More Than Arrays

- 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.

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

Memory

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

Mystery

```java
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?

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

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

```java
int[] a2 = new int[2]; //o2
a2.x = 100;
a2 = a1;
a2.x = 5;
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

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

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.

How can we use code to make this list?

```java
ListNode list = new ListNode();
list.data = 5;
list.next = new ListNode();
list.next.data = 10;
list.next.next = new ListNode();
list.next.next.data = 15;
```

What does this code do to our list?

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

This isn't quite what we want.

What's wrong?
- The left side of the assignment is an arrow.
- The right side of the assignment is a node.

Working With Linked Lists
When we call `.next`, we follow an arrow in the list. What happens if we have this list:

```
  5 15 10
```

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!

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

    public ListNode(int data) {
        this(data, null);
    }

    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}
```

What list does this code make?

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

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

```java
ListNode list = new ListNode(1, new ListNode(2, new ListNode(3, null)));
```
Does That Make Sense? 1

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

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

Another ListNode Example 2

Before:

```
1 2 3 4
0 1 2 3
list
4
list2
5
```

After:

```
1 2 4 3
0 1 2 3
list
4
list2
5
```

How many ListNode are there in the before picture?

There are FOUR. Each box is a ListNode.

How many references to ListNode are there?

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

Another ListNode Example (Solution) 3

Before:

```
list

1
2
3
4
list2
```

After:

```
list

1
2
3
list2
```

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

Printing a LinkedList 4

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);
2 list = list.next;
3 System.out.println(list.data);
4 list = list.next;
5 System.out.println(list.data);
6 list = list.next;
```

Printing a LinkedList

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

Does That Make Sense? 1

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

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

Another ListNode Example 2

Before:

```
1 2 3 4
0 1 2 3
list
4
list2
5
```

After:

```
1 2 4 3
0 1 2 3
list
4
list2
5
```

How many ListNode are there in the before picture?

There are FOUR. Each box is a ListNode.

How many references to ListNode are there?

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

Another ListNode Example (Solution) 3

Before:

```
list

1
2
3
4
list2
```

After:

```
list

1
2
3
list2
```

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

Printing a LinkedList 4

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);
2 list = list.next;
3 System.out.println(list.data);
4 list = list.next;
5 System.out.println(list.data);
6 list = list.next;
```
What if our list has 1000 nodes? That would be horrible to write.

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

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

```
ListNode current = list
while (current != null) {
    System.out.println(current.data);
    current = current.next;
}
```

```
LinkedIntList

- 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 ArrayIntList!
```