**线性表**

**线性表的通用特性：线性表的通用特性在List接口中定义。**

**1.从线性表中提取一个元素。**

**2.向线性表中插入一个元素。**

**3.从线性表中删除一个元素。**

**4.找出线性表中元素的个数。**

**5.确定线性表中是否包含某个元素。**

**6.确定线性表是否为空。**

**数组线性表**

**数组一旦创建之后，它的大小就没法改变。尽管如此，仍然可以使用数组来实现动态的数据结构。处理的方法是，当数组不能再存储线性表中的新元素时，创建一个更大的新数组来替换当前的数组。**

**链表**

**链表采用链接结构实现。**

**由于ArrayList是用数组实现的，所以get和set方法可以通过下标访问和修改元素，也可以用add方法在线性表末尾添加元素，它们是高效的。但是，add和remove方法的效率很低，因为这两个方法需要移动潜在的大量元素。为提高在表中开始位置添加和删除元素的效率，可以采用链式结构来实现线性表。**

**栈和队列**

**可以使用数组线性表实现栈，使用链表实现队列。**

**两种方法可用来设计栈和队列的类。**

**1.使用继承：可以通过继承数组线性表类ArrayList来定义栈类，通过继承链表类LinkedList来定义队列类。**

**2.使用组合：可以将数组线性表定义为栈类中的数据域，将链表定义为队列类中的数据域。**

**组合更好一些，因为它可以定义一个全新的栈类和队列类，而不需要继承数组线性表类与链表类中不必要和不合适的方法。**

**如下，组合方式实现队列类GenericQueue**

**public class GenericQueue<E> {**

**private java.util.LinkedList<E> list**

**= new java.util.LinkedList<E>();**

**public void enqueue(E e) {**

**list.addLast(e);**

**}**

**public E dequeue() {**

**return list.removeFirst();**

**}**

**public int getSize() {**

**return list.size();**

**}**

**@Override**

**public String toString() {**

**return "Queue: " + list.toString();**

**}**

**}**

**优先队列**

**可以使用堆实现优先队列。**

**普通的队列是一种先进先出的数据结构，元素在队列尾追加，而从队列头删除。在优先队列（priority queue）中，元素被赋予优先级。当访问元素时，具有最高优先级的元素最先删除。**

**可以使用堆来实现优先队列，其中根结点是队列中具有最高优先级的对象。**

**如下：**

**public class MyPriorityQueue<E extends Comparable<E>> {**

**private Heap<E> heap = new Heap<E>();**

**public void enqueue(E newObject) {**

**heap.add(newObject);**

**}**

**public E dequeue() {**

**return heap.remove();**

**}**

**public int getSize() {**

**return heap.getSize();**

**}**

**}**

**小结：**

**1.定义一个数据结构本质上是定义一个类。为数据结构定义的类应该使用数据域来存储数据，并提供方法来支持诸如插入和删除等操作。**

**2.创建一个数据结构是从该类创建一个实例。这样就可以将方法应用在实例上来处理数据结构，比如插入一个元素到数据结构中，或者从数据结构中删除一个元素。**

**3.采用堆来实现一个优先队列**