Client.java

```
public class Client {

   public static final int MAX_RUNS = 1;

   // find average time of multiple runs together
   public static void main(String[] args) {
      for(int i = 100; i <= 100000; i *= 10) {
         double arrayTotal = 0.0;
         double linkTotal  = 0.0;

         for(int j = 0; j < MAX_RUNS; j++) {
            arrayTotal += arrayBasedTest(i);
            linkTotal  += linkBasedTest(i);
         }

         arrayTotal /= (double)MAX_RUNS;
         linkTotal  /= (double)MAX_RUNS;

         System.out.println("" + i + " objects");
         System.out.printf("    Array based: %10f seconds\n", arrayTotal/1000000000.0);
         System.out.printf("    Link based:  %10f seconds\n", linkTotal /1000000000.0);
      }
   }

   public static double arrayBasedTest(int ELEMENTS) {
      ArrayQueue<Integer> queue = new ArrayQueue<>(ELEMENTS);
      ArrayStack<Integer> stack = new ArrayStack<>(ELEMENTS);

      long start_time = System.nanoTime();

      // place elements on the queue
      for(int i = 0; i < ELEMENTS; i++)
         queue.enqueue(i);

      // take elements from queue and put them on the stack
      while(queue.size() != 0)
         stack.push(queue.dequeue());

      // take elements from stack and put them back on the queue
      while(stack.size() != 0)
         queue.enqueue(stack.pop());

      long end_time = System.nanoTime();
      double delta = (double)(end_time - start_time);

      return delta;
   }
```

```
    public static double linkBasedTest(int ELEMENTS) {
        LinkedQueue<Integer> queue = new LinkedQueue<>();
        LinkedStack<Integer> stack = new LinkedStack<>();

        long start_time = System.nanoTime();

        // place elements on the queue
        for(int i = 0; i < ELEMENTS; i++)
            queue.enqueue(i);

        // take elements from queue and put them on the stack
        while(queue.size() != 0)
            stack.push(queue.dequeue());

        // take elements from stack and put them back on the queue
        while(stack.size() != 0)
            queue.enqueue(stack.pop());

        long end_time = System.nanoTime();
        double delta = (double)(end_time - start_time);

        return delta;
    }
}
```

SinglyLinkedList.java

```
public class SinglyLinkedList<T> {

    /**
     * @param <T>
     */
    private class ListNode<T> {
        public ListNode<T> next;
        public T data;

        public ListNode(ListNode<T> next, T data) {
            this.next = next;
            this.data = data;
        }
    }

    private int count = 0;
    private ListNode<T> head;

    /**
     * Constructor
     */
    public SinglyLinkedList() {
        head = null;
    }

    /**
     * @return
     */
    public int getCount() {
        return count;
    }

    /**
     * @return first element w/o removing it from the list
     */
    public T peekHead() {
        if(count == 0)
            return null;

        return head.data;
    }

    /**
     * @return return the last element w/o removing it from the list
     */
    public T peekTail() {
        if(count == 0)
            return null;
```

```
    ListNode<T> tmp = head;
    while(tmp.next != null)
        tmp = tmp.next;

    return tmp.data;
}

/**
 * @param data data to add to beginning of list
 */
public void addHead(T data) {
    head = new ListNode(head, data);
    count++;
}

/**
 * @param data data to add to end of list
 */
public void addTail(T data) {
    if(count == 0) {
        head = new ListNode(head, data);
        count++;
        return;
    } else {

        ListNode<T> tmp = head;

        while(tmp.next != null)
            tmp = tmp.next;

        tmp.next = new ListNode<>(null, data);
        count++;
    }
}

/**
 * @return removed data
 */
public T removeHead() {
    if(count == 0)
        return null;

    count--;

    T data = head.data;
    ListNode tmp = head.next;
    head = null; // assist gc
    head = tmp;
```

```
        return data;
    }

    /**
     * @return removed data
     */
    public T removeTail() {
        if(count == 0)
            return null;

        if(count == 1) {
            T data = head.data;
            head = null;
            count--;
            return data;
        }

        // iterate to end of list
        ListNode<T> tmp = head;
        while(tmp.next != null) {
            tmp = tmp.next;
        }
        T data = tmp.data;

        // need to apply null to end of list again
        ListNode<T> nullNode = head;
        while(nullNode.next != tmp) {
            nullNode = nullNode.next;
        }

        nullNode.next = null;

        count--;
        return data;
    }
}
```

Queue.java

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

    /**
     * @return tell if there are zero elements in the queue
     */
    boolean isEmpty();

    /**
     * @param e element to place in the queue
     */
    void enqueue(E e);

    /**
     * @return return first element w/o removing it from the queue
     */
    E first();

    /**
     * @return remove element and return it
     */
    E dequeue();
}
```

ArrayQueue.java

```
public class ArrayQueue<E> implements Queue<E> {

    private E[] data;
    private int f = 0;
    private int sz;
    private static final int CAPACITY = 1000;

    /**
     * constructor w/ a default capacity
     */
    public ArrayQueue() { this(CAPACITY); }

    /**
     * constructor with a specified capacity
     * @param capacity max number of items in the ArrayQueue
     */
    public ArrayQueue(int capacity) {
        this.data = (E[])new Object[capacity];
    }

    @Override
    public int size() {
        return sz;
    }

    @Override
    public boolean isEmpty() {
        return (sz == 0);
    }

    @Override
    public void enqueue(E e) {
        if(sz == data.length)
            throw new IllegalStateException("Queue is full");
        int avail = (f + sz) % data.length;
        data[avail] = e;
        sz++;
    }

    @Override
    public E first() {
        if(isEmpty())
            return null;
        return data[f];
    }

    @Override
    public E dequeue() {
```

```
        if(isEmpty())
            return null;
        E answer = data[f];
        data[f] = null;
        f = (f + 1) % data.length;
        sz--;
        return answer;
    }
}
```

LinkedQueue.java

```
public class LinkedQueue<E> implements Queue<E> {

    private SinglyLinkedList<E> list = new SinglyLinkedList<>();
    public LinkedQueue() { ; }

    @Override
    public int size() {
        return list.getCount();
    }

    @Override
    public boolean isEmpty() {
        return (size() == 0);
    }

    @Override
    public void enqueue(E e) {
        list.addTail(e);
    }

    @Override
    public E first() {
        return list.peekHead();
    }

    @Override
    public E dequeue() {
        return list.removeHead();
    }
}
```

Output - CSCI-161-Lab05 (run)

```
run:
100 objects
    Array based:   0.000190 seconds
    Link based:    0.016303 seconds
1000 objects
    Array based:   0.003002 seconds
    Link based:    0.022644 seconds
10000 objects
    Array based:   0.012947 seconds
    Link based:    0.284137 seconds
100000 objects
    Array based:   0.014985 seconds
    Link based:   21.986262 seconds
BUILD SUCCESSFUL (total time: 22 seconds)
```