**Flyweight Design Pattern**

The Flyweight design pattern is a structural pattern used to minimize memory usage or computational expenses by sharing as much data as possible with similar objects. It is particularly useful when a large number of objects must be created and the majority of their state can be shared.

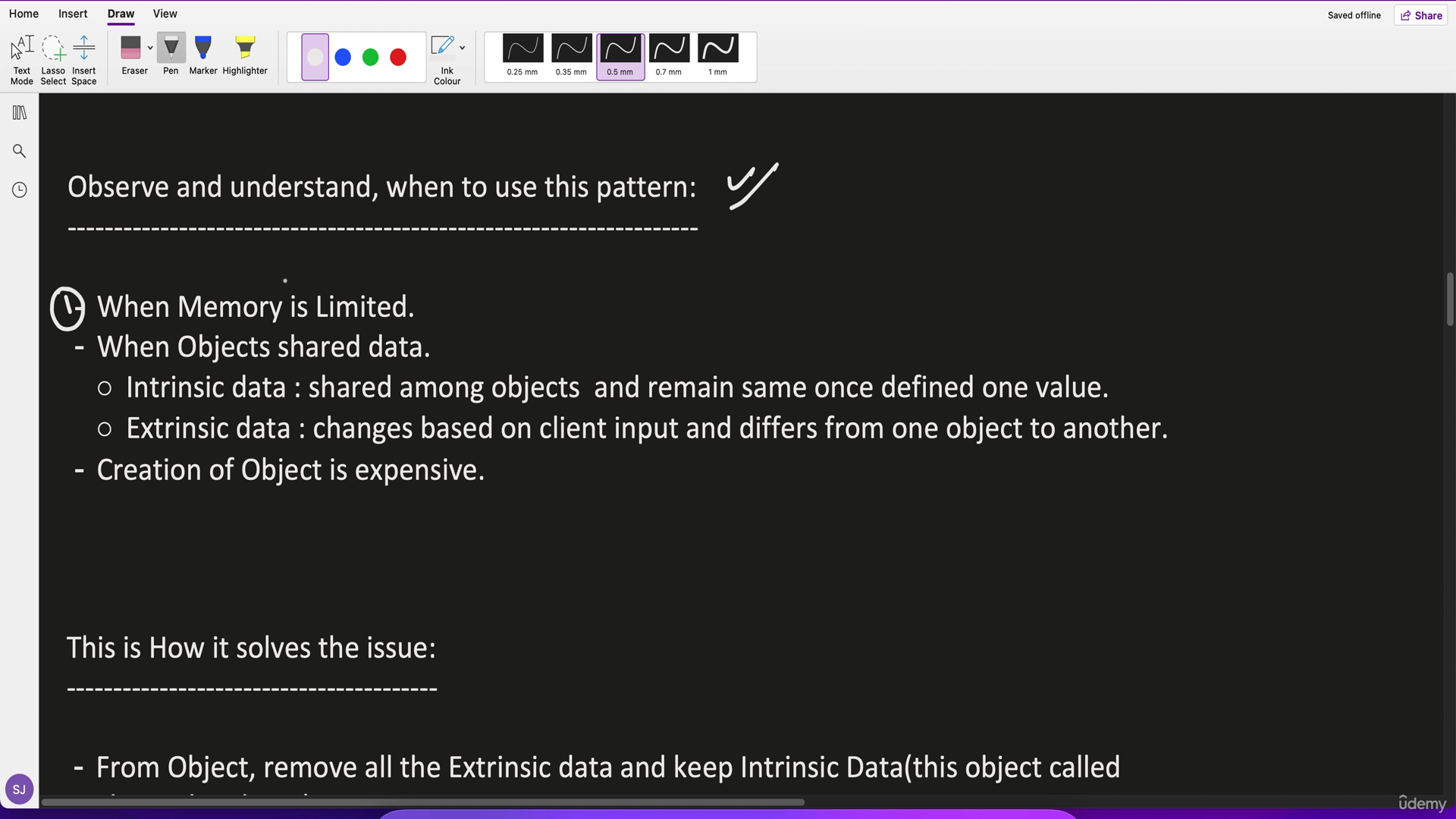
**Definition**

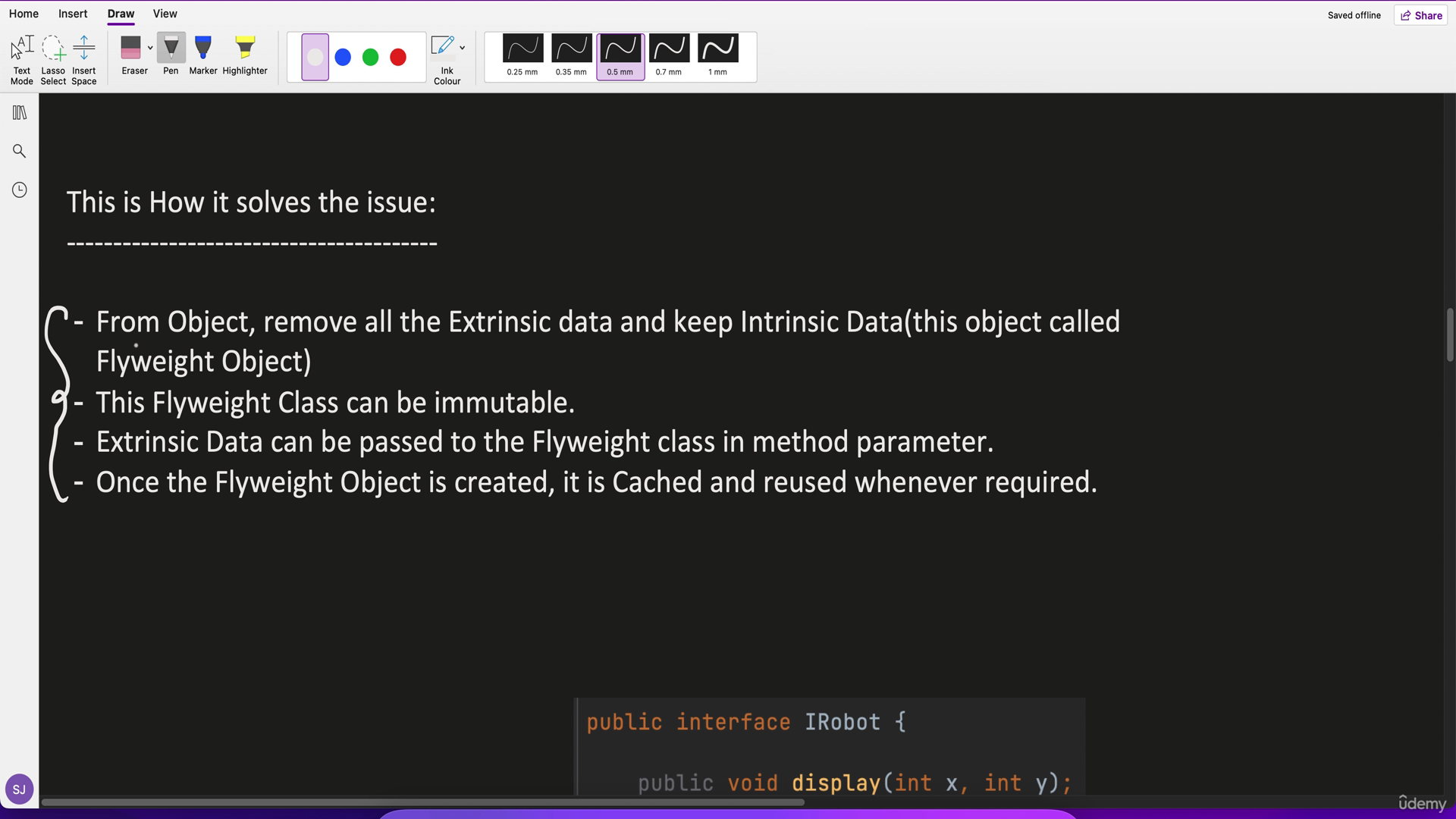
The Flyweight pattern reduces memory usage by sharing common parts of object state among multiple objects, instead of storing them separately. This pattern divides the object state into intrinsic (shared) and extrinsic (unique) states, allowing many fine-grained objects to share common data.

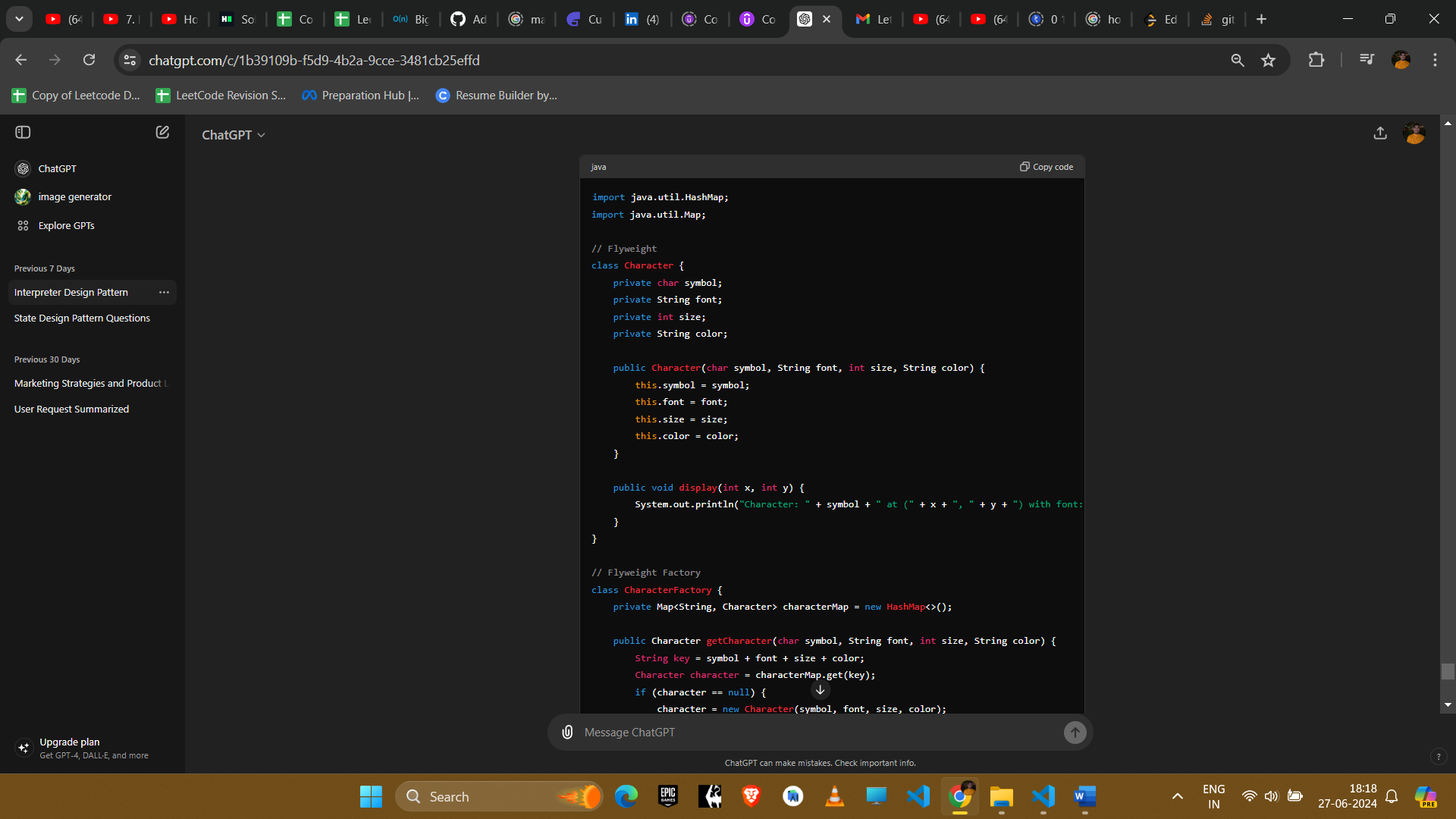
**When to use:** Design word processor, text editor, games

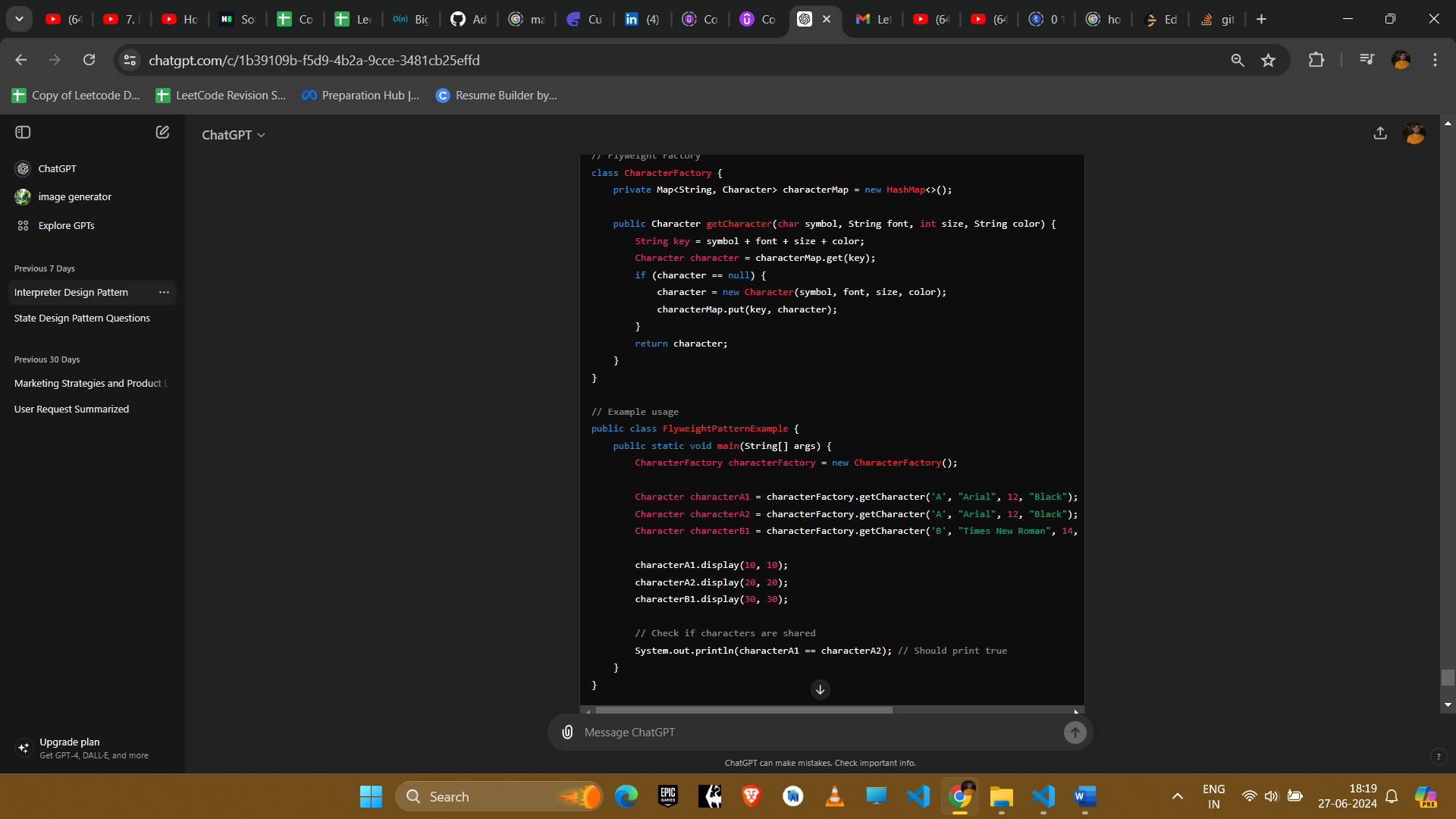
**Example**

Consider a scenario where we need to render a large number of characters in a text editor, where each character has formatting like font, color, and size.









**Explanation**

1. **Flyweight**: The Character class encapsulates the intrinsic state (symbol, font, size, color) that can be shared.
2. **Flyweight Factory**: The CharacterFactory class ensures that characters are shared by storing them in a Map and reusing existing ones when requested.
3. **Client Code**: The FlyweightPatternExample demonstrates how the CharacterFactory creates and shares character objects.

**Example Uses in Amazon Interviews**

1. **Text Rendering**
   * **Scenario**: Rendering large documents with repeated characters.
   * **Implementation**: Use the Flyweight pattern to share character objects with the same formatting, reducing memory usage.
2. **Game Development**
   * **Scenario**: Managing a large number of game objects like trees, buildings, or bullets.
   * **Implementation**: Use the Flyweight pattern to share common data among game objects, optimizing memory usage.
3. **Caching**
   * **Scenario**: Caching objects that are expensive to create and used frequently.
   * **Implementation**: Use the Flyweight pattern to share cached objects, improving performance and reducing memory usage.
4. **Graphical User Interfaces**
   * **Scenario**: Creating multiple instances of UI components with similar properties.
   * **Implementation**: Use the Flyweight pattern to share common properties among UI components, reducing memory footprint.

**Conclusion**

The Flyweight pattern is an effective way to manage memory usage and improve performance when dealing with a large number of similar objects. By sharing common data, it allows the creation of many fine-grained objects without incurring the cost of storing redundant data. This pattern is particularly useful in scenarios where many objects need to be created and their state can be partially shared.