CSE143 Lecture Questions for Wednesday, 4/14/21

Question	Answer
What's the relationship between queues and linkedlists? Are they the same thing? Ok thank you. So we've already worked with linked lists in the past week before these lectures specifically about linked lists? I see, so last week was client side and now we're working on implementation side? Thanks!	In the Java collections framework the LinkedList<E> class implements the Queue<E> interface, so they aren't the same thing. For the Guitar Hero program you are using the built-in LinkedList<E> class and we were using it last week. But now we're talking about how to implement one and we're creating our own version called LinkedIntList. Yes.
What happens if you make ListNode private in the ListNode class and public in the LinkedIntList class (basically the reverse)? So does it matter more that in at least one class, the field is private? Ok, thanks.	Then you would be giving the client of LinkedIntList the ability to reach in and manipulate the list directly, but it won't matter because nobody can access the fields in the node anyway. No. You want it the way I've shown with the private field in LinkedIntList and public fields in ListNode. The alternative for ListNode would be to include getters and setters and make the fields private.
For a class like ListNode, would proper commenting still be important? If the only client is yourself. A little more lax, then. Thanks!	The commenting rules for private methods and for a node class are somewhat different. For example, you can talk about implementation details. But they should still be commented.

In the last example of class today does the front list change to include the 17? Or is it just the current list that changes to include the new value? I guess I am kind of confused if front and current are two different lists. That makes more sense. Thank you.	I'm not sure what you're asking about. In the last minute of class I showed the complete code that uses an if/else to distinguish between changing front when the initial list is empty versus the loop solution that finds the last node and attaches the new node to that. The field called front keeps a reference to the first node in the list that we're working with. The variable current is assigned to point to the same node initially, which means that it also is referring to the list we're working with. They aren't different lists, they're two different references to the same list.
When will assignment 1 be graded?	It is being graded now. You'll get it Thursday before homework 2 is due.
Is there a data structure like a linked list but with nodes that point both forward and backward? Would that be more useful/less useful in any situation? What about a circular or looping linked list that points back to its head? Oh okay. Thank you! My TA said it wasn't really useful so I just wanted to clarify.	That is a variation of a linked list known as a doubly-linked list. It is a common data structure. In fact, the built in LinkedIntList is a doubly linked list. I wanted to keep it simple, so we're studying a singly linked list. Yes, a circular list is another variation that is sometimes used.
For Linked Lists, we say that adding/removing is much more efficient than in conventional lists. But it still takes a loop to traverse through the list to get to an index. Is this payoff worth it, say, to make LL a useful alternative to conventional lists? Thanks!	Different applications will have different needs, so it's good to have alternatives to choose from. If you need random access, you'd want to use an array-based solution. If that's not as important as adding and removing, then you'd be more inclined to use a linked list.