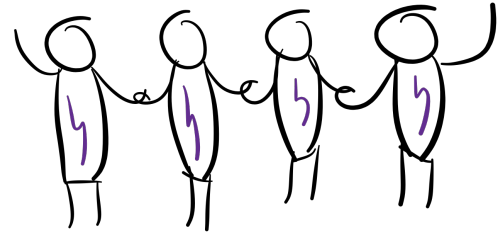


CSE 143

Computer Programming II

Linked Nodes



Quick Note on Interfaces and Declarations

1

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

```
1 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

2

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

3

- 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

4

- 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

Memory

5

Consider the following two documents in a text editor:

- A normal book
- A "choose your own adventure" book

Which tasks are easy/hard in each type of book?

- Find the last page
- Add a new page in the middle of the book
- Add a new page at the end of the book

Books as Data Structures

- Arrays are stored in memory like a normal book; it's **contiguous**, and **random-access**
- For the next three lectures, we'll discuss the data structure equivalent to a "choose your own adventure" book

Mystery

6

```
1 List<Integer> list1 = new ArrayList<Integer>();
2 list1.add(8);
3 list1.add(3);
4 List<Integer> list2 = new ArrayList<Integer>();
5 list2.add(100);
6 List<Integer> list3 = list2;
7 list2 = list1;
8 list2.add(5);
9 list1.add(2);
10 System.out.println("A: " + list1);
11 System.out.println("B: " + list2);
12 System.out.println("C: " + list3);
```

What does this code print?

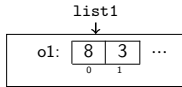
_____ OUTPUT _____

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

Mystery Explained

7

```
1 List<Integer> list1 = new ArrayList<Integer>(); //o1
2 list1.add(8);
3 list1.add(3);
```



```
4 List<Integer> list2 = new ArrayList<Integer>(); //o2
5 list2.add(100);
```



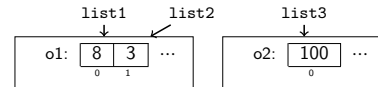
```
6 List<Integer> list3 = list2;
```



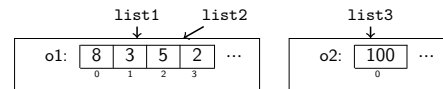
Mystery Explained (cont.)

8

```
7 list2 = list1;
```



```
8 list2.add(5);
9 list1.add(2);
```



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).

ListNode

9

ListNode Class

```
1 public class ListNode {
2     int data;
3     ListNode next;
4 }
```

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.

ListNode

10

ListNode Class

```
1 public class ListNode {
2     int data;
3     ListNode next;
4 }
```

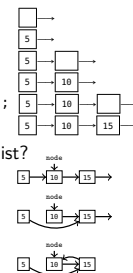
How can we use code to make this list?

list: [5] → [10] → [15] →

```
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;
```

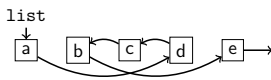
What does this code do to our list?

```
1 ListNode node = list.next;
2 list.next = list.next.next;
3 list.next.next = node;
```

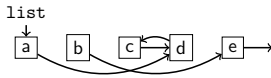


This isn't quite
What's wrong?

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



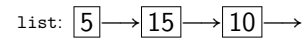
The code sets **the arrow** coming out of c to **the node d**.



The **left side** of the assignment is **an arrow**.

The **right side** of the assignment is **a node**.

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



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);
```

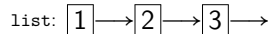
The first one prints `null`. The second throws a `NullPointerException`.

`null` means "end of the list"!

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



Can we do this without ever using `.next`?

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