CSE143X Lecture Questions for Friday, 12/4/20

Time (e.g., 12:45)	Question	Answer
8:00	Could you explain why doubling the capacity is constant time again? thx.	Consider the case where the capacity is 100 and you are adding a 101st item. There isn't room for it. So it creates a new array 200 long and copies the 100 values over to it. That's a lot of work. So you'd say that it was expensive to add the 101st element. But then you don't need to do any more expansion to add another and another and another up to 200. That's 99 add operations that will be cheap after that one expensive one. If you divide the cost of expansion (copy to an array 200 long) by the 100 add operations that can be done, it's only 2 operations per add (a small constant).
16:-ish	In the case of ArrayIntListIterator at least, so each ArrayIntList has essentially 2 copies of its list? (one in iterator). Are there any concerns of space/redundancy here? So the assignment of this.list = list in the iterator's constructor still references the original list? Makes sense. Thanks :)	No, that's not right. There is only one copy. The iterator just keeps an index into an existing list, so it's not a copy. Yes.
17:00	Would we be able to do this for .next()? return list.get(position++); Would the post-increment still happen, even though it's during the return statement?	I think the fact that you're not sure whether it works is an indication that it's not a good solution. :-) Yes, it would work.
23:00	Why don't we just declare Iterator<Integer> iter = new ArrayIntListInterator<>(numbers) ? And do you think it's useful to look at some contents of the Iterator<Integer> interface now? Oh it's not inheritance.	I mention this later. You want to give the list structure the flexibility to define it any way it wants to. We don't want to know as clients what kind of class is used to implement the iterator. It's better to just know that it implements the interface.

	The JavaDoc for Iterator says that remove() is an “optional operation.” I assume this means that some iterators are not capable of removing items. However, I’m wondering if that is communicated to the caller in any way. Would a non-removing iterator throw an exception on remove()?	It throws an exception (something like UnsupportedOperationException). That’s a bad way of doing things, but that’s what they chose.
31	The AbstractIntList class would still have to mention get and other methods that are not similar in all the classes but they would be left hollow right? Isnt it the idea that if we implement an interface then we are guaranteeing that we will have those methods so we have to mention them? Like we do for compareTo in Comparable	It doesn’t need to mention them because they’re included in the IntList interface. So when AbstractIntList says that it implements the IntList interface, it is saying that it will have all of those methods. When they aren’t filled in, they remain hollow (from the interface). When you declare a class to be abstract, then you don’t need to mention every method. By implementing an interface, you are adding a set of behaviors to what clients can expect of you. And if you’re an abstract class, you don’t have to mention the methods specifically for Java to figure that out.

46	Then why don’t we iterate on the other list for removeAll()?
	`Iterator<Integer> itr = other.iterator();`
	`While (itr.hasNext()) {`
	`    Int n = itr.next();`
	`    For (int i : list) {`
	`        If (i == n)`
	`    }`
	`}`
	
	I see, thank you.
		If you want to propose some code (or pseudocode), I’ll tell you why it might be inefficient.
		
		Lots of problems. Your if(contains) would have to be while(contains) because there might be more than one. Your call on remove(n) would have to be a call on remove at an index. That is going to involve tons of searching of this list to see if it contains something and, if so, where it is. Can’t use a foreach loop while you’re changing the structure. Foreach loop won’t give you an index. This other approach just doesn’t work well.

31:50	Cant we make the AbstractIntList class have all the methods in the IntList interface instead of implementing it, and make the ones that we don't know the definition for abstract and define the ones we know? Ohhh so the primary reason we use an interface instead of an abstract class is to not restrict the class' inheritance?	I think you're proposing getting rid of the interface and having abstract methods in AbstractIntList. We could do that, but then someone who wanted to define their own IntList class would be required to extend our abstract class. That's a big constraint to put on people. It's better to give them the option of implementing the interface. Yes, but I'd word it as the reason we do BOTH an abstract class and an interface is to give flexibility to people who want to include their own classes along with ours.
48:50	Why do we need ArrayIntList.this.remove but not ArrayIntList.this.size()? Gotcha.	For code in the inner class there are two places where you might find the appropriate method. You might find it in the inner class or you might find it in the outer class. For the size method, there is such a method in the outer class, but not the inner class, so you don't need to say ArrayIntList.this.size (although you can if you want to). But for remove, Java gets confused because there are remove methods in both the inner and outer class. It really shouldn't get confused because they have different signatures, but they chose not to fix this.
45	Is the reason we can call itr.remove() without concurrent modification issues because iterators adjust the position pointer as they do the remove? Ok thanks	My version of the iterator doesn't check for concurrent modification. The way it's done with the built-in ArrayList<E> is that it uses a field called modCount that keeps track of how many modifications have been made to the underlying structure. The iterator knows what that should be if it is the only object that is modifying the structure. When it sees a discrepancy, it throws the concurrent modification exception.

40	Before the for each loop, you added Iterator<Integer> iterator to IntList. How will we specify what the iterator method itself does since we only say .iterator But that doesnt have the iterator method itself right? So how would IntList know what .iterator() means Ahh okay got it thanks	The description in the Iterator interface mentions what an iterator should do. https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html The ArrayIntList class would have to implement the iterator() method in such a way that it returns an appropriate object (an object whose behavior matches the description in the Iterator interface documentation).
