CSE143X Lecture Questions for Friday, 11/6/20

Time (e.g., 12:45)	Question	Answer
41:03	You are actually using a data structure, it's just implicit. All recursive calls take place on the program's stack.	Yes, I mention the call stack later.
3:40	What is the difference between an invariant and a precondition? I see, thank you.	A precondition is a constraint on a single method. Invariants are true all the time, so they apply to every method (and must be maintained by each method).
18	Once we have added the compareTo method in our program, do we still need to implement the Comparable interface? Ok, but even if we don't have it, won't our client method simply call the compareTo method, or does it not work like that? Got it, thank you!	Yes. It's not enough to have the method. You have to have the "implements" clause in the class header. Yes, but you won't get past the compiler. To call a method like Collections.sort, you have to convince the compiler that you can have values that can be compared.
	`Do recursion buddhists aim to achieve boolean zen? Re:` 阿弥陀佛	I think that boolean zen resonates particularly with recursion buddhists. And we'll have recursion zen soon.
17	What is the purpose of us implementing the Comparable interface if we have to write the compareTo method ourselves?	The compiler wants to know that your various method calls will work. In Collections.sort, the code calls compareTo. How do I know you'll have a compareTo method? I want to have a general sorting method that works on all kinds of data. What all those different types of objects have in common is that they have to have a method called compareTo. That's what an interface is perfect for (guaranteeing that it has a particular behavior).
17	Is this compareTo method the same as the String one?	No. Each class defines its own compareTo. For the String class, it examines the sequence of characters in the two strings.

	For String[] arr, if I want to use Arrays.sort(arr), do I need to add my own compareTo() function? Then for Angle class? Thanks!	The String class implements the Comparable interface, which means it has a compareTo method. That means you can call Arrays.sort to sort an array of String values and you can call Collections.sort to sort a list of String values without writing a compareTo yourself. The final version of Angle that had compareTo and implemented Comparable allows it to be sorted by Arrays.sort if you have an array of Angle objects.
	recursion to me has always seemed like memorizing a bunch of clever tricks to break down problems. Do you have any recommended reading to kinda know how to "think more recursively"?	Eric Roberts wrote a book called Thinking Recursively but it seems to be out of print and expensive to buy a used copy. I recommend just paying attention to the examples we'll cover. We will see a lot of interesting examples of recursion before the quarter is over. Also...chapter 19 on "Functional Programming with Java 8" is a great chapter to understand the computer scientists who love recursion.
14	So does a TreeSet implement the comparable interface and have its own compareTo? Edit: so all wrapper classes implement it?	No. TreeSet assumes that all values added to the set implement the Comparable interface, so it assumes that all of the values stored in the set have a compareTo method. It happens to be true that all wrapper classes implement Comparable, although Java didn't have to necessarily make it that way.
40	Do you need to write n = n - 1 ok	No. Each recursive call knows which value of n to use, so we don't need to change n before we make a call. This is a good example of where recursion is different from iteration. With a loop, we'd have a shared variable n that we'd have to change as we go through the loop. But with recursion, each method has its own copy of n, so we don't have to change any of those individual values of n.

	I just don't understand how Angle could implement Comparable<Angle>, I mean, how a class could implement an interface of that class. I know there are TreeSet implement Set, ArrayList implement List. But in the two examples, the class and interface are of the same type, but not the class is the type of the implemented interface. I see.	A class can implement many different interfaces. It may seem odd that Angle implements Comparable<Angle>, but it just means the Comparable interface has the option to fill in a type for what you're comparing to and you can compare objects to other objects of the same type.
	Is recursion an intentional language design choice or just the natural by product of being able to call functions within themselves? Was there recursion with assembly code?	Language designers have to make a decision about whether or not to support recursion. The old-style programming language BASIC had subroutines (GOSUB), but that wasn't real recursion because you didn't get an independent version of each method call (each with its own local variables and its own "program counter" to keep track of where you are in program execution).
48:00	You ripping up all those poor robots is making me tear up a 'lil bit. For the "reverstance!"	Java has no sympathy for recursive invocations that have finished executing. Maybe they're happier in call stack heaven because they know they accomplished their purpose in life.
45:37	How about changing the order of String line = input.nextLine() and reverse(input)? The same result? Oh so I can say, before calling recurse, I always need to do something else at first to terminate the recursion, right? Gotcha, thx.	No, if you switched the order you'd get infinite recursion. It would ask if there is a line, then it would recurse, ask if there is a line, recurse, ask if there is a line, recurse, etc. No progress. Like asking every robot, "Do you know what row he's in?" Your recursive call has to be on a simpler case than the original, so you have to do something to get you closer to being done (like reading a line of the file in the case of the reverse method).