CSE143X Lecture Questions for Wednesday, 12/2/20

| Time (e.g., 12:45) | Question | Answer |
|---|---|---|
| | My IDE freaks out when I don't add @Override to method overrides, but is it absolutely necessary? It works fine without. Is it a Java thing?<br><br>Ok, thanks! | No, that's not a Java thing. It's your IDE. It is optional for you to include that annotation. |
| | Is there a reason why Java doesn't have multiple inheritance?<br><br>Makes sense why they would be strict on that.<br><br>Is the trend of new languages coming out after C++ having multiple inheritance still a big point of contention between developers? Something like python still has it.<br><br>In this case, Riel's quote is sorta just one philosophy out of many right? | Inheritance is not easy to implement. You have to make sure that the subclass includes all of the state and behavior of the superclass. It's pretty ugly in C++. I have mentioned Arthur Riel and his design heuristics. Here's one:<br>54. If you have an example of multiple inheritance in your design, assume you have made a mistake and then prove otherwise.<br>He is saying this somewhat jokingly.<br><br>Yes, different languages do very different things. Mixins is another approach.<br><br>Yes about philosophy/Riel. |
| 29 | Why do we ever declare our variables the more generic(superclass) type if we cant access the specific methods in the subclass? | There are a lot of situations where you might do that. We defined a Shape abstract class with subclasses. Obviously it's helpful to treat things as generic shapes most of the time, but sometimes you might want to see if something is of some specific type to do some special action on that object. |
| 23 | Can I analogy Type A = new TypeB() with List<Integer> l = ArrayList<>().<br><br>okk. | Yes, later in the lecture I mention this. |
| 45 | If my Four class did not override method1 but called super.method1 then it would print One1 right? | Yes. |

| | | |
|---|---|---|
| 42 | When calling method1, why in class Four, it uses super.method1, but in class three, just method1, the super. Could be omitted?<br><br>In practice, we would meet so complex situations to deal with the code reuse problem?<br>ok. | In class Three, the call on method1 was in method2. In class Four, the call on method1 is inside of method1. So you don't want it to be recursive in class Four.<br><br>The code reuse problem comes up often in practice in real code. |
| | Class Four inherit Three, Three inherit One. When calling super.method in class Four, we refer to class Three or One? Or it depends where the method is in?<br><br>thanks. | The Four class inherits from Three, so a call on super.method1 is a call on method1 from the Tree class. But that method is itself inherited from the One class. It basically executes the first method it finds going up the inheritance chain. |
| | How does Java deserialize (json, xml, etc.) into a class with respect to inheritance? Getting the deserializer to choose the right subclass has been a problem for me in other languages.<br><br>Yes, I am asking about how Java recreates a class based on serialized json/xml/etc. when that class can be a super class, or a range of subclasses. For example, in a video game, if the server was to send a list of entities to the Java client, does Java have a way to store each entity in the appropriate subclass of Entity, such as either Player, Box, or Enemy?<br><br>Looking into online resources, I see that standard Java serialization uses a proprietary format that can probably store which specific subclass was used to encode the data. (as opposed to standard JSON/XML formats which would not contain that information) | Objects in Java know what their type is. But are you asking about recreating intermediate superclasses?<br><br>You're asking about implementation details for the Java runtime system that are not normally available to us. You could implement things somewhat differently. But Java does have an object for each class and if you have an inheritance hierarchy and you serialize an object, then it would have to include all relevant class objects for it to recreate it.<br><br>This is an area of Java implementation that I'm not very familiar with. |
| 33:40 | When executing the method, the actual object means the new &lt;object&gt; that was created or the casted object?<br>I see, thank you. | The actual object is the one being constructed with the call on new. |

| | | |
|---|---|---|
| 19:30 | Wouldn’t object be considered the superclass here? | If you’re asking about the Object class, it is the one and only class in Java that has no superclass. |