Command Design Pattern

Nidhi V Kulkarni
Priyanka V Rao

What is *Command* design pattern?

- An object encapsulates all ***information*** needed to perform an action.

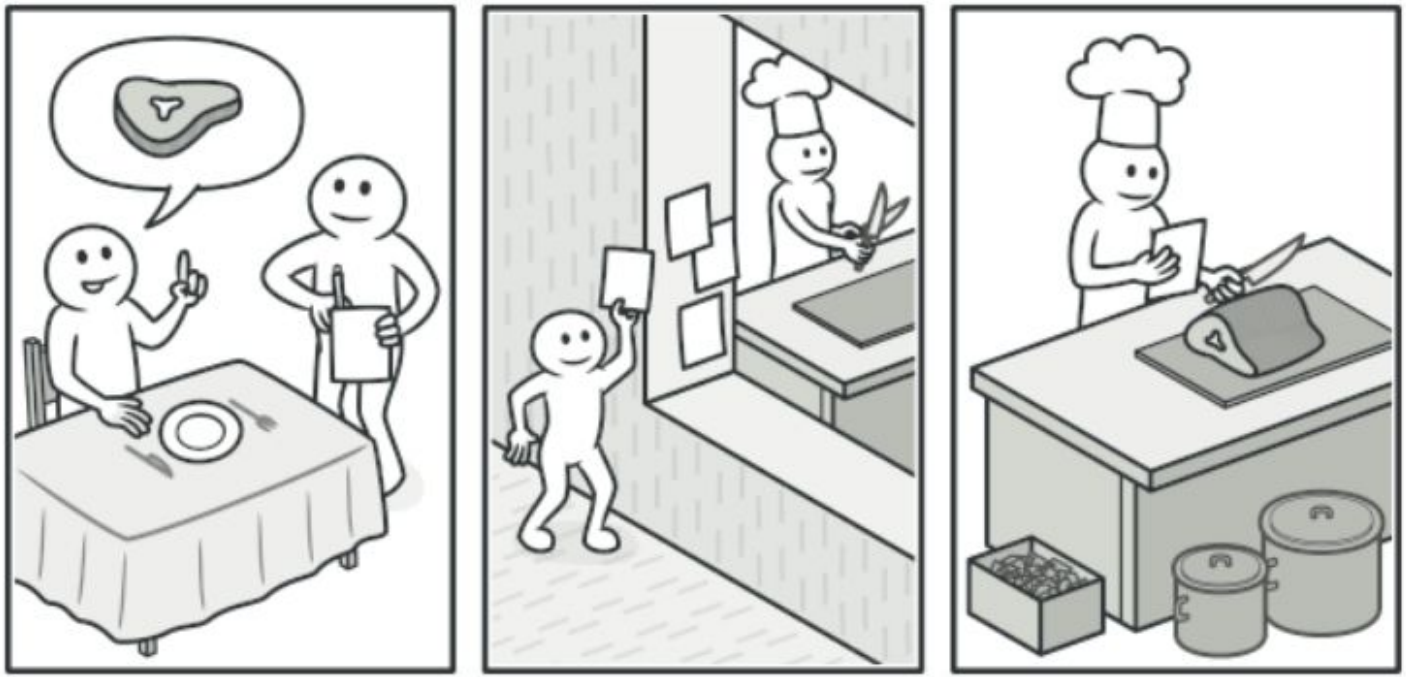
- Issue requests to objects without knowing -

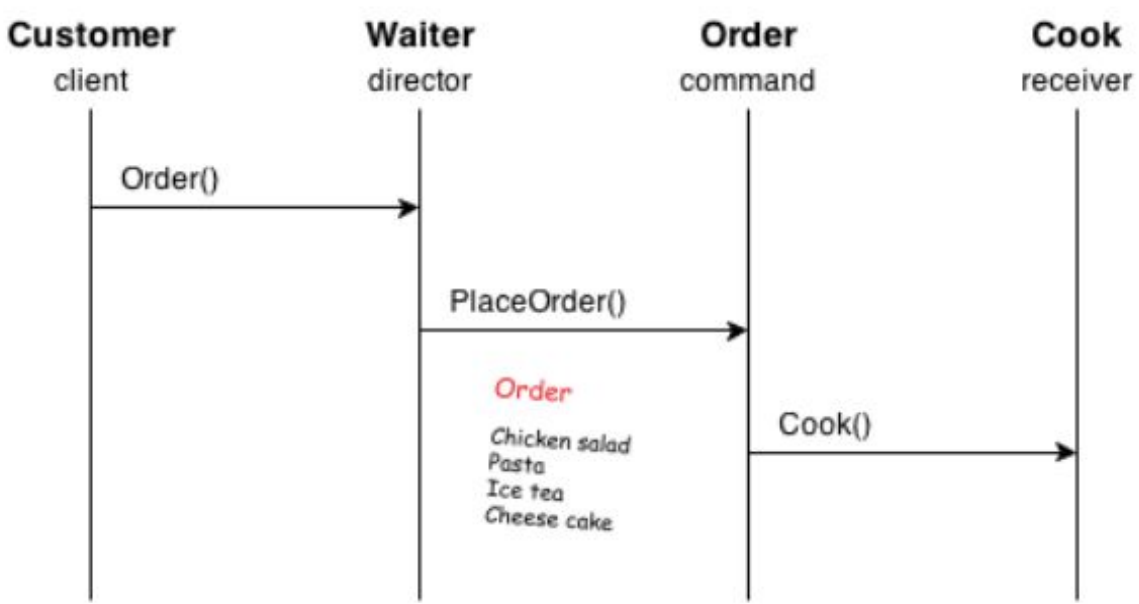
- The operation being requested
- The receiver of the request.

Information-

- Method name
- The object that owns the method
- Values for the method parameters.

Real time example




Customer
client
Waiter
director
Order
command
Cook
receiver
Order()
PlaceOrder()
Order
Chicken salad
Pasta
Ice tea
Cheese cake
Cook()

Four terms-

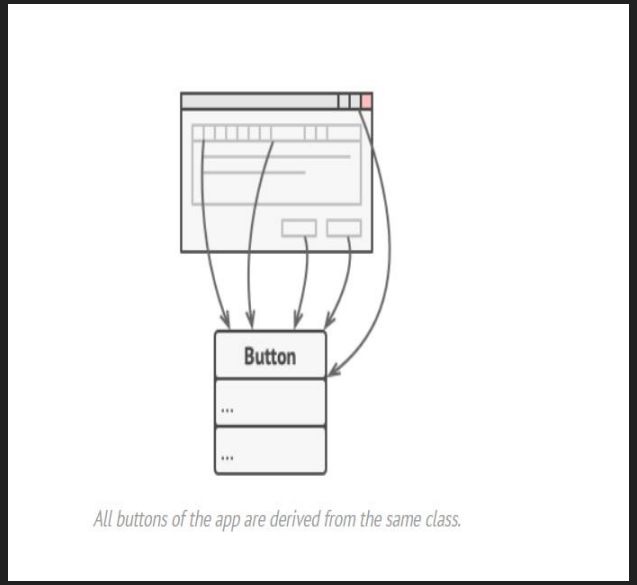
- Client
- Command
- Invoker
- Receiver

Technical example

- Text editor app-> Create toolbar -> Many buttons
- **Button** - Base class
- Buttons look similar, but different operations

Code for the various click handlers of these buttons?

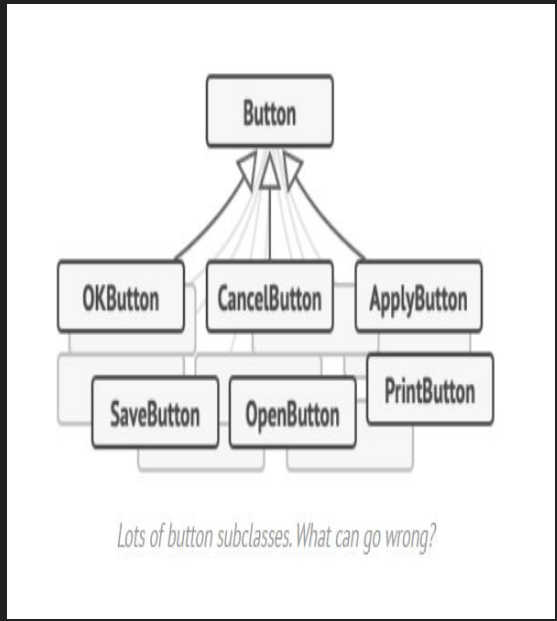
- Create subclasses for each button.
- Subclasses contain code for the button click.



Flaw 1

- Many number of classes
- Risk breaking the code in subclasses - *Button* class

GUI code is dependent on business logic code



Flaw 2

Copy, paste, print or save would need to be invoked from multiple places.

Print button -

- Toolbar
- The context menu,
- Hit Ctrl+P on the keyboard- *shortcut*

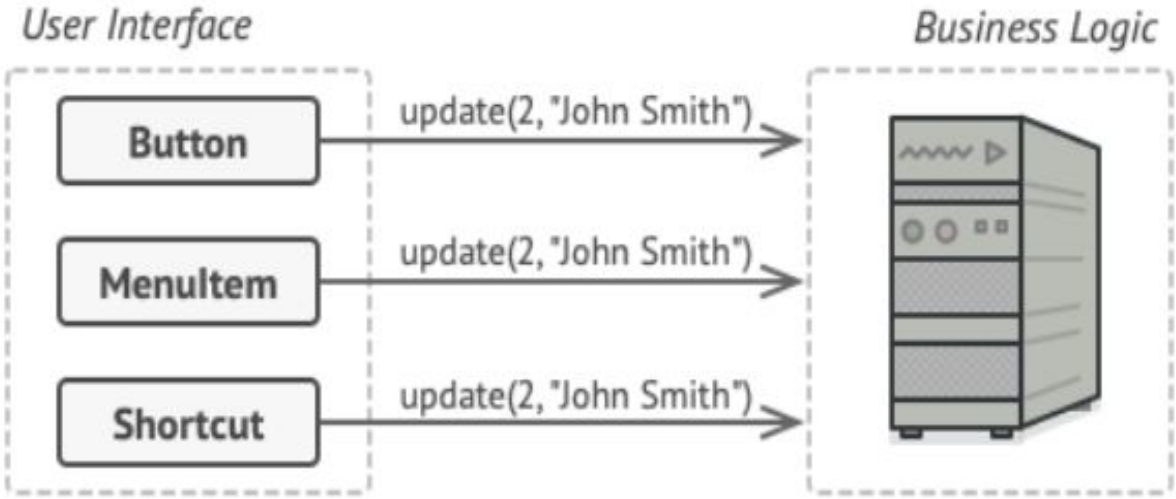
Solution?

Duplicate the operation’s code in many classes

Make menus dependent on buttons.

Solution?

- Good software design - ***Principle of separation of concerns***
- Breaking an app into layers.
- Example - a layer for the GUI and another layer for the business logic.
- A GUI object calls a method of a business logic object
- This is done by passing it some arguments.
- This process is usually described as one object sending another a request.



The GUI objects may access the business logic objects directly.

Ideal approach

The **Command** pattern- GUI objects shouldn’t send these requests directly.

Instead, you should extract all of the request details-

- The object being called
- The name of the method
- Arguments in a command class with a method that triggers the request.

Object called- **Business logic**

Name of the method - **Update**

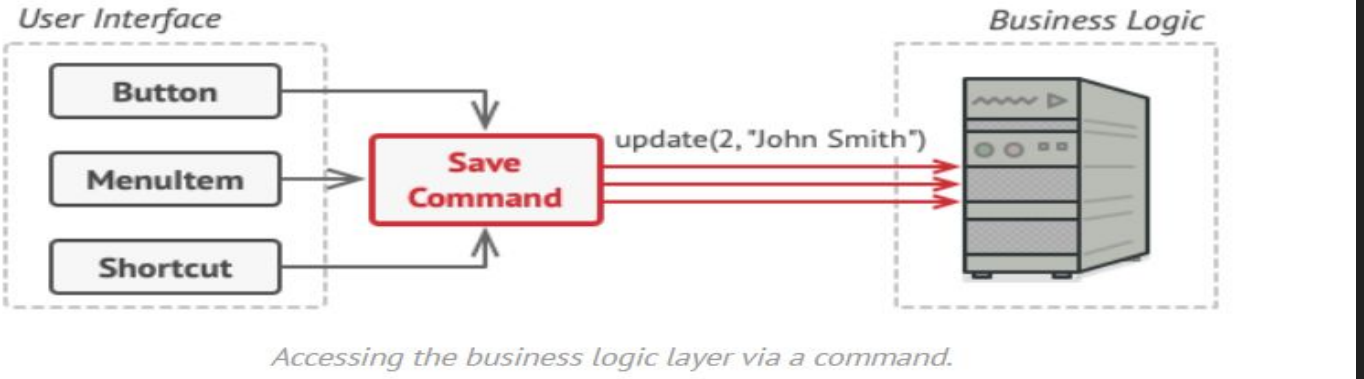
Arguments- **2, “John Smith”**

Command objects- links between various GUI and business logic objects.

The GUI object doesn’t need to know -

- What business logic object
- What request and how it’ll be processed.

The GUI object just triggers the **command**, which handles all the details.



Accessing the business logic layer via a command.

After we apply the Command pattern-

- No longer need all those button subclasses
- Put a single field into the base **Button** class
- Stores a reference to a command object
- The button execute that command on a click.

Commands -

- Middle layer
- Reduces coupling between the GUI and business logic layers

Real world applications

- GUI buttons and Menu items
- Multi-level Undo/Redo
- Remote Control Devices
- Transactional behavior

Pros and Cons

Pros

- Encapsulation
- Decoupling
- Undo/Redo functionality
- Sequencing of commands
- Logging

Cons

- Increased complexity
- Increased memory usage
- Performance overhead

References

https://refactoring.guru/design-patterns/command

https://www.tutorialspoint.com/design_pattern/command_pattern.htm

https://en.wikipedia.org/wiki/Command_pattern#:~:text=In%20object%2Doriented%20programming%2C%20the,values%20for%20the%20method%20parameters.

https://sourcemaking.com/design_patterns/command