UnorderedArrayPriorityQueue.java

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

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

public class UnorderedArrayPriorityQueue<E> implements PriorityQueue<E> {

  /* Functions Included
   * ====PUBLIC====
   * constructors        //Ln.30
   * insert(object)      //Ln.40
   * remove()            //Ln.47
   * peek()              //Ln. 68
   * contains(Object)    //Ln. 83
   * size()              //Ln. 91
   * clear()             //Ln. 96
   * isEmpty()           //Ln. 102
   * isFull()            //Ln. 107
   * iterator()          //Ln. 112
   */

  // Variable Declarations
  private int size, maxSize, modCtr;
  private E[] vectorArray = (E[])new Object[DEFAULT_MAX_CAPACITY];

  //Constructor
  public UnorderedArrayPriorityQueue() {
    size = modCtr = 0;
    maxSize = DEFAULT_MAX_CAPACITY;
  } //End constructor
  public UnorderedArrayPriorityQueue(int max) {
    size = modCtr = 0;
    maxSize = max;
  } //End constructor

  //Insert will insert an object at the end of the array.
  public boolean insert(E object) {
    if(isFull()) return false;
    vectorArray[size++] = object;
    return true;
  } //End insert()

  //Will remove the highest priority object.
  //Ties are broken by which was inserted first.
  public E remove() {
    if(isEmpty()) return null;
    int lastIndex = 0;

    //Finding what to remove: compare priority.
    int comp;
    for(int i=1; i<size; i++) {
      cmp=((Comparable<E>)vectorArray[lastIndex]).compareTo(vectorArray[i]);
      if(comp>0) lastIndex = i;
    }

    //Cleanup and retrun: Shift array and return object.
    E temp = vectorArray[lastIndex];
    size--;
    for(;lastIndex<size;lastIndex++)
      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)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
```