Class Notes: Pizza Shop Software Implementation

In this class, we delved into several important software engineering concepts, particularly focusing on design patterns like the Decorator Design Pattern, and the concepts of association, aggregation, and composition within object-oriented programming (OOP).

Overview

The main focus of the class was implementing a pizza shop system using the Decorator Design Pattern. This pattern allows for adding functionality to objects dynamically, which is essential for scenarios like a pizza shop where customers can add various toppings to their pizzas.

Decorator Design Pattern

1. **Basic Concept:** The decorator pattern allows behavior to be added to individual objects, either statically or dynamically, without affecting the behavior of other objects from the same class.
2. **Application in Pizza Shop:**
   - **Base Pizza Class:** We begin with a `Pizza` abstract class that has a `getPrice` method.
   - **Concrete Implementations:** A `BasePizza` class inherits from `Pizza` and implements the `getPrice` method, returning a base price.
   - **Add-ons as Decorators:**
     - Classes like `Cheese` and `Mushroom` are created, each implementing the `getPrice` method that adds their own price on top of the pizza's price.
     - These add-ons are decorators since they enhance the functionality of the base pizza by incorporating additional costs dynamically.
3. **Implementation Details:**

- Each decorator has an association with a pizza, and it requires a pizza instance to calculate its full price【4:7†source】.
- This setup allows the system to be adaptable and maintainable, so if the price of a component changes, only that specific class needs modification, adhering to the Open/Closed Principle (OCP) 【4:19†source】.

Object-Oriented Concepts

Association, Aggregation, and Composition

1. **Association**:
   - It is a relationship where all objects have their lifecycle, and there is no owner. In the context of our pizza system, the `Cheese` and `Mushroom` classes are associated with `Pizza`.
   - Example: A `Pizza` class having instances of various `Topping` or `Addon` classes.
2. **Aggregation**:
   - It is a specialized form of association with a loose relationship and is often described as a "has-a" relationship. For instance, `Pizza` "has" toppings, but these toppings can exist independently.
   - Example: An aggregation can be having a `Basket` class containing multiple `Fruit` objects where the fruits can exist independently of the basket【4:17†source】.
3. **Composition**:
   - It is a strong form of association with strict dependency (if the parent object is deleted, so is the child), seen as part-whole relationship. In the pizza example, the pizza cannot exist without its base component.
   - The `Cheese` and `Mushroom` decorators show composition since they need a `Pizza` object to function, implying strong lifecycle dependency 【4:17†source】.

UML Diagrams

- **Class Diagrams**:

`Mushroom`.

- Attributes of classes, visibility (public/private), and methods (like `getPrice`) are detailed within these diagrams【4:16†source】.

Conclusion

The workshop provided an insightful application of theoretical concepts into practical design patterns that solve real-world problems in software development by demonstrating a solution for managing and extending features without altering existing code structures significantly. This enhances the flexibility, scalability, and maintainability of the system.