CSE143 Lecture Questions for Friday, 4/2/21

| Question | Answer |
|---|---|
| What is the difference between using an iterator and a regular for loop? For example:<br><br>`// regular for loop traversal`<br>`for (int i = 0; i < list1.size(); i++) {`<br>`  if (list1.get(i) % 2 == 0) {`<br>`  list1.remove(i);`<br>`  }`<br>`}`<br><br>`// Iterator traversal`<br>`Iterator<Integer> itr = list1.iterator();`<br>`while (itr.hasNext()){`<br>`  int next = itr.next();`<br>`  if (next % 2 == 0) {`<br>`  itr.remove();`<br>`  }`<br>`}`<br><br>I tried this and they do the same thing, so what would be the benefit of doing one over the other?<br>Ah that makes sense, thank you! | That's two ways to accomplish the same thing. In one case you're doing a lot of the work by keeping track of an index variable. In the other case, you have the iterator do most of the work. For some structures, you don't have an option like this. For example, with a set, there is no first option because it doesn't have a get method (no indexing scheme). |
| Is the 'Iterator' an interface? If so, is that why we say Iterator<...>, are there different "sub-interfaces" an iterator has?<br>Oh that sounds cool, thanks! | There is an Iterator<E> interface in Java and it is used to store references to iterator objects. There are variations on the Iterator interface that allow you more flexibility in how you move through the structure (e.g., moving both forwards and backwards). |
| In the radio analogy, would making sound be an interface or would that just be a behaviour?<br>I see. So just in short, interface is just a list of things that you'd expect a program to be doing in order to 'qualify' as a subclass of a class?<br><br>Right. Thanks so much! | It's hard to explain. :-)<br><br>It would be something the radio does, but it isn't a behavior in the sense I'm describing. I'm talking about how the client interacts with the object. So it wouldn't be part of the interface except in the sense that you would describe how the radio controls change that observed activity. As you change the volume, you expect to hear the sound more loudly, for example. It's not a perfect analogy.<br><br>You have that mostly right. It's a list of methods you expect to be able to call along with a description of what effect they have and you have to have all of those to qualify as being of that type. |

| | |
|---|---|
| Quick question but does the homework only revolve around what we have covered so far is it in from the span of Friday to Wednesday for each week? | No. Most weeks I'll hand out the homework on Friday. This week we finished a bit early and I didn't want to delay allowing you to start on the homework, so I gave it out on Wednesday. |
| Outside of the verification, is there any other way we can tell if the coding we have meets the code quality outside of eyeballing and comparing to the unoffiial style guide? Or is that something we can do with TA'S? | We won't give you feedback on your homework before you turn it in. We call that "pregrading." We'll explain the style guidelines, but then you are expected to apply them to the program. Most students in 143 lose some style points. I wouldn't worry about it. The key is to learn from any of those mistakes and do better in the future. |
| At the beginning of lecture you had a list that showed translations from an array to an arraylist. Could you explain the line that says "new String[10] => new ArryList<>()"?<br><br>So its a general example for an unspecified array type?<br><br>Oh, that makes sense! Thanks! | I was trying to show the ArrayList version of what you might have learned with arrays. So you know that to construct an array, you say something like:<br>`new String[10]`<br>The equivalent for an ArrayList is to say something like:<br>`new ArrayList<>()`<br><br>It's a general example for an array of String values or an ArrayList of String values.<br><br>I think perhaps you are being confused by the thought that this is a complete line of code. It would tend to be in a line of code like this:<br>`String[] data = new String[10];`<br>Versus a line of code like this:<br>`List<String> list = new ArrayList<>();` |

| | |
|---|---|
| I'm still having trouble wrapping my head around "interface"; so we would make sure there is an interface by starting something with List? And not with ArrayList?<br><br>So are there other interfaces other than List? And is there some sort of pattern like [interface] and Array[interface], or is that just for List and ArrayList? Ok, thank you! | ArrayList is a class. It is a complete description of an object. There is also an interface called List. The interface is not complete. It contains a list of methods, but no information about how to implement those methods. The issue of where to use List versus ArrayList is a bit of a style issue. You need to use ArrayList when you call new because you are constructing an actual object. But in the other places, we can be more general and use the List interface instead. That makes our code more flexible because we can replace our ArrayList objects with other objects that also implement the List interface.<br><br>There are many interfaces in Java. It is common to have names like List/ArrayList. In the lecture I also show Set/TreeSet. But that's just a matter of style. There is no particular reason that they have to match. |
| How many late days do we have? | 10 free late days, but remember that any one program can be turned in up to 4 days late. |
| Is there a rubric for each assignment? How many points can be lost to style versus each required component of the assignment?<br><br>Are those rubrics available to students? Like prior to submission?<br><br>Thank you! | We do have grading rubrics for the various assignments. Approximately half of the points are for style and half for what we call "external correctness."<br><br>No, they are not available to students. |
| Can we make an ArrayList storing multiple types of objects?<br><br>Oh so subtypes of objects can be stored in an ArrayList of the supertype?<br><br>Okay thank you! | Yes and no. You get to pick the type of element. We generally try to store the same kind of values in any given ArrayList. But you could say that it stores Object and then you could put any kind of value into the list. That is not usually what we want to do. But sometimes there are subtypes we are interested in, like an ArrayList of Critter objects that are slightly different kinds of critters.<br><br>yes... ArrayList<Object> could store any kind of object values. |

| | |
|---|---|
| Does the abstract keyword in Java have to do with abstraction? | Yes it does. We're going to talk about that later in the quarter. The quick version is that it allows you to describe high level information without low-level details. For a method, that would be its header (name and parameters) but no method body. |
| What's the main difference between iterator and list?<br><br>Ohhhhh I seeee!! Thank you!! | The list is the big data structure that has all of the list values stored in it. An iterator is what we call a "light weight" object. It can be used to traverse the list, but it isn't the list itself. If you've been to a pharmacy, you know there are staff members who can wander around the pharmacy and access the drugs stored there. The pharmacy is like the list (big, with lots of stuff in it). The iterator is like the staff member (has access, but isn't itself the pharmacy). |
| Are List and ArrayList different? | Yes. ArrayList is a class (complete definition). List is an interface (only mentions method headers, without implementation details). |
| When are we supposed to use List vs ArrayList | You use ArrayList when you construct the object and List for variables, parameters, return types, and fields. |
| At 18:00 when you say "this is the API" are you talking about the individual methods like the "indexOf()" method?<br><br>OR are you saying all the List<E> methods together is the ONE API?<br><br>So all the methods in List<E> together = the 1 API? (sorry I'm a little confused by your answer) | We use the term API to refer to the description of how to use some software component. That would include method headers and descriptions of what the methods do. It might include other details as well. The List<E> method descriptions I was showing constitute the bulk of the API.<br><br>I'm not sure what you mean by "the 1." Each software component has its own API. You can say that the collection of method descriptions for List<E> are a pretty description of its API. |
| I think this was asked earlier but I just wanted to make sure, are we allowed to use foreach for HW #1? Or is that only for HW #2? Great! Thanks. | You can use the foreach loop for homework 1. |

| | |
|---|---|
| Can you explain why we use List<String> instead of ArrayList<String>? You explained how it's to make it more generic with the cookie analogy. I'm not seeing the connection with the cookie analogy. Like why is it important to make it more generic for Array in particular.<br><br>Gotcha! Thanks! | It has to do with Java's type system. Java cares a lot about what type of data you are working with. If you say you are working with data of type ArrayList, then you are restricted to just that kind of object. If you instead say that it can be anything of type List, then it will work for ArrayList and for other types of lists. We're trying to write the best code we can, so it's better to make it more general. The cookie analogy had to do with what type of customer are you willing to work with. Most businesses would want to accept a wide range of customers and not be overly specific. |