231026 컴파운드 패턴

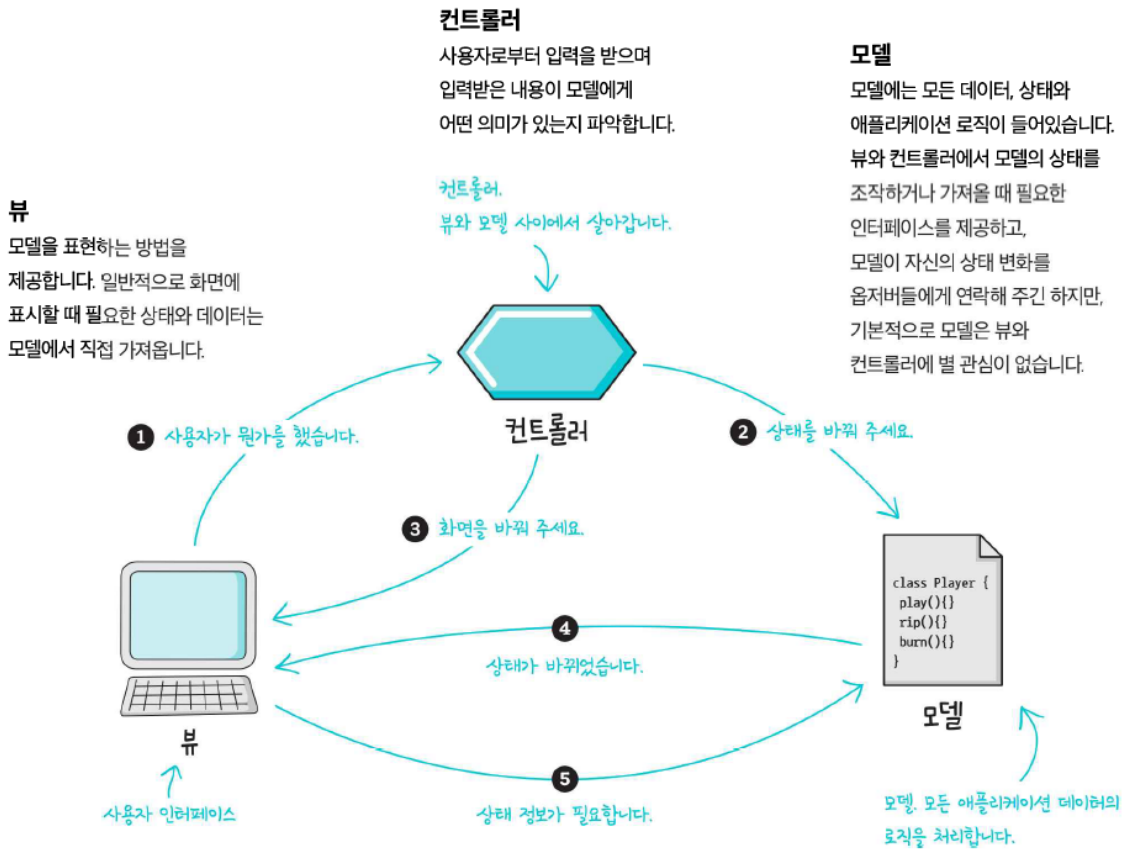
두 개 이상의 패턴을 결합하여 일반적으로 자주 등장하는 문제들에 대한 해법을 제공하는 패턴

대표적 컴파운드 패턴 : MVC 패턴

Model : 어플리케이션의 데이터, 자료

View : 사용자에게 보여지는 UI

Controller : Model과 View 사이를 잇는 징검다리



Model

1. 사용자가 편집하길 원하는 모든 데이터를 가지고 있어야함
2. 뷰나 컨트롤러에 대해서 어떠한 정보도 알지 말아야함
    a. 데이터 변경이 일어났을때 모델에서 화면 UI를 직접 조정해서 수정할 수 있도록 뷰를 참조하는 내부 속성값을 가지면 안됨
3. 변경이 일어나면 변경통지에 대한 처리방법을 구현해야함
    a. 옵저버 패턴 사용 가능

View

1. 모델이 가지고 있는 정보를 따로 저장해서는 안됨
2. 모델이나 컨트롤러와 같이 다른 구성요소들을 몰라야함
3. 변경이 일어나면 변경통지에 대한 처리방법을 구현해야함

Controller

1. 모델이나 뷰에대해서 알고 있어야 함
    a. 모델과 뷰는 서로 몰라야함
2. 모델이나 뷰의 변경을 모니터링 해야함

왜 사용할까?

- 하나의 역할만 진행하여 효율성 높힘 (분업)
- 중복코딩 사라짐
- 각자의 역할에 집중하여 유지보수성, 애플리케이션의 확장성, 유연성 증가

단점

- 복잡함
- 파일 분할이 많아짐
- 일관성을 유지하는것에 대한 비용

    - 예기치못한 상황에 대처하다 시간을 뺏기는 경우가 있음

함께 생각해보기

- MVP 패턴
    - presenter는 모델과 뷰의 간접적인 참조마저 허용하지않음
- MVVM 패턴
    - 데이터 바인딩을 통해 view와 view model이 동기화

보너스. 유니티에서 활용하기

https://programmingdev.com/unity-디자인-패턴-mvc-mvp-mvvm-비교-및-예시-코드/

https://everyday-devup.tistory.com/66