As with previous chapters, we are going to lay out the learning objectives for this chapter, and split them into what you should get on the first reading vs the second reading.

1 First Reading: Understand Key Ideas

On your first reading, the goal is to get the big ideas, and build the mental scaffolding in which to hold the details on your second reading—the learning objectives for your first reading are generally **Remember** and **Understand** objectives. If you are missing a few details (but not completely lost), it is ok to move ahead, and come back to them the next time through—you might ask questions on the course forums too! Note you should **sleep at least one night between the first and second reading** to give your brain time to process the information.

- **Object Layout**
  - **Explain** how POD types are laid out in memory in C++.
  - **State** the *suboject rule* and **explain** why it is needed.
  - **Define** *vtable*.
  - **Explain** how the vtable pointer for an object is setup.
  - **Explain** how the vtable is used to dynamically dispatch methods.
  - **Explain** how/why the subobject rule applies to the layout of the vtable.
  - **Explain** the performance overheads of dynamic dispatch.
- **Multiple Inheritance**
  - **Define** *multiple inheritance*.
  - **Recognize** that you need to carefully consider if a class exhibits is-A relationships with all parent classes you plan to use (multiple-)inheritance with. (Note: as with Ch 18, we note that there is a lot more to OO design and thinking about inheritance hierarchies that we don't cover here, but a Software Engineering class would cover).
  - **State** the syntax for a class to inherit from multiple parent classes.
  - **Define** *primary parent*.
  - **State** the syntax for disambiguating between methods or fields inherited from multiple parents.
  - **Explain** how objects are constructed and destructed when their classes inherit from multiple parents.
  - **Explain** the difficulty we run into if we naively try to lay out objects of a class that uses multiple inheritance.
  - **Explain** how to lay out objects of a class that uses multiple inheritance.
  - **Define** *primary vtable* (and **identify** it in a drawing of an object layout).
  - **Explain** how the pointer is adjusted when assigning a pointer to an object to a pointer to its non-primary parent type (using subtype polymorphism).

  - **Explain** how the `this` pointer is adjusted when a method call is made on an object that is polymorphed to a non-primary parent type.
  - **Explain** the performance overheads of polymorphism with a non-primary parent.

- **Virtual Multiple Inheritance**
  - **Explain** what problem arises when a class inherits from two classes such that the parent classes both share a common ancestor class (*e.g.*, B extends A, C extends A, and D extends B+C).
    - Note: you may hear this called the *diamond problem*.
  - **Define** *virtual inheritance*.
  - **State** the syntax for virtual inheritance, including where (which class) it needs to be declared in.
  - **Explain** how objects are laid out when they use virtual inheritance.
  - **Explain** the additional performance costs of virtual inheritance.
  - **Recognize** that there is a clear, well-defined ordering for how objects are constructed/destructed when using virtual inheritance (you don't need to know the rule, but may need to come back to it at some point in the future).

- **Mixins**
  - **Define** *mixin*.
  - **Explain** why mixins can be useful.

2 Second Reading

Before you delve into your second reading, we want to say that we aren't going to make use of multiple inheritance in this course. We do, however, want you to know it exists (in its various forms) in case you need it in the future. Likewise, we aren't going to use mixins, however, having heard of them is great if you need them in the future.

We want you to understand object layout and vtables for a couple reasons. The first is that we know several students who have had these come up in interviews (vtables aren't C++-specific, every OO language uses the idea). The second, is that we want to de-mystify how/why a lot of things work. Why do you have to request dynamic dispatch explicitly? Because there is a slight performance overhead which was a big deal in 1983. Why aren't templates and inheritance fully composable (Section 18.7) because of how inheritance and templates are implemented...

Accordingly, we really only have one learning objective for your second reading:

- **Draw** the layout for an object, possibly including inheritance, multiple inheritance, and/or virtual inheritance.

You might or might not need to use multiple inheritance in your future designs—but we want you to know about it in case you do.