Command ID

Student:

Izabela Kuźniar

Teacher:

Andrea Corradini

Course:

Software Design Patterns

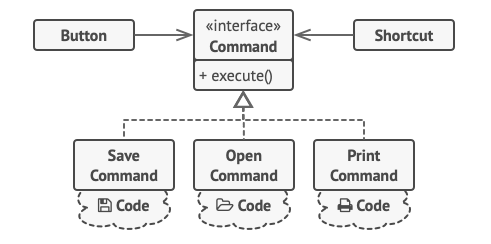
# Name and category

Command is a behavioral pattern.

# Intent:

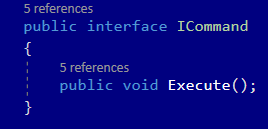
Decouples the object from an operation.

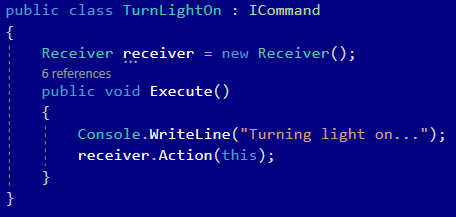
# Structure as a UML class diagram

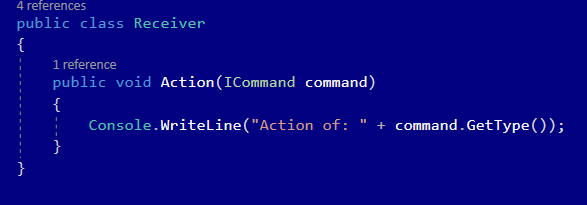


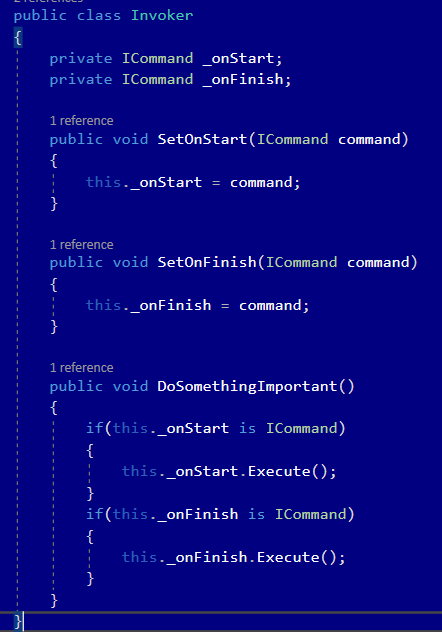
# Implementation:

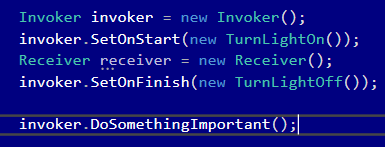
To implement Command patter we need an Command Interface with classes that inherit this interface, such as: TurnLightsOn and TurnLightsOff. We need also an Invoker and a Receiver.











# Consequences:

Benefits:

* Single Responsibility Principle. You can decouple classes that invoke operations from classes that perform them.
* Open/Closed Principle. You can introduce new commands into the app without breaking existing client code.
* Ability to implement undo/redo
* Ability to implement deferred execution of operations.
* Ability to assemble a set of simple commands into a complex one.

Drawbacks:

* Possibility of making the code even more complicated by introducing a new level between senders and receivers.

# Known uses

* To implement Callbacks
* Eliminate switch statements
* Thread pools

# Related patterns

1. Chain of Responsibility, Command, Mediator and Observer address various ways of connecting senders and receivers of requests:
   1. Chain of Responsibility passes a request sequentially along a dynamic chain of potential receivers until one of them handles it.
   2. Command establishes unidirectional connections between senders and receivers.
   3. Mediator eliminates direct connections between senders and receivers, forcing them to communicate indirectly via a mediator object.
   4. Observer lets receivers dynamically subscribe to and unsubscribe from receiving requests.
2. Handlers in Chain of Responsibility can be implemented as Command. In this case, you can execute a lot of different operations over the same context object, represented by a request.
3. Command and Strategy may look similar because you can use both to parametrize an object with some action. However, they have very different intents.
   1. Command can be used to convert any operation into an object. The operation’s parameters become fields of that object. The conversion lets you defer execution of the operation queue it, store the history of commands, send commands to remote services, etc.
   2. On the other hand, Strategy usually describes different ways of doing the same thing, letting you swap these algorithms within a single context class.
4. Prototype can help when you need to save copiers of Command into history.
5. Visitor can be treated as a powerful version of Command pattern. Its objects can execute over various objects of different classes.