Problem Sheet #2

Solutions

1. Given the following s**tack** class declaration, write the following methods using an array-based implementation:

- public void push(E data)
- public E pop()
- public boolean isFull()
- public boolean isEmpty()

Note: there is no currentSize variable here. Do we need it?

```
class Stack<E>  {
   private int head;
   private int maxSize;
   private Object [] stack;

   public Stack(int size) {
       maxSize = size;
       head = -1;
       stack = new Object[maxSize];
   }

   public void push(E data)


   public E pop()


   public boolean isFull()


   public boolean isEmpty()


}
```

2. Given the following array based **Queue** class declaration, write the methods specified. Both the enqueue and dequeue method must run in O(1) time.

```
public void enqueue(E n);
```

- `public E dequeue();`
- `public boolean isFull();`
- `public boolean isEmpty();`

```
public class Queue<E> {
    private int maxSize;
    private int currentSize;
    private Object [] storage;
    private int front, rear;

    public Queue(int size) {
        maxSize = size;
        currentSize = 0;
        storage = (E[]) new Object[maxSize];
        front = rear = 0;
    }

    public void enqueue(E obj)


    public E dequeue()


    public boolean isFull()


    public boolean isEmpty()

}
```

3. Some circular queue implementations use the **mod** operator % in enqueue and dequeue operations. Explain why this is inefficient.

4. If a queue is implemented using a singly linked list with a head and tail pointer, you should always insert at the tail and remove from the head. Explain why this is so.

5. What is a Priority Queue, and how does it differ from a standard queue?

6. Priority Queues are almost always implemented with an ordered data structure. Why?

7. Write the following methods for a standard unordered singly linked list. There is a head pointer, but no tail pointer.

// inserts the given key into the first position in the list (head)

```
public void insertFirst(int key)
```

// inserts the given key at the end of the list (there is no tail pointer)

```
public void insertLast(int key)
```

inserts the key1 into the position immediately following the last instance of key2
// if key2 exists in the list. Otherwise, the key1 is inserted at the end of the list.

```
public void insertAfter(int key1, int key2)
```

// removes the given key from the list if it exists, otherwise does nothing

```
public void delete(int key)
```

// removes the first element (head) in the list

```
public void deleteFirst()
```

// deletes the last instance of the key in the list

```
    public void deleteLastInstance(int key)
```

// removes the last element in the list (there is no tail pointer).

```
    public void deleteLast() {
```

/ / reverses the order of the nodes in the list. i.e. if the list contains:
// HEAD->A->B->C->D
// then the method modifies the list so that it becomes:
// HEAD->D->C->B->A
// The method does not create a new list, but reverses the nodes by manipulating the links in the existing list.

```
    public void reverseList(){
```

8. Both Stacks and Queues can be implemented with either arrays or linked lists. Discuss the advantages and disadvtages of each implementation.