Maps/HeapsCompetency#9

Heap	Mastered 1 point	Progressing 0.001 points	Novice 0 points	Criterion Score
Can design and implement a heap				/ 1

Explain what is hashing:

- A different way to search through data structures by an integer. Hashing navigates the data structure by comparing an integer, which has been referenced by an element with its key. Instead of searching the data structure with element, it searches it with an integer
-

Explain the differences among linear probing, quadratic probing, and double hashing:

- Linear probing is searching inside the hash table
- Quadratic probing is searching quadratically inside the hash table
- Double hashing is searching inside the hash table by hashing a key twice

Heap.java

```
1  package MapsHeapsWeek;
2
3  import java.util.NoSuchElementException;
4
5  public class Heap<E extends Comparable>{
6      private Object[] data;
7      private int manyItems;
8
9      public Heap(int manyItems) {
10         data = new Object[manyItems];
11         manyItems = 0;
12     }
13
14     public void insert(E e) {
15         data[manyItems++] = e;
16
17     }
18
19     public E remove() {
20         Object output = data[0];
21         data[0] = data[manyItems - 1];
22         manyItems--;
23         heapifyDown(0);
24         return (E) output;
25     }
26
27     public E peek() {
28         if(isEmpty()) {
29             throw new NoSuchElementException();
30         } else {
31             return (E) data[1];
32         }
33     }
34
35     public boolean isEmpty() {
36         boolean status = false;
37         if(data.length == 0) {
38             status = true;
39         }
40
41         for(Object o: data) {
42             if(o != null) {
43                 status = false;
44             }
45         }
46         return status;
47
48     }
49
```

```
49
50      private void heapifyUp(int index) {
51          int parentIndex = parent(index);
52          if(index > 0 && ((Comparable) data[index]).compareTo(data[parentIndex]) > 0) {
53              swap(index, parentIndex);
54              heapifyUp(parentIndex);
55          }
56      }
57
58      private void swap(int index1, int index2) {
59          Object temp = data[index1];
60          data[index1] = data[index2];
61          data[index2] = temp;
62      }
63
64      private void heapifyDown(int index) {
65          int largest = index;
66          int left = left(index);
67          int right = right(index);
68
69          if(left < manyItems && ((Comparable) data[left]).compareTo(data[largest]) > 0) {
70              largest = left;
71          }
72
73          if(right < manyItems && ((Comparable) data[right]).compareTo(data[largest]) > 0) {
74              largest = right;
75          }
76
77          if(largest != index) {
78              swap(largest, index);
79              heapifyDown(largest);
80          }
81      }
82
83      private int left(int parentIndex) {
84          return 2 * parentIndex + 1;
85      }
86
87      private int right(int parentIndex) {
88          return 2 * parentIndex +2;
89      }
90
91      private int parent(int childIndex) {
92          return (childIndex -1) /2 ;
93      }
94  }
95
```

HeapDriver.java

PriorityQueue/src/MapsHeapsWeek/HeapDriver.java

```java
public class HeapDriver {

    public static void main(String[] args) {
        Heap<String> heap = new Heap(10);
        heap.insert("Hello");
        heap.insert("World");
        heap.insert("Tuesday");



        PriorityQueue queue = new PriorityQueue(heap);
    queue.add("Wednesday");
queue.poll();
System.out.println(queue.peek());
System.out.println(queue.isEmpty());



    }

}

```

PriorityQueue.java

```java
package MapsHeapsWeek;

public class PriorityQueue<E extends Comparable> {
    private Heap<E> heap;

    public PriorityQueue(Heap<E> heap) {
        this.heap = heap;
    }

    public void add(E e) {
        heap.insert(e);
    }

    public E poll() {
        return heap.remove();
    }

    public E peek() {
        return heap.peek();
    }

    public boolean isEmpty() {
        return heap.isEmpty();
    }

}

```