CSE143 Lecture Questions for Monday, 5/3/21

Question	Answer
Will there be an assignment due next Thursday (5/13)? Thanks! Follow up question-- will the midterm be due Tuesday night at 11pm? Thank you.	No. Tuesday, 11:59 pm.
You mentioned that the inheriting class TypeB takes all the public methods of TypeA, but what if a public method requires a private helper method as well? I see. So writing code in the inheriting class would require the author to re-write the helper method if need be.	The original public method can call the private method because it is in the same class. The inheriting class would not have access to the private method, though. It's as if the inheriting class is a client (so it works to call the public method, although it can't access the private method). It could require rewriting the private method, although it might be enough to have access to the public methods.
Is Recursion the last topic to be present on the Midterm? Or can it also include what we are learning today? Okay! Maybe I haven't watched it all the way! Thank You So Much!	The midterm includes inheritance. The question I go over in lecture is an example of what to expect on the midterm.
If we overwrite a method in a subclass, but declare the variable as the superclass type, and run the method that was overwritten, which one runs? I see. So it wouldn't matter even if we declared it as a superclass.	Objects always behave the same way no matter how you refer to them. So an object of the subclass would always execute it's own version of any method. It never matters what type of variable is used to refer to an object in terms of how it behaves. This is how things are in the real world as well. You wouldn't find a legal secretary doing things the way a secretary does.
Why can't we just declare x to be a Type B variable assigned to a new Type B object instead of having to cast it from a Type A variable again? Or am I missing some idea/concept here? That makes sense! Thank you!	Certainly it's simpler when the variable is of the same type as the object, but it is often the case in an object-oriented program where they differ. For example, the critter simulator has tons of code that acts on objects of type Critter even though the actual critters we include tend to be of a different type that inherits from Critter.

Does extending a class also extend it's private methods? Or only it's public methods? Also, how does inheritance affect fields?	The subclass inherits all methods and fields of the superclass, although it does not have access to private fields and methods.
The "default" inheritance is extends Object. Is there a default implements parallel to this?	No. If you don't have an implements clause, then you don't implement any interfaces.
Suppose I have var5 declared as Object var5 = new Three(); and I have another class between One and Three that also has a method 2. If I call ((Three) var5).method2(); which method2 will execute? Wait so even if I cast it to the class between One and Three, it will execute the method2 for Three? Okay, thank you!	Objects always behave in the same way, so you don't have to look at a variable or a cast to figure out how it will behave. A Three object will always do exactly the same thing when you call its method2. Yes, an object always behaves the same way. Casting has no effect on that.
Does the compiler only verify that there is a cast no matter what the variable is being casted to? In the example that you gave, casting to TypeA would pass the compiler check to call bOnlyMethod(), but TypeA doesn't have that method, right? I am still confused by this. Okay, thank you for clarifying!	The first step is the compiler verifying that the class you are casting to has the method you are calling. The second step is to make sure the cast is legal. There are some cases where the compiler will determine that a cast has no hope of working. I don't test those situations because it's pretty subtle. No, casting to TypeA would not have worked. The first step is the compiler checking to make sure that the class you are contracting for has the method you are calling. When there is a cast, you use the class that you are casting to. So the compiler would ask whether TypeA has a bOnlyMethod and the answer is no. So it would lead to a compiler error. Casting changes which class the compiler checks, but it doesn't somehow allow everything to pass its test.