Assignment 3 SOLUTIONS

1) StackArray

```
StackArray.java
import java.util.Arrays;
/**
 * Implements Stack using an Array
 *
 * @author (Chris Ewing)
 * @version (4/6/13)
 */
public class StackArray
{
    private Object [] array; // object array we store items in
    private int top;        // the cell where the current top of stack is located
    private static final int DEFAULT_CAPACITY = 10;

    public StackArray()
    {
        array=new Object[DEFAULT_CAPACITY];
        top=-1;
    }


    /**
     * Pushes a new item onto top of stack
     * @param item the item to add to the stack
     */
    public void push(Object item){ //O(1)
        if(top==DEFAULT_CAPACITY)
        {
            array=Arrays.copyOf(array, 2*array.length);
        }
        array[top+1]=item;
        top++;
    }

    /**
     *  Pops an item off the top of stack
     *  @return the top item
     */
    public Object pop() // O(1)
    {
        Object temp=array[top];
        array[top]=null;
        top--;
        return temp;
    }

    /**
     *  Returns the top item from stack without removing
     *  @return the top item
     */
    public Object peek() // O(1)
    {
        Object temp;
        temp=array[top];
        return temp;
    }

    /**
     * Checks for empty stack
     * @return true iff stack is empty
     */
    public boolean isEmpty(){ //O(1)
        if(top==-1){
            return true;
        }
        else
            return false;
    }

    /**
     * Gets the number of items in the stack
     * @return the number of items in stack
     */
    public int size(){ //O(1)
        return top+1;
    }

    /**
     * Creates and returns a string version of stack
     * @return a string reprenting items in stack
     */
    public String toString(){ //O(N)
        String result = "";
        for(int k=top;k>=0;k--)
        {
            result += array[k]+" ";
        }
        return result;
    }
}
```

2) StackLink

StackLink.java

```
/**
 * Implements Stack using Linked Nodes
 *
 * @author (Chris Ewing)
 * @version (4/6/13)
 */
public class StackLink
{
    private Node top;    // reference to the top node of our stack
    int size;
    public StackLink(){
        top=null;
        size=0;

    }

    /**
     * Pushes a new item onto top of stack
     * @param item the item to add to the stack
     */
    public void push(Object item){//O(1)
        Node newNode=new Node(item,null);
        if(top==null){
            top=newNode;
        }
        else{
            newNode.next=top;
            top=newNode;
        }
        size++;
    }

    /**
     *  Pops an item off the top of stack
     *  @return the top item
     */
    public Object pop(){ //O(1)
        Object temp=top.data;
        if(temp!=null){
            top=top.next;
            size--;
            return temp;
        }
        return null;
    }

    /**
     *  Returns the top item from stack without removing
     *  @return the top item
     */
    public Object peek(){  //O(1)
        Object temp=top.data;
        return temp;
    }

    /**
     * Checks for empty stack
     * @return true iff stack is empty
     */
    public boolean isEmpty(){ //O(1)
        if(top==null)
        return true;
        else
        return false;

    }

    /**
     * Gets the number of items in the stack
     * @return the number of items in stack
     */
    public int size(){ //O(1)
        return size;

    }

    /**
     * Creates and returns a string version of stack
     * @return a string reprenting items in stack
     */
    public String toString(){ //O(N)
        String result ="";
        Node current=top;
        while(current!=null){
            result=current.data+""+result;
            current=current.next;
        }
        return result;
    }
    private class Node {
        public Object data;     // data field -- the data stored in this particular node
        public Node next;  // next field -- reference to next SLLNode in list, or null

             // Constructor
         Node (Object data, Node next)
         {  this.data = data;
            this.next = next;
         }

    }
}
```

3) LinkedQueue

```
/**
 * Implements a Queue using Linked Nodes
 *
 * @author (Chris Ewing)
 * @version (4/6/13)
 */
public class LQueue
{

    private Node front;   // reference to the front node of the queue (the next to be
dequeued)
    private Node back;    // reference to the back node of the queue (the last item that
was added)
    int size;
    // Define Constructor here
    public LQueue(){
        front=null;
        back=null;
        size=0;

    }

    public void enqueue(Object value){ //O(1)
        Node newNode= new Node(value, null);
        if(front==null&&back==null){
            front=newNode;
            back=newNode;
        }else{
            back.next=newNode;
            back=back.next;
        }
        size++;
    }
    // post: the value is added to the tail of the structure

    public Object dequeue(){ //O(1)
        if(front==null){
            return null;
        }
        Node removed =front;
        if(front.next!=null){
            front=front.next;
        }
        else{
            front=null;
            back=null;
        }
        size--;
        return removed.element;
    }
    // pre: the queue is not empty
    // post: the head of the queue is removed and returned

    public Object getFront(){ //O(1)
        return front.element;

    }
    // pre: the queue is not empty
    // post: the element at the head of the queue is returned

    public boolean isEmpty(){ //O(1)
```

```
        if(front==null&&back==null){
            return true;
        }
        return false;

    }
    // post: returns true if and only if the queue is empty

    public int size(){ //O(1)
        return size;
    }
    // post: returns the number of elements in the queue

    public String toString(){ //O(N)
        String result="";
        Node current=front;
        while(current!=null){
            result=result+""+current.element;
            current=current.next;
        }
        return result;
    }


    private class Node                // Facilitator class for the LQueue class
    {
        // Data members
        public Object element;         // Queue element
        public Node next;         // Pointer to the next element

        // because there are no access labels (public, private or protected),
        // access is limited to the package where these methods are declared

        // Constructor
        Node ( Object elem, Node nextPtr )
        { element = elem; next = nextPtr;  }

    } // Class Node

}
```

4) VectorQueue

```
VQueue.java
import java.util.Vector;
/**
 * Implements a Queue using a Vector
 *
 * @author (Chris Ewing)
 * @version (4/6/13)
 */
public class VQueue<E>
{
    // declare a Vector<E> queue here to hold the data in the VQueue
    Vector<E> myVector;

    public VQueue(){
        myVector= new Vector<E>();
    }

    public void enqueue(E value){ //O(1)
        myVector.add(value);
    }
    // post: the value is added to the tail of the structure

    public E dequeue(){ //O(1)
        E removed=myVector.get(0);
        myVector.remove(0);
        return removed;
    }
    // pre: the queue is not empty
    // post: the head of the queue is removed and returned

    public E getFront(){ //O(1)
        return myVector.get(0);

    }
    // pre: the queue is not empty
    // post: the element at the head of the queue is returned

    public boolean isEmpty(){ //O(1)
        if(myVector.size()==0){
            return true;
        }
        return false;
    }

    // post: returns true if and only if the queue is empty

    public int size(){ //O(1)
        return myVector.size();
    }


    // post: returns the number of elements in the queue
    public String toString(){ // O(N)
        String result="";
        for(int i=0;i<myVector.size();i++){
           result=result+" "+myVector.get(i);
        }
        return result;
    }
}
```