```java
/**
 * EECS233 Written HW3
 * Tung Ho Lin
 */

public class PriorityQueue<T extends Comparable<T>> {

  private T[] items;

  private int numItems;

  private int maxItems;

  public PriorityQueue(int maxSize) {
    items = (T[]) new Comparable[maxSize];
    maxItems = maxSize;
    numItems = 0;
  }

  private boolean isEmpty() {
    return numItems==0;
  }

  public void insert(T item) {
    if(numItems == maxItems) {
      T[] olditems = items;
      items = (T[]) new Comparable[numItems*2 + 1];
      for(int i=0; i<olditems.length; i++)
        items[i] = olditems[i];
    }
    items[numItems] = item;
    numItems++;
    siftUp(numItems-1);
  }

  public T removeMax() {
    T toRemove = items[0];
    items[0] = items[numItems-1];
    numItems--;
    siftDown(0);
    return toRemove;
  }

  public void siftUp(int i) {
    T toSift = items[i];
    int child = i;
    int parent = (i-1)/2;
    while(parent > 0 && items[child].compareTo(items[parent]) > 0) { //if the child is larger than the parent
      items[child] = items[parent];
      items[parent] = toSift;
      child = parent;
      parent = (child-1)/2;
    }
    items[parent] = toSift;
  }

  public void siftDown(int i) {
    T toSift = items[i];
    int parent = i;
    int child = parent*2 + 1;  //child to compare with; start with left child
    while(child < numItems) {
      if(child + 1 < numItems && items[child].compareTo(items[child + 1]) < 0) //if the right child exists and is larger than the left child
        child += 1;
      if(toSift.compareTo(items[child]) >= 0) //if the parent is larger or equal to the child
        break;  //siftDown is complete
      items[parent] = items[child];
      items[child] = toSift;
```

```
68        parent = child;
69        child = parent*2 + 1;
70      }
71        items[parent] = toSift;
72    }
73  }
74
75
76
77
```