How to implement a Priority Queue?

- **Keep them sorted!**
  (Haven’t we implemented it already?)
  - Appropriate if the number of items is small

- Sorted Array-based implementation

- Linked List-based implementation

- Binary search tree implementation

Priority “Queue”

Reading LDC Ch 18
A minheap is a complete binary tree in which each node’s element is less than or equal to both of its children.

A minheap keeps the smallest element readily available.

A maxheap is a complete binary tree in which each node’s element is greater than or equal to both of its children.

A maxheap keeps the greatest element readily available.

Three primary operations for maxheaps:
- Add new element to the heap
- Find the maximum element
- Remove the maximum element

Which Data Structure to extend to create a heap?

Maxheap, Minheap, or Neither?

```java
package javafoundations;
public interface MaxHeap<T extends Comparable<T>>
    extends BinaryTree<T> { }
```

```java
// MaxHeap.java  Java Foundations
// Defines the interface to a max heap.

public interface MaxHeap<T extends Comparable<T>>
    extends BinaryTree<T> { }
```
• Insert new course: 349 in a maxheap!

- **Strategy**
  - Insert newItem into the bottom of the tree
  - “Trickle up” new item to appropriate spot in the tree

Heaps: heapify

- **Step 3:** Transform the semiheap back into a heap by “trickling down” the smallest-of-three element
  - Performed by the recursive calls to heapify

- Remove old course: 331.

- **Step 1:** Delete (& remember to return) the item in the root
  - Results in disjoint heaps

- **Step 2:** Copy the item from the last node into the root, resulting in a *semiheap*

- Using LinkedBinaryTree

- Using ComputedLinkArrayTree

- Which is better?
Can we use a Heap to sort?

- **Algorithm:**
  
  - Efficiency?
  
  - More efficient strategy
    - The second half of the array represents a bunch of (one-node) heaps
    - Use heapify() to fix the first-half nodes

---

```java
package javafoundations;
public interface MaxHeap<T extends Comparable<T>> extends BinaryTree<T>
{
    // Adds the specified object to the heap.
    public void add(T obj);

    // Returns a reference to the element with the highest value in
    // the heap.
    public T getMax();

    // Removes and returns the element with the highest value in the
    // heap.
    public T removeMax();
}
```

Since `PriorityQueue` implements `Queue`...
package javafoundations;
import javafoundations.exceptions.*;

public class PriorityQueue<T> extends Comparable<T> implements Queue<T> {
    private LinkedMaxHeap<T> heap;

    public PriorityQueue() {
        heap = new LinkedMaxHeap<T>();
    }

    public T first() {
        return heap.getMax();
    }

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

    public int size() {
        return heap.size();
    }

    public String toString() {
        return heap.toString();
    }

    public void enqueue(T element) {
        heap.add(element);
    }

    public T dequeue() {
        try {
            return heap.removeMax();
        }
        catch(EmptyCollectionException ece) {
            System.out.println(ece);
        }
        return null;
    }
}

(more...)