Revision Notes: Strategy and Observer Design Patterns

Introduction

This class covered two important design patterns used extensively in software engineering: the Strategy Pattern and the Observer Pattern. Both patterns are behavioral design patterns, focusing on how classes and objects interact and communicate with each other.

Agenda

The agenda for the class included a deep dive into:

- Strategy Design Pattern
- Observer Design Pattern

Strategy Design Pattern

Concept

The Strategy Pattern involves the selection of an algorithm at runtime. It defines a family of algorithms, encapsulates each one, and makes them interchangeable. It allows the algorithm to vary independently from the clients that use it.

Use Case Examples

1. **Sorting Algorithms**: For performing tasks like sorting, where multiple algorithms (like quicksort, mergesort) can be applied for the same task, the Strategy Pattern is highly useful .
2. **Transportation in Maps**: In applications like Google Maps, different strategies (like car, bike, walking) determine the route, depending on user preferences .

Implementation

- A context class delegates the implementation of some behavior to different strategy classes.
- For example, in a payment processing system, a payment strategy might allow the user to use different mechanisms (credit card, debit card) based on the specific strategy chosen.

Analogies

- **Google Maps**: Similar to choosing different transport modes (car, walking, biking) based on user preference.

Advantages

- Provides an alternative to subclassing.
- Eliminates conditional statements.
- Increases extendability.

Observer Design Pattern

Concept

The Observer Pattern is a way to achieve a publish-subscribe relationship between objects. A subject maintains a list of observers, which must be notified of any state changes.

Use Case Example

- **Weather Monitoring System**: A weather station subject notifies various display components (observers) about changes in weather data such as temperature or humidity .

Implementation

- **Observer**: Implements an update interface to receive state changes from the subject.

Analogies

- **Weather Station**: Instead of each display manually fetching weather data, they get notified when there's a change in the weather data .
- **HR Application Process**: Instead of the applicant polling the HR system, they are notified when there's an update on their application .

Technical Considerations

- Observers need to have an update method. Using abstract classes ensures that all observers implement necessary behavior .
- To dynamically add or remove observers without changing the existing code, methods `registerObserver` and `removeObserver` are used .

Advantages

- Promotes a loose coupling between the subject and the observers .
- Flexible and dynamic interaction model.

Conclusion

The Strategy and Observer design patterns provide flexible solutions to common problems in software engineering allowing behavior to be selected and varied dynamically, and facilitating dynamic interaction between objects, respectively.

This class provided examples, use cases, and analogies to clarify these concepts for practical implementation. As design patterns are foundational to building scalable and maintainable software, understanding and applying them effectively is crucial for any software engineer.