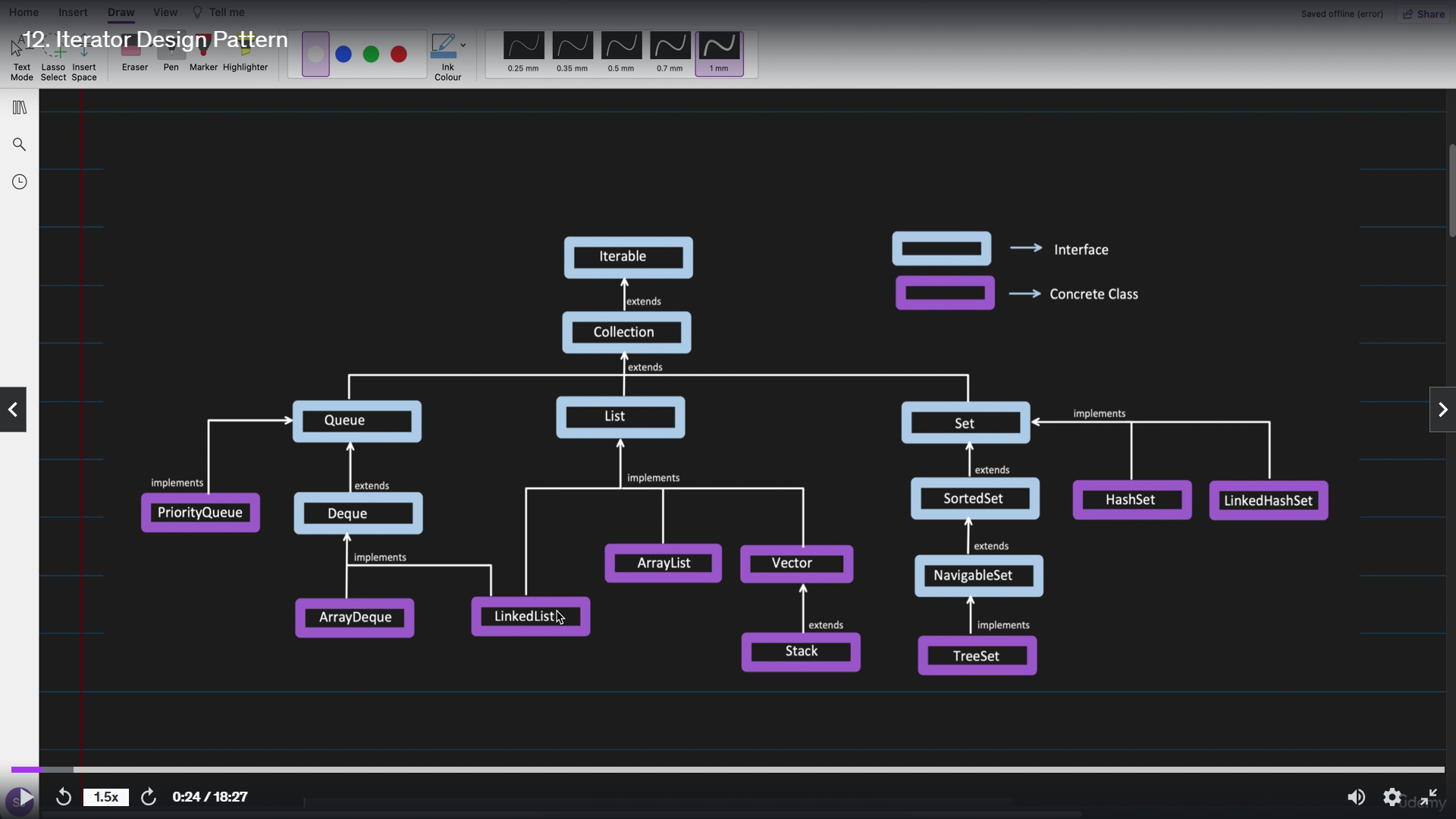
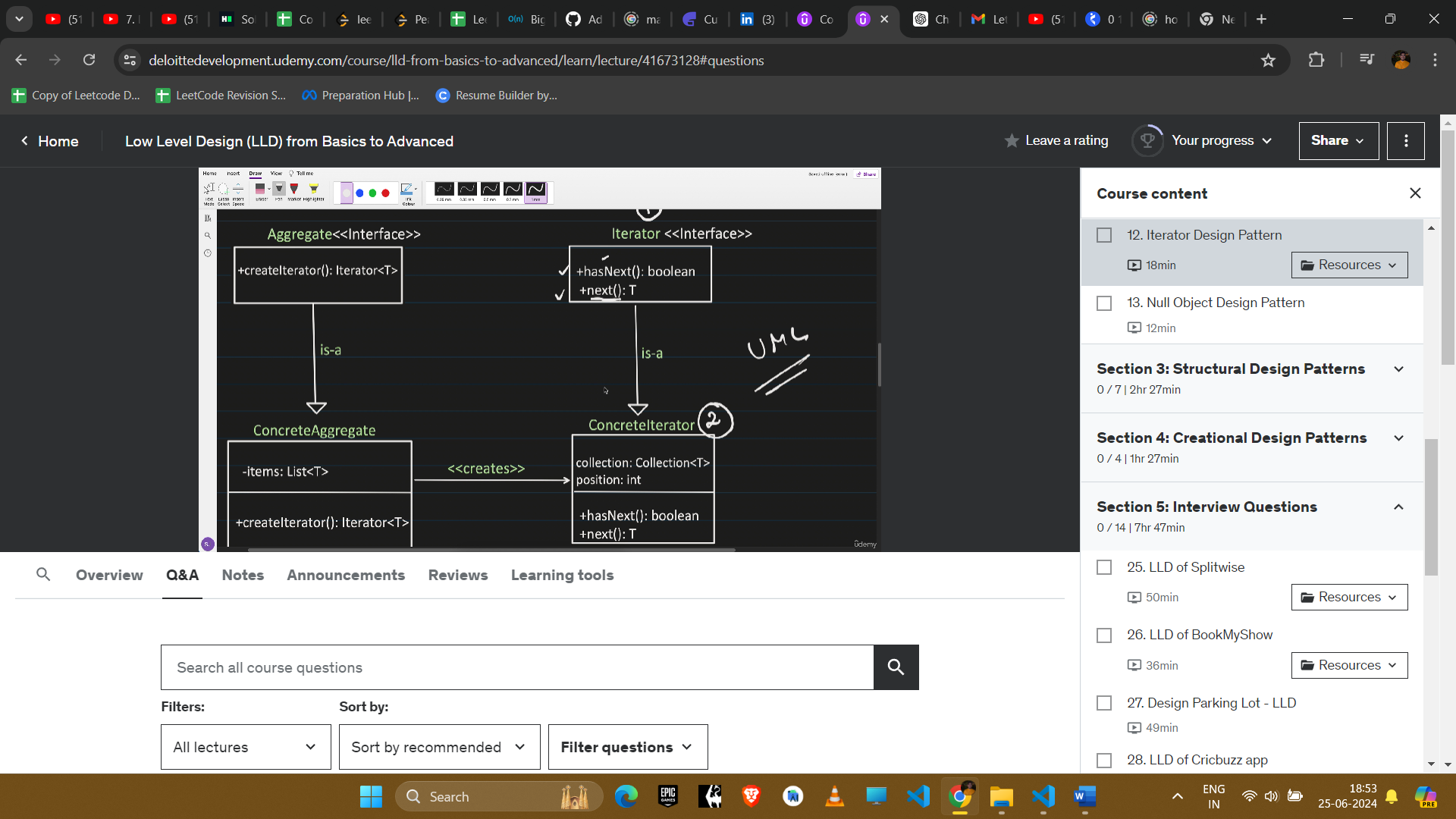
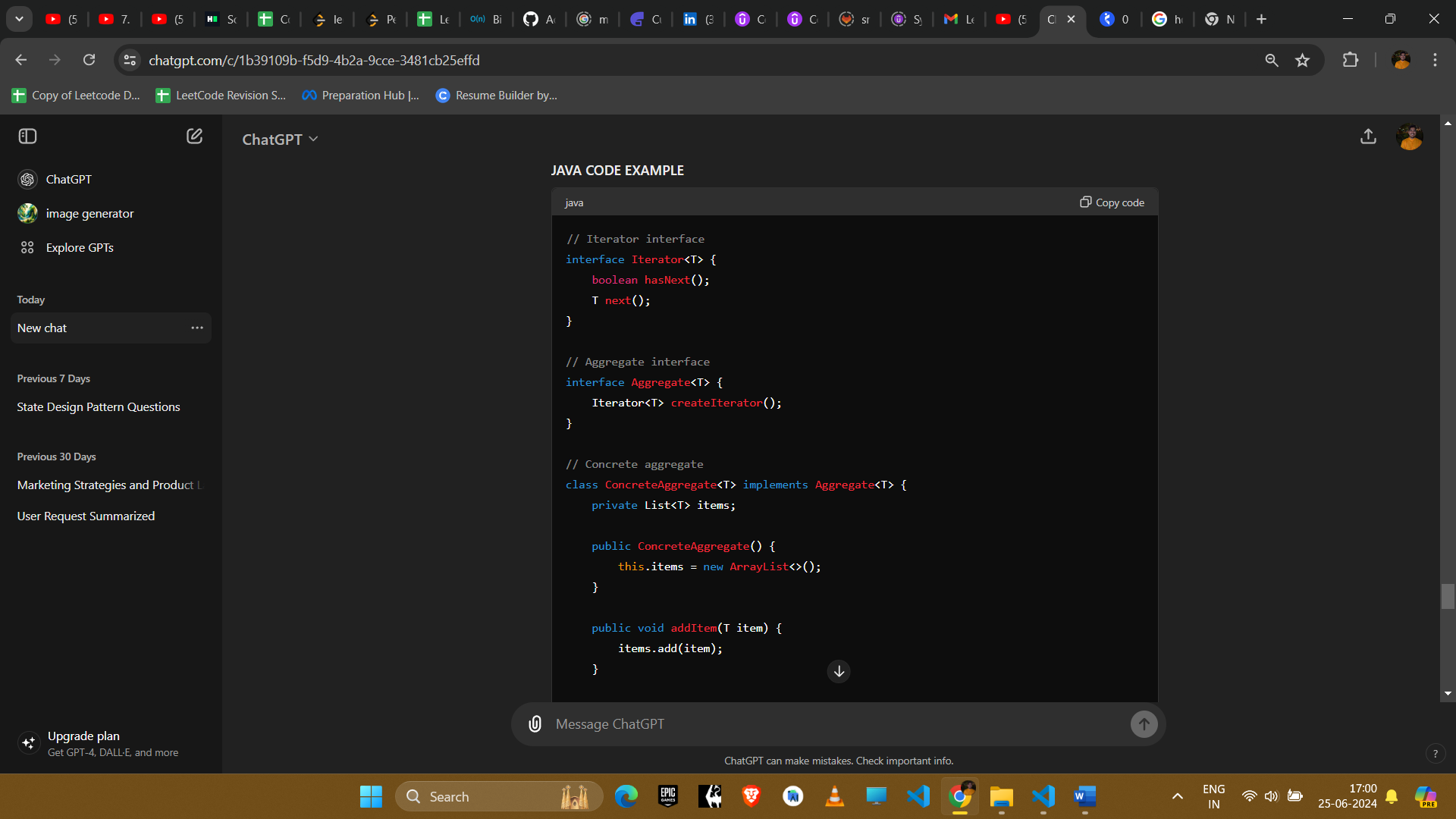
**Iterator Design Pattern**

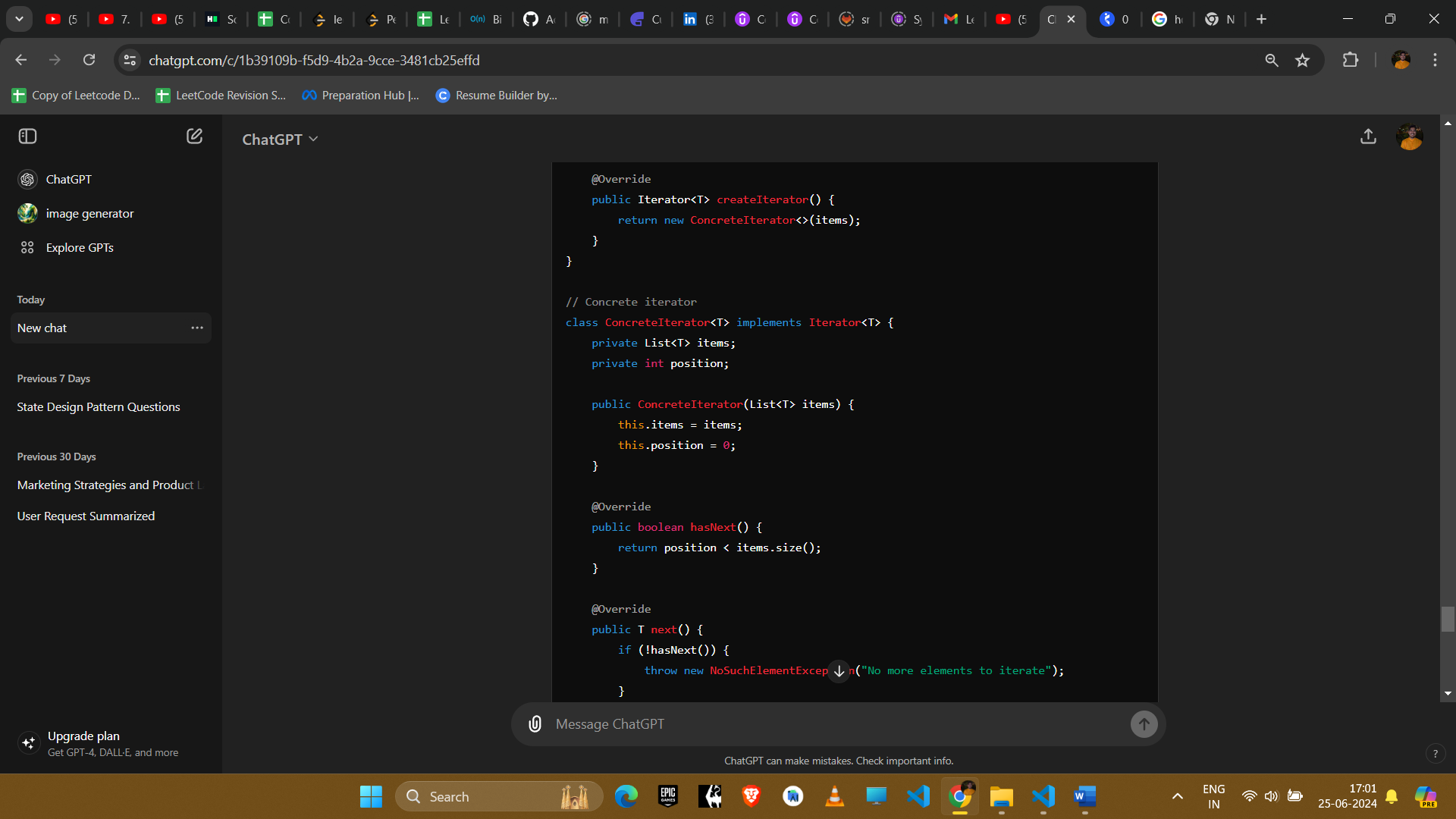
The Iterator design pattern is a behavioural pattern that provides a way to access elements of an aggregate object sequentially without exposing its underlying representation. The Iterator pattern allows you to loop through a collection of objects (like a list or array) without needing to understand the internal structure of that collection. It provides a common interface for different types of collections, ensuring that you can use the same methods to traverse any collection, regardless of its specific implementation.

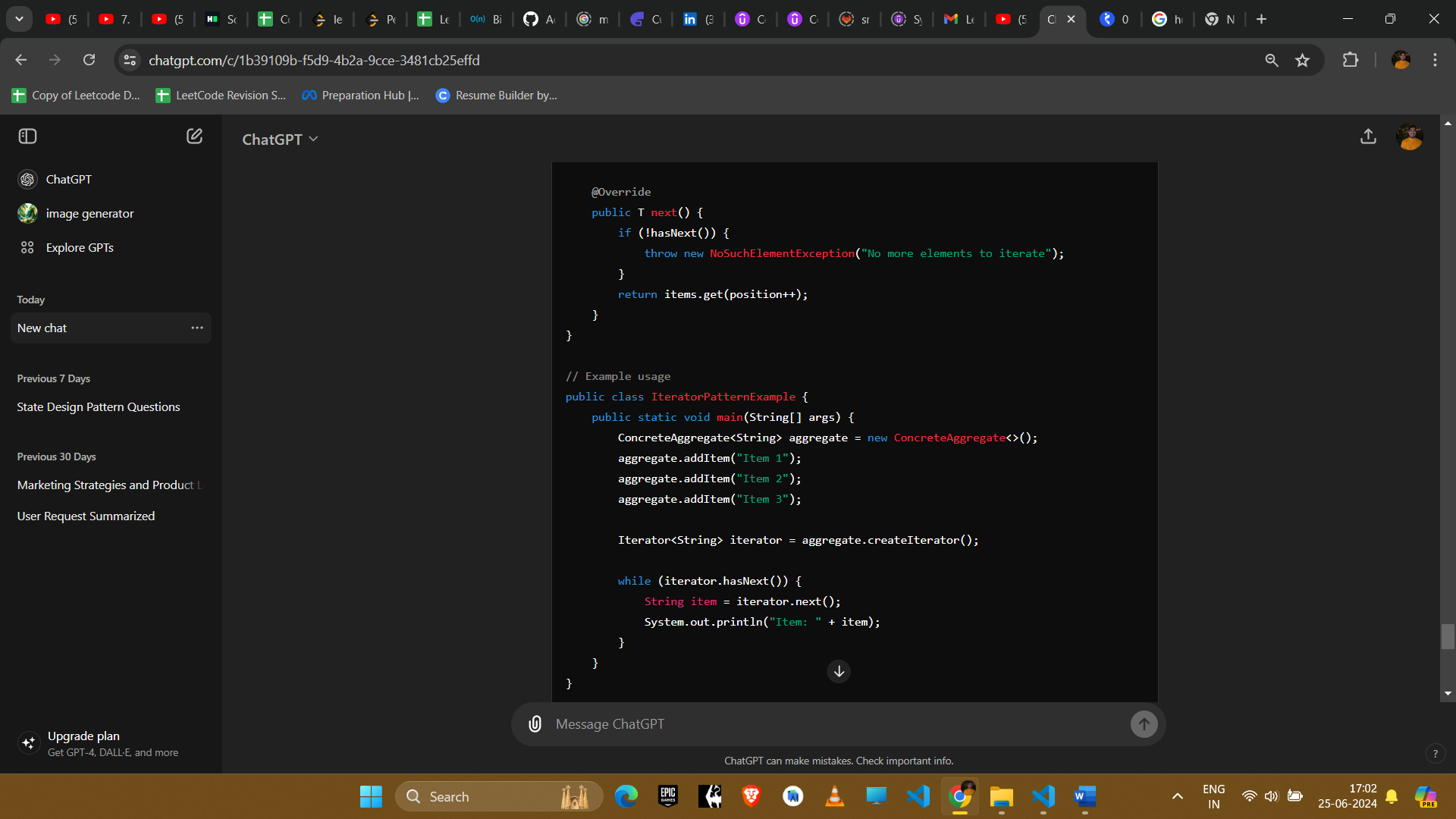
Eg usage: Java collections











#### Explanation

1. **Iterator Interface**: Defines methods to access elements sequentially (hasNext() and next()).
2. **Aggregate Interface**: Defines a method to create an iterator.
3. **Concrete Aggregate**: Implements the aggregate interface and provides the collection of items.
4. **Concrete Iterator**: Implements the iterator interface and traverses the collection of items.

#### Example Uses in Amazon Interviews

1. **Collection Traversal**
   * **Scenario**: Iterating through elements of a collection without exposing its internal structure.
   * **Implementation**: Iterators provide a uniform way to traverse various types of collections (lists, trees, etc.).
2. **File System Navigation**
   * **Scenario**: Navigating through files and directories in a file system.
   * **Implementation**: File system objects are represented as aggregates, and iterators facilitate traversal.
3. **Database Query Results**
   * **Scenario**: Iterating over query results fetched from a database.
   * **Implementation**: Iterators manage the retrieval and traversal of database query results efficiently.
4. **Custom Data Structures**
   * **Scenario:** You have a custom data structure, such as a graph or a linked list, and want to provide a way to traverse it.
   * **Implementation:** Implement an iterator for your custom data structure to allow easy and consistent traversal.

#### Conclusion

The Iterator pattern simplifies the traversal of complex data structures and enhances code readability by separating the collection from the traversal logic. It promotes flexibility and reusability by allowing different types of aggregates to use the same iterator interface, making it ideal for scenarios where collections may vary in structure or size.