Queue vs Priority Queue

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

MaxHeaps, MinHeaps and Heaps

- 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 minheap keeps the smallest valued element readily available
  - A maxheap keeps the ______ valued element readily available

- Which data Structure to extend to create a heap?
- Two binary tree extensions are needed in a heap
  - It is a binary tree (a complete one)
  - 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?

javafoundations.MaxHeap

```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();
}
```

Adding an Element to a MaxHeap

- Insert new course: 349 in a maxheap!

Removing the Max from a MaxHeap

- Remove old course: 331.
Heapify: Fixing a heap with a small problem

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

Can we use a Heap to sort?

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

(Max)Heapsort example

Heapsort efficiency?
Next: Use LinkedMaxHeap to implement PQs

```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...

```java
package javafoundations;
public interface Queue<T> {
    // Adds the specified element to the rear of the queue.
    public void enqueue(T element);
    // Removes and returns the element at the front of the queue.
    public T dequeue();
    // Returns a reference to the element at the front of the queue without removing it.
    public T first();
    // Returns true if the queue contains no elements and false otherwise.
    public boolean isEmpty();
    // Returns the number of elements in the queue.
    public int size();
    // Returns a string representation of the queue.
    public String toString();
}
```

The PriorityQueue Class

```java
package javafoundations;
import javafoundations.exceptions.*;
public class PriorityQueue<T extends Comparable<T>> implements Queue<T> {
    private LinkedMaxHeap<T> heap;
    // Creates a new, empty priority queue.
    public PriorityQueue() {
        heap = new LinkedMaxHeap<T>();
    }
    // Enqueues an 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...)
The PriorityQueue Class

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

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

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

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