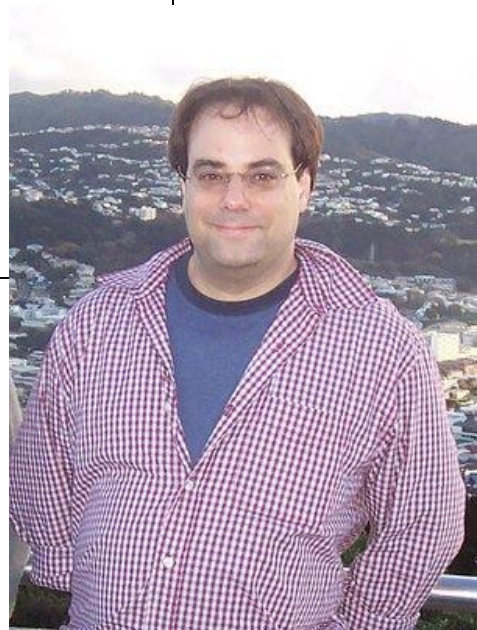
Topic 11
Linked Lists

"All the kids who did great in high school writing pong games in BASIC for their Apple II would get to college, take CompSci 101, a data structures course, and when they hit the pointers business their brains would just totally explode, and the next thing you knew, they were majoring in Political Science because law school seemed like a better idea."

-Joel Spolsky



Thanks to Don Slater of CMU for use of his slides.

Clicker 1

- What is output by the following code?

```
ArrayList<Integer> a1 = new ArrayList<>();
ArrayList<Integer> a2 = new ArrayList<>();
a1.add(12);
a2.add(12);
System.out.println(a1 == a2);
```

A. `false`

B. `true`

C. No output due to syntax error

D. No output due to runtime error

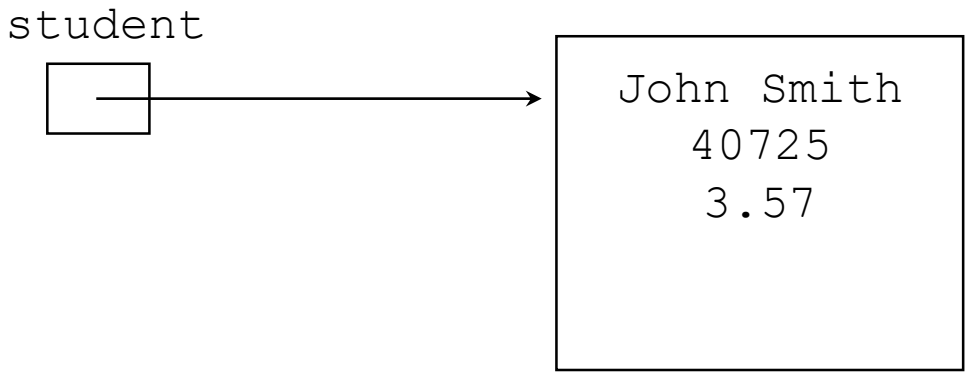
E. Varies from one run of the program to the next

Dynamic Data Structures

- *Dynamic* data structures
  - They grow and shrink one element at a time, normally without some of the inefficiencies of arrays
  - as opposed to a static container such as an array
- Big O of Array Manipulations
  - Access the kth element
  - Add or delete an element in the middle of the array while maintaining relative order
  - adding element at the end of array? space avail? no space avail?
  - add element at beginning of an array

Object References

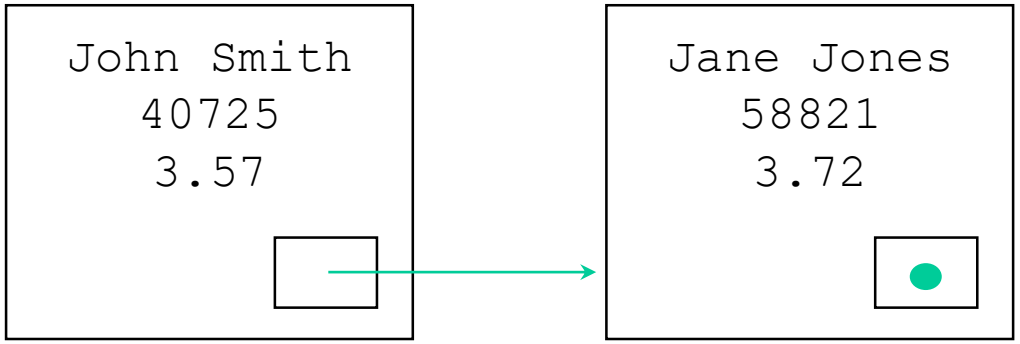
- Recall that an *object reference* is a variable that stores the address of an object
- A reference can also be called a *pointer*
- They are often depicted graphically:



References as Links

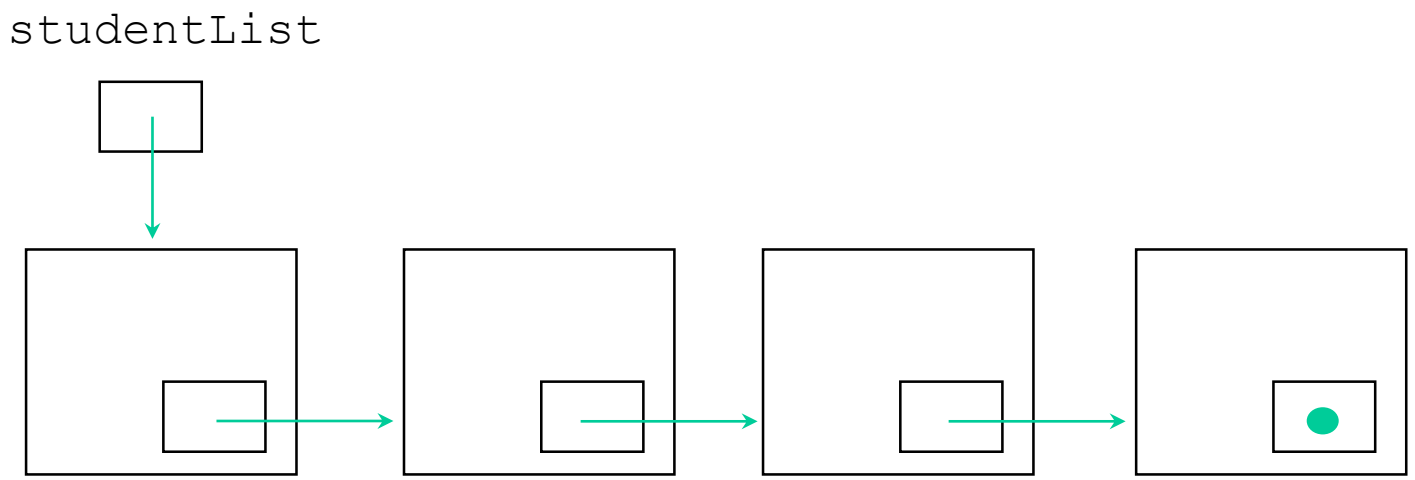
- Object references can be used to create *links* between objects

- Suppose a `Student` class contained a reference to another `Student` object



References as Links

- References can be used to create a variety of linked structures, such as a *linked list*:



Linked Lists

- A **linear** collection of self-referential objects, typically called nodes, connected by other links
  - linear: **for every node in the list, there is one and only one node that precedes it (except for possibly the first node, which may have no predecessor,) and there is one and only one node that succeeds it, (except for possibly the last node, which may have no successor)**
  - self-referential: **a node that has the ability to refer to another node of the same type, or even to refer to itself**
  - node: **contains data of any type, including a reference to another node of the same data type, or to nodes of different data types**
  - **Usually a list will have a beginning and an end; the first element in the list is accessed by a reference to that class, and the last node in the list will have a reference that is set to `null`**

Advantages of linked lists

- Linked lists are dynamic, they can grow or shrink as necessary

- Linked lists are *non-contiguous;* the logical sequence of items in the structure is decoupled from any physical ordering in memory

Nodes and Lists

- A different way of implementing a list
- Each element of a Linked List is a separate Node object.
- Each Node tracks a single piece of data plus a reference (pointer) to the next
- Create a new Node very time we add something to the List
- Remove nodes when item removed from list and allow garbage collector to reclaim that memory

A Node Class

```
public class Node<E> {
        private E myData;
        private Node<E> myNext;

        public Node()
        {       myData = null; myNext = null;     }

        public Node(E data, Node<E> next)
        {       myData = data; myNext = next; }

        public E getData()
        {       return myData;       }

        public Node<E> getNext()
        {       return myNext;       }

        public void setData(E data)
        {       myData = data;       }

        public void setNext(Node<E> next)
        {       myNext = next;       }
}
```

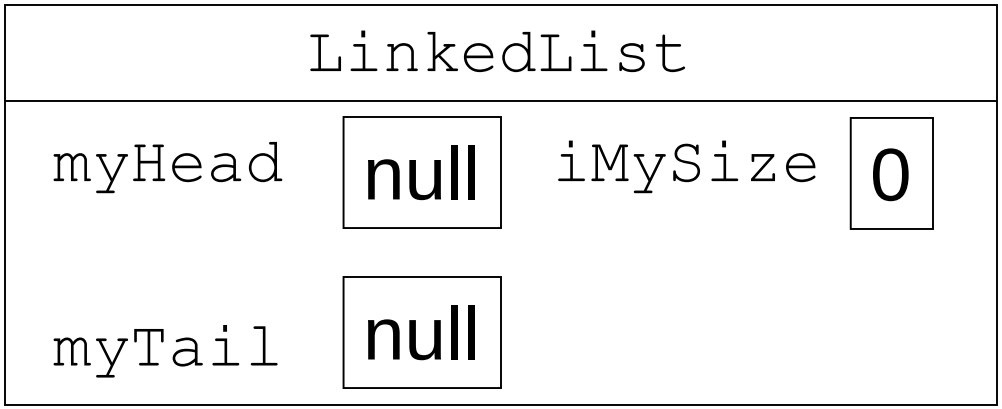
One Implementation of a Linked List

- The Nodes show on the previous slide are *singly linked*
  - a node refers only to the next node in the structure
  - it is also possible to have *doubly linked* nodes.
  - The node has a reference to the next node in the structure and the *previous* node in the structure as well
- How is the end of the list indicated
  - myNext = null for last node
  - a separate dummy node class / object

A Linked List Implementation

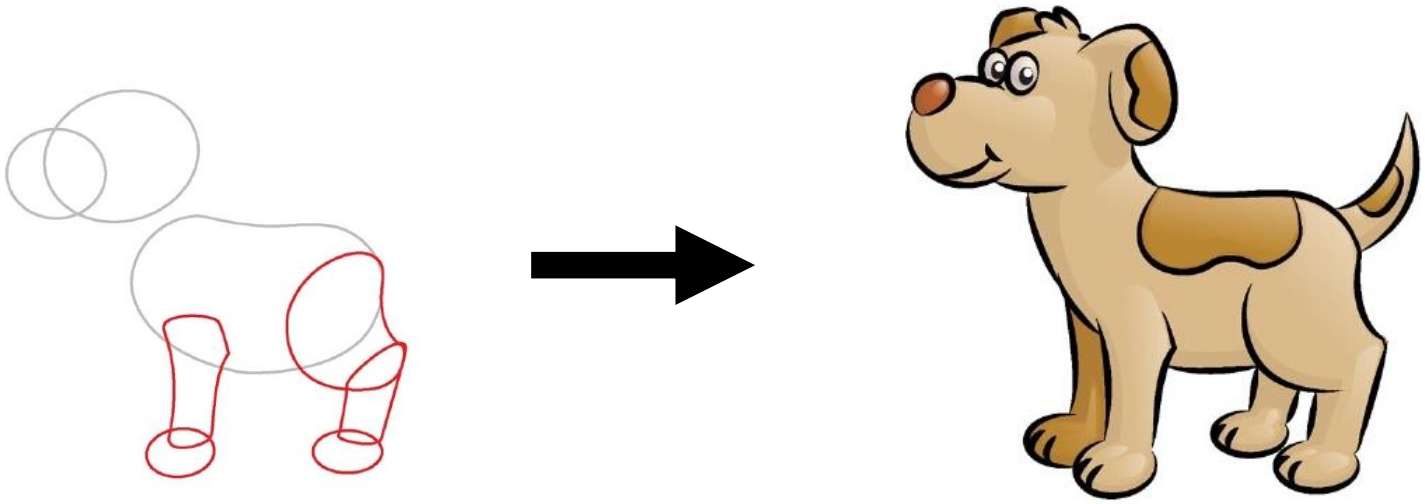
```
public class LinkedList<E> implements IList<E>
	private Node<E> head;
	private Node<E> tail;
	private int size;

	public LinkedList(){
		head = null;
		tail = null;
		size = 0;
	}
}
LinkedList<String> list = new LinkedList<String>();
```



Writing Methods

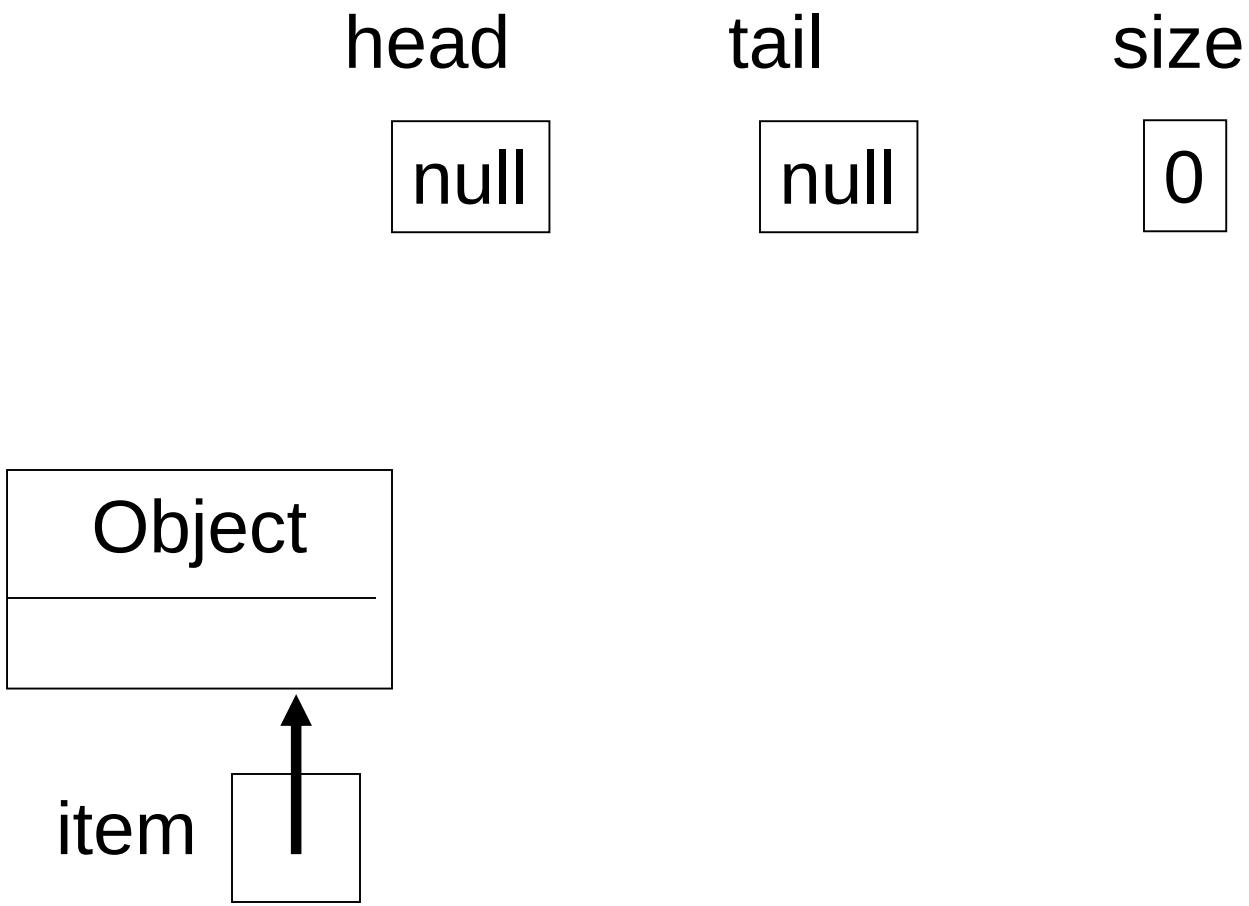
- When trying to code methods for Linked Lists ***draw pictures!***
  - If you don't draw pictures of what you are trying to do it is very easy to make mistakes!



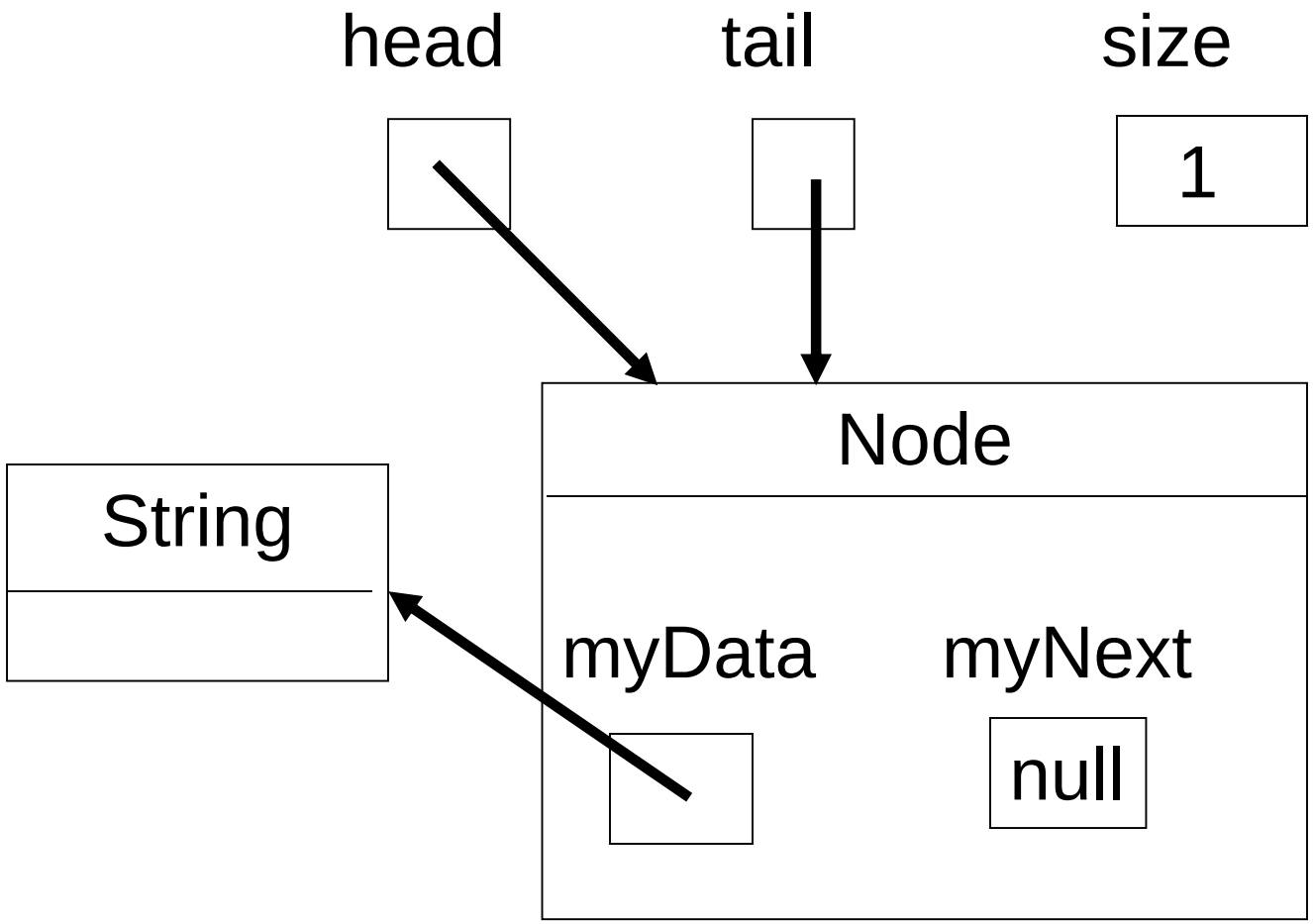
add method

- add to the end of list
- special case if empty
- steps on following slides
- public void add(E obj)

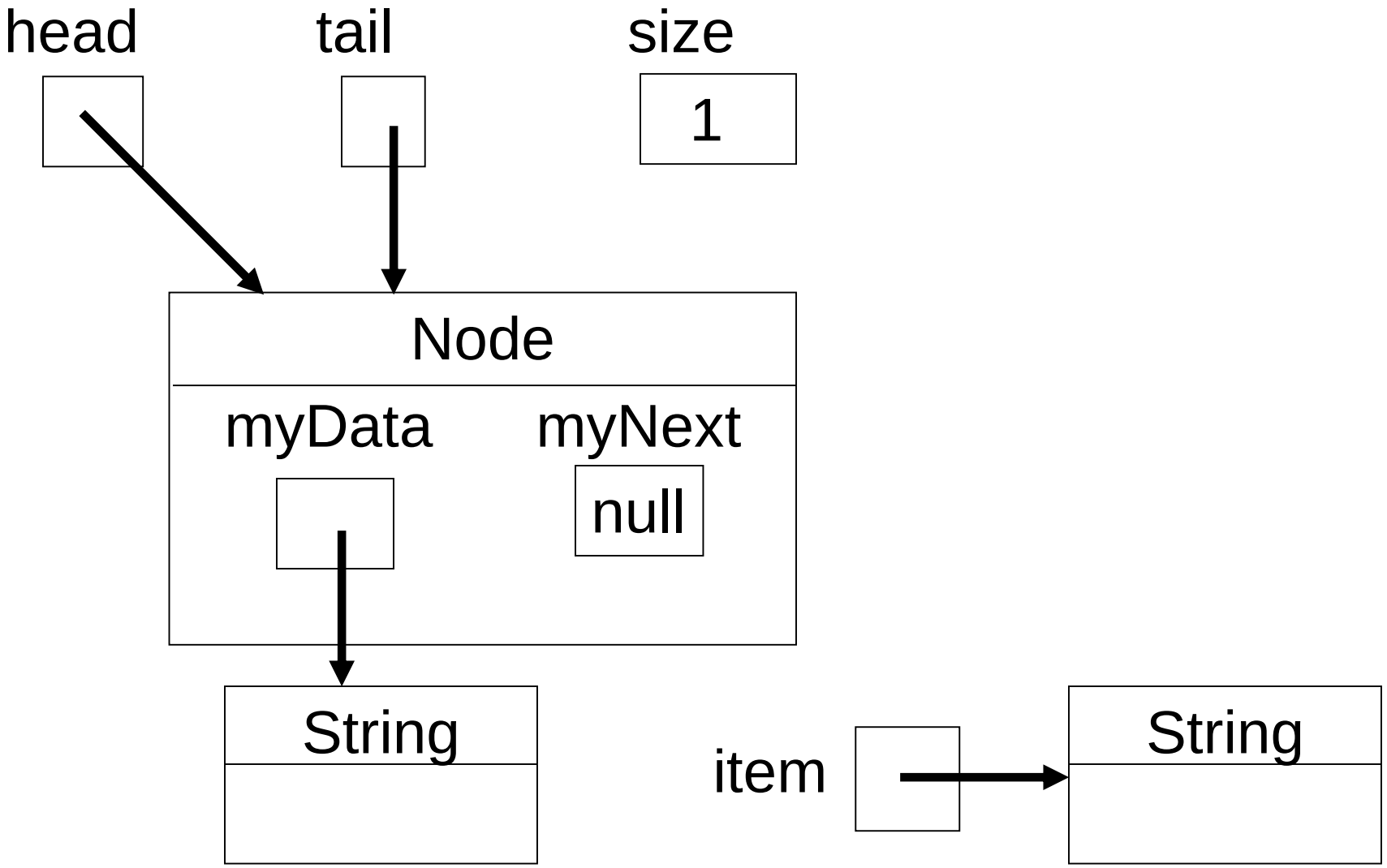
Add Element - List Empty (Before)



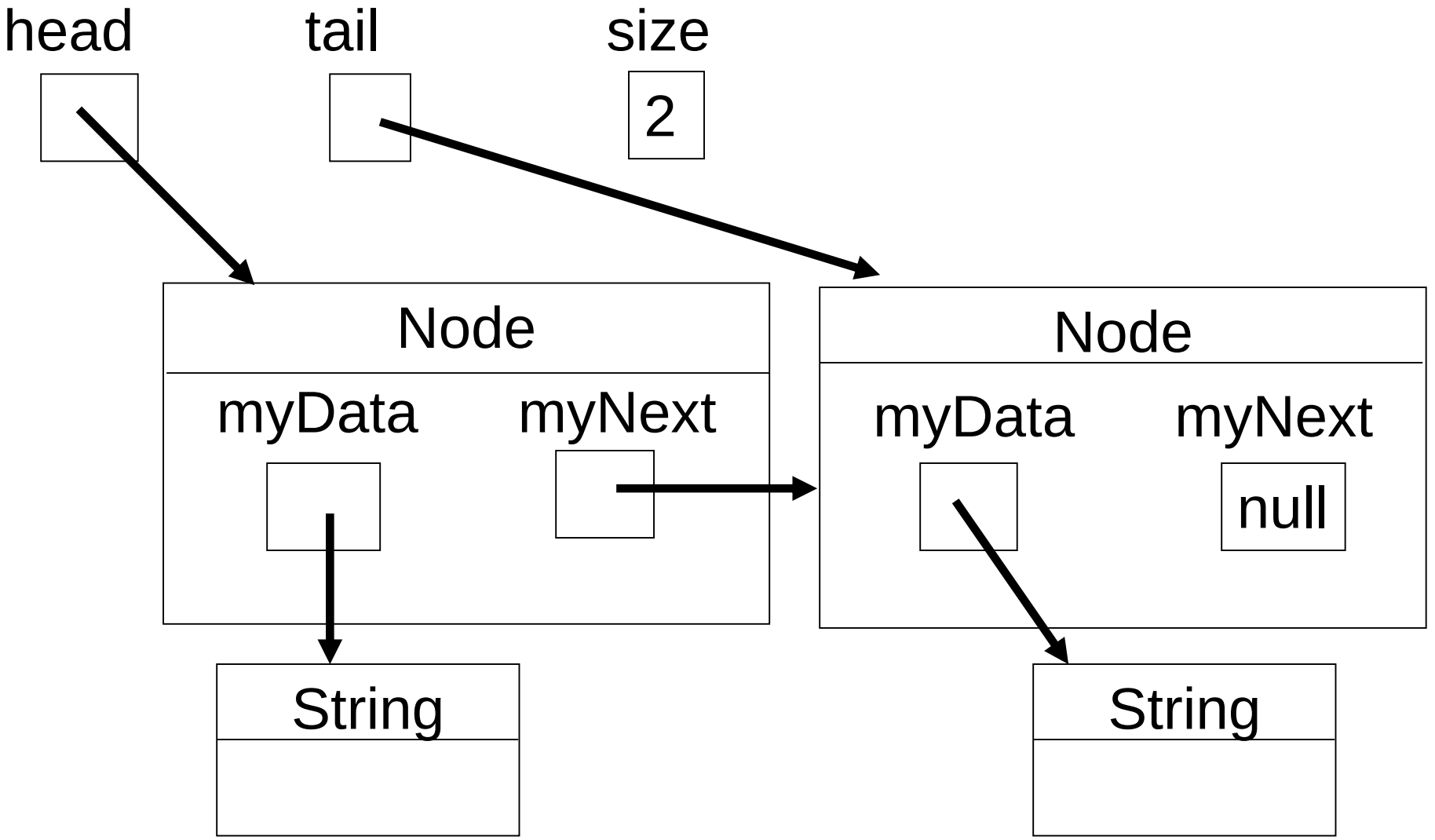
Add Element - List Empty (After)



Add Element - List Not Empty (Before)



Add Element - List Not Empty (After)



Code for default add

- public void add(E obj)

Clicker 2

- What is the worst case Big O for adding to the end of an array based list and our LinkedList314 class? The lists already contain N items.

	Array based	Linked
A.	O(1)	O(1)
B.	O(N)	O(N)
C.	O(logN)	O(1)
D.	O(1)	O(N)
E.	O(N)	O(1)

Contains method

- Implement a contains method for our Linked List class

public boolean contains(E val) // val != null

Code for addFront

- add to front of list
- public void addFront(E obj)
- How does this compare to adding at the front of an array based list?

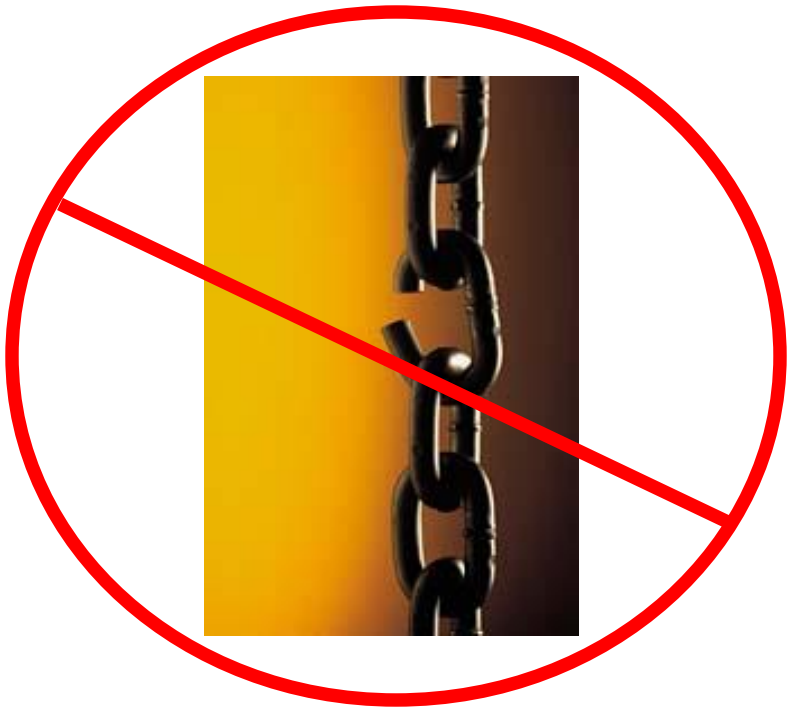
Clicker 3

- What is the Big O for adding to the front of an array based list and a linked list? The lists already contain N items.

	Array based	Linked
A.	O(1)	O(1)
B.	O(N)	O(1)
C.	O(logN)	O(1)
D.	O(1)	O(N)
E.	O(N)	O(N)

Code for Insert

- public void insert(int pos, E obj)
- Must be careful not to break the chain!
- Where do we need to go?
- Special cases?



Clicker 4

- What is the Big O for inserting an element into the middle of an array based list and into the middle of a linked list? Each list already contains N items.

	Array based	Linked
A.	O(1)	O(1)
B.	O(1)	O(N)
C.	O(N)	O(1)
D.	O(N)	O(N)
E.	O(N)	O(logN)

Clicker Question 5

- What is the Big O for getting an element based on position from an array based list and from a linked list? Each list contains N items. In other words `E get(int pos)`

	Array based	Linked
A.	O(1)	O(1)
B.	O(1)	O(N)
C.	O(N)	O(1)
D.	O(logN)	O(N)
E.	O(N)	O(N)

Code for get

- public E get(int pos)
- The downside of Linked Lists

Code for remove

- public E remove(int pos)

Clicker 6

- What is the order to remove the last element of a singly linked list with references to the first and last nodes of the linked structure of nodes?
  The list contains N elements

A. O(1)

B. O(logN)

C. O(N^0.5)

D. O(N)

E. O(NlogN))

Why Use Linked List

- What operations with a Linked List faster than the version from ArrayList?

Clicker 7 - Getting All Elements in Order From a Linked List

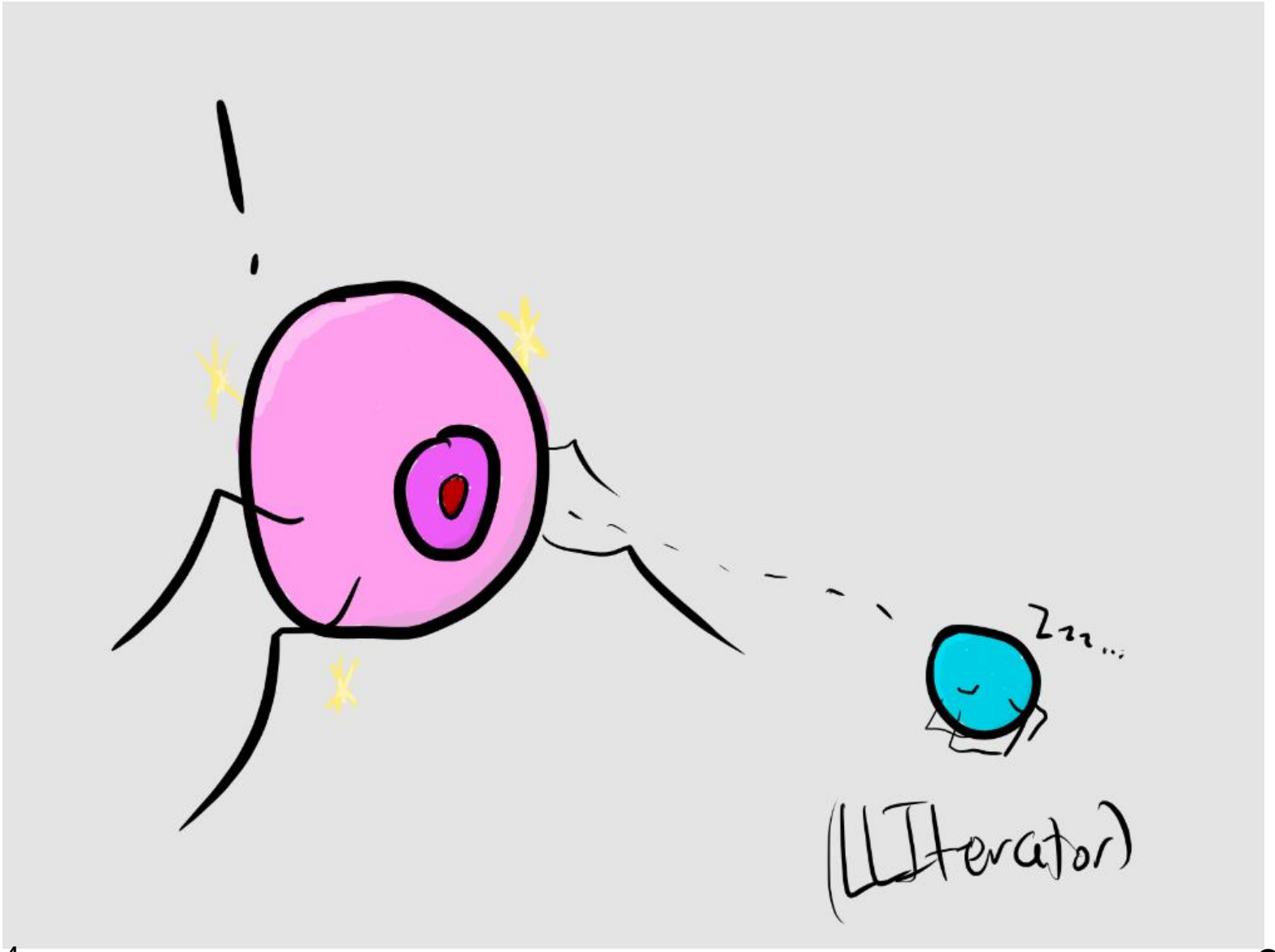
- What is the Order (Big O) of the following code?

```
LinkedList314<Integer> list;
list = new LinkedList314<Integer>();
// code to fill list with N elements
int total = 0;
//Big O of following code?
for(int i = 0; i < list.size(); i++)
   total += list.get(i);
```

A. O(N) B. $O(2^N)$ C. O(NlogN)

D. $O(N^2)$ E. $O(N^3)$

Iterators to the Rescue



Other Possible Features of Linked Lists

- Doubly Linked
- Circular
- Dummy Nodes for first and last node in list

```
public class DLNode<E> {
	private E myData;
	private DLNode<E> myNext;
	private DLNode<E> myPrevious;

}
```

Dummy Nodes

- Use of Dummy Nodes for a Doubly Linked List removes most special cases
- Also could make the Double Linked List circular

Doubly Linked List add

- public void add(E obj)

Insert for Doubly Linked List

- public void insert(int pos, E obj)