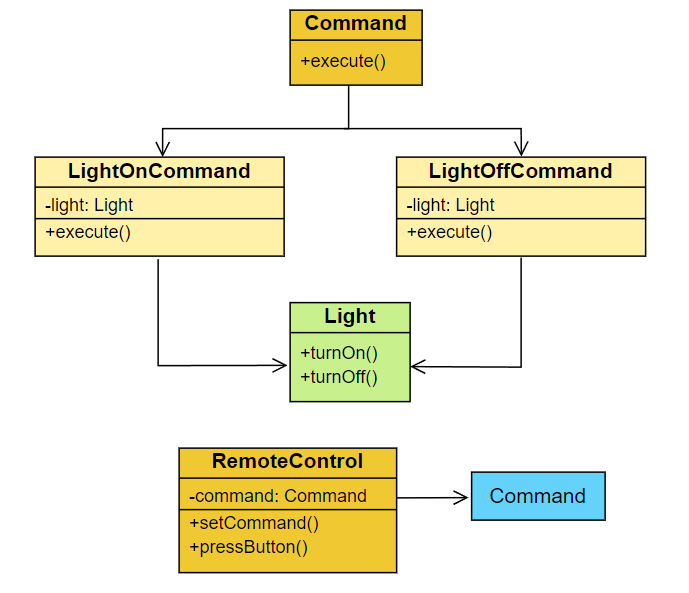
**Exercise 9: Implementing the Command Pattern**

The Command pattern implementation demonstrates a home automation system where a *RemoteControl* can turn a Light on and off. The system uses a Command interface with concrete *LightOnCommand* and *LightOffCommand* classes, each holding a reference to a Light object. The *RemoteControl* acts as an invoker, holding a Command reference which can be set and executed. The Light class serves as the receiver, containing the actual on/off functionality. In the main method, a Light object is created along with corresponding on and off commands. These commands are then set on the *RemoteControl* and executed, demonstrating how the pattern allows for flexible control of devices without the *RemoteControl* needing to know the specifics of each operation or device.

Each and every class description :

1. **Command (Interface):**

* Defines the contract for all concrete commands.
* Contains a single method execute().
* Acts as the base for *LightOnCommand* and *LightOffCommand*.
* Allows for easy addition of new commands in the future.
* Decouples the sender (*RemoteControl*) from the receiver (Light).

1. **LightOnCommand:**

* Concrete implementation of the Command interface.
* Holds a reference to the Light object (receiver).
* Implements the *execute()* method to turn the light on.
* Encapsulates the action of turning the light on.
* Acts as a bridge between the *RemoteControl* and Light.

1. **LightOffCommand:**

* Concrete implementation of the Command interface.
* Holds a reference to the Light object (receiver).
* Implements the *execute()* method to turn the light off.
* Encapsulates the action of turning the light off.
* Acts as a bridge between the *RemoteControl* and Light.

1. **RemoteControl:**

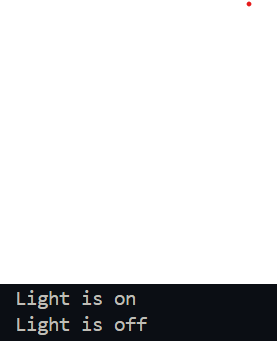
* Acts as the invoker in the Command pattern.
* Holds a reference to a Command object.
* Has methods to set the command and execute it.
* Doesn't know about the specific command implementation.
* Provides a way to issue commands without knowing the details.

1. **Light:**

* Acts as the receiver in the Command pattern.
* Contains the actual implementation of turning the light on and off.
* Has methods *turnOn()* and *turnOff().*
* Represents the device being controlled.
* Is decoupled from the *RemoteControl* and Command objects.

Here is the Github code repo -

Here is the output of the code



This implementation demonstrates the Command pattern, allowing for flexible and extensible control of devices in a home automation system.