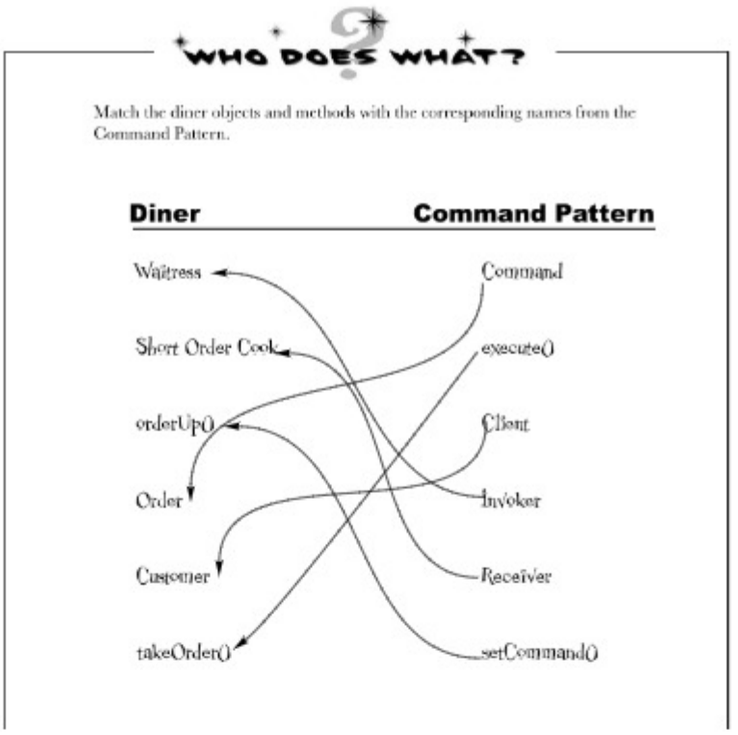
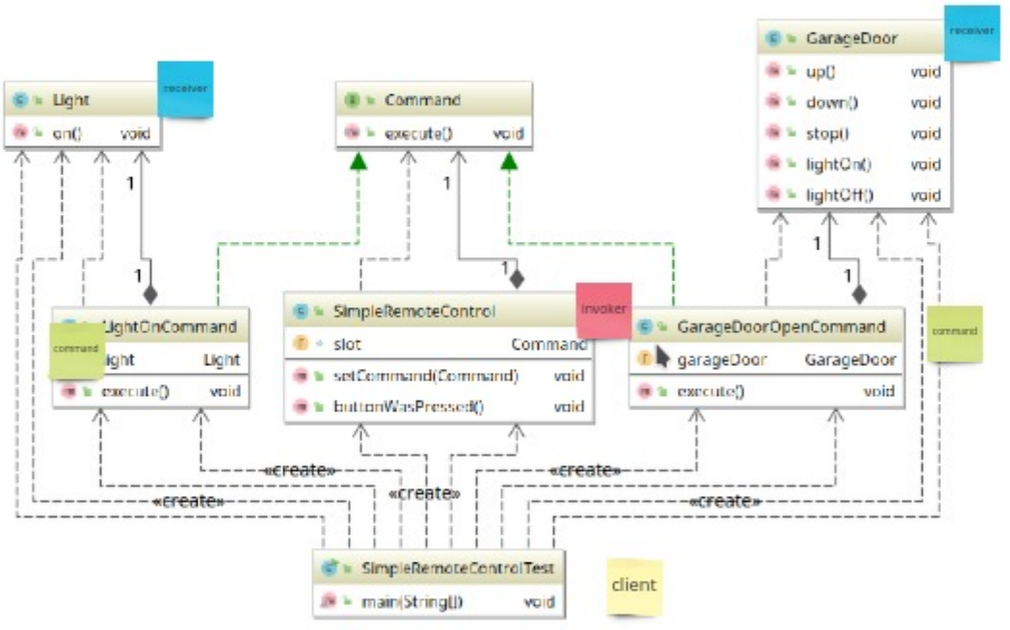
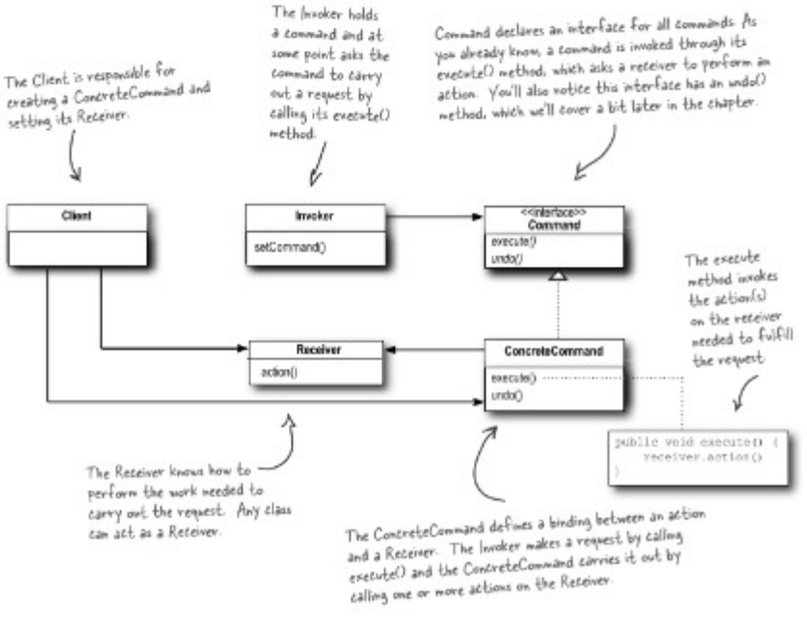
The Command Pattern (idea used by ngrx with reducers and actions)

When you need to decouple an object making requests from the objects that know how to perform the requests, use the Command Pattern.





the class diagram



The Command Pattern encapsulates a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations.

ex: Waitress parametrized with different Orders
- SimpleRemoteControl parametrized with different commands: lightOn, garageDoorOpen

An encapsulated request.

= binds together a set of actions on a specific receiver
= action + receiver in one object, expose just one method: "execute"
-> calling execute causes the action to be invoked on the receiver
-> from the outside, no other objects know WHAT actions get performed ON WHAT receiver

there are no Dumb Questions

Q: Do I always need a receiver? Why can't the command object implement the details of the execute() method?

A: In general, we strive for "dumb" command objects that just invoke an action on a receiver; however, there are many examples of "smart" command objects that implement most, if not all, of the logic needed to carry out a request. Certainly you can do this; just keep in mind you'll no longer have the same level of decoupling between the invoker and receiver, nor will you be able to parameterize your commands with receivers.

Q: How can I implement a history of undo operations? In other words, I want to be able to press the undo button multiple times.

A: Great question! It's pretty easy actually; instead of keeping just a reference to the last Command executed, you keep a stack of previous commands. Then, whenever undo is pressed, your invoker pops the first item off the stack and calls its undo() method.