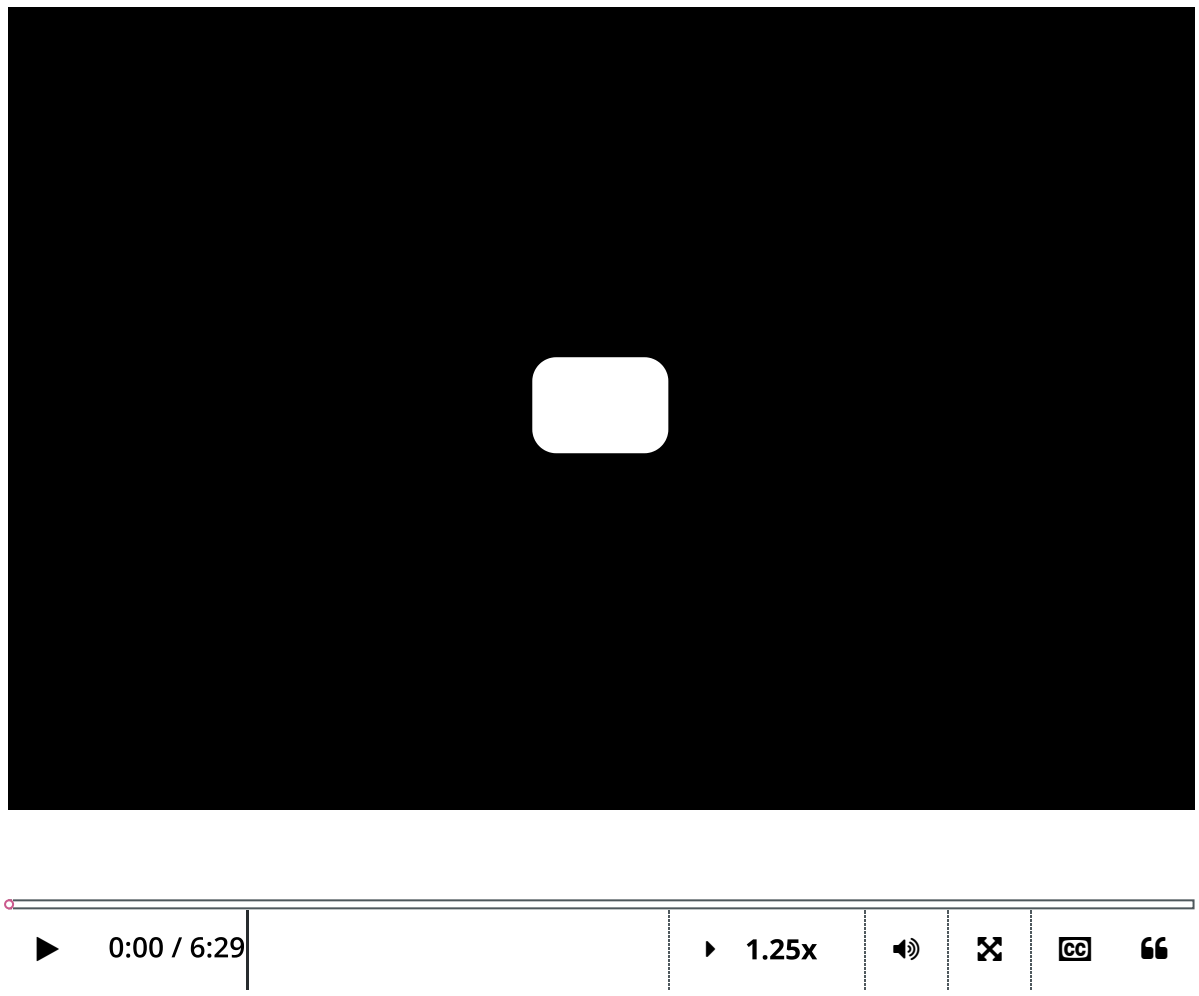
Part 7: A More Complex Calculator

Bookmark this page

Part 7: A More Complex Calculator

As you watch this video, feel free to type along if you wish. This will be a good way to practice for the coming assignment.

A More Complex Calculator



Start of transcript. Skip to the end.

[MUSIC PLAYING]

For this video, I'm going to begin creating a more complex calculator.

I want to show here, what the interface for that calculator should look like.

The calculator has this stack property that we can use

to look inside the calculators memory.

So you enter numbers into the calculator,

after it's been instantiated.

Video
Download video file

Transcripts
Download SubRip (.srt) file
Download Text (.txt) file

What could go wrong with addition and subtraction? Well, at the end of this example we found something that could go wrong! I was expecting that with 11 in the calculator, entering a 2 and then subtracting would produce 9, not -9.

At this point, let's **define** what it means for our module's behavior to be correct:

1. An **operator** is a class instance which provides a method named *calc*. The *calc* method shall accept two arguments and operate on its arguments in the "typical" order:
   Adder().calc(a, b) = a + b
   Subtracter().calc(a, b) = a - b
   etc...
2. A **calculator** is a class instance which provides an *enter_number* method and *add, subtract, multiply,* and *divide* methods.
3. The constructor of a *calculator* shall accept four *operators*: an *adder, subtracter, multiplier,* and *divider*. The calculator shall use these operators to perform its *add, subtract, multiply*, and *divide* methods.
4. *enter_number(a), enter_number(b), operator* should be the same as *a operator b*. In other words:
   enter_number(a), enter_number(b), add() = a + b
   enter_number(a), enter_number(b), subtract() = a - b
   etc...
5. The *add, subtract, multiply,* and *divide* methods shall both:
   1. Return the result of the operation
   2. Enter the result of the operation back into the calculator

This makes it possible to perform the following sequences of operations:

1. *calculator.enter_number(2)*
2. *calculator.enter_number(3)*
3. *calculator.add()* *# Returns 5, and now 5 is now 'in the calculator'*
4. *calculator.enter_number(1)*
5. *calculator.subtract()* *# Returns 4 because 5 - 1 = 4*
6. What should happen if a user enters only one number and then tries to perform an operation? Let's decide that the calculator should throw an **InsufficientOperands** exception if there are fewer than two numbers in the stack.

Based on this definition, we can see that the final result of our calculation in the video should have been 9!

‹ Previous Next ›

There are some situations that our definition above don't explicitly cover:

- What should a *calculator* do if two numbers have already been entered onto the stack and the user enters a third?
- What if there are more than two numbers "in the calculator" when the user performs an operation?
- When a user adds 2 and 3, that enters 5 into the calculator. But are 2 and 3 still in the calculator, or do they get removed?
- Etc..

For the purposes of this exercise, we'll leave the behavior in these circumstance *undefined*. Any person who wants to program a *calculator* could choose to write their code such that:

- a *calculator* throws an error when a third number gets entered into the calculator
- or it could just ignore any new numbers entered if it already has two numbers
- or it could replace one of the two numbers with the third

Any of these choices would be OK in that they would not explicitly violate the six behavioral rules we defined above.

Our next step is to add tests that can automatically tell us if our classes are conforming to those six behavioral rules.