## Queue Declaration

```
public class arrayQueue{
    // State variables of queue

    protected Object[] elements;
    protected int front, rear;
    protected boolean full;

    // Construtor for queue

    public arrayQueue(int capacity){
    elements = new Object[capacity];
    front = 0;
    rear=elements.length-1;
    }

    // Queue Methods

}
```

## Queue Methods

```
// Add modulo one function
public int addOne(int i){
    return (++i % elements.length);
}
// Testing queue empty condition
public boolean isEmpty(){
    return (front == addOne(rear));
}
// Testing queue full condition
public boolean isFull(){
    return (front ==
            addOne(addOne(rear)));
}
```

## Queue Method: Insert

```
// Insert at the rear of the queue
public void insert(Object o){
   if(!isFull()) {
      elements[rear = addOne(rear)] = o;
   }
   else
      System.out.println("queue full");
}
```

## Queue Method: Remove

```
// Remove from the front of the queue
public Object remove(){
   int index;
   if(!isEmpty()){
      index = front;
      front = addOne(front);
      return (elements[index]);
   }
   else{
      System.out.println("queue empty");
      return null;
   }
}
```

# Node Declaration

```
public class node{
    // two fields are:  info, next
    protected Object info;
    protected node next;

    // create an empty node
    public node(){
        next = null;
        info = null;
    }

    // create & initialize a new node
    public node(Object o,node n){
        info = o;
        next = n;
    }

    //Other operations on nodes

}
```

# Node Methods

```
// Set next of a node
public void setNext(node n){
    next = n;
}

// Set info of a node
public void setInfo(Object o){
    info = o;
}

// Retrieve next of a node
public node getNext(){
    return next;
}

// Retrieve info of a node
public Object getInfo(){
    return info;
}

public String toString(){
    return ""+info;
}
```

## Linked List Declaration

```
import node;

public class linkedList{
    // Linked list represented by
    // head.  Size gives the number
    // of nodes.
    protected node head;
    protected int size;

    // creates an empty linked list
    public linkedList(){
        head = null;
        size = 0;
    }

    // Operations on linked list

}
```