CSE143X Lecture Questions for Friday, 10/30/20

Time (e.g., 12:45)	Question	Answer
8:50	What is front defined as? Okay so it is a whole node not a .next Ok got it	We're exploring how to write code for a class called LinkedIntList. It has a field called "front" of type ListNode. It is basically a variable that can store a reference to a node. It is not itself a node.
	If there is a method in the implementation view that is not client side (eg: to reduce code redundancy) do we have to comment on it? I'm sorry, I didn't quite get what you mean in the last sentence. What does "discuss the implementation" mean and how can a method do that? One last question: When you say "can", does that mean it's optional? Ok, thank you for the clarification.	Yes, you can end up with methods that are used in the implementation to eliminate redundancy but which you don't want to include in the client view. You should declare all such methods as "private". There is an example in Monday's ArrayIntList that has a method called checkIndex. You can introduce as many private methods as you want. Private methods can include comments about the implementation because a client wouldn't see them. Yes, optional.
52:00	Video cuts off :(is the inchworm okay D: Re: thanks!	Yes, you can use inchworm. Not much was cut off. I include both versions in handout #12: https://courses.cs.washington.edu/courses/cse143x/20au/handouts/12.html
7:48	Is the if statement that checks if front is null considered a pre condition? Understood, thank you.	No. A precondition is something that has to be true. For example, addSorted can't work with a list that isn't sorted. But the code in the if that handles the front case means you can deal with that, so it doesn't need to be a precondition.
41:44	Should it be while(current.next != null && current.next.data < value), the order inside is important? OK.	Wait for it...

17:20	Changing a .next link means changing a .next link of front? But the .nexts will eventually always stem from front, right? Ok, thank you.	Not necessarily. The field called front stores a reference to the first node. You might change the .next field of that node. Or you might change the .next field of a node later in the list. All nodes can be reached from the front, but they don't lead back to front. Yes, all nodes are reached by starting at front and moving forward.
	Can you explain robust/sensitive tests a little more? (what they are, how they differ) Re: makes a lot more sense! Thank you	A sensitive test is one that can generate an error. For example, testing whether current.next.data has some value will be a problem if current.next is null. That would throw a NullPointerException. To avoid that, you'd make sure that current.next is not null. So it is more robust to first test the value of current.next before testing current.next.data. Here's a simpler example. Suppose you have an int variable called n and you want to test whether 10 / n is 2. You would see if (10 / n == 2). But what if n is 0? Then you get an exception thrown because of division by 0. You can always test whether n is 0 (that's a robust test). You can't always test whether 10 / n == 2 (that's a sensitive test). So you test something like: `if (n != 0 && 10 / n == 2) {...}` That puts the robust test first.