CSE143 Lecture Questions for Monday, 4/5/21

Question	Answer
As a guideline, how many times should you have a chunk of code before you should make it into a method? Thank you!	I tend to introduce the method once I have two occurrences of the same code.
Is the order of parameters considered in the signature? If I had method (int i, String str) vs method (String str, int i) would that work? Cool. Thanks!	Order is part of the signature. (int, String) is a different signature from (String, int)
For the code at min 27, can you set up the for loop so it has i = size; i>0; i++ in the for loop instead? Okay, thanks	Yes...keep watching.
For removeAll, would it be more efficient to use an Iterator object or can we not since we haven't implemented that for ArrayIntList? Thanks!	At the point where I showed it, we hadn't implemented the iterator. It wouldn't be more efficient.
Why can't next() return a value of type int? Is it just because the Iterator<Integer> interface interacts with Integer and not int? Yep! Just wondering about return types. Thanks	Yes, it's because of the Iterator<E> interface. That requires that the return type be Integer. But as I've mentioned, Java will turn it into int for you.
If I'm still feeling confused about building a constructor, would it be better to watch/read/look at more examples of constructors, or is there some other pieces of information that I could be missing that I should study more first? Thanks, I will look into those!	You could look at the resources from last quarter's 142 class about constructors. There was a lecture where it was discussed, there are lecture slides, and there is a short extra video for 3/1: https://courses.cs.washington.edu/courses/cse142/21wi/calendar.shtml
Just to make sure I'm understanding constructors-- so at the start of the class code, there will be some fields being declared; and will all those fields always need to be included in the constructor? So in some of the examples, we declare size = 0 in the constructor anyway, even though it's already 0 by default; is this just a special case for size? So the safest thing to do when coding a constructor is to include all the fields, even if some of them *should* be 0 anyway? Or is it just situational? Ok thanks!	Java will auto-initialize all fields to the zero equivalent for the type. If that's good enough, then you don't need to set the field in the constructor. Often that isn't good enough. For example, with ArrayIntList, you have to construct an array. No, it isn't necessary to set fields like that to 0 but I feel that it improves readability to include that. It's more a personal style choice. It's up to you whether you want to include the lines of code that set fields to their default value.

Is “implements” keyword a way to access the interface when you can’t do it on default So to use a specific interface, you use implements to tell Java the class has it? Oh ok, thank you	It is telling Java that the class you are writing is something that implements a specific interface. It also creates a relationship between the two. For example, ArrayList implements List, which means that every ArrayList object is also considered to be of type List. Yes, to tell Java that it implements that interface. We’ll see more examples of this in Friday’s lecture.
For the clear() method, could we just do elementData = new int[capacity] assuming we’ve stored capacity. Okay, thanks! The solution in the video was more elegant anyways haha.	You would still have to reset size to 0 even in that case and it would be very inefficient because it would take a lot of time to construct that array and initialize its elements to 0.
What does the implements keyword do while defining a class? Will it throw an exception if it does not have all the same methods (as defined in the interface)? Okay thank you!	It lets Java know that this class implements the given interface. Java verifies that it does implement the interface and then the class is considered to be of that type (e.g., ArrayList is also of type List). We’ll see more examples of this in Friday’s lecture. If it doesn’t include all of the methods, then it won’t compile.
19:40 so you’re choosing to scan the other list because it’s shorter? Also couldn’t you use the “contains” method in your list instead and it would still work? Why wouldn’t you be able to know this list’s location but you will know the locations of the other list? Thanks!	It is possible to write the code either way and you can come up with scenarios where one way or the other works better. But it turns out to be much more convenient to write it the second way. For example, the contains method doesn’t help you for removing from this list **because you need to know its location.** You could use indexOf, but that is highly inefficient because it scans over and over to get the multiple occurrences. Just try writing the code the other way and you’ll see that it’s very difficult. You don’t need to know locations in the other list because you’re not changing the other list. You need locations in this list because it’s the one that needs to be changed.

What is the difference between Array of type Int versus Array of type Integer? When would you use type integer instead of type int? Thanks!	I mentioned this in Friday's lecture. An Integer object is a "wrapper" for an int. It has one and only one field of type int. But in practical terms, you can treat it as if it is type int and it will behave as expected most of the time. As I said in Friday's lecture, you have to use type Integer for structures like ArrayList<E>. The same is true for objects like iterators.