CSE143X Lecture Questions for Wednesday, 11/25/20

Time (e.g., 12:45)	Question	Answer
4:43	ADT is interface in java? Oh thx.	ADT is a technical term used by computer scientists to describe an abstraction. But in Java, the ADTs are implemented as interfaces. Interfaces can be used for other things as well, as we will see later in the lecture today.
9:10	Could you explain again why stack interface only has stack class? Lol. When designing programming language, the reasonable order is interface first and then corresponding classes? Thank you.	The Stack class was introduced early with Java and the software engineers weren't being as careful in their programming at that time. So they didn't make a separate Stack interface. It's bad design, but the Java folks didn't want to fix it. That's the problem with a practical language...it always has some warts. Yes, you should introduce interfaces when you are introducing a new kind of data structure that you expect to be implemented in different ways.
25	Are there other implementations for queues besides linkedLists? Ok thanks. Will we be using any of them in class besides linkedlist?	Yes. If you're interested, you can go to the Java documentation for Queue and it includes a list of known implementing classes. https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html Yes, you will use something called a priority queue for the final homework.
13:51	If I am understanding correctly, "q.remove()" will return the value removed and also alter the "q"? Thanks!	Yes. I worded it as "removes and returns the front of the queue" in the short description I showed.
29:20	Petition to start CSE 144?	I taught a version of the course once: https://courses.cs.washington.edu/courses/cse190l/07sp/

28:20	At the end of the first queueToStack, the stack is the full [3, 6, 9, 18,15,12]. So wouldnt the call to stackToQueue reverse the whole stack and not just the last 3 elements? Oh right my bad i was still working with the stack from the for loop. Thanks!	There were two changes that I made to the code. One problem was that it was processing only half of the values (18, 15, and 12). I fixed that by changing the for loop to a while loop. Once you do that, the loop leaves the queue with the values [18, 15, 12, 9, 6, 3]. So then we put them back to the stack, but they're in reverse order. That's why you need an extra stackToQueue followed by queueToStack.
34	Does the comparable interface have something that automatically orders the objects based on whether compareTo returns a negative or positive int? Ohh I must've missed that, thank you!	The interface itself doesn't have that, but client code using the interface does. So in the Arrays class there is a sort method that depends on results from calls on compareTo for it to put things in sorted order.
	Instead of an interface can't you make a parent class for all circle squarr and rectangle? and put compareTo in that parent class? right :)) for the class Shape if we just put something random inside area() instead of using abstract, would the child classes's area methods still overwrite it? so by using abstract it will return an error? ok, thank you!	Wait for it… Yes, if you gave a default definition for the area method in Shape (e.g., returning 0), then the various area methods in the other Shape classes would override it. But what if some Shape class doesn't include an area method? It will be returning 0. That's not a good idea. Better to be clear that you must provide a definition for the area method. Yes, by making it abstract, classes like Square will be required to override area. If they don't, you won't be able to create instances of them.
	I don't understand why you need use super when move toString() to the abstract class but don't need to use super when move compareTo to the Shape. What's the difference? Sorry I missed some part. I see now.	The issue has to do with the constructor I added to the Shape class that takes a String representing the name of the shape. Because the Shape class has a constructor that requires a String, you can no longer rely on a call to: `super();` As a way to handle setting up the superclass part of the object.

	If the interface could be replaced by an abstract class, why do we need interface? Or why don't we relax the restrictions of interface which could only "say hello", but could define the methods inside. What's the purpose to design these fancy concepts? Make sense. thanks.	Java has single inheritance. You can extend only one class. Multiple inheritance is tricky and Java decided to avoid it. But you don't want to be limited to just one such relationship. It is easier to allow multiple interface relationships because there is no concrete code to inherit from the superclass.