Gilded Rose Refactoring – UML Diagram and Design Explanation

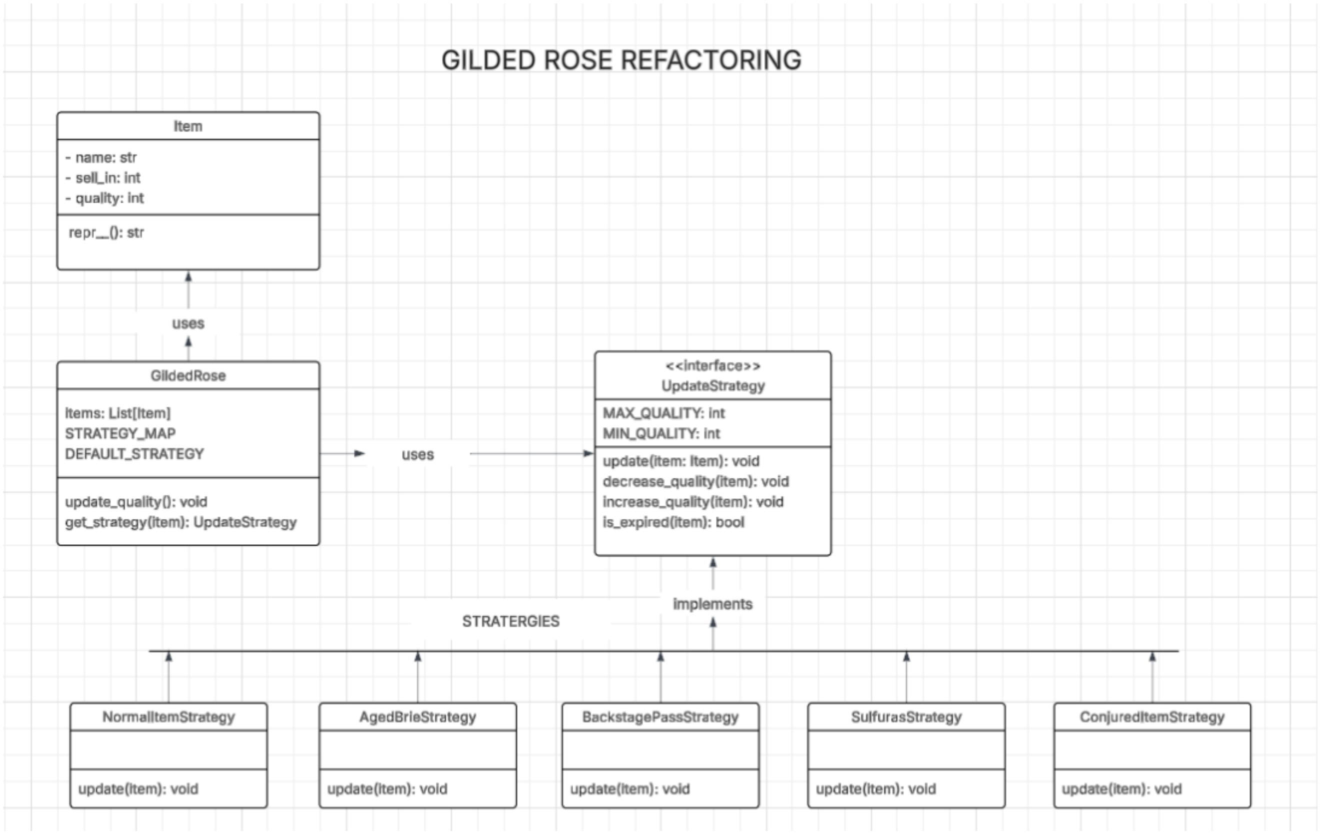
Name: Pooja Malakappa Nuchchi

Issues with Original Code

- Multiple nested if/else statements: The code goes like 4-5 levels deep with conditions inside conditions. Makes hard to follow and requires multiple rounds of reading.
- Everything is crammed into one method: This one function is trying to handle the logic for all Strategies (e.g.: Aged Brie) at once.
- Adding new items is difficult: If I wanted to add a new item type like "Conjured," I have to go into this already messy method and add even more if/else statements. Risk of breaking something that's already working.

My Solution: Strategy Pattern

UML Diagram



Why Strategy Pattern?

Instead of one giant method with multiple if/else checks, each item type gets its own class. Each strategy class is short, easy to read, and can be tested independently. Adding new items requires minimal changes to existing code. Each item type's logic is in its own file.

Explanation

- **UpdateStrategy (interface):** Defines methods like increasing or decreasing quality.
- **Strategy classes:** NormalItemStrategy, AgedBrieStrategy, BackstagePassStrategy, SulfurasStrategy, ConjuredItemStrategy - each handles its own item type.
- **Item class**: Stays the same.
- **GildedRose class**: It uses UpdateStrategy and Item. Uses the appropriate strategy to update each item.

Potential Drawbacks of Strategy Pattern

- Need to look at multiple files to understand the whole system.
- If there are fixed number of Items, its not a good idea of using a Strategy Pattern. It is best suited if new item types need to be added later.