Priority Queue Implementations

- Keep them sorted! (Have we implemented it already?)
  - Appropriate if the number of items is small

- Sorted Array-based implementation

- Linked List-based implementation

- Binary search tree implementation
A minheap is a complete binary tree in which each element is less than or equal to both of its children.

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

A minheap keeps the smallest valued element readily available.

A maxheap keeps the largest valued element readily available.

Which Data Structure to extend to create a heap?

Two binary tree extensions are present in a heap:

- It is a complete tree
- Its elements must be Comparable

Three primary operations for minheaps:

- Add new element to the heap
- Find the minimum value
- Remove the minimum value

Maxheap, Minheap, or Neither?

- 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

```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();
}
```
- 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"

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

How many times do we need to call heapify?

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

- Using LinkedBinaryTree
- Using ComputedLinkArrayTree
- Which is better?
Since PriorityQueue implements Queue...

```java
package javafoundations;
public interface PriorityQueue<T> extends Comparable<T>>
  implements Queue<T> {
  private LinkedMaxHeap<T> heap;
  // Creates a new, empty priority queue.
  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();
}

//-------------------------------------------------------------
// Enqueues a Comparable element.
//-------------------------------------------------------------
public void enqueue(T element)
{
    heap.add(element);
}

//-------------------------------------------------------------
// Dequeues the max of all elements in the heap.
//-------------------------------------------------------------
public T dequeue()
{
    try {
        T temp = heap.removeMax();
        return temp;
    } catch (EmptyCollectionException ece) {
        System.out.println(ece);
    }
    return null;
}

(more...)