Behavioral Design Pattern - Command

**Group Members**: Chih-hao, Elham, George, Guilherme

**Description:**  
Sometimes we have multiple elements calling similar actions (logic) and we want to keep such actions decoupled from their callers so they can be invoked by any trigger without awkward dependencies or volatile code.  
The “command” pattern achieves that by using an object to encapsulates the information needed to perform an action. Essentially a behaviour wrapped in a class that later fills a slot with its functionality.

**Problem:**Imagine you have a category of objects attempting to perform a list of actions. Let’s use remote controls as an example, you might have one for the lights, another for AC, and another for the radio.   
You might think that it’s a good idea to create subclasses for each, but what if the user now decides to use the light switch on the wall instead of the remote control, would you repeat the code? Or create an instance of the remote control just for the for the wall switch to use its methods? The responsibility of the remote control or the switch should be to send a signal, not to modify the state of things.

Another example would be a gun, it might shoot different types of bullets, even though it’s using the same trigger. These different types of bullets might also be used by other guns (other triggers); we wouldn’t want to add the bullet behaviour inside of the gun. This would make them coupled.   
But we could during runtime decide to populate our gun with a different type of bullet, giving it a different behaviour.

Imagine you have a category of objects attempting to perform a list of actions. You don’t want to create variations (subclasses) for every different action. This could cause such objects to be responsible for multiple areas, which is not ideal. Plus you would have to repeat your code whenever different objects share the same actions.

specially if such actions might also be controlled from other sources.

**Solution:**Decouple elements by wrapping behaviour into a class and allowing invokers to call them. (Command pattern)  
For example: A remote control can be a single class that is populated with

You might have multiple elements attempting to perform similar actions.   
You have a category of elements that perform different actions. Let’s use the example a “Gun”  
In the attempt to isolate their actions, we might create multiple subclasses.

This offers scalability, expansion, and decoupling while allowing every class to maintain their own responsibility.

Method name, method owner, values for method parameters

This offers scalability, expansion, and decoupling while allowing every class to maintain their own responsibility.

Example:

Alexa:

Slots:

Light Command 1:

Light Command 2:

Music Command 3: On/Off/setVolume/setPlaylist