

UnorderedArrayPriorityQueue.java

```
1  /* Vincent Chan
2   * masc0264
3   */
4  package data_structures;
5
6  import java.util.Iterator;
7  import java.util.NoSuchElementException;
8  import java.util.ConcurrentModificationException;
9
10 public class UnorderedArrayPriorityQueue<E> implements PriorityQueue<E> {
11
12     /* Functions Included
13     * ====PUBLIC====
14     * constructors           //Ln.30
15     * insert(object)         //Ln.40
16     * remove()               //Ln.47
17     * peek()                 //Ln. 68
18     * contains(Object)       //Ln. 83
19     * size()                 //Ln. 91
20     * clear()                //Ln. 96
21     * isEmpty()              //Ln. 102
22     * isFull()               //Ln. 107
23     * iterator()             //Ln. 112
24     */
25
26     // Variable Declarations
27     private int size, maxSize, modCtr;
28     private E[] vectorArray = (E[])new Object[DEFAULT_MAX_CAPACITY];
29
30     //Constructor
31     public UnorderedArrayPriorityQueue() {
32         size = modCtr = 0;
33         maxSize = DEFAULT_MAX_CAPACITY;
34     } //End constructor
35     public UnorderedArrayPriorityQueue(int max) {
36         size = modCtr = 0;
37         maxSize = max;
38     } //End constructor
39
40     //Insert will insert an object at the end of the array.
41     public boolean insert(E object) {
42         if(isFull()) return false;
43         vectorArray[size++] = object;
44         return true;
45     } //End insert()
46
47     //Will remove the highest priority object.
48     //Ties are broken by which was inserted first.
49     public E remove() {
50         if(isEmpty()) return null;
51         int lastIndex = 0;
52
53         //Finding what to remove: compare priority.
54         int comp;
55         for(int i=1; i<size; i++) {
56             comp=((Comparable<E>))vectorArray[lastIndex].compareTo(vectorArray[i]);
57             if(comp>0) lastIndex = i;
58         }
59
60         //Cleanup and retrun: Shift array and return object.
61         E temp = vectorArray[lastIndex];
62         size--;
63         for(;lastIndex<size;lastIndex++)
64             vectorArray[lastIndex] = vectorArray[lastIndex+1];
```

```

65     return temp;
66 } //End remove()
67
68 //Returns object of highest priority,
69 //does NOT remove it.
70 public E peek() {
71     if(isEmpty()) return null;
72     int lastIndex = 0;
73
74     int comp;
75     for(int i=1; i<size; i++) {
76         comp=((Comparable<E>)vectorArray[lastIndex]).compareTo(vectorArray[i]);
77         if(comp>0) lastIndex = i;
78     }
79
80     return vectorArray[lastIndex];
81 } //End peek()
82
83 //Will check the queue to see if
84 //the specified object is a component.
85 public boolean contains(E obj) {
86     for(int i=0; i<size; i++)
87         if(((Comparable<E>)obj).compareTo(vectorArray[i])==0) return true;
88     return false;
89 } //End contains()
90
91 //Returns the current size of the queue.
92 public int size() {
93     return size;
94 } //End size()
95
96 //Clears the queue.
97 public void clear() {
98     vectorArray = (E[])new Object[maxSize];
99     size = 0;
100 } //End clear()
101
102 //Returns true if queue is empty.
103 public boolean isEmpty() {
104     return size==0;
105 } //End isEmpty()
106
107 //Returns true if the queue is full.
108 public boolean isFull() {
109     return size == maxSize;
110 } //End isFull()
111
112 //Returns an iterator to use for iterating
113 public Iterator<E> iterator() {
114     return new IteratorHelper();
115 } //End iterator()
116
117 class IteratorHelper implements Iterator<E> {
118     int iterIndex;
119     int modChk;
120
121     public IteratorHelper() {
122         iterIndex = 0;
123         modChk = modCtr;
124     }
125
126     public boolean hasNext() {
127         return iterIndex<size;
128     }
129
130     public E next() {
131         if(!hasNext()) throw new NoSuchElementException();
132         if(modCtr != modChk) throw new ConcurrentModificationException();

```

```
133         return vectorArray[iterIndex++];
134     }
135
136     public void remove() {
137         throw new UnsupportedOperationException();
138     }
139 }
140 } //End UnorderedListPriorityQueue
```