데이터 독립

이름 : 조대훈

목차

0. 개요

0.1 추구하는 프로그래밍 방향성

- 게임 클라이언트는 작성해야되는 코드의 양이 다른 개발자에 비해서 많습니다.
- 보다 효율적인 프로그래밍을 통해서 개발 속도를 높이고자 하였습니다.
- 저는 보다 효율적인 프로그래밍의 조건으로 '가시성'에 많은 신경을 썼습니다.
- '가시성'이 좋은 코드를 만들기위해서 '주석'과 기능 개발 전후로 'UML 작성'에 항상 일정 시간을 사용하였습니다.

0.2 문제의 발견

- 수 많은 UML을 그리다 보니, 매번 그리는 UML들에 문제가 많다는게 느껴졌습니다.
- 바로 **클래스 간의 참조 선이 너무 많아서 UML이 복잡해 진다는 것이였습니다.**
- UML을 그리기 전에는 잘 몰랐지만, UML로 그려진 코드는 스파게티 코드 그 자체였습니다.
- 저는 클래스 간의 참조가 왜 이렇게 많아졌는지 고민하였고, 원인을 발견하게 되었습니다.

1. 스파게티 코드가 되는 원인과 해결 방안

1.1 원인

① 데이터 참조

- 특정 클래스의 멤버 변수의 사용이 필요해 참조하는 경우가 있습니다.

② 기능 참조

- 특정 클래스에 구현된 기능을 사용하기 위해서 해당 객체를 참조하는 경우가 있습니다.

1.2 해결 방안

① 데이터 참조

- 기능을 수행하는데 필요한 **데이터를 독립적으로 존재**하게 함으로써 해결할 수 있습니다.

② 기능 참조

- '**독립된 데이터에 Observer 패턴을 사용**'함으로써 해결할 수 있습니다.

2. 독립된 데이터

- **'통일성 있는 유일성과 접근성'**을 만족하는 데이터를 **'독립된 데이터'**라 표현하였습니다.

2.1 유일성

- 특정 기능 구현에 있어서, 기존 데이터들을 가공하여 사용하는 경우가 대부분입니다.
- 만약 이때, 같은 성격의 새로운 데이터를 만들어 사용하면 '유일성'이 잃어버립니다.

2.1.1 유일성 손실

- 개발자들은 가끔 중복된 데이터 클래스를 만듭니다.
- 데이터 클래스가 너무 많고, 분산되어 존재하고 있어서 중복된지 모르는 것입니다.
- 이러한 '유일성 손실'은 정상적이지 않는 기능의 출력이라는 오류를 만나게 됩니다.

2.1.2 기존의 잘못된 접근

- 이러한 오류를 방지하기 위해, 개발자는 해당 데이터가 존재하는 객체를 참조하게 됩니다.
- 이러한 객체 참조 관계는 점차 스파게티 코드를 만들게 됩니다.

2.1.3 새로운 해결 방안

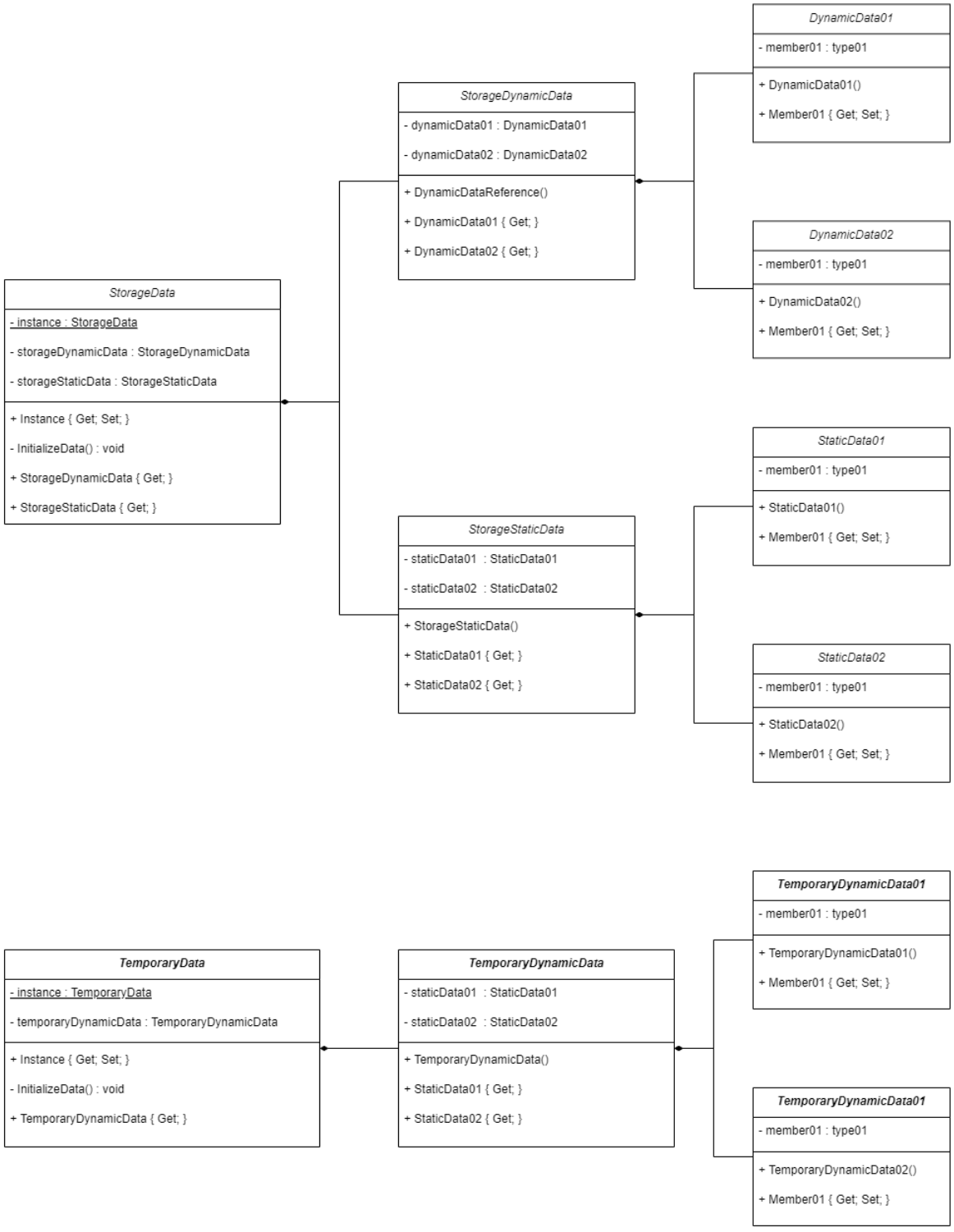
- 여기서 데이터를 데이터 클래스 집단에 몰아 넣어 관리하자는 생각을 하게 되었습니다.
- 하나의 집단이 데이터를 관리하니, 중복된 데이터 클래스 확인이 용이해 졌습니다.
- 이전에 작성한 '데이터 분류'법을 통해서 데이터 집단을 나누면, 가시성이 더욱 좋아집니다.

2.2 접근성

- 데이터를 하나의 집단에 몰아 넣고 관리하자,
  해당 집단에 대한 접근법에 대한 고민이 자연스럽게 하게 되었습니다.

- '유일성'과 '접근성'에 용이한 디자인 패턴인 'Singleton 패턴'을 사용하기로 하였습니다.-

2.3 'Singleton 패턴'을 사용한 데이터 독립



3. 독립된 데이터에 Observer 패턴 사용

- **'독립된 데이터 객체에 Observer 패턴을 사용'**함으로써 '기능 참조'를 위한 객체 참조를 최소화 할 수 있습니다.

3.1 Observer 패턴이란?

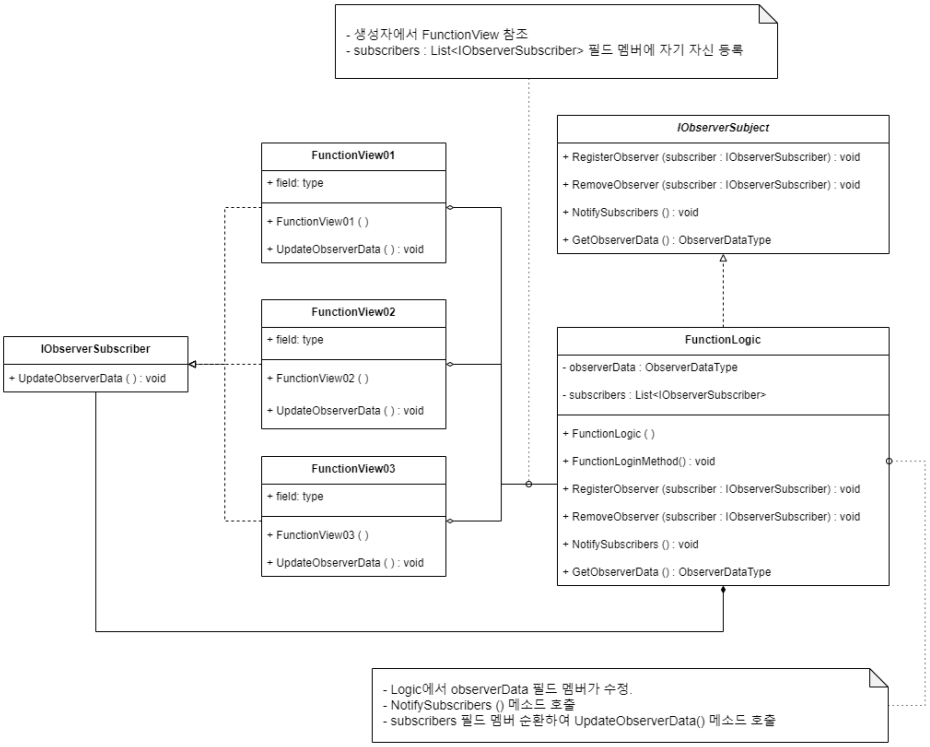
- 하나의 객체(발행자, Subject)의 데이터 변경이 다른 객체(구독자, Subscriber)에게 통지되고, 이로 인해 관련된 동작이 수행되도록 하는 디자인 패턴입니다.

- Subject가 관리하는 데이터를 이용하기 위해 Subscriber 클래스 측에서 Subject 클래스를 짧은 시간(생성자 또는 OnEnable()과 같은 활성화 시점) 참조합니다.

- Subject와 Subscriber는 1대 N의 관계를 갖을 수 있습니다.

- Subject와 Subscriber 객체는 서로 다른 클래스이며, 확장과 수정에 있어 자유롭습니다.

(Subject는 데이터만 잘 갱신하면되고, Subscriber는 데이터만 잘 전달 받으면 됩니다.)



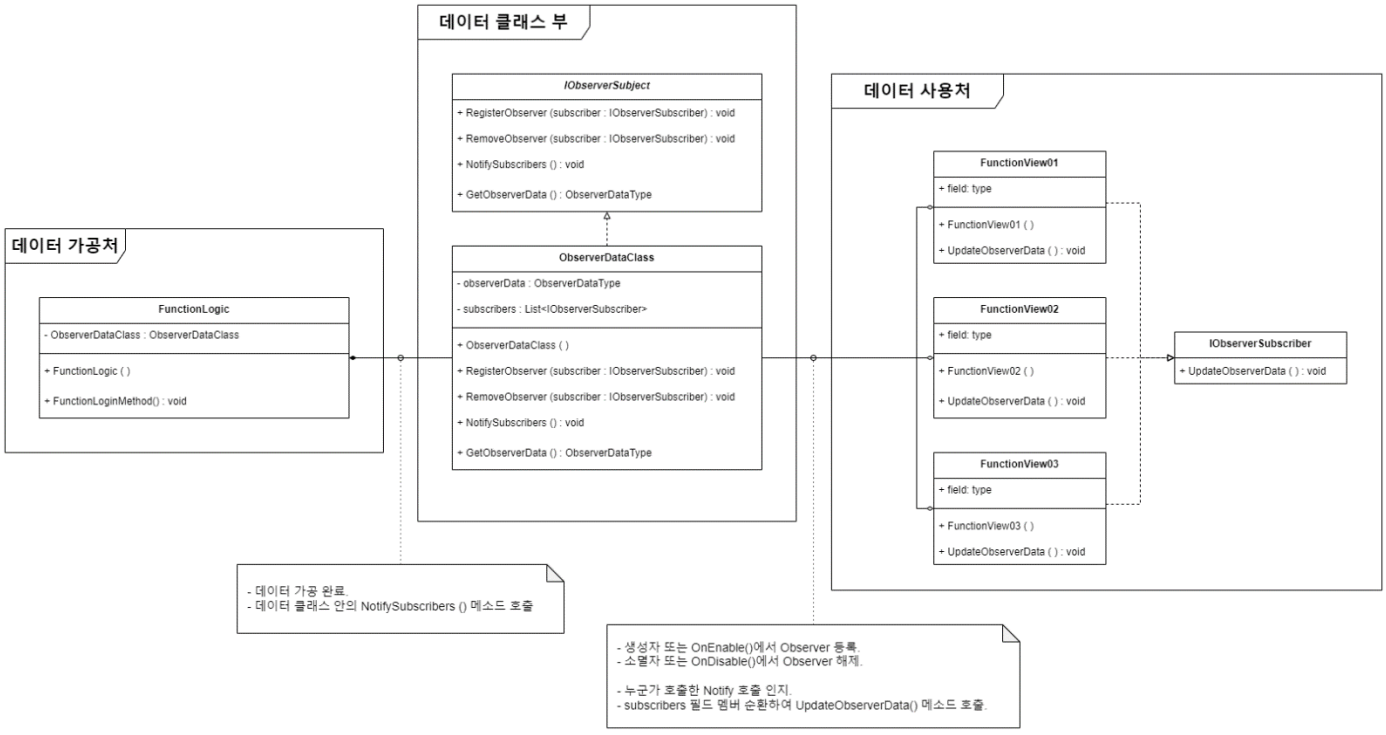
<Observer 패턴 예시>

3.2 Observer 패턴의 잘못된 접근

- Observer 패턴을 알려주는 많은 예시들이 Logic 구현 코드와 Observer 패턴 구현 코드를 같은 클래스 내에 작성하는 경우가 있습니다.
- 하지만 그럴경우, View가 불필요한 Logic 구현부를 참조하는 상황이 발생합니다.
- Observer 패턴의 강점은, 독립된 데이터 객체에서 사용됨으로써 부각됩니다.

3.3 독립된 데이터 객체에 Observer 패턴 사용

- 데이터가 Logic 클래스로부터 독립이 되면 많은 이점이 있습니다.
- Logic 클래스는 자신이 가공하는 데이터가 어디에 사용되는지 아무것도 몰라도 됩니다.
- View 클래스는 자신이 사용되는 데이터가 언제 수정되는지 몰라도 됩니다.
- 즉, Logic과 View 클래스들은 자신의 일에 충실하면 되는 것입니다.



<수정된 Observer 패턴 예시>