Thanks, Josh.
So we’re going to clarify our problem before making a different pattern. In our problem, we don’t need to provide global access to our variables. We simply need to handle transaction requests as they come, as accurately as possible. And there is a design pattern that can fit this scenario much better than the singleton pattern: the command pattern!

In this pattern, a request becomes an object. The command class is an interface that concrete commands can be created from. A client can create commands, send it to the receiver - the receiver can then send these commands to the invoker, which will process the commands through a queue.

How does this work with our bank pattern? Well, if our client needs to withdraw some money and view a summary, the client class can just make a receiver object, and pass withdraw and summary command objects to it. The invoker then queues up these commands and does them in that order! Seems more useful than a singleton pattern.

This demonstrates how this pattern was able to suit our problem better. Now I’ll hand things off to Miguel to wrap this up. Miguel?