题目：

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

**Example:**

**Input:** 1->2->4, 1->3->4

**Output:** 1->1->2->3->4->4

先说说Queue这个容器吧。

queue：队列。

队列是一种特殊的线性表，是一种先进先出（FIFO）的数据结构。它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作。进行插入操作的端称为队尾，进行删除操作的端称为队头。队列中没有元素时，称为空队列。

在java5中新增加了java.util.Queue接口，用以支持队列的常见操作。该接口扩展了java.util.Collection接口。

在Java里面Queue是一个接口，它只是定义了一个基本的Queue应该有哪些功能规约。

基本功能：

offer 添加一个元素并返回true 如果队列已满，则返回false

poll 移除并返问队列头部的元素 如果队列为空，则返回null

peek 返回队列头部的元素 如果队列为空，则返回null

put 添加一个元素 如果队列满，则阻塞

take 移除并返回队列头部的元素 如果队列为空，则阻塞

element 返回队列头部的元素 如果队列为空，则抛出NoSuchElementException异常

add 增加一个元索 如果队列已满，则抛出IIIegaISlabEepeplian异常

remove移除并返回队列头部的元素 如果队列为空，则抛出NoSuchElementException异常。

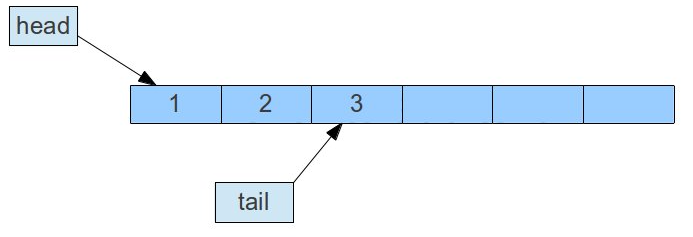
Queue使用时要尽量避免Collection的add()和remove()方法，而是要使用offer()来加入元素，使用poll()来获取并移出元素。它们的优点是通过返回值可以判断成功与否，add()和remove()方法在失败的时候会抛出异常。 如果要使用前端而不移出该元素，使用element()或者peek()方法。

Queue接口与List、Set同一级别，都是继承了Collection接口。LinkedList实现了Queue接 口。我们可以把LinkedList当成Queue来用。但是Queue接口窄化了对LinkedList的方法的访问权限（即在方法中的参数类型如果是Queue时，就完全只能访问Queue接口所定义的方法 了，而不能直接访问 LinkedList的非Queue的方法），以使得只有恰当的方法才可以使用。

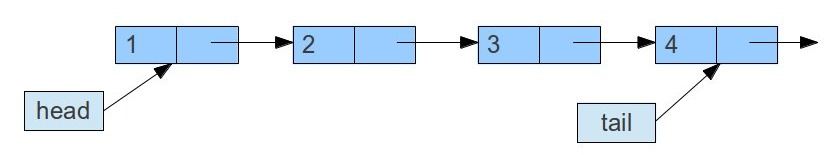
java中具有Queue功能的类主要有如下几个：AbstractQueue, ArrayBlockingQueue, ConcurrentLinkedQueue, LinkedBlockingQueue, DelayQueue, LinkedList, PriorityBlockingQueue, PriorityQueue和ArrayDqueue。最基本的是LinkedList和ArrayDqueue。PriorityQueue是优先级队列。

Queue本身是一种先入先出的模型(FIFO)，和我们日常生活中的排队模型很类似。根据不同的实现，他们主要有数组和链表两种实现形式。如下图：

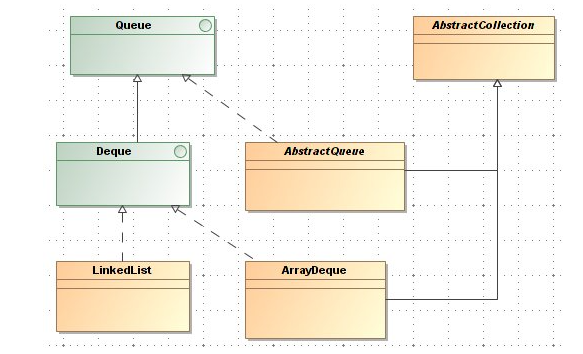
数组：ArrayDeque



链表：LinkedList



在jdk里几个常用队列实现之间的类关系图如下：



Queue作为一个接口，它声明的几个基本操作无非就是入队和出队的操作。具体操作如下：

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

boolean add(E e); // 添加元素到队列中，相当于进入队尾排队。

boolean offer(E e); //添加元素到队列中，相当于进入队尾排队.

E remove(); //移除队头元素

E poll(); //移除队头元素

E element(); //获取但不移除队列头的元素

E peek(); //获取但不移除队列头的元素

}

Queue接口加入的offer和poll方法虽然和collection接口中的add和remove方法作用大致相同，但是更符合场景。

Deque也是一个接口，它继承了Queue的接口规范。Deque是一个双向队列，意味着它不过是对Queue接口的增强。我们要讨论的LinkedList和ArrayDeque都是实现Deque接口。

PriorityQueue是个基于优先级堆的极大优先级队列。

此队列按照在构造时所指定的顺序对元素排序，既可以根据元素的自然顺序来指定排序（参阅 Comparable）， 也可以根据 Comparator 来指定，这取决于使用哪种构造方法。优先级队列不允许 null 元素。 依靠自然排序的优先级队列还不允许插入不可比较的对象（这样做可能导致 ClassCastException）。 此队列的头是按指定排序方式的最小元素。如果多个元素都是最小值，则头是其中一个元素——选择方法是任意的。

PriorityQueue内部实现是在数组的基础上实现的。优先级队列是无界的，但是有一个内部容量，控制着用于存储队列元素的数组的大小。 它总是至少与队列的大小相同。随着不断向优先级队列添加元素，其容量会自动增加。无需指定容量增加策略的细节。但是此实现不是同步的。不是线程安全的。如果多个线程中的任意线程从结构上修改了列表， 则这些线程不应同时访问 PriorityQueue 实例，这时请使用线程安全的PriorityBlockingQueue 类。

注意：

1. 优先级队列是用数组实现，但是数组大小可以动态增加，容量无限。

2. 线程不安全。PriorityBlockingQueue是线程安全的优先级队列。

3. 不允许使用 null 元素。

4. 方法iterator()中提供的迭代器并不保证以有序的方式遍历优先级队列中的元素。 PriorityQueue对元素采用的是堆排序，头是按指定排序方式的最小元素。堆排序只能保证根是最大（最小），整个堆并不是有序的。 方法iterator()中提供的迭代器可能只是对整个数组的依次遍历。也就只能保证数组的第一个元素是最小的。

如果需要按顺序遍历，请考虑使用 Arrays.sort(pq.toArray())。

本道题我用的是优先级队列，但是看看了具体内容发现可能方法不是很好，因为在第一次排序的时候能排好，但是如果你从堆中取出一个元素，该堆为了维持最小堆便不能保证是按顺序排列的了，只能保证第一个元素为最小元素。

我的解题想法就是，把两个链表中的数字都拿出来然后排好序再构成链表。想图省事用到了优先级队列。

下面就主要说说答案吧。

就是逐次比较两个链表中的单元数值，较小的添加到结果的链表中，然后较小元素的那条链向后遍历一个单元在和下一条链的单元比，小加入，大的话第二条链子的那个单元加入到结果链中，然后向后遍历一个单元和上一条链子单元比较。直到其中一条链子结束为止。若1链结束，2链就直接加入到结果的链子中。代码如下：

public static ListNode mergeTwoLists\_A(ListNode l1, ListNode l2) {

ListNode head = new ListNode();

ListNode p = new ListNode();

p=head;

l1=l1.next;

l2=l2.next;

while(l1!=null&&l2!=null){

if(l1.val<l2.val){

p.next=l1;

l1=l1.next;

}

else {

p.next=l2;

l2=l2.next;

}

p=p.next;

}

if (l1 == null) {

p.next = l2;

} else {

p.next = l1;

}

return head;

}