects will be inserted according to default natural sorting order.

PriorityQueue q = new PriorityQueue(int intialCapacity);

PriortyQueue q = new PriorityQueue(SortedSet s);

PriorityQueue q = new PriorityQueue(Collection c);

* **Example:**

import java.util.\*;

class PriorityQueueDemo{

public static void main(String[] args){

PriorityQueue q = new PriorityQueue();

System.out.println(q.peek());// null

System.out.println(q.element());//NoSuchElementException

for(int i=0;i<=10;i++){

q.offer(i);

}:

System.out.println(q);//[0,1,2,…3]

System.out.println(q.poll()); //0

System.out.println(q); //[1,2,3,4…10]

}

}

Note:

Some platforms won’t provide proper support for Thread priorities and PriorityQueue’s

* **Example\_02:**

import java.util.\*;

class PriorityQueueDemo{

public static void main(String[] args){

PriorityQueue q = new PriorityQueue(15, new MyComparator());

q.offer(“A”);

q.offer(“Z”);

q.offer(“L”);

q.offer(“B”);

System.out.println(q); // [Z,L,B,A]

}

}

class MyComparator implements Comparator{

public int compare(Object obj1, Object obj2){

String s1 = (String) obj1;

String s2 = obj2.toString();

return s2.compareTo(s1);

}

}