Assignment 2: Priority Queues

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

Code

OrderedArrayPriorityQueue.java

```
1  /*
2   * Vincent Chan
3   * masc0264
4   */
5  package data_structures;
6
7  import java.util.Iterator;
8  import java.util.NoSuchElementException;
9  import java.util.ConcurrentModificationException;
10
11 public class OrderedArrayPriorityQueue<E> implements PriorityQueue<E> {
12   /*Functions Included
13    * ====PUBLIC====
14    * Constructor                          //Ln.32
15    * insert(Object)                       //Ln.44
16    * remove()                             //Ln.56
17    * peek()                               //Ln.65
18    * contains(Object)                     //Ln.73
19    * size()                               //Ln.79
20    * clear()                              //Ln.84
21    * isEmpty()                            //Ln.90
22    * isFull()                             //Ln.96
23    * iterator()                           //Ln.100
24    *
25    * ====PRIVATE====
26    * find(Object, startIndex, endIndex)       //Ln.109
27    * binSearch(Object, startIndex, endIndex)  //Ln.118
28    */
29   // Variable Declarations
30   private int size, maxSize, modCtr;
31   private E[] vectorArray;
32
33   //Constructor
34   public OrderedArrayPriorityQueue() {
35     size = modCtr = 0;
36     maxSize = DEFAULT_MAX_CAPACITY;
37     vectorArray = (E[])new Object[DEFAULT_MAX_CAPACITY];
38   } //End constructor
39   public OrderedArrayPriorityQueue(int max) {
40     size = modCtr = 0;
41     maxSize = max;
42     vectorArray = (E[])new Object[maxSize];
43   } //End constructor
44
45   //Inserts a new object into the priority queue.
46   //Returns False if the queue is full.
47   public boolean insert(E object) {
48     if(isFull()) return false;
49     int insertLoc = find(object, 0, size-1);
50     for(int i=size; i>insertLoc; i--) vectorArray[i] = vectorArray[i-1];
51     vectorArray[insertLoc] = object;
52     size++;
53     modCtr++;
54     return true;
55   } //End insert()
56
```

```
57    //Removes the top priority object that has been
58    //in the queue the longest.
59    //returns null if empty
60    public E remove() {
61      if(isEmpty()) return null;
62      modCtr++;
63      return vectorArray[--size];
64    } //End remove()
65
66    //Returns but does not remove the object
67    //with highest priority that has been in
68    //the queue the longest. Returns null if empty
69    public E peek() {
70      if(isEmpty()) return null;
71      return vectorArray[size-1];
72    } //End peek()
73
74    //Returns true of the object is in the array
75    public boolean contains(E obj) {
76      return binSearch(obj, 0, size-1);
77    } //End contains()
78
79    //Returns the current size.
80    public int size() {
81      return size;
82    } //End size()
83
84    //Clears the array and replaces it with a new one.
85    public void clear() {
86      size = 0;
87      vectorArray = (E[])new Object[maxSize];
88    } //End clear()
89
90    //Returns true if the array is empty.
91    public boolean isEmpty() {
92      return size==0;
93    } //End isEmpty()
94
95    //Returns true if the array is full.
96    public boolean isFull() {
97      return size==maxSize;
98    } //End isFull()
99
100   //Returns an iterator to use in iterating
101   public Iterator<E> iterator() {
102     return new IteratorHelper();
103   } //End iterator()
104
105   //This will return the index of where an element should be inserted
106   private int find(E obj, int low, int hi) {
107     //Termination condition: Found the insertion point.
108     if(hi<low) return low;
109
110     //Compare the middle of the array to the sought object
111     int mid = (low+hi)/2;
112     int comp = ((Comparable<E>)obj).compareTo(vectorArray[mid]);
113
114     //If not found, recursively call the function with a refined search area.
115     return (comp>0)?find(obj, low, mid-1) : find(obj, mid+1, hi);
116   } //End find()
117
118   private boolean binSearch(E obj, int low, int hi) {
119     if(hi<low) return false;
120     int mid = (low+hi)/2;
121     int comp = ((Comparable<E>)obj).compareTo(vectorArray[mid]);
122     if(comp==0) return true;
123     //If not found, recursively call the function with a refined search area.
124     return (comp>0)?binSearch(obj, low, mid-1) : binSearch(obj, mid+1, hi);
```

```
125     }
126
127
128     class IteratorHelper implements Iterator<E> {
129       int iterIndex, modChk;
130
131       public IteratorHelper() {
132         modChk = modCtr;
133         iterIndex = size-1;
134       }
135
136       public boolean hasNext() {
137         return iterIndex>=0;
138       }
139
140       public E next() {
141         if(!hasNext()) throw new NoSuchElementException();
142         if(modCtr!=modChk) throw new ConcurrentModificationException();
143         return vectorArray[iterIndex--];
144       }
145
146       public void remove() {
147         throw new UnsupportedOperationException();
148       }
149     }
150   } //End OrderedArrayPriorityQueue
```