You shall not pass.

Ordinary Java statements are perishable

* Cannot undo a filed assignment
* Cannot directly serialize a sequence of actions (calls)

Want an object that represents an operation

* X should change its field Y to value Z
* X should do w()

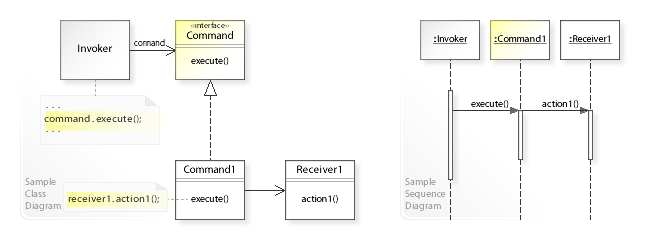
Uses: GUI commands, multi-level undo/redo

An object which represents an instruction to perform a particular action and contains all the information necessary for the action to be taken.

**GOF Definition**

**Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations**

Class Diagram



**Design Participants**

Participants for command design pattern are:

* **Command interface** – for declaring an operation.
* **Concrete command classes** – which extend the Command interface, and has execute method for invoking business operation methods on receiver. It internally has reference of the receiver of command.
* **Invoker** – which is given the command object to carry out the operation.
* **Receiver** – which execute the operation

In command pattern,

* The invoker is decoupled from the action performed by the receiver.
* The invoker has no knowledge of the receiver.
* The invoker invokes a command, and the command executes the appropriate action of the receiver.
* Thus, the invoker can invoke commands without knowing the details of the action to be performed.
* In addition, this decoupling means that changes to the receiver’s action don’t directly affect the invocation of the action.

**Command vs Strategy**

Typically the Command pattern is used to make an object out of *what needs to be done* -- to take an operation and its arguments and wrap them up in an object to be logged, held for undo, sent to a remote site, etc. There will tend to be a large number of distinct Command objects that pass through a given point in a system over time, and the Command objects will hold varying parameters describing the operation requested.

The Strategy pattern, on the other hand, is used to specify *how* something should be done, and plugs into a larger object or method to provide a specific algorithm. A Strategy for sorting might be a merge sort, might be an insertion sort, or perhaps something more complex like only using merge sort if the list is larger than some minimum size. Strategy objects are rarely subjected to the sort of mass shuffling about that Command objects are, instead often being used for configuration or tuning purposes.

The main difference is that the command does some action over the object. It may change the state of an object while Strategy decides how to process the object. It encapsulates some business logic.

**Command vs CQRS**

Command pattern in its raw form is like GOF says.

In CQRS the command is just a DTO (data transfer object) because CQRS is, in most cases, implemented with events or message bus that handles the command. In CQRS you send the command to the system and the system has some kind of bus or event architecture that allows autonomous component to subscribe to handle the command; in this way you can create a responsibility chain and it is more SOLID to, for example, work with read and write models.

It is more a chain of responsibility pattern that command pattern but you keep the advantages of command pattern because you still have commands and can, for example, implement UNDO an action easily like in command pattern.

**Command vs Chain of responsibility**

Command is basically just a command encapsulated in an object. Chain of responsibility is more an object trying to handle something & if not, pass it onto the next one in the 'chain'.