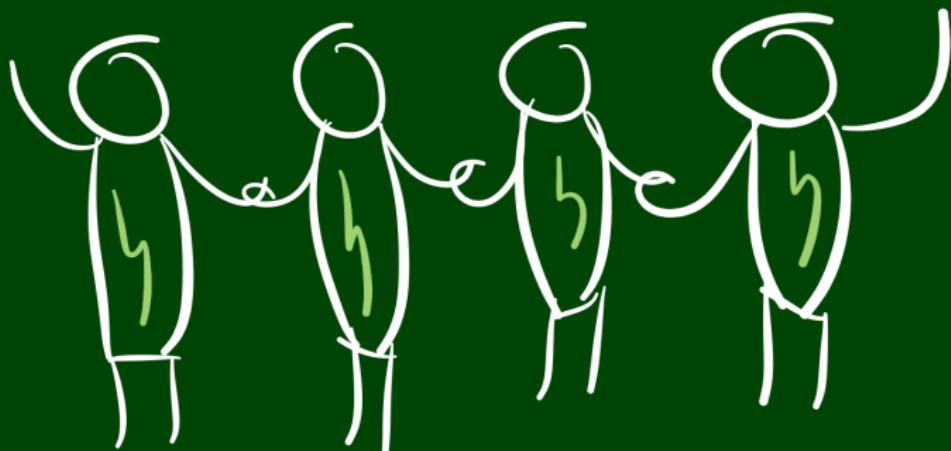


CSE
143X

Accelerated Computer
Programming I/II

List Nodes



- So far, the only “real data structure” we’ve seen is **arrays**
- What are some limitations of arrays?
 - You need to know the size before declaration
 - Adding/removing can be annoying
 - They have no methods
- This is where the idea of a **list** comes in

- 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

Consider the following two documents in a text editor:

- A normal book
- A “choose your own adventure” book

What happens to the page numbers when we...

- 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

```
1 int[] a1 = new int[2];
2 a1.x = 8;
3 a1.y = 3;
4 int[] a2 = new int[2];
5 a2.x = 100;
6
7 int[] a3 = a2;
8 a2 = a1;
9 a2.x = 5;
10 a1.y = 2;
11 System.out.println("A: " + a1.x + ", " + a1.y);
12 System.out.println("B: " + a2.x + ", " + a2.y);
13 System.out.println("C: " + a3.x + ", " + a3.y);
```

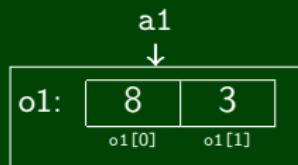
What does this code print?

OUTPUT

```
>> A: 5, 2
>> B: 5, 2
>> C: 100, 0
```

Mystery Explained

```
1 int[] a1 = new int[2]; //o1
2 a1.x = 8;
3 a1.y = 3;
```



```
4 int[] a2 = new int[2]; //o2
5 a2.x = 100;
```



```
6 int[] a3 = a2;
```



Mystery Explained (cont.)

```
7 a2 = a1;
```



```
8 a2.x = 5;  
9 a1.y = 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 (a1,a2,a3).

ListNode Class

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

A ListNode is:



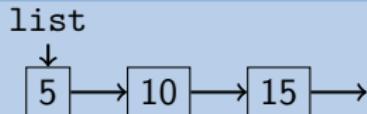
The **box** represents data, and the **arrow** represents next.

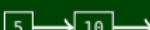
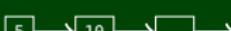
Since next is of ListNode type, the arrow can either point to nothing (null) or another ListNode.

ListNode Class

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

How can we use code to make this list?

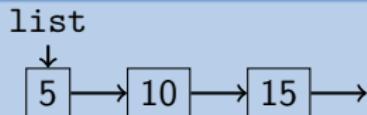


- 1 `ListNode list = new ListNode();`  →
- 2 `list.data = 5;`  →
- 3 `list.next = new ListNode();`  →
- 4 `list.next.data = 10;`  →
- 5 `list.next.next = new ListNode();`  →
- 6 `list.next.next.data = 15;`  →

ListNode Class

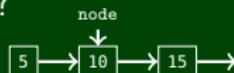
```
1 public class ListNode {  
2     int data;  
3     ListNode next;  
4 }
```

How can we use code to make this list?

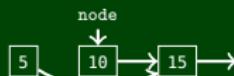


What does this code do to our list?

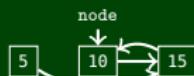
```
1 ListNode node = list.next;
```



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



```
3 list.next.next = node;
```



This isn't quite

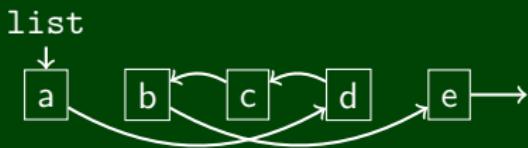


What's wrong?

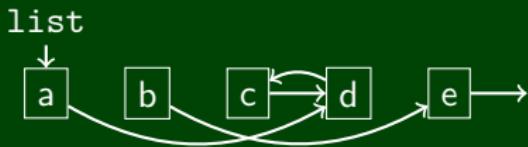
Working With Linked Lists

10

```
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)));
```

Linked Lists I



Outline

- 1 Get more familiar with `ListNode`
- 2 Learn how to run through the values of a `LinkedList`
- 3 Learn how `LinkedList` is implemented

Does That Make Sense?

1

Quick Note: When I say “does that make sense?” . . .

- If it does make sense, yell “yes”
- Otherwise, say nothing.

Outline

- 1 Get more familiar with `ListNode`s
- 2 Learn how to run through the values of a `LinkedList`
- 3 Learn how `LinkedList` is implemented

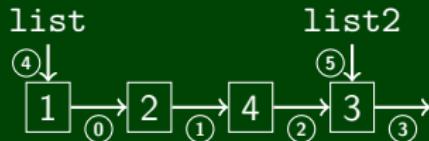
Another ListNode Example

2

Before:



After:



How many `ListNode`s are there in the before picture?

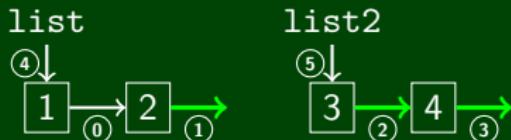
There are FOUR. Each box is a `ListNode`.

How many references to `ListNode`s are there?

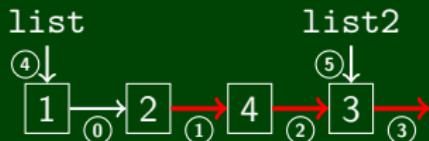
There are SIX. Every arrow is a reference to a `ListNode`.

Another ListNode Example (Solution)

Before:



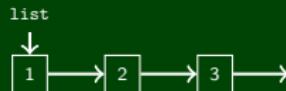
After:



```
1 list.next.next = list2.next
2 list2.next.next = list2;
3 list2.next = null;
```

Outline

- 1 Get more familiar with `ListNode`
- 2 Learn how to run through the values of a `LinkedList`
- 3 Learn how `LinkedList` is implemented



Printing a LinkedList Manually

```
1 System.out.println(list.data);
2 System.out.println(list.next.data);
3 System.out.println(list.next.next.data);
```

Now, note that we can use a variable to keep track of where we are:

```
1 System.out.println(list.data);    list
                                ↓
[1] → [2] → [3] →
2 list = list.next;            list
                                ↓
[1] → [2] → [3] →
3 System.out.println(list.data);  list
                                ↓
[1] → [2] → [3] →
4 list = list.next;            list
                                ↓
[1] → [2] → [3] →
5 System.out.println(list.data);  list
                                ↓
[1] → [2] → [3] →
6 list = list.next;            list
                                ↓
```

What if our list has 1000 nodes? That would be horrible to write.



Printing a **BIG** LinkedList

```
1 while (list != null) {  
2     System.out.println(list.data);  
3     list = list.next;  
4 }
```

But that destroys the list; so, use a temporary variable instead:

Printing a **BIG** LinkedList Correctly

```
1 ListNode current = list  
2 while (current != null) {  
3     System.out.println(current.data);  
4     current = current.next;  
5 }
```

Outline

- 1 Get more familiar with `ListNode`
- 2 Learn how to run through the values of a `LinkedList`
- 3 Learn how `LinkedList` is implemented

- No generics (only stores ints)
- Fewer methods: `add(value)`, `add(index, value)`, `get(index)`,
`set(index, value)`, `size()`, `isEmpty()`, `remove(index)`,
`indexOf(value)`, `contains(value)`, `toString()`
- This is the same idea as when we implemented `ArrayList`!