Assignment 2: Priority Queues

Writeup by **Vincent Chan**, ID **815909699**, account **masc0264**

Code

OrderedArrayPriorityQueue.java

```java
/*
 * Vincent Chan
 * masc0264
 */
package data_structures;

import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.ConcurrentModificationException;

public class OrderedArrayPriorityQueue<E> implements PriorityQueue<E> {
  /*Functions Included
   * ====PUBLIC====
   * Constructor                           //Ln.32
   * insert(Object)                        //Ln.44
   * remove()                              //Ln.56
   * peek()                                //Ln.65
   * contains(Object)                      //Ln.73
   * size()                                //Ln.79
   * clear()                               //Ln.84
   * isEmpty()                             //Ln.90
   * isFull()                              //Ln.96
   * iterator()                            //Ln.100
   *
   * ====PRIVATE====
   * find(Object, startIndex, endIndex)         //Ln.109
   * binSearch(Object, startIndex, endIndex)    //Ln.118
   */
  // Variable Declarations
  private int size, maxSize, modCtr;
  private E[] vectorArray;

  //Constructor
  public OrderedArrayPriorityQueue() {
    size = modCtr = 0;
    maxSize = DEFAULT_MAX_CAPACITY;
    vectorArray = (E[])new Object[DEFAULT_MAX_CAPACITY];
  } //End constructor
  public OrderedArrayPriorityQueue(int max) {
    size = modCtr = 0;
    maxSize = max;
    vectorArray = (E[])new Object[maxSize];
  } //End constructor

  //Inserts a new object into the priority queue.
  //Returns False if the queue is full.
  public boolean insert(E object) {
    if(isFull()) return false;
    int insertLoc = find(object, 0, size-1);
    for(int i=size; i>insertLoc; i--) vectorArray[i] = vectorArray[i-1];
    vectorArray[insertLoc] = object;
    size++;
    modCtr++;
    return true;
  } //End insert()

  //Removes the top priority object that has been
  //in the queue the longest.
  //returns null if empty
  public E remove() {
    if(isEmpty()) return null;
    modCtr++;
    return vectorArray[--size];
  } //End remove()

  //Returns but does not remove the object
  //with highest priority that has been in
  //the queue the longest. Returns null if empty
  public E peek() {
    if(isEmpty()) return null;
    return vectorArray[size-1];
  } //End peek()

  //Returns true of the object is in the array
  public boolean contains(E obj) {
    return binSearch(obj, 0, size-1);
  } //End contains()

  //Returns the current size.
  public int size() {
    return size;
  } //End size()

  //Clears the array and replaces it with a new one.
  public void clear() {
    size = 0;
    vectorArray = (E[])new Object[maxSize];
  } //End clear()

  //Returns true if the array is empty.
  public boolean isEmpty() {
    return size==0;
  } //End isEmpty()

  //Returns true if the array is full.
  public boolean isFull() {
    return size==maxSize;
  } //End isFull()

  //Returns an iterator to use in iterating
  public Iterator<E> iterator() {
    return new IteratorHelper();
  } //End iterator()

  //This will return the index of where an element should be inserted
  private int find(E obj, int low, int hi) {
    //Termination condition: Found the insertion point.
    if(hi<low) return low;

    //Compare the middle of the array to the sought object
    int mid = (low+hi)/2;
    int comp = ((Comparable<E>)obj).compareTo(vectorArray[mid]);

    //If not found, recursively call the function with a refined search area.
    return (comp>0)?find(obj, low, mid-1) : find(obj, mid+1, hi);
  } //End find()

  private boolean binSearch(E obj, int low, int hi) {
    if(hi<low) return false;
    int mid = (low+hi)/2;
    int comp = ((Comparable<E>)obj).compareTo(vectorArray[mid]);
    if(comp==0) return true;
    //If not found, recursively call the function with a refined search area.
    return (comp>0)?binSearch(obj, low, mid-1) : binSearch(obj, mid+1, hi);
    }


    class IteratorHelper implements Iterator<E> {
      int iterIndex, modChk;

      public IteratorHelper() {
        modChk = modCtr;
        iterIndex = size-1;
      }

      public boolean hasNext() {
        return iterIndex>=0;
      }

      public E next() {
        if(!hasNext()) throw new NoSuchElementException();
        if(modCtr!=modChk) throw new ConcurrentModificationException();
        return vectorArray[iterIndex--];
      }

      public void remove() {
        throw new UnsupportedOperationException();
      }
    }
  } //End OrderedArrayPriorityQueue
```