Quick Note on Interfaces and Declarations

First, we haven’t actually covered interfaces yet! We’re covering them in depth in three weeks.

```java
List<String> list = new ArrayList<String>();
```

When you declare a variable, its type is always what’s on the left. In the above, `list` is a `List`.

A Tiny Bit of Learning Theory

Some of you want me to stop giving index cards. I’m not going to, and here’s why.

As humans, we can’t pay attention for more than about 20 minutes at a time. We can increase that span by context-switching.

Some Other Things

- An easy way to get rubber duckies is come to office hours, or solve the problems on my door.

- As a general rule, I’m going to start showing up to lecture between 30 minutes and an hour early. Please ask questions!

Today’s Goals

- Get familiar with the idea of “references” (things that point to objects)

- Define and explore `ListNode`

- Learn about `null`

- Practice modifying linked lists

- Get familiar with matching up code and pictures of linked lists
A ListNode is:

The box represents data, and the arrow represents next.

Since `next` is of ListNode type, the arrow can either point to nothing (null) or another ListNode.

What's Going On?
- The keyword `new` creates an actual new object to point to (o1, o2).
- All the other variables just point to objects that were created with `new (list1,list2,list3)`.

What does this code print?

```
>> A: [8, 3, 5, 2]
>> B: [8, 3, 5, 2]
>> C: [100]
```

Class ListNode

```java
public class ListNode {
    int data;
    ListNode next;
}
```

How can we use code to make this list?

```
list: 5 10 15
```

What does this code do to our list?

```
1 ListNode list = new ListNode();
2 list.data = 5;
3 list.next = new ListNode();
4 list.next.data = 10;
5 list.next.next = new ListNode();
6 list.next.next.data = 15;
```

What's wrong?
- This isn't quite right.
Working With Linked Lists

list.next.next.next = list.next;

list

The code sets the arrow coming out of c to the node d.

Dereferencing

When we call .next, we follow an arrow in the list. What happens if we have this list:

list: 5 \rightarrow 15 \rightarrow 10 \rightarrow

And we call the following code:

1 System.out.println(list.next.next.next);

Or this code:

1 System.out.println(list.next.next.next.data);

The first one prints null. The second throws a NullPointerException.
null means "end of the list!"

Constructors!

1 public class ListNode {
2     int data;
3     ListNode next;
4 
5     public ListNode(int data) {
6         this(data, null);
7     }
8 
9     public ListNode(int data, ListNode next) {
10         this.data = data;
11         this.next = next;
12     }
13 }

What list does this code make?

ListNode list = new ListNode(1, null);
ListNode next = new ListNode(2, null);
list.next = new ListNode(3, null);

list: 1 \rightarrow 2 \rightarrow 3 \rightarrow

Can we do this without ever using .next?

ListNode list = new ListNode(1, new ListNode(2, new ListNode(3, null)));

The left side of the assignment is an arrow.
The right side of the assignment is a node.