Command Design Pattern

what is it?

Command pattern is a data driven design pattern and falls under behavioral pattern category. A request is wrapped under an object as command and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.

Intended use

- Encapsulate a request as an object, thereby letting you parametrize clients with different requests, queue or log requests, and support undoable operations.
- Promote "invocation of a method on an object" to full object status
- An object-oriented callback

Vocabulary

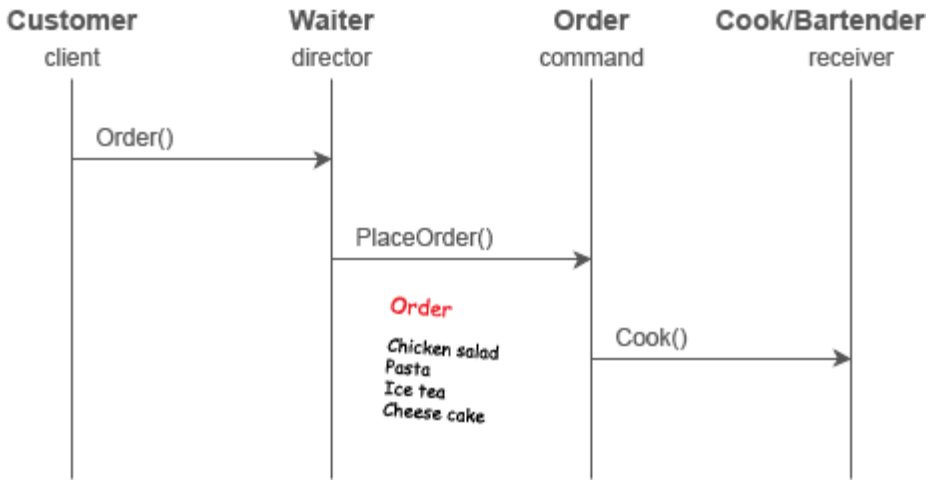
- Command
    - The command that is being sent
- Client
    - The sender of the command
- Invoker
    - the one that performs the operation
- Receiver
    - The one who gets the command

Structure/runtime

- The client implements a command class where there is normally an execute function
- The invoker calls the execute command of the command object

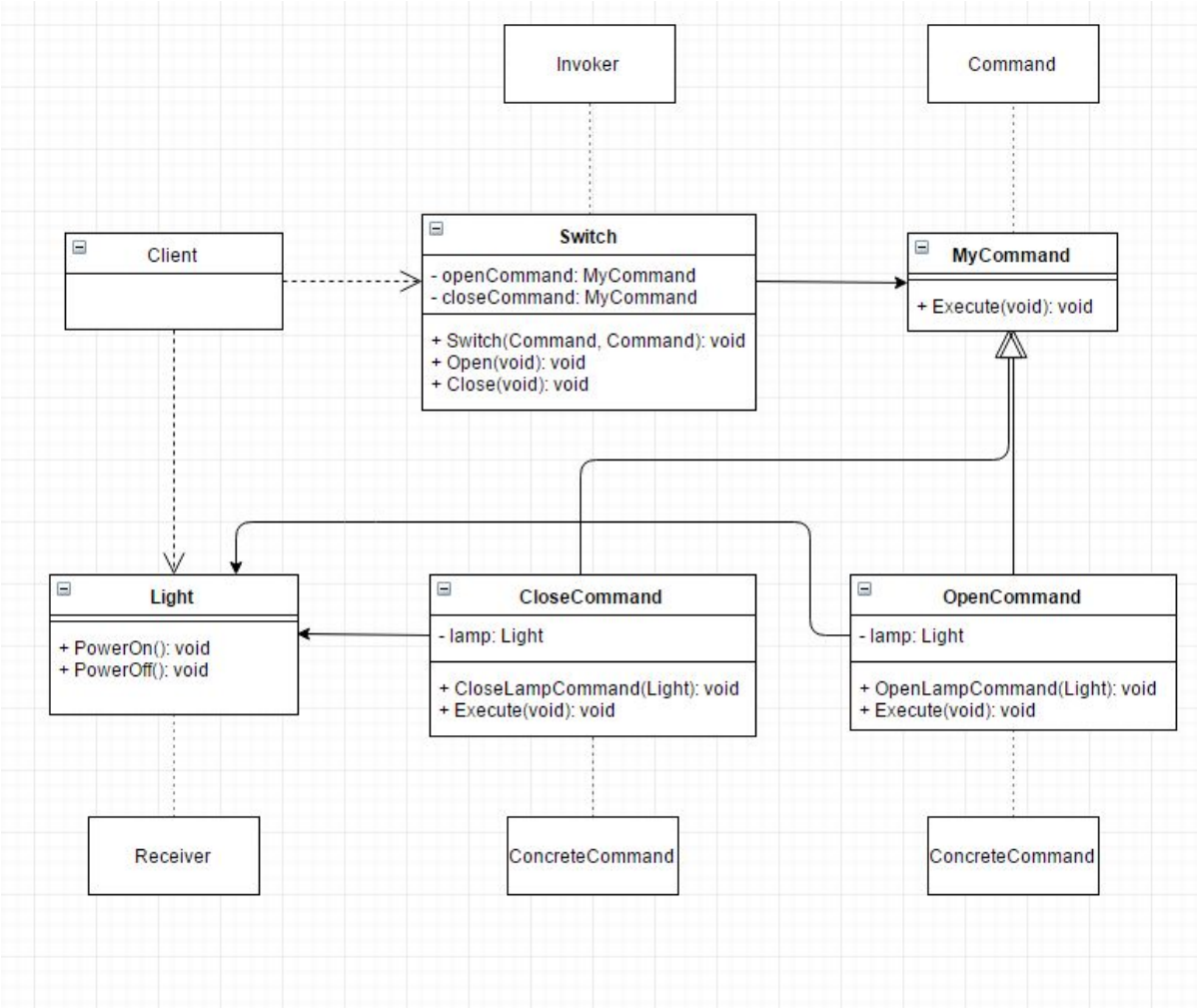
Consequences

- Decouples the invoker from the object that knows how to perform the operation
- commands are first-class objects
- commands can be assembled into composite
    - TCP/IP layers
- Adding new command is easy



- Pick two students as volunteers
- Setting is in a resturant
- One student is the customer
- One student is the cook
- One student is the bartender
- One student is the waiter (invoker)
- Pass the peice of paper as the order to the customer
- customer (client) writes down a drink order on the paper and give to waiter
- waiter gives the order (invoke command) to the barender (receiver)
- bartender makes the food
- customer (client) writes down food order on the paper and give to waiter
- waiter gives the order (invoke command) to the cook (receiver)
- cook makes the food (executes the command)

Switch example
https://en.wikipedia.org/wiki/Command_pattern#Java



Additional examples.
Stock order example
http://www.oodesign.com/command-pattern.html

NFP's

- Extendability/Abstraction
    - The abstract command class or command interface allows a command to be executed by different receivers.
- Modularity
    - Each command is an independent class(Module), allowing it to be stored and passed separately.
- Security
    - It can achieve complete decoupling between the sender and the receiver. Thus, the sender has no knowledge of the Receiver's interface.
- Maintainability/Debuggability
    - Since every command is an independent class, debugging and maintaining each command become easier.

Pros & Cons

- Pros
    - Decreasing the dependency/coupling of the system
    - New commands can be added into the system easily
    - Macro Instructions are easier to implement now.
    - Making it easier to undo and redo commands. e.g. use Memento to maintain the states
- Cons
    - Using command design pattern may requires more effort on implementation, since each command requires a concrete command class, which will increase the number of classes significantly.