1. R-16.22
In our circular array implementation of a queue, can you compute the value of the currentSize from the values of the head and tail fields? Why or why not?
No. Because there are several edge cases which we can not use tail minus head to get size. For example, if head = 0, tail = 0, in this case, queue size can be full or 0, it is non-deterministic.

2. R-16.24
Suppose you are stranded on a desert island on which stacks are plentiful, but you need a queue. How can you implement a queue using two stacks? What is the big-Oh running time of the queue operations?
For two stacks:

```
//initialize size
size = 0;
//initialize two empty stack
Stack<T> A = new Stack<>();
Stack<T> B = new Stack<>();
//Implement my own add()
void add(T e) {
        while(!B.isEmpty()) {
                A.push(B.pop());
        }
        A.push(e);
         size ++;
}
//Implement my own remove()
T remove() throws Exception {
        while(!A.isEmpty()) {
                B.push(A.pop());
        }
        //get return value
        T ret = B.pop();
        size--;
        return ret;
}
//Implement my own size()
int size() {
        return size;
}
//Implement my own isEmpty()
boolean isEmpty() {
        return size==0;
}
```

For running time:
add(): amortized O(1), remove():amortized O(1), size(): O(1), isEmpty(): O(1)

3. R-16.25
Suppose you are stranded on a desert island on which queues are plentiful, but you need a stack. How can you implement a stack using two queues? What is the big-Oh running time of the stack operations?

```
//initialize size
size = 0;
//initialize two empty queue
Queue<T> A = new Queue<>();
Queue<T> B = new Queue<>();
//implement my own push()
void push(T e) {
        if(!A.isEmpty()) {
                A.add(e);
        } else {
                B.add(e);
        }
}
//implement my own pop()
T pop () throws Exception{
        int temp = size - 1;
        if (!A.empty()) {
                while (temp>0) {
                        B.add(A.remove());
                        temp - -;
                }
                return A.remove();
        } else if (!B.empty()) {
                while (temp>0) {
                        A.add(B.remove());
                        temp - -;
                }
                return B.remove();
        } else {
                throw new EmptyStackException();
        }
    }

//Implement my own size()
int size() {
        return size;
}
```

```
//Implement my own isEmpty()
boolean isEmpty() {
        return size==0;
}
```
For running time:
push(): O(1), pop(): O(n), size(): O(1), isEmpty(): O(1)

4. Suppose you have a stack S containing n elements and a queue Q that is initially empty. Describe how you can use Q to scan S to see if it contains a certain element x, with the additional constraint that your algorithm must return the elements back to S in their original order. You may not use an array or linked list—only S and Q and a constant number of reference variables. What is the running time of your algorithm? (5 points)
O(n).

```
boolean contains(T x) {
    boolean ret = false;
    //reverse the order to insert elements into Q and check elements
    while(!S.isEmpty()) {
        temp = S.pop();
        if (temp .equals(x)) ret = true;
        Q.add(temp);
    }
    //to insert elements into S
    while(!Q.isEmpty()){
        S.push(Q.remove())
    }
    //reverse the order
    while(!S.isEmpty()){
        Q.add(S.pop())
    }
    //to insert elements into S
    while(!Q.isEmpty()){
        S.push(Q.remove())
    }
    return temp;
}
```