◈ C# 어셈블리 ◈

📄 1. 어셈블리가 뭐죠

어셈블리어는 아니다.
컴파일되서 나온 파일을. C# 에서는 어셈블리(Assembly) 라고 부른다

- **.exe, .dll**
  > exe 는 Main() 메서드를 포함하는(진입점) 형태이다.
  >
  > - 콘솔 실행이 가능하다.
  >
  > dll 은 반대로 진입점이 없는 형태이다.

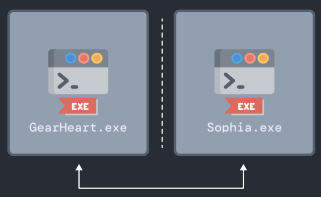
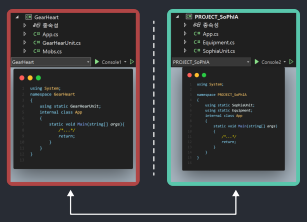
📄 2. ...그래서? 🤔

- ✨ 이거 하나만 이해하자. ✨
- **어셈블리가 같은 프로젝트여야 internal 한정자 접근이 가능하다!!**

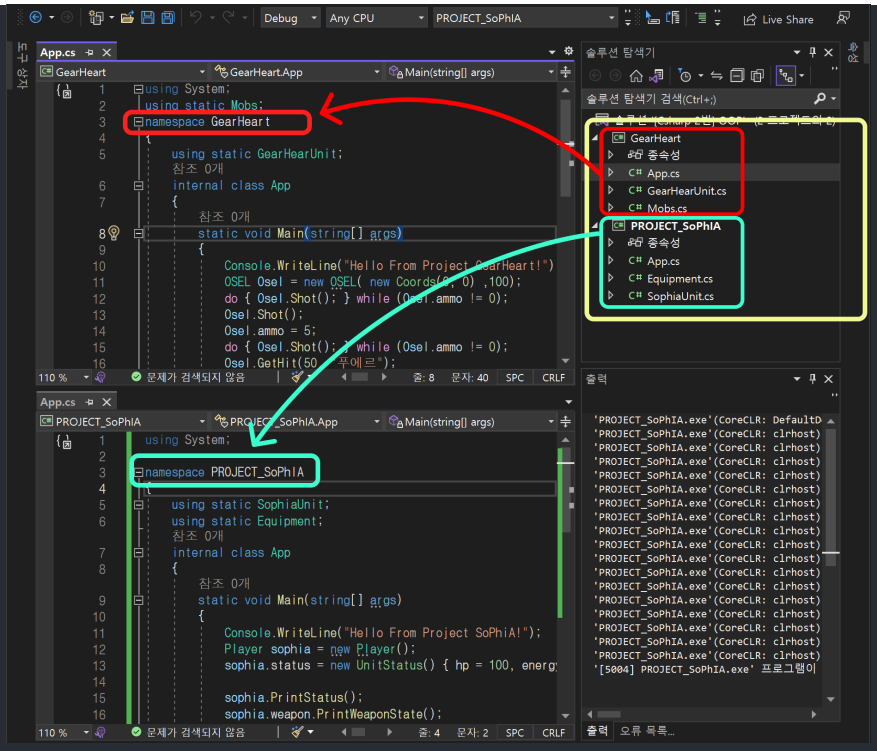
> *...뭐가 대단한데*

3. 어셈블리가 다른 [프로젝트 & 파일]

GearHeart VS SoPhIA

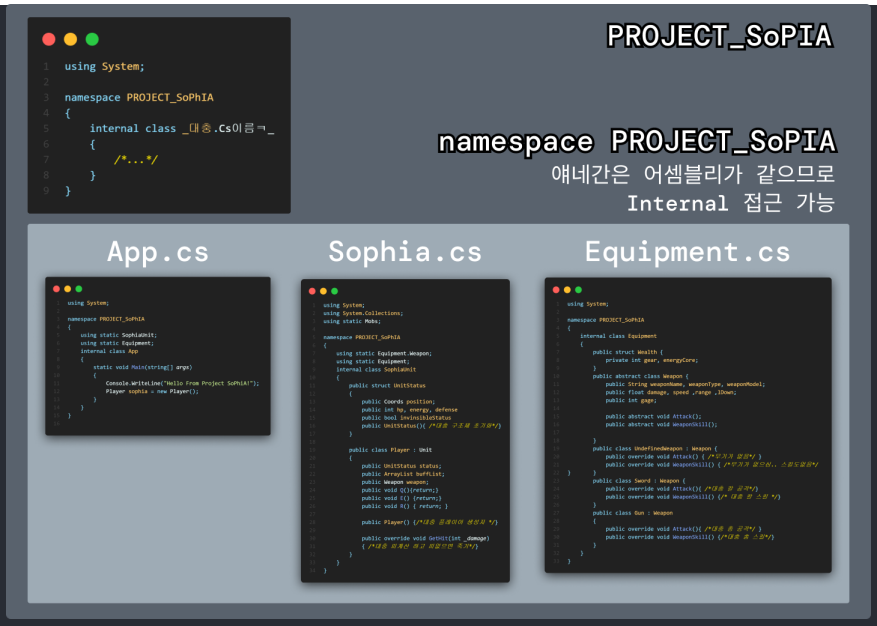
파일관점	코드관점
	
다른 exe라서	*다른 프로젝트라서*

우측 솔루션에 두개의 각각 코드의 Namespace가 다른것을 확인 할 수 있다.



4. 어셈블리가 같은 [프로젝트 & 파일]

Namespace가 같은것이 같은 어셈블리이다.



5. 결론

- 1. 프로젝트(exe) 가 각각 다른 놈은 Internal 접근이 불가능.
- 2. namespace가 다르면 Internal 접근이 불가능

6. namespace가 없을수도 있다.

프로젝트에 구애받지 않고 어디서든 접근 가능하다는뜻이다.

1. GearHeart에 있는 *Unit* 클래스가 너~무 잘 만들어 졌다. 😇

2. 그래서 다음 프로젝트(SoPhIA)에 Unit을 사용해보고 싶다 🧐

3. 아싸! *Player*를 *Unit* 클래스로 만들어야징 ~ 👍

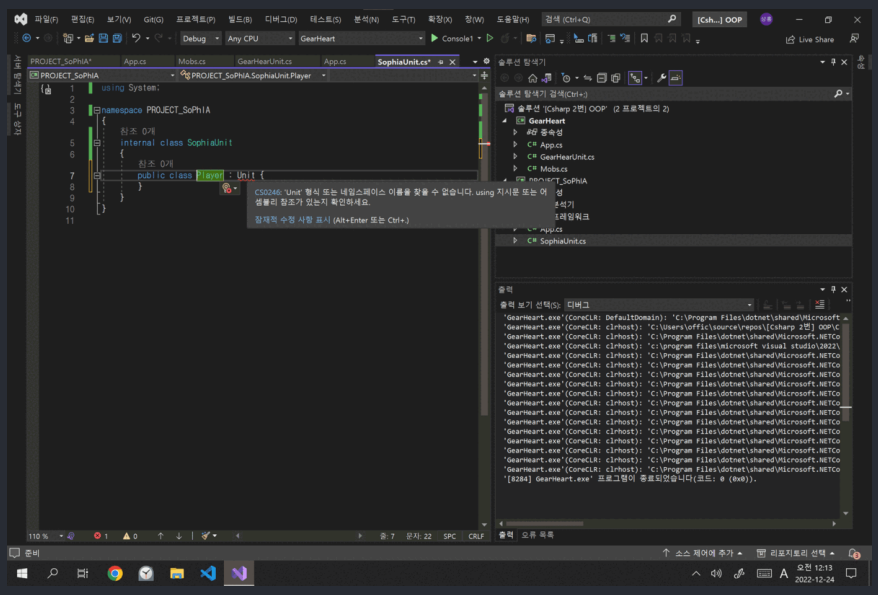
4. GearHeart에서 SoPhIA로 복사 붙여넣기 하면 되는거 아닌가? ㅋㅋ 🤡

5. 조용히 하세요!! 🤬

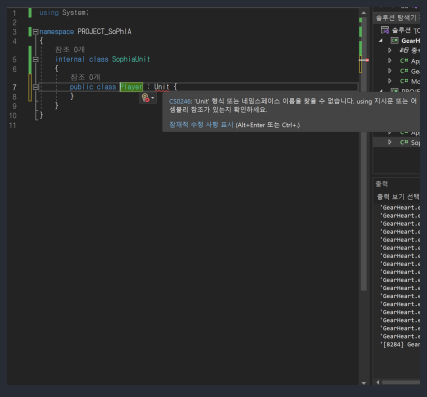
지금 **Public은 정말 어셈 상관없이 어디서든 접근 가능한지** 실험하잖아!!!

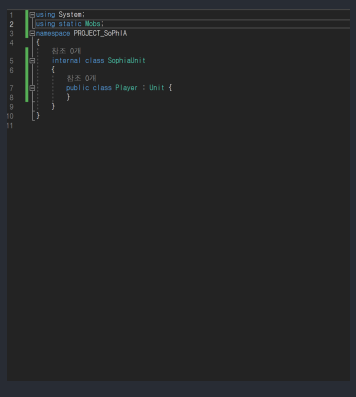
```
//namespace GearHaert { //namespace 없엤다
    public class Mobs   //public으로 바꿨다 Internal
    {
        public interface IHitable { void GetHit(int
        public interface IInvincible { bool SetInvi
        public interface IDie{ void Die(); }

        public struct Coords
        {
            public Coords(int x, int y) { this.x =
            public int x {get;set;}
            public int y { get; set; }
        }

        public class Unit : IHitable, IDie
        {
            public int helthPoint = 0;
            public float forwardAngle = 0.0f;
            public bool invincibleState = false;
            public Coords position = new Coords(0,

            public virtual void GetHit(int _damage,
                this.helthPoint -= _damage; //데미지
                if (this.helthPoint < 0) { Die(); }
            }
            public void Die() {
```

```
                Console.WriteLine("힝 나 죽었어 이펙
            }
        }
    }
//}
```



전	후
	`using static Mobs;` 가 생김





namespace가 없고, Public으로 정의된 클래스는 어디서든 접근이 진짜 가능하구나..