[3주 차]

상속과 프로그래밍 패턴

2024.04.10

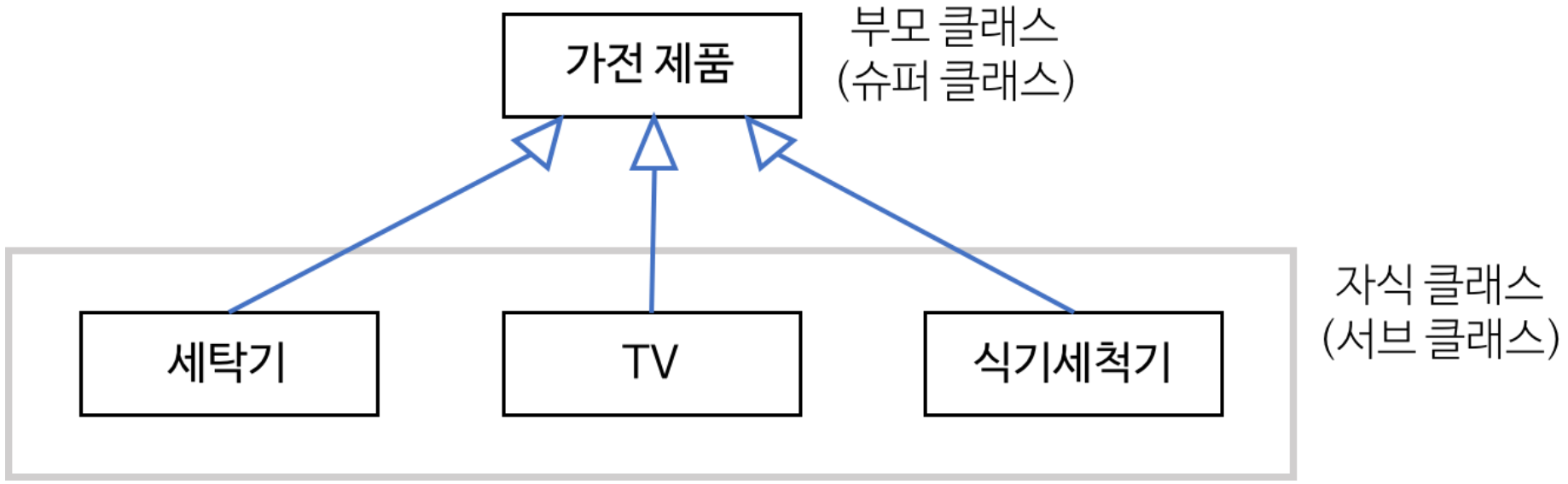
목차

1. 상속

1.1. 부모 클래스와 자식 클래스

객체 지향 → 세상의 모든 물체들을 추상화시켜 상태와 행위를 가진 객체를 만드는 것, 그리고 그 객체들끼리 상호 작용을 하는 것

상속 → 객체들의 공통된 부분을 찾아 부모 클래스와 자식 클래스로 나누고, 그 특성을 물려받을 수 있도록 한 것



1. 상속

1.2. 오버로딩과 오버라이딩

오버로딩 → 매개변수만 다른 같은 이름의 메서드를 여러 개 선언하는 것

오버라이딩 → 부모 클래스에서 가지고 있는 메서드를 자식 클래스에서 다시 정의하는 것

구분	Overriding	Overloading
접근 제어자	부모 클래스의 메소드의 접근 제어자보다 **더 넓은 범위의 접근 제어자**를 자식 클래스의 메소드에서 설정할 수 있다.	**모든 접근 제어자**를 사용할 수 있다.
리턴형	동일해야 한다.	달라도 된다.
메소드명	동일해야 한다.	동일해야 한다.
매개변수	동일해야 한다.	달라야만 한다.
적용 범위	**상속관계**에서 적용된다.	**같은 클래스** 내에서 적용된다.

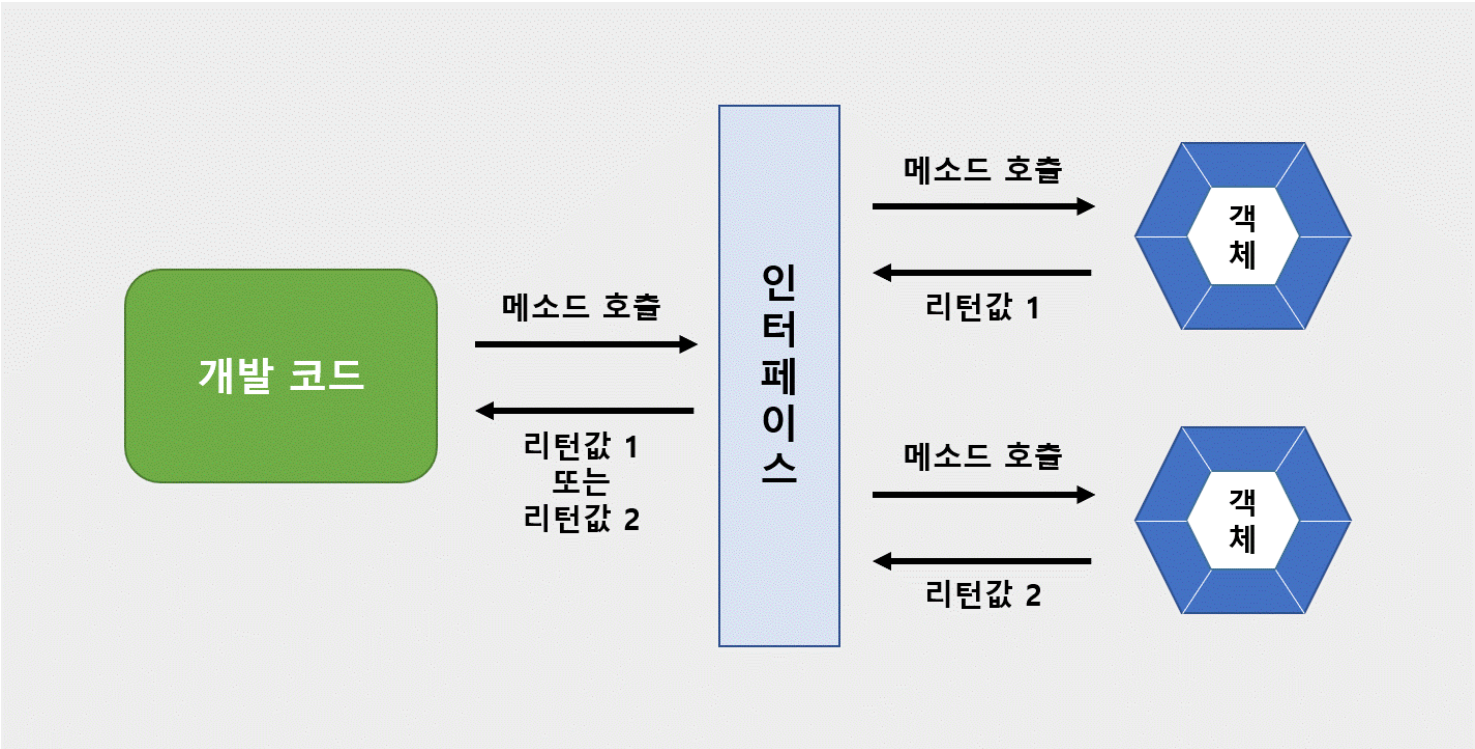
1. 상속

1.3. 인터페이스

→ 추상적인 메서드만 존재하는 추상 클래스

→ 기능을 정의하기만 하고 구현하지는 않음

→ 여러 객체들의 공통된 부분을 추상화해 인터페이스를 통해서 다양한 객체를 쉽게 접근/조작할 수 있도록 함



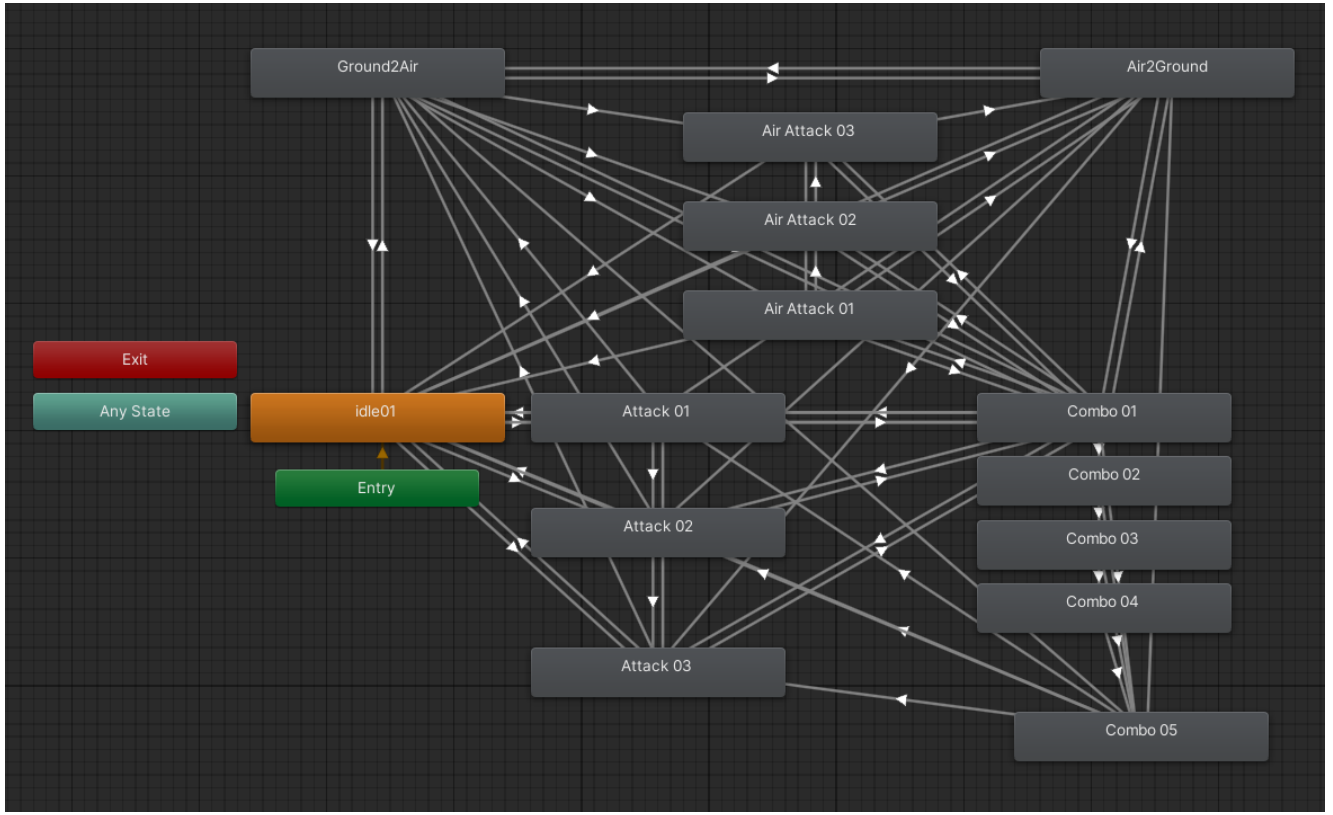
2. 디자인 패턴

2.1. FSM (Finite State Machine)

→ 유한 상태 기계

→ 객체의 다양한 상태를 관리하는 패턴

→ 유니티에서 애니메이션을 관리할 때 자주 사용



2. 디자인 패턴

2.2. Singleton 패턴

→ 하나의 객체가 하나의 인스턴스만을 가져야 할 때 사용

→ 유니티에서는 매니저 클래스에서 많이 사용함

→ (장점) 메모리가 절약된다

→ (장점) 데이터 공유가 쉽다

→ (단점) 객체 지향과는 거리가 멀다

→ (단점) 자식 클래스를 만들 수 없다

→ (단점) 멀티 스레딩 환경에서 취약하다

```
public class GameManager : MonoBehaviour
{
    private static GameManager _instance;
    0 references
    public static GameManager Instance
    {
        get
        {
            return _instance;
        }
    }
}
```

3. 실습

3.1. 상속, 오버로딩, 오버라이딩, 인터페이스

<git project>/sdh-202403/Project/SDH2024/Assets/20240410/01

3. 실습

3.2. 디자인 패턴과 상속의 결합 (MonoSingleton)

<git project>/sdh-202403/Project/SDH2024/Assets/20240410/02

감사합니다