```java
import java.util.NoSuchElementException;
import java.util.Random;

@SuppressWarnings({"unchecked"})

public class Heap {
    private Comparable[] heapArray;
    private static final int SIZE = 1000;
    private int size = 0;

    //I wasn't sure where to put these since we aren't allowed
    //making other testing classes.
    private static Random random = new Random();
    private static final String[] CATEGORY = {
        "Life-threatening", "Chronic", "Major fracture", "Walk-in"
    };

    public Heap() {
            heapArray = new Comparable[SIZE];
    }

    private void bubbleUp(){
        for(int childIndex = size-1, parentIndex = childIndex/2;
                childIndex != parentIndex
                && heapArray[childIndex].compareTo(heapArray[parentIndex]) < 0;
                childIndex = parentIndex, parentIndex /= 2){

            System.out.println("bubbleUp():  "
                    + childIndex + "    " + parentIndex);

            Comparable temp = heapArray[childIndex];
            heapArray[childIndex] = heapArray[parentIndex];
            heapArray[parentIndex] = temp;
        }

        System.out.println("bubbleUp() complete");
    }

//wont work for uneven elements
    private void bubbleDown(){
        for(int parentIndex = 0, childToCompare = -1;
                true;
                parentIndex = childToCompare){

            int firstChildIndex = 2 * parentIndex + 1;
            int secondChildIndex = firstChildIndex + 1;

            //if just one child, skip the first compareTo().
            //This means the first born is the last element in the array.
            if(firstChildIndex >= this.size()){ //reached the end
                break;
            } else if(secondChildIndex >= this.size()){
                childToCompare = firstChildIndex;
            } else { //parent has two chidren
                int result =
                        heapArray[firstChildIndex]
                        .compareTo(heapArray[parentIndex]);

                childToCompare =
                        (result < 0) ? firstChildIndex : secondChildIndex;
            }

            int result =
                    heapArray[childToCompare].compareTo(heapArray[parentIndex]);

            if(result < 0){ //swap the parent and the child
                Comparable temp = heapArray[childToCompare];

                heapArray[childToCompare] = heapArray[parentIndex];
                heapArray[parentIndex] = temp;
            } else {
                break;
            }
```

```
74
75          }
76      }
77
78      public void insert(Comparable item){
79          try{
80              heapArray[size] = item;
81              size++;
82          } catch(ArrayIndexOutOfBoundsException x){
83              increaseSize();
84              insert(item);
85  //      } catch(NullPointerException x){
86              //only one element in array
87          }
88
89          bubbleUp();
90      }
91
92      public boolean isEmpty(){
93          return size == 0;
94      }
95
96      public int size(){
97          return size;
98      }
99
100     public Comparable getRootItem(){
101         return heapArray[0];
102     }
103
104     private void increaseSize(){
105         Comparable[] newHeapArray = new Comparable[heapArray.length * 2];
106
107         for(int index=0; index<size; index++){
108             newHeapArray[index] = heapArray[index];
109         }
110
111         heapArray = newHeapArray;
112     }
113
114     public Comparable removeRootItem(){
115         Comparable toReturn = heapArray[0];
116 //double check if decreases size AFTER the assignment
117         heapArray[0] = heapArray[--size];
118
119         bubbleDown();
120
121         return toReturn;
122     }
123
124     public static void main(String[] aaarg){
125         Heap heap = new Heap();
126
127         for(int index=10; index>0; index--){
128             System.out.println("insert():  " + index);
129 //          heap.insert(random.nextInt());
130             heap.insert(index);
131             System.out.println("getRootItem();  "
132                     + ((Integer)(heap.getRootItem())).intValue());
133 //          heap.insert(new TestingPatient());
134         }
135
136         System.out.println("insert() items:");
137         heap.printArray();
138
139         System.out.println("removeRootItem() a few times:");
140         heap.removeRootItem();
141         heap.removeRootItem();
142         heap.removeRootItem();
143
144         heap.printArray();
145
146     }
```

```
147
148     private void printArray(){
149
150         for(int index=0; index<this.size(); index++){
151             System.out.println(
152                     "heapArray[" + index + "] = " + heapArray[index]);
153         }
154
155         System.out.println();
156     }
157
158     private class TestingPatient extends ER_Patient{
159
160         public TestingPatient() {
161             super(CATEGORY[random.nextInt(CATEGORY.length)]);
162         }
163     }
164 }
165
166 /*
167     insert() items:
168     heapArray[0] = 1
169     heapArray[1] = 2
170     heapArray[2] = 3
171     heapArray[3] = 5
172     heapArray[4] = 4
173     heapArray[5] = 8
174     heapArray[6] = 9
175     heapArray[7] = 6
176     heapArray[8] = 10
177     heapArray[9] = 7
178
179     removeRootItem() a few times:
180     heapArray[0] = 6
181     heapArray[1] = 7
182     heapArray[2] = 3
183     heapArray[3] = 10
184     heapArray[4] = 4
185     heapArray[5] = 8
186     heapArray[6] = 9
187 */
```