```
public class PriorityDemo

{


    public static void main(String args[])

    {

        //Define PriorityQueue's class object is q

        PriorityQueue q=new PriorityQueue();


        //Input values

         q.add("X", 10);

         q.add("Y", 1);

         q.add("Z", 3);


         //Output

        System.out.println(" " +q.remove()); // Returns X

        System.out.println(" " +q.remove()); // Returns Z

        System.out.println(" " +q.remove()); // Returns Y


    }

}

import java.util.*;

public class PriorityQueue

{


    private ArrayList queueArray;

    private ArrayList priorityArray;


    //Constructor

    public PriorityQueue()
```

```
{
        queueArray=new ArrayList();
        priorityArray=new ArrayList();
}
//Adds the items in the queue list by using priority
public <T> void add(T itemAdd,int priorityAdd)
{
        queueArray.add(itemAdd);
        priorityArray.add(priorityAdd);

}
public boolean isEmpty()
{
        return queueArray.size()==0;

}

public <T> T remove()
{
        int max=0,maxPriority=0;
        for(int i=0;i<queueArray.size();i++)
        {
                if(priorityArray.get(i) > maxPriority)
                {
                        max=i;
                        maxPriority=priorityArray.get(i);
                }
        }
        priorityArray.remove(max);
```

```
                return (T) queueArray.remove(max);

        }


}
```

Implement Priority Queue using Linked Lists.

- push(): This function is used to insert a new data into the queue.
- pop(): This function removes the element with the highest priority form the queue.
- peek() / top(): This function is used to get the highest priority element in the queue without removing it from the queue.
    - The list is so created so that the highest priority element is always at the head of the list. The list is arranged in descending order of elements based on their priority. This allow us to remove the highest priority element in O(1) time. To insert an element we must traverse the list and find the proper position to insert the node so that the overall order of the priority queue is maintained. This makes the push() operation takes O(N) time. The pop() and peek() operations are performed in constant time.
    - **Algorithm :**

```
PUSH(HEAD, DATA, PRIORITY)
Step 1: Create new node with DATA and PRIORITY
Step 2: Check if HEAD has lower priority. If true follow Steps 3-4 and end. Else goto Step 5.
Step 3: NEW -> NEXT = HEAD
Step 4: HEAD = NEW
Step 5: Set TEMP to head of the list
Step 6: While TEMP -> NEXT != NULL and TEMP -> NEXT -> PRIORITY > PRIORITY
Step 7: TEMP = TEMP -> NEXT
[END OF LOOP]
Step 8: NEW -> NEXT = TEMP -> NEXT
Step 9: TEMP -> NEXT = NEW
Step 10: End
POP(HEAD)
Step 2: Set the head of the list to the next node in the list. HEAD = HEAD -> NEXT.
Step 3: Free the node at the head of the list
Step 4: End
PEEK(HEAD):
```

```
Step 1: Return HEAD -> DATA
Step 2: End
```

- **CODE:**

```
import java.util.* ;

class Solution
{


// Node
static class Node {
   int data;

   // Lower values indicate higher priority
   int priority;

   Node next;

}

static Node node = new Node();

// Function to Create A New Node
static Node newNode(int d, int p)
{
   Node temp = new Node();
   temp.data = d;
   temp.priority = p;
   temp.next = null;
```

```
    return temp;
}

// Return the value at head
static int peek(Node head)
{
    return (head).data;
}

// Removes the element with the
// highest priority form the list
static Node pop(Node head)
{
    Node temp = head;
    (head) = (head).next;
    return head;
}

// Function to push according to priority
static Node push(Node head, int d, int p)
{
    Node start = (head);

    // Create new Node
    Node temp = newNode(d, p);

    // Special Case: The head of list has lesser
    // priority than new node. So insert new
    // node before head node and change head node.
    if ((head).priority > p) {

        // Insert New Node before head
        temp.next = head;
        (head) = temp;
    }
    else {

        // Traverse the list and find a
        // position to insert new node
        while (start.next != null &&
            start.next.priority < p) {
            start = start.next;
        }

        // Either at the ends of the list
        // or at required position
        temp.next = start.next;
        start.next = temp;
    }
    return head;
}

// Function to check is list is empty
static int isEmpty(Node head)
{
    return ((head) == null)?1:0;
}
```

```
// Driver code
public static void main(String args[])
{
    // Create a Priority Queue
    // 7.4.5.6
    Node pq = newNode(4, 1);
    pq =push(pq, 5, 2);
    pq =push(pq, 6, 3);
    pq =push(pq, 7, 0);

    while (isEmpty(pq)==0) {
        System.out.printf("%d ", peek(pq));
        pq=pop(pq);
    }

}
}
```

Output:

```
7  4  5  6
```