CSE143X Lecture Questions for Wednesday, 10/28/20

Time (e.g., 12:45)	Question	Answer
0:00	Mic sounds silky smooth. Nice upgrade	Cool beans
2:22	Correction from Monday, dressing up as LinkedList for Halloween instead. Re: It do be like that sometimes.	I'm hollow inside, so I'd dress up as an IntList interface.
	So we don't use some sort of extra data structure to 'contain' all of the nodes, but instead work with the first node in the LL? Re: is that the reason LL are bad at random access, because you'd need to traverse from the start each time?	That's right. You just store a reference to the first node and because of the links you can get to all of the others from there. Exactly...random access is not possible because of the need to follow all of those links.
11:40	So you mean the difference between the ListNode and the referral to a ListNode(the next field inside)? Make sense.	I'm not sure exactly what part you're referring to, but I was talking about the difference between the node object and a reference to the node object (like the difference between a person and a phone number used to contact that person).
15:45	If the list variable stores the address of the new ListNode, why list's type is not an int cause i think the address should be a number. I see.	The address will probably be an integer, but Java does not allow you to manipulate addresses as integers. They preferred to have this notion of a "reference" where they don't tell you exactly what it is.

19:51	Could you explain the difference between uninitialized and null again?
	
	So null has a specific value?
	
	Thanks!
		Suppose I say just this in my Java code:
		`ListNode list;`
		At that point I have declared a variable but haven't given it a value. So if I tried to test it's value, I'd get an error. But if I say:
		`list = null;`
		Then the variable has a value and can be tested with code like:
		`while (list != null) {...}`
		
		Yes, null is a specific value that is recognized and can be tested against.

	When you use the word “error” in your above response, are you implying that this occurs at compile time, as opposed to runtime (where it would be an “exception”)? If so, is it possible for “uninitialized” variables to exist at runtime?	The Java compiler verifies that variables are initialized before they are accessed, so as you suggest, it should never happen that a variable is uninitialized at runtime.
	Can we store different data types in the different nodes of a linked list? Got it! Thanks	I’m not sure what you’re asking. You can have more than one data field, and they could be of different types. And the built-in LinkedList<E> allows you to store different types of data.
	This is based on the above question: Could a LinkedList<Integer> node point to a LinkedList<String> node?	No.
	So I know it was mentioned that setting up Linked Lists with for-loops isn’t exactly *pretty*... but is it off limits? (is it generally just considered super sloppy?)	It’s a matter of personal choice, but it gets quite ugly, especially as the loop tests start getting longer, as we’ll see in Friday’s lecture.
43	If I’m getting this right, there’s a node class that ListNode implements whose fields are private?	No. We define two classes. The first is called ListNode and it has two public fields. So it’s a dangerous class (not well encapsulated). There is a second class called LinkedIntList that has one private field that stores a reference to the front of the list.
24:05	“How many different places are there here where im storing a reference to a listnode?” Can you explain this a little more? I thought it would be the arrows creating new references (so 4) but this is wrong? Re: gotcha!! Wasnt counting the null references, thanks	Two of them are set to null initially (the links that appear at the ends of the two lists). But there are 6 different places where a ListNode value could be stored.

	How does Java address the issue of garbage collection when it comes to linked lists? Does it have to traverse the entire list every time? Yes, but I was asking about the performance implications of an GC algorithm that may have to look at the entire list to determine which ones cannot be reached. Does Java optimize that at all?	Java has a process known as the garbage collector that periodically looks for objects that cannot be reached by code. When an object no longer has a reference to it, it sort of floats away and the garbage collector reclaims it. I've sometimes used the analogy that they are like helium balloons that you let go of. Java does not commit itself to a specific garbage collection algorithm. It's true that you might need to look at all of those nodes to figure out that they can be reached.
13:55	If we don't use "new" to a class name variable, then this variable is always store a reference to an object of that class? gotcha	If you have a variable whose type is something like Foo, then it stores a reference to an object of type Foo.
	Is ListNode a built in class or just something we've written?	Something we've written.
	The ListNode could be relaxing cause it implements what? Maybe because the ListNode class has a ListNode field inside? Then why do we need the LinkedIntList class rather than use ListNode class directly? If I change the public field of ListNode to private, then we don't need LinkedIntList? I can see that later, thx.	No. The point is that the client uses LinkedIntList and there is no way for the client to get access to a node object. So there is no way the client could damage a node. If you didn't have the LinkedIntList class, then your client would be using ListNode, which is what we want to avoid. You want both classes. There are things you want to associate with the overall list and things you want to associate with each individual node. Nodes are very simple components that are used by the list class to store the data.
	While creating a list node, is there a way to know how many nodes in that list, or do we have to go to every ".next" to get the length? Thanks!	We are using a simple implementation where you'd have to go through each .next link as you suggest to find the length. But typically you would store the list length as a field in LinkedIntList.