Command Pattern

Margin Razvan Cristian

# 1. Intent

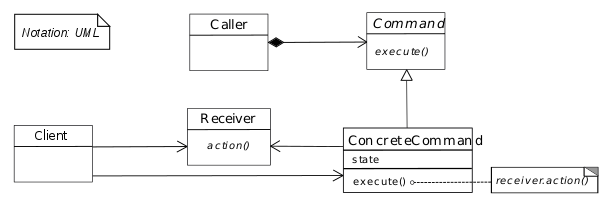
* Encapsulate in an object all the data required for performing a certain action (command)
* Decouple objects that produce the commands from their consumers
* Promote “invocation of a method on an object”
* Object-oriented callback

# 2. Explanation

The Command Pattern tries to resolve the problem of issuing requests to objects without knowing anything in advance about the operation being requested or the receiver of the request.

The Command Pattern requires implementing 4 components:

* Command
  + Objects whose role is to store all info required for executing an action, including the method to call and the object that implements the method (receiver)
  + Contains an execute() method
  + It decouples the object that invokes the operation from the one that knows how to perform it
* Receiver
  + Object that performs a set of actions, i.e performs the actual action when the command’s method is called( when execute() is called)
* Invoker
  + Object that knows how to execute a given command but does not know how that command has been implemented
  + Can store queues commands, aside from executing them
* Client
  + Object that controls the command execution process by specifying what commands to execute and at what stages of process to execute them
  + Treat each object as a “black box” by simply invoking the execute() method



The client that creates a command is not the same client that executes it. This separation provides flexibility. Command objects can be thought as “tokens” that are created by one client that knows what needs to be done, and then passed to another client that has the resources for doing it.

Regarding the SOLID principles, I believe that the Command Pattern respects the Single Responsibility Principle, because each class is assigned with only one responsibility( Invoker’s responsibility is to know how to execute a command, The client’s responsibility is to manage the commands and control them). Furthermore, The Command Pattern makes its design open against change of adding new behavior with each behavior being usable in one context, whereas it is not open against changing existing commands. In addition, it also respects the Interface segregation principle in the sense that the Commands are not forced to implement other method except execute().

# 3. Related Patterns

* Chain of Responsibility, Mediator and Observer address how you can decouple senders and receivers, but with different trade-offs, whereas Command normally specifies a sender-receiver connection with a subclass
* Chain Of Responsibility can use Command to represent requests as objects
* Command can use Memento to maintain the state required for an undo operation
* A Command that must be copied before being placed on a history list acts as a Prototype

# 4. Common situations of use

Usually the Command Pattern is used in applications that need requests to be encapsulated as objects, thereby allowing clients to be parametrized with different requests. Therefore, it is mainly applied in client-server applications.

* GUI buttons and menu items
  + In swing, an action is a command object. In addition to the ability to perform the desired command, an action may have an associated icon, keyboard shortcut, etc. A toolbar button or menu item component may be initialized using only the Action Object
* Networking
  + It is possible to send whole command objects across the network to be executed on the machines, for example player actions in computer games
* Parallel Processing
  + The commands are written as tasks to a shared resource and executed by many threads in parallel

# 5. References

<http://www.baeldung.com/java-command-pattern>

<https://en.wikipedia.org/wiki/Command_pattern>

<https://sourcemaking.com/design_patterns/command>

<https://softwareengineering.stackexchange.com/questions/307620/command-pattern-and-open-closed-principle>