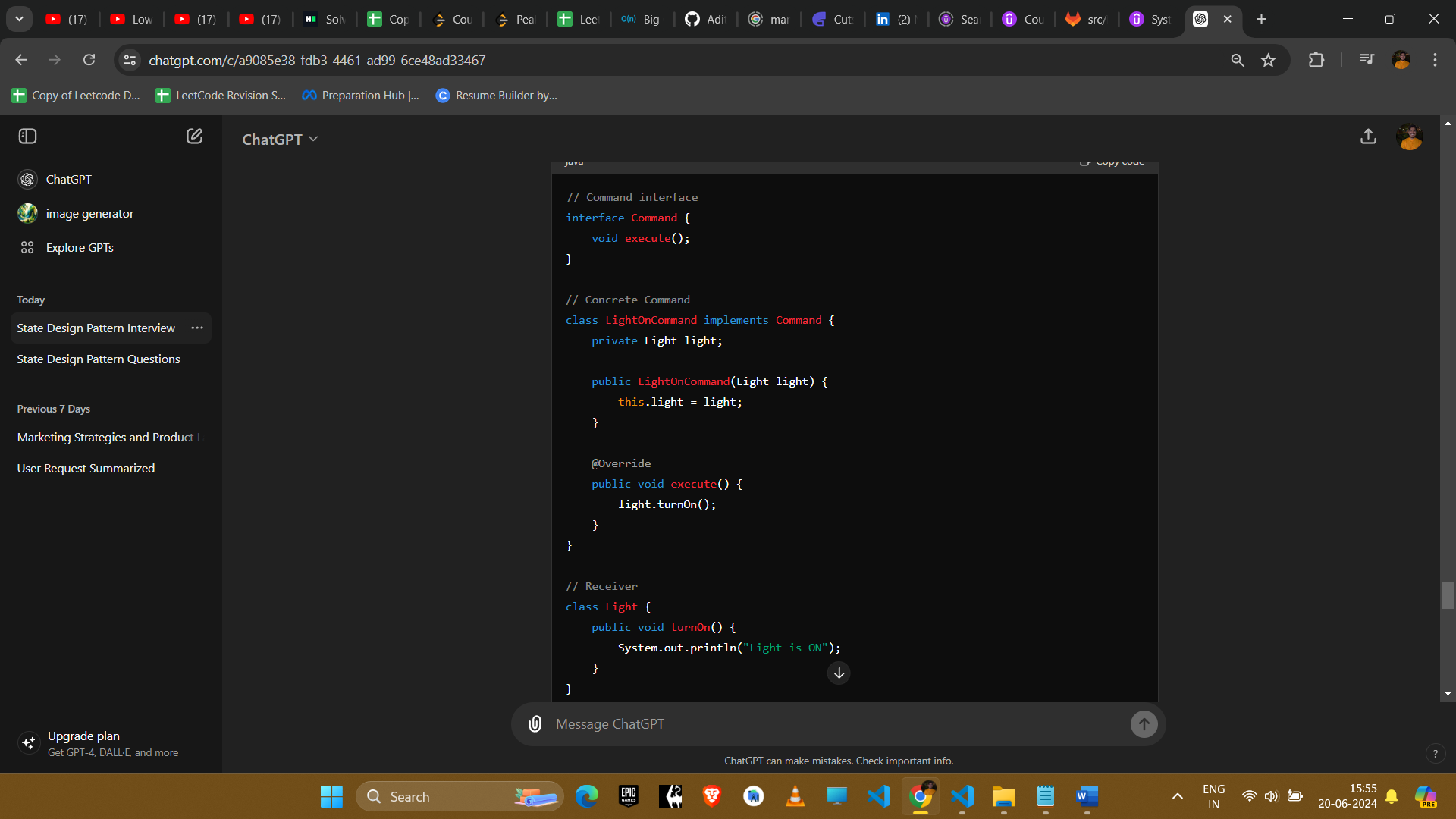
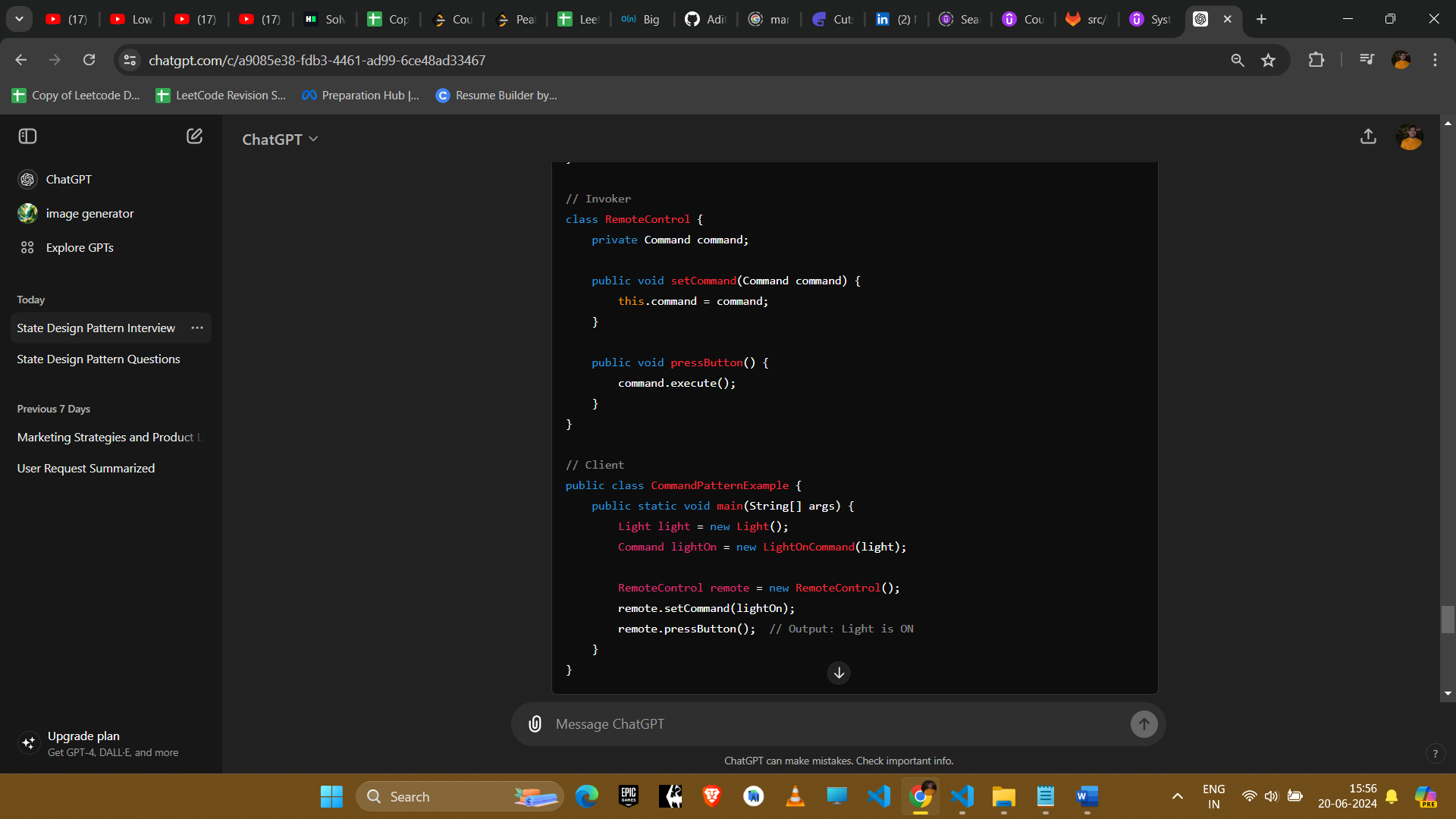
**Command Design Pattern**

The Command design pattern is a behavioral pattern that turns a request into a stand-alone object containing all information about the request. This decouples the sender of the request from the receiver, allowing for operations like queuing requests, logging, and undoing actions.





### Explanation

1. **Command Interface**: Declares the execute method.
2. **Concrete Command**: Implements the Command interface and defines the binding between the action and the receiver.
3. **Receiver**: Contains the actual business logic (in this case, the Light class).
4. **Invoker**: Triggers the command (in this case, the RemoteControl class).
5. **Client**: Configures the commands and the invoker.

### Example Uses in Amazon Interviews

#### 1. **Undo/Redo Functionality**

* **Scenario**: Implementing an undo/redo feature in a text editor or any application with reversible actions.
* **Implementation**: Each action (e.g., typing, deleting) is encapsulated as a command. Commands are stored in a history list to allow undoing and redoing.

#### 2. **Transaction Management**

* **Scenario**: Managing a series of financial transactions that need to be executed or rolled back as a unit.
* **Implementation**: Each transaction operation (e.g., debit, credit) is a command. Commands can be committed or rolled back as needed.

#### 3. **Task Scheduling System**

* **Scenario**: Scheduling tasks to be executed later or periodically.
* **Implementation**: Each task is a command object. The scheduler invokes the command at the appropriate time.

#### 4. **Remote Control System**

* **Scenario**: Controlling multiple devices (e.g., TV, AC, Lights) with a remote control.
* **Implementation**: Each button on the remote is associated with a command object that performs a specific action on a device.

#### 5. **Macro Recording**

* **Scenario**: Recording a series of user actions to be played back later.
* **Implementation**: Each user action is encapsulated as a command. Commands are recorded in sequence and can be replayed in the same order.

### Conclusion

The Command design pattern is useful for scenarios where actions need to be parameterized, queued, logged, or undone. It decouples the sender of the request from the receiver, providing flexibility and extensibility in handling requests.