COMP20010

Data Structures and Algorithms I

09 - Tutorial: Stacks

Dr. Aonghus Lawlor
aonghus.lawlor@ucd.ie

Queues

- Implement the Queue ADT using an Array type
- you should implement the following interface:

```
public interface Queue<E> {
     /**
      * Returns the number of elements in the queue.
      * @return number of elements in the queue
      */
     int size();

     /**
      * Tests whether the queue is empty.
      * @return true if the queue is empty, false otherwise
      */
     boolean isEmpty();

     /**
      * Inserts an element at the rear of the queue.
      * @param e  the element to be inserted
      */
     void enqueue(E e);

     /**
      * Returns, but does not remove, the first element of the queue.
      * @return the first element of the queue (or null if empty)
      */
     E first();

     /**
      * Removes and returns the first element of the queue.
      * @return element removed (or null if empty)
      */
     E dequeue();
   }
```

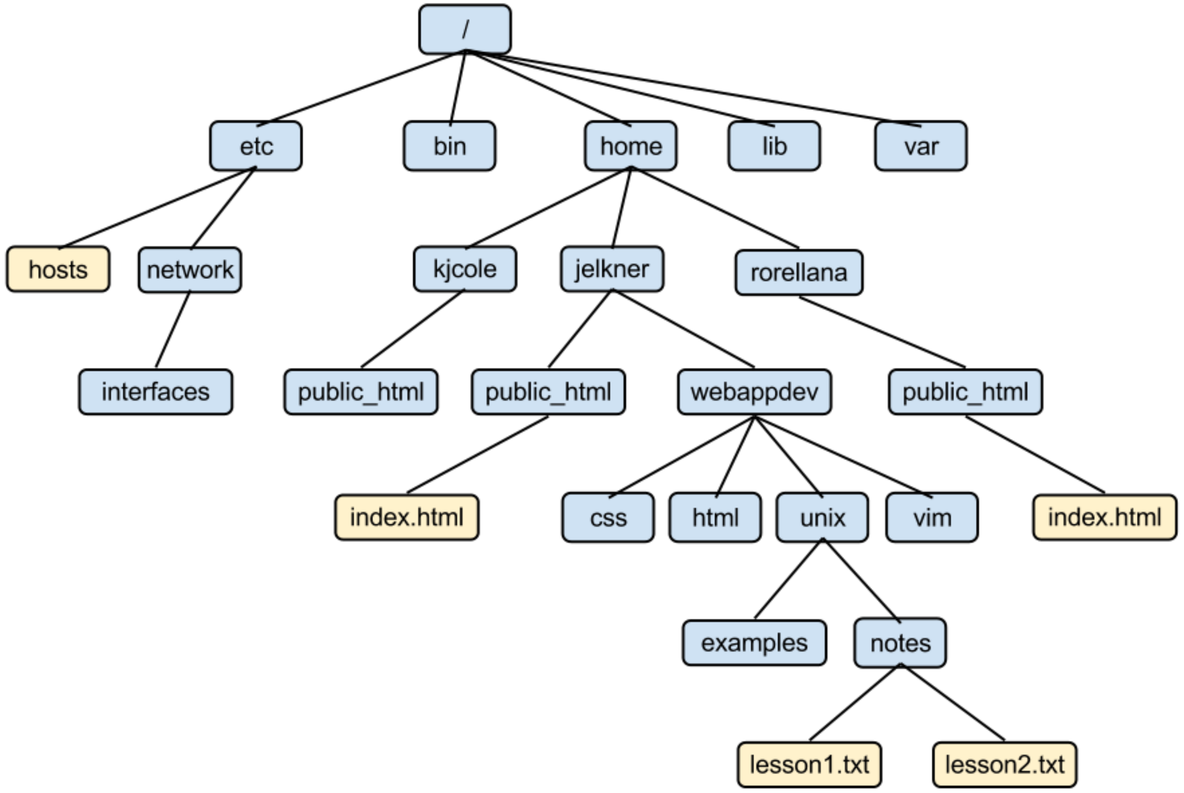
Queues

- Implement the Queue ADT using an Array type
- you should implement the following interface:
- use this code for testing:

```
public static void main(String [] args) {
   ArrayQueue_src<Integer> q = new ArrayQueue_src<Integer>();
   for(int i = 0; i < 10; ++i) {
      q.enqueue(new Integer(i));
   }
   System.out.println("q:" + q); // q:(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

   q.dequeue();
   System.out.println("q:" + q); // q:(1, 2, 3, 4, 5, 6, 7, 8, 9)

   q.dequeue();
   System.out.println("q:" + q); // q:(2, 3, 4, 5, 6, 7, 8, 9)

   q.enqueue(-1);
   System.out.println("q:" + q); // q:(2, 3, 4, 5, 6, 7, 8, 9, -1)
}
```

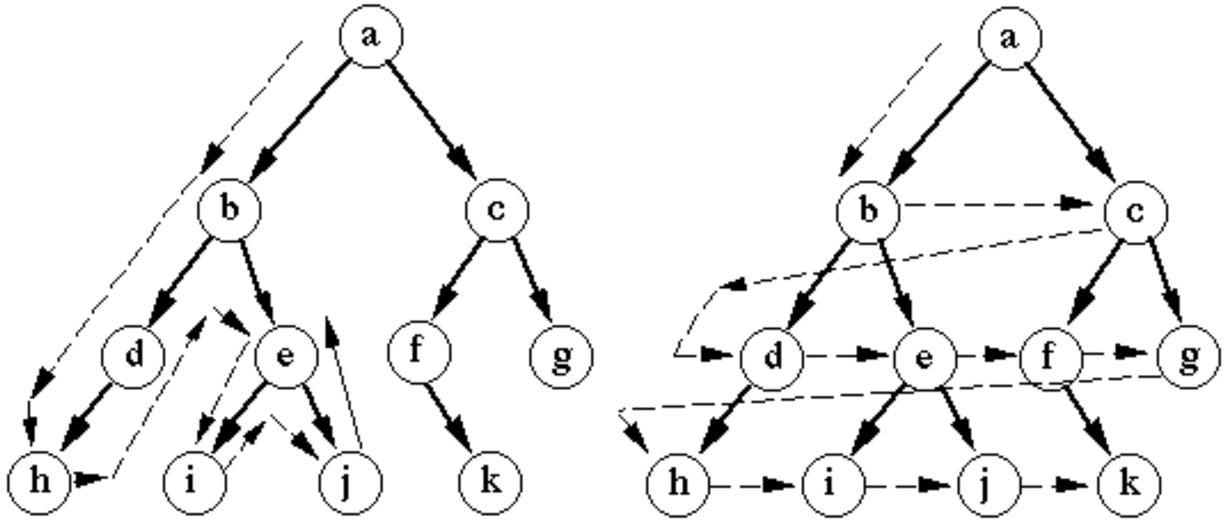
Queues

- Review the java implementation of file counting
- directories are tree structures
- can hold files or directories
- reimplement your iterative file counting using queues, instead of stacks.
- Comment on the differences



Queues

- Review the java implementation of file counting
- directories are tree structures
- can hold files or directories
- reimplement your iterative file counting using queues, instead of stacks.
- Comment on the differences



Priority Queues

- You can implement the priority queue with an Entry(key, value) object, but you should prefer instead to use the method on the next slide.

✘
```
public interface PriorityQueue<K, V> {
```

✔
```
public interface PriorityQueue<E> {
```

```
public interface PriorityQueue<K, V> {

   /**
    * Returns the number of items in the priority queue.
    *
    * @return number of items
    */
   int size();

   /**
    * Tests whether the priority queue is empty.
    *
    * @return true if the priority queue is empty, false otherwise
    */
   boolean isEmpty();

   /**
    * Inserts a key-value pair and returns the entry created.
    *
    * @param key   the key of the new entry
    * @param value the associated value of the new entry
    * @return the entry storing the new key-value pair
    * @throws IllegalArgumentException if the key is unacceptable for this queue
    */
   Entry<K, V> insert(K key, V value) throws IllegalArgumentException;

   /**
    * Returns (but does not remove) an entry with minimal key.
    *
    * @return entry having a minimal key (or null if empty)
    */
   Entry<K, V> min();

   /**
    * Removes and returns an entry with minimal key.
    *
    * @return the removed entry (or null if empty)
    */
   Entry<K, V> removeMin();
}
```

Priority Queues

- Implement the Priority Queue ADT using a list (you can use java.util.list)
- you should implement the following interface:

```
public interface PriorityQueue<E> {

    /**
     * Returns the number of items in the priority queue.
     *
     * @return number of items
     */
    int size();

    /**
     * Tests whether the priority queue is empty.
     *
     * @return true if the priority queue is empty, false otherwise
     */
    boolean isEmpty();

    /**
     * Inserts a value
     *
     * @param value the associated value of the new entry
     */
    void insert(E value);

    /**
     * Returns (but does not remove) an entry with minimal key.
     *
     * @return entry having a minimal value (or null if empty)
     */
    E min();

    /**
     * Removes and returns an entry with minimal key.
     *
     * @return the removed entry (or null if empty)
     */
    E removeMin();
}
```

Priority Queues

- Implement the Priority Queue ADT using a list (you can use java.util.list)
- The objects you store in the Priority Queue should implement the Comparable interface:
- Here is an example of a Tweet object with a compareTo which does a comparison on the `num_followers`.

```
public class Tweet implements Comparable<Tweet> {

   private String username;
   private int id;
   private int num_followers;

   public Tweet(String username, int id, int num_followers) {
      this.username = username;
            this.id = id;
            this.num_followers = num_followers;
   }
   public static void main(String[] args) {
      Tweet t0 = new Tweet("ucdcomputerscience", 10023834, 10204);
      Tweet t1 = new Tweet("insightcentre", 598498434, 4010);

      System.out.println(t0.compareTo(t1));
   }
   @Override
   public int compareTo(Tweet other) {
      if(this.num_followers < other.num_followers) {
         return -1;
      }
      else if(this.num_followers > other.num_followers) {
         return +1;
      }
      return 0;
   }
}
```