옵저버 패턴(Observer Pattern)

옵저버 패턴의 개념과 역할

옵저버 패턴(Observer Pattern)은 객체의 상태 변화를 관찰하는 디자인 패턴 중 하나로, 한 객체의 상태가 바뀌면 그 객체에 의존하는 다른 객체들에게 연락이 가고 자동으로 내용이 갱신되도록 하는 패턴입니다.

옵저버 패턴에서는 관찰 대상이 되는 객체(Subject)와 관찰자(Observer)가 있습니다. Subject 객체에는 관찰자들을 등록하고, 상태가 변경되면 등록된 모든 Observer 객체에게 상태 변경을 통지합니다. Observer 객체는 이 상태 변화를 처리하고 필요한 동작을 수행합니다. 이런 방식으로 Subject 객체와 Observer 객체들은 느슨하게 결합되어 있어 유연하게 상호작용할 수 있습니다.

예를 들어, 블로그 서비스에서는 게시글을 작성하는 사용자가 있고, 이 사용자가 작성한 게시글의 내용이 바뀌면 이를 구독한 다른 사용자들에게 알림을 보내주는 기능을 구현할 수 있습니다. 여기서 게시글 작성자가 Subject 객체가 되고, 알림을 받을 사용자들이 Observer 객체가 됩니다. 게시글 작성자는 게시글을 작성할 때마다 이를 구독한 Observer 객체들에게 변경을 통지하고, Observer 객체들은 변경된 내용을 확인하고 알림을 보내는 등의 작업을 수행합니다. 이를 옵저버 패턴을 이용하여 구현할 수 있습니다.

옵저버 패턴의 구성 요소

Subject(주체)

옵저버 패턴에서 주체는 상태를 가지고 있으며, 이 상태에 따라 Observer들에게 변경사항을 알리는 역할을 합니다. Subject 인터페이스를 구현하며, 등록된 옵저버들을 추가하고 삭제할 수 있는 메서드를 제공합니다.

Observer(관찰자)

옵저버 패턴에서는 주체의 상태 변경을 알고 싶은 객체들을 Observer라고 합니다. 이 객체들은 주체의 상태를 관찰하고 있다가 상태가 변경되면 주체로부터 알림을 받아 처리합니다. Observer 인터페이스를 구현하며, 주체로부터 상태 변경 알림을 받는 update() 메서드를 제공합니다.

옵저버 패턴의 동작 방식

옵저버 패턴은 Subject(주체)와 Observer(관찰자)로 구성되어 있습니다. Subject는 데이터의 변화를 감지하고, 그 변화를 Observer에게 알려주는 역할을 합니다. Observer는 Subject에서 발생한 변화에 대해 처리하는 역할을 합니다.

Subject는 상태를 저장하고 있는 데이터를 갖고 있으며, 상태가 변경되면 등록된 Observer에게 알립니다. 이때 Subject는 Observer에게 직접 알리는 것이 아니라 Observer가 구현한 특정 메서드를 호출해 알립니다.

Observer는 Subject에서 등록된 상태 변화에 대한 알림을 받으면 처리를 수행합니다. Observer는 여러 개일 수 있으며, Subject에서 등록 및 삭제를 할 수 있습니다.

이러한 방식으로 Subject와 Observer가 연결되어 상호작용하면서 데이터의 변화를 감지하고 처리할 수 있습니다.

옵저버 패턴의 장단점

장점

- Subject와 Observer가 각각 독립적으로 확장될 수 있어 유연성이 높습니다.
- 한 객체의 상태 변화가 다른 객체의 동작을 바로 발생시킬 수 있어, 객체간의 결합도를 낮출 수 있습니다.
- 객체간의 연결 고리를 끊어주어 객체의 생명 주기 관리를 편리하게 할 수 있습니다.

단점

- Observer가 많을수록 각 Observer마다 연산이 추가되기 때문에, 프로그램의 복잡도와 속도를 떨어뜨릴 수 있습니다.
- Subject와 Observer간의 인터페이스 구현이 복잡할 수 있습니다.

옵저버 패턴과 MVC 패턴의 관계

MVC 패턴은 소프트웨어 디자인 패턴 중 하나로, 사용자 인터페이스와 애플리케이션 로직을 분리하는 패턴입니다. 모델(Model)은 애플리케이션의 데이터와 상호작용을 담당하고, 뷰(View)는 모델의 상태를 시각적인 형태로 보여주며, 컨트롤러(Controller)는 사용자의 입력에 응답하여 모델과 뷰를 업데이트합니다.

이때 모델은 데이터에 대한 변화가 발생하면 뷰나 컨트롤러에게 알려주어야 합니다. 이때 옵저버 패턴을 사용하면 모델과 뷰, 컨트롤러 간의 의존성을 낮추면서, 모델의 상태 변화를 뷰나 컨트롤러에게 알리는 것이 가능합니다. 이는 옵저버 패턴에서 Subject(주체)가 모델에 해당하고, Observer(관찰자)가 뷰와 컨트롤러에 해당합니다. 모델은 상태가 변할 때마다 등록된 옵저버에게 알리고, 옵저버는 이를 받아서 처리합니다.

따라서, MVC 패턴에서 옵저버 패턴을 적용하면 모델과 뷰, 컨트롤러 간의 결합도를 낮추고, 유연성과 확장성을 높일 수 있습니다.

옵저버 패턴의 구현 방법과 주의사항

1. 주제(Subject) 인터페이스 정의하기: 주제(Subject)는 옵저버(Observer)들의 등록과 삭제, 그리고 상태 변화를 알리는 메서드를 정의합니다. 일반적으로 `registerObserver()`, `removeObserver()`, `notifyObservers()` 등의 메서드가 포함됩니다.
2. 옵저버(Observer) 인터페이스 정의하기: 옵저버는 주제(Subject)로부터 상태 변화를 전달받는 `update()` 메서드를 정의합니다.
3. 주제(Subject) 클래스 구현하기: 주제 클래스는 주제(Subject) 인터페이스를 구현하며, 등록된 옵저버(Observer)들에게 상태 변화를 알리는 `notifyObservers()` 메서드를 구현합니다.
4. 옵저버(Observer) 클래스 구현하기: 옵저버 클래스는 옵저버(Observer) 인터페이스를 구현하며, `update()` 메서드를 구현합니다.
5. 주제(Subject) 클래스에서 옵저버(Observer)들의 상태를 추적하고 변경사항이 있을 때 `notifyObservers()` 메서드를 호출합니다.
6. 옵저버(Observer) 클래스에서는 `update()` 메서드를 통해 주제(Subject) 클래스에서 전달된 변경사항을 처리합니다.

주의사항으로는, 주제(Subject) 클래스와 옵저버(Observer) 클래스 사이에 서로 강한 결합이 형성되어선 안 됩니다. 이를 위해 인터페이스를 사용하여 느슨한 결합을 유지하는 것이 좋습니다. 또한, 주제(Subject) 클래스에서 옵저버(Observer) 클래스를 직접 생성하면 의존성이 생기므로, 이를 해결하기 위해 주제(Subject) 클래스에서는 옵저버(Observer) 인터페이스만 참조하도록 해야 합니다.