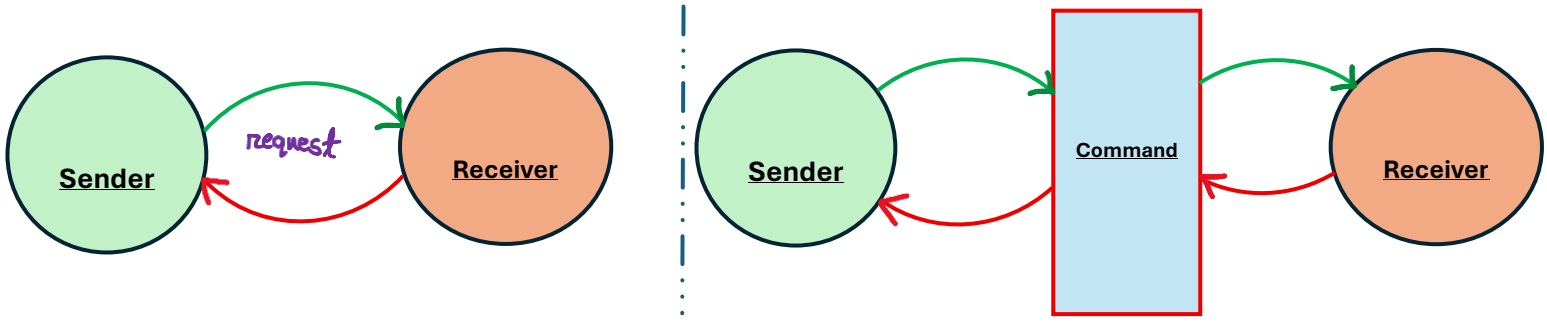
Command Design Pattern

Defn: Instead of sending ***a request*** directly from the **sender** to the **receiver**, the ***Command pattern*** **encapsulates** the **request** as an ***object***. This approach allows the **request** to be ***stored***, ***tracked***, and ***managed***—enabling features such as ***queuing***, ***logging***, and support for ***undo*** and ***redo*** **operations**.



Use Case: Smart Home Automation System:

In a ***smart home*** automation system, imagine an ***application*** with **buttons** to **control** various **smart devices** such as ***lights***, ***fans***, and ***air conditioners***. These buttons are **mapped** to a ***remote control interface*** used by the user to interact with the devices.

A naive approach would be to implement a ***Remote class*** that holds ***direct references*** to the **device objects** and provides methods like ***turnOnLight()***, ***turnOffFan()***, etc. While this approach may work for simple scenarios, it leads to ***tight coupling*** between the **remote** and the **devices**.

This tight coupling introduces several ***design issues***:

- It **violates the Open/Closed Principle (OCP)**, as any **modification** to **device behaviour** requires **changes** to the ***Remote class***.
- It ***limits*** **scalability** and **flexibility**, making it **difficult** to **add** ***new devices*** or **modify** ***existing actions***.

Solution: *Command Design Pattern*

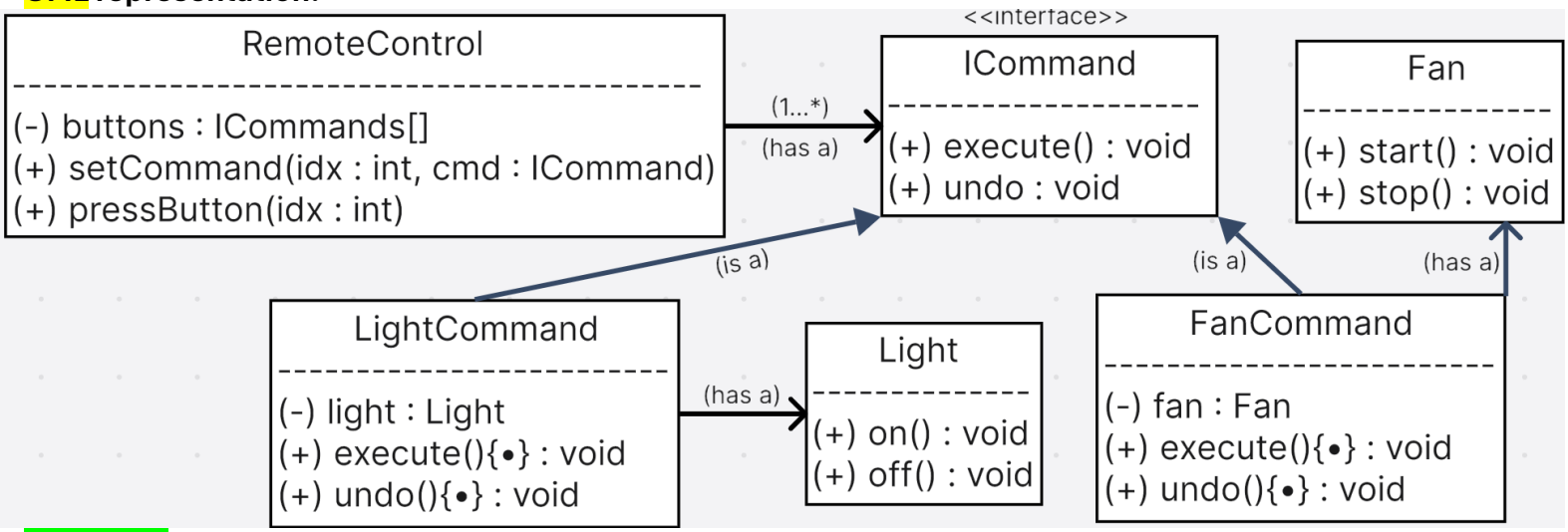
To address these concerns, the ***Command Design Pattern*** recommends **encapsulating** each **request** (e.g., turning on a light) as a **separate** ***command object***. Instead of the ***remote*** **directly invoking device methods**, it **delegates** the **responsibility** to ***command objects***.

Each **button** on the **remote** is associated with ***a command***, and each **command** knows ***how to perform*** the requested **action** on the appropriate device. This ***decouples*** the ***invoker*** (remote) from the ***receiver*** (device), enabling:

- ***Easy modification*** or ***extension*** of behaviour without altering existing code.
- Support for **additional features** such as ***undo***, ***redo***, and ***command queuing***.

By introducing a ***command layer*** between the ***user interface*** and the ***smart devices***, the system becomes **more** ***modular***, ***maintainable***, **and** ***extensible***.

UML representation:



Code Link: https://github.com/sibasundarj8/-System-Design-/tree/main/Codes/15__Command%20Design%20Pattern%20code