**Builder Design Pattern**

The Builder design pattern is a creational pattern that provides a way to construct complex objects step by step. It separates the construction of an object from its representation, allowing the same construction process to create different representations.

**Limitation:** Cannot handle dynamic requests like Decorator Design Pattern.

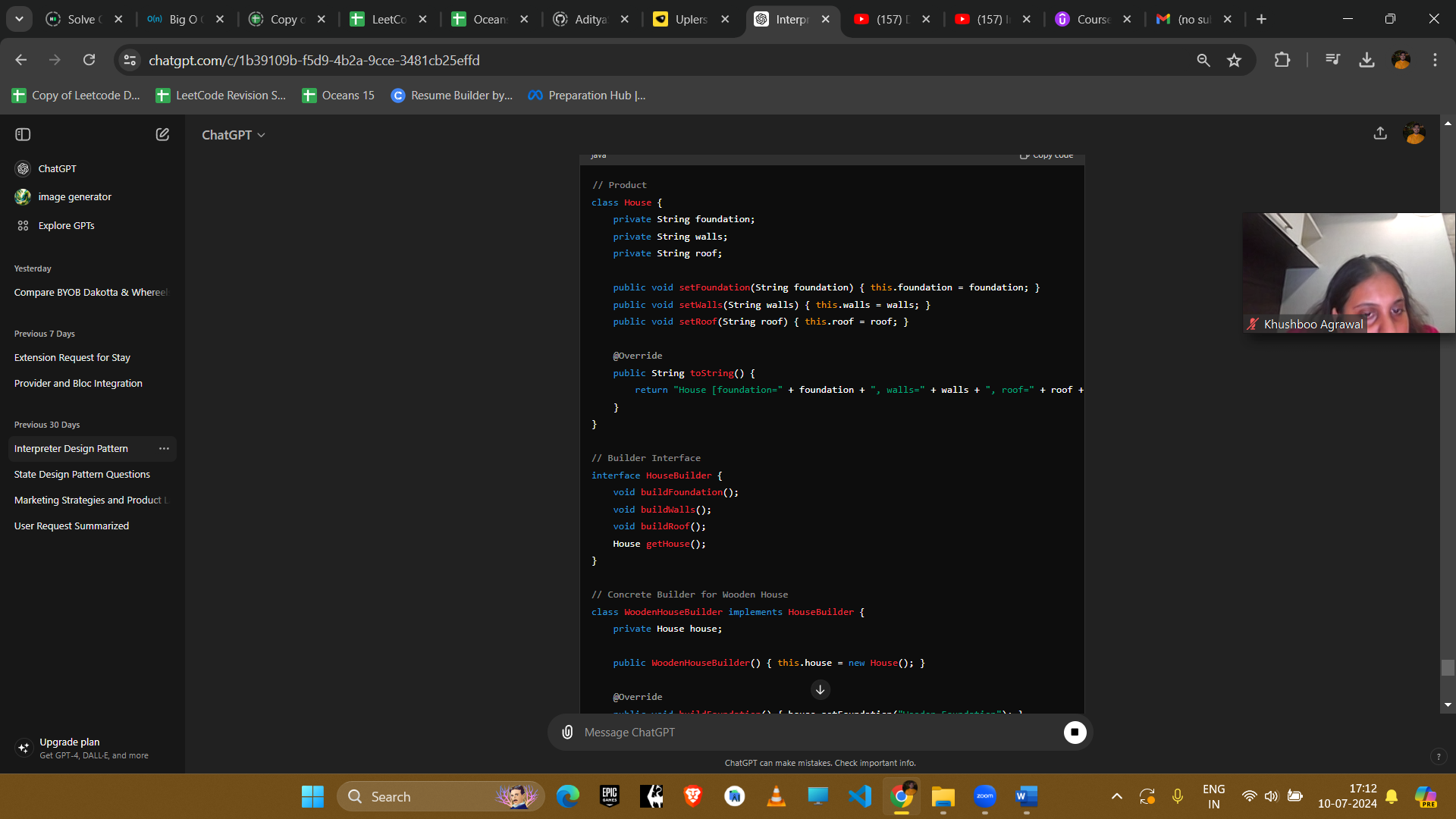
**Definition**

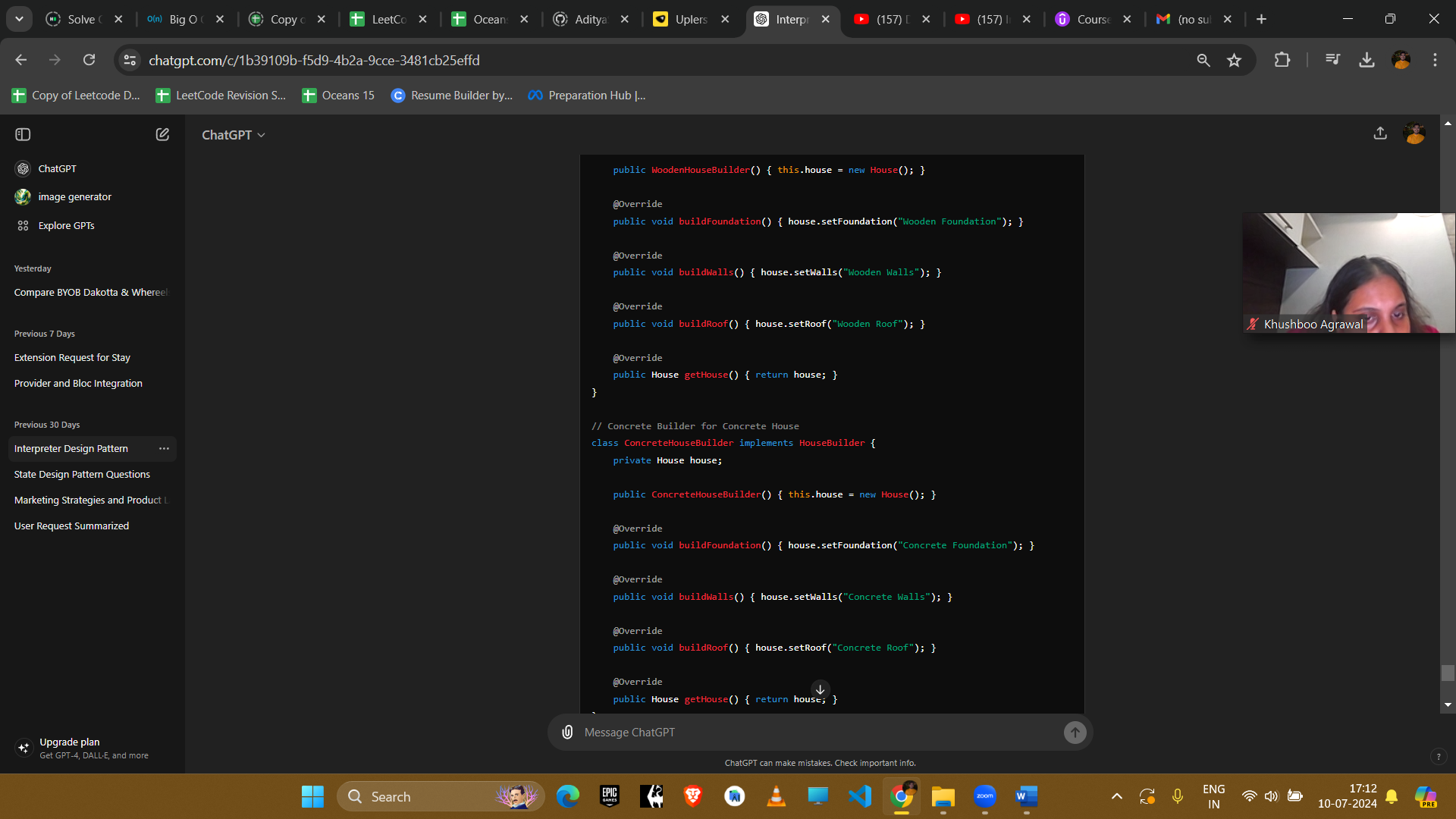
The Builder pattern allows you to create complex objects by specifying their type and content through a step-by-step process. It separates the construction logic from the actual object representation, enabling the construction of different types and forms of objects using the same building process.

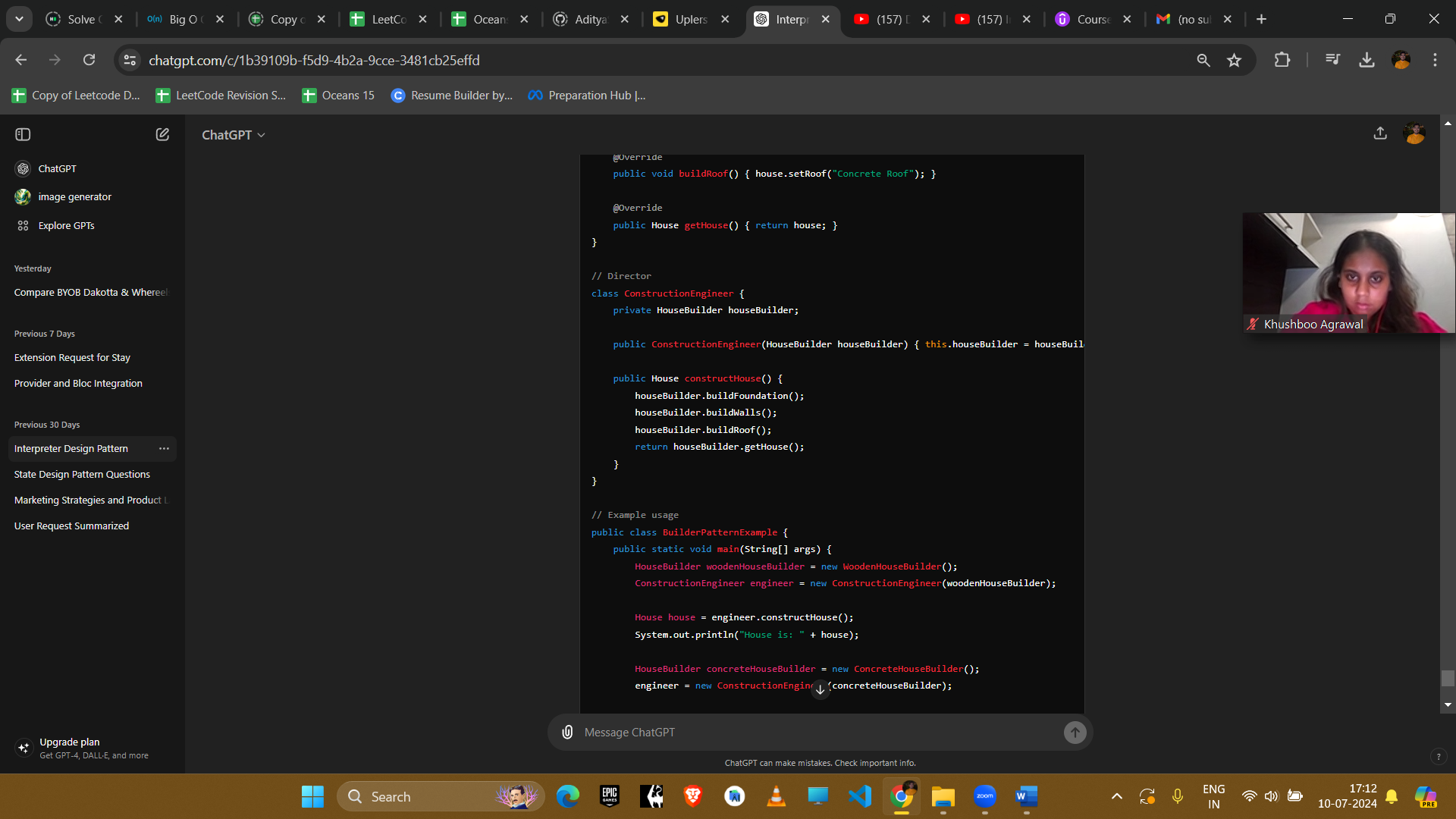
**What problem does it solve?** CHECK VIDEO

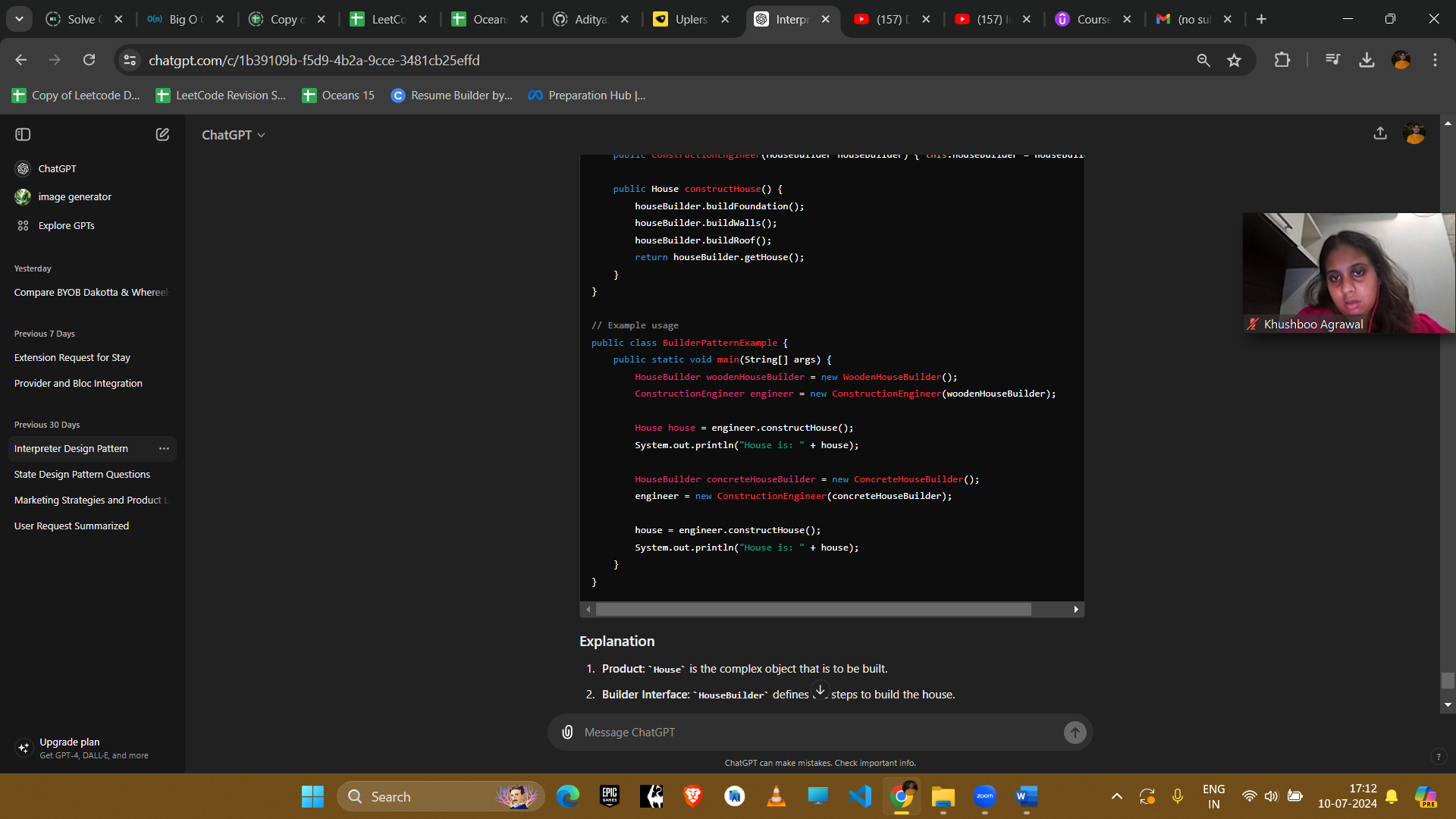
**Example**

Consider a scenario where we need to create different types of houses (e.g., wooden house, concrete house) with various parts like walls, roof, and foundation.









**Explanation**

1. **Product**: House is the complex object that is to be built.
2. **Builder Interface**: HouseBuilder defines the steps to build the house.
3. **Concrete Builders**: WoodenHouseBuilder and ConcreteHouseBuilder implement the HouseBuilder interface to construct different types of houses.
4. **Director**: ConstructionEngineer is responsible for managing the construction process using a HouseBuilder.

**Example Uses in Amazon Interviews**

1. **Complex Object Construction**
   * **Scenario**: Building complex objects like vehicles, meal plans, or computer configurations.
   * **Implementation**: Use the Builder pattern to construct these objects step by step, allowing customization and flexibility.
2. **Immutable Objects**
   * **Scenario**: Creating immutable objects with multiple attributes, like configuration settings or user profiles.
   * **Implementation**: Use the Builder pattern to set the attributes and construct the immutable object.
3. **UI Components**
   * **Scenario**: Constructing complex UI components with various options.
   * **Implementation**: Use the Builder pattern to build UI components step by step, allowing different configurations.
4. **Test Data Creation**
   * **Scenario**: Creating complex test data objects for unit tests.
   * **Implementation**: Use the Builder pattern to construct test data objects with various attributes and states.

**Conclusion**

The Builder pattern is beneficial for constructing complex objects with multiple parts or configurations. It promotes flexibility, readability, and maintainability by separating the construction process from the object representation. This pattern is especially useful when dealing with complex objects that require detailed and flexible construction processes.