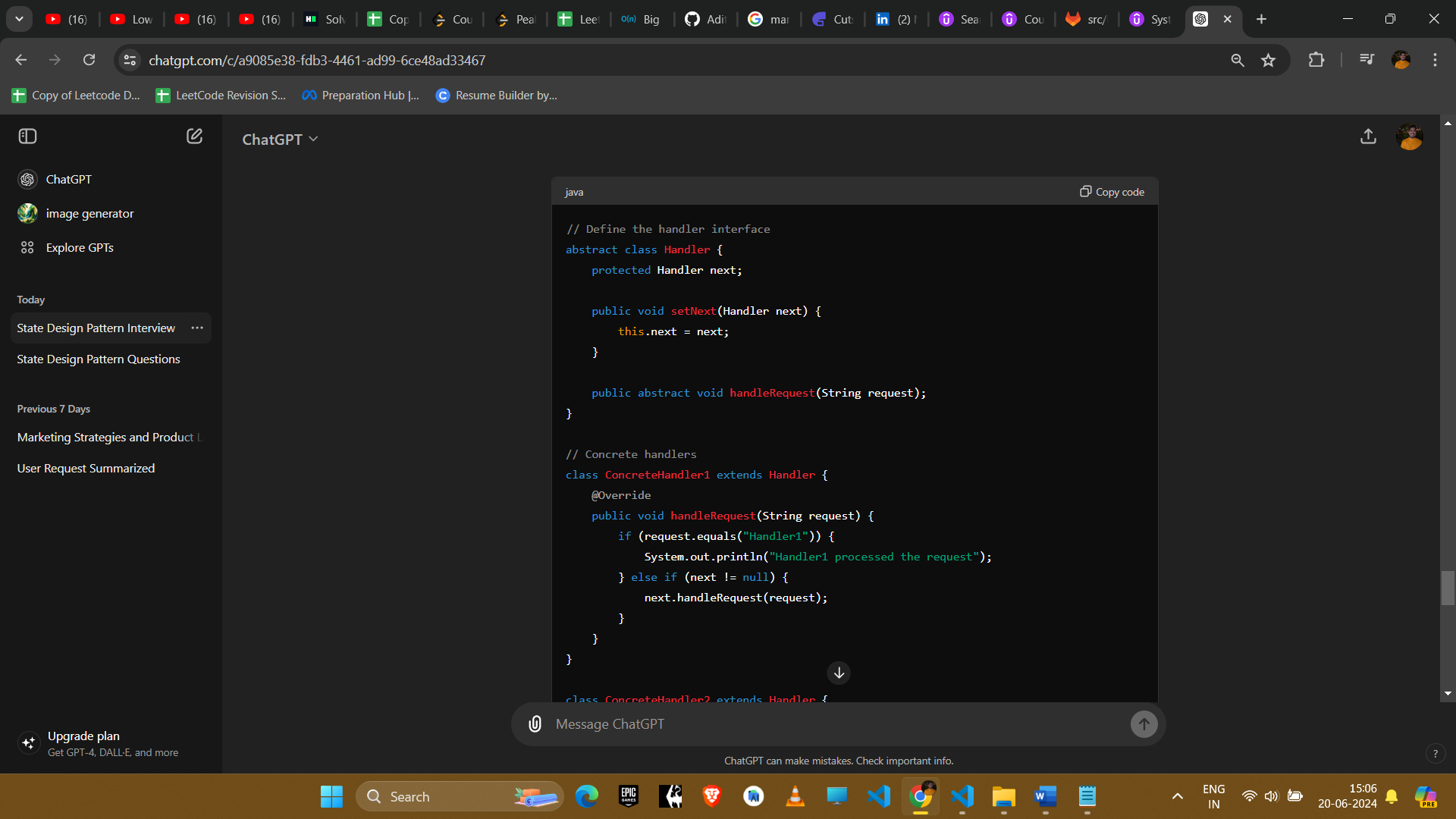
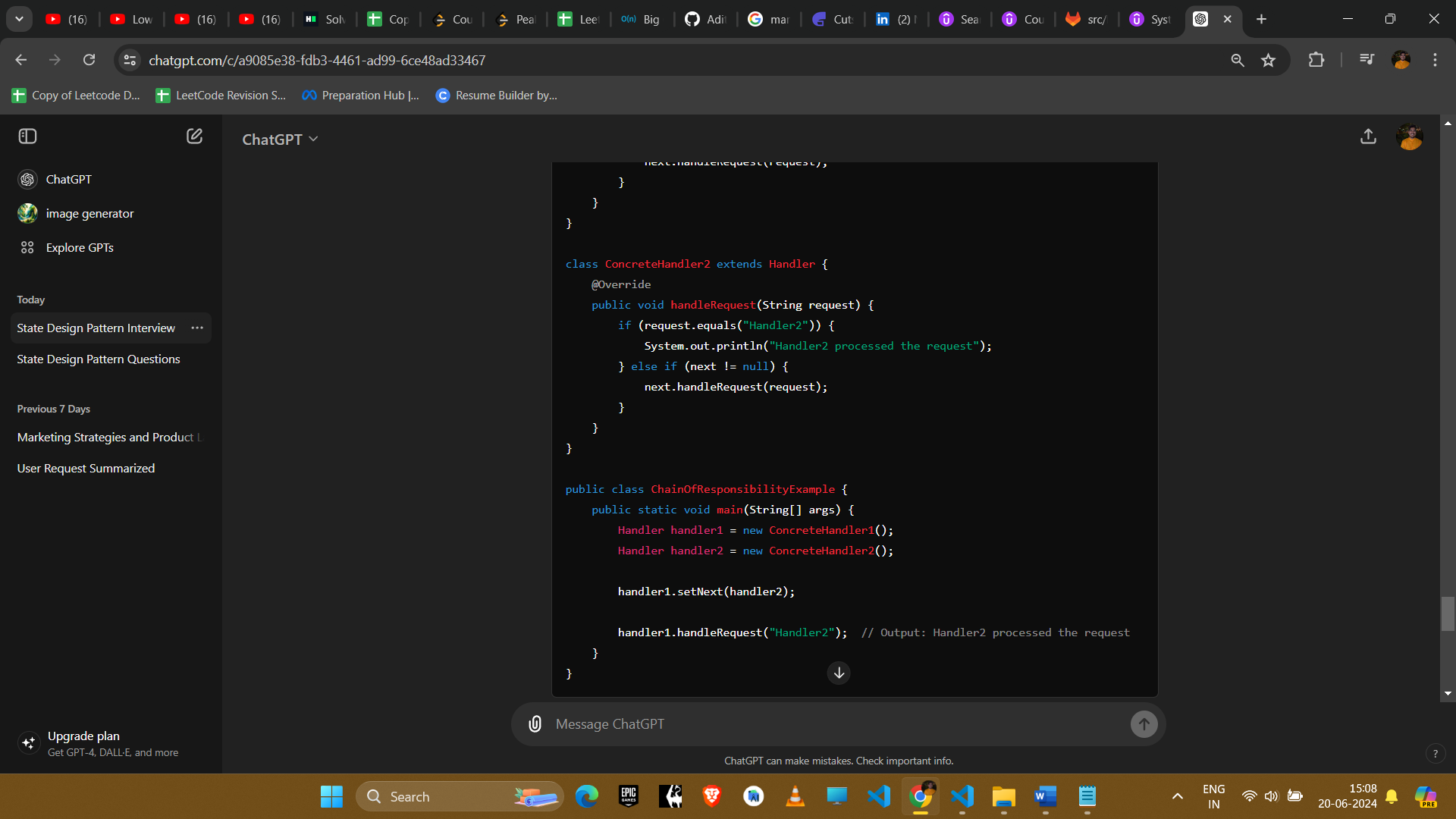
**Chain of Responsibility Design Pattern**

The Chain of Responsibility design pattern is a behavioral pattern used to pass a request along a chain of handlers. Each handler decides either to process the request or pass it to the next handler in the chain. This pattern decouples the sender of a request from its receivers, giving multiple objects a chance to handle the request.



Eg use cases : ATM, Vending Machine, Logger (Amazon)



### Explanation

1. **Handler Interface**: Defines an interface for handling requests and a method to set the next handler.
2. **Concrete Handlers**: Implement the handler interface and decide whether to process the request or pass it to the next handler.
3. **Client Code**: Creates the handlers, chains them together, and sends a request.

### Example Uses in Amazon Interviews

#### 1. **Technical Support System**

* **Scenario**: Different levels of support (e.g., Level 1, Level 2, Level 3) handle customer issues.
* **Implementation**: Each support level checks if it can handle the issue; if not, it passes the issue to the next level.

#### 2. **Logging System**

* **Scenario**: Different loggers (e.g., Console Logger, File Logger, Error Logger) process log messages.
* **Implementation**: Each logger checks if it should log the message; if not, it passes the message to the next logger.

#### 3. **Request Filtering**

* **Scenario**: Different filters (e.g., Authentication, Authorization, Validation) process HTTP requests.
* **Implementation**: Each filter checks and processes the request; if it can't fully handle it, it passes it to the next filter.

#### 4. **Event Handling System**

* **Scenario**: Different event handlers process user actions in a UI.
* **Implementation**: Each handler checks if it can handle the event; if not, it passes it to the next handler.

### Conclusion

The Chain of Responsibility pattern is useful for scenarios where multiple objects might handle a request and the handler isn't known in advance. It promotes loose coupling and flexibility in assigning responsibilities to objects.