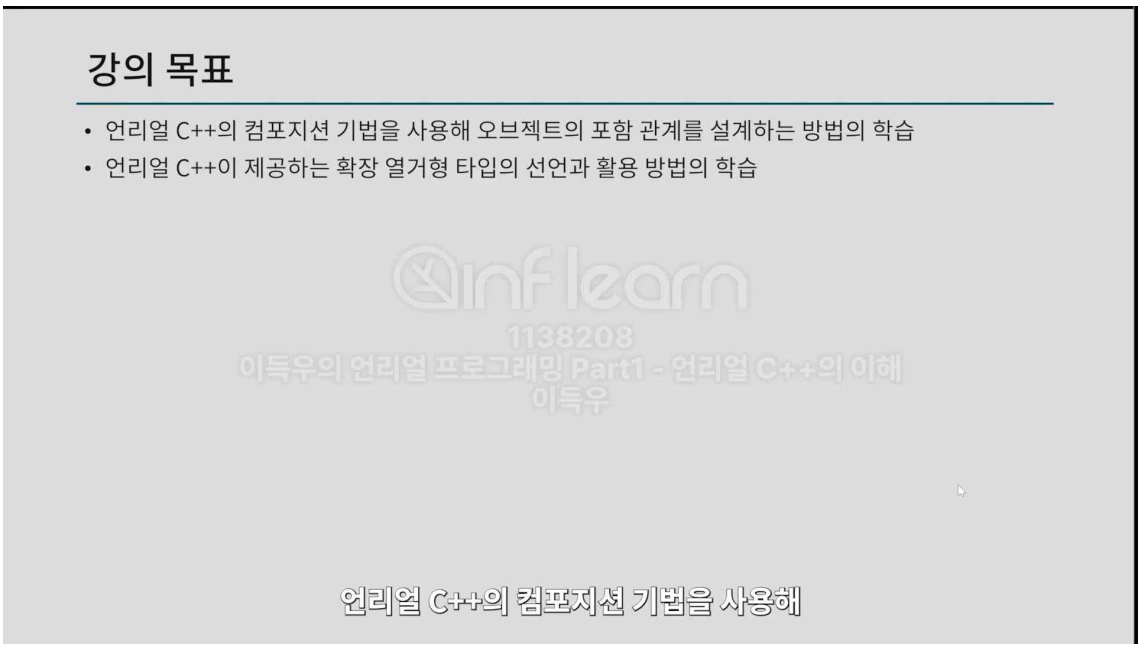
언리얼 프로그래밍 Part1-8

제목:언리얼 C++ 설계 II - 컴포지션

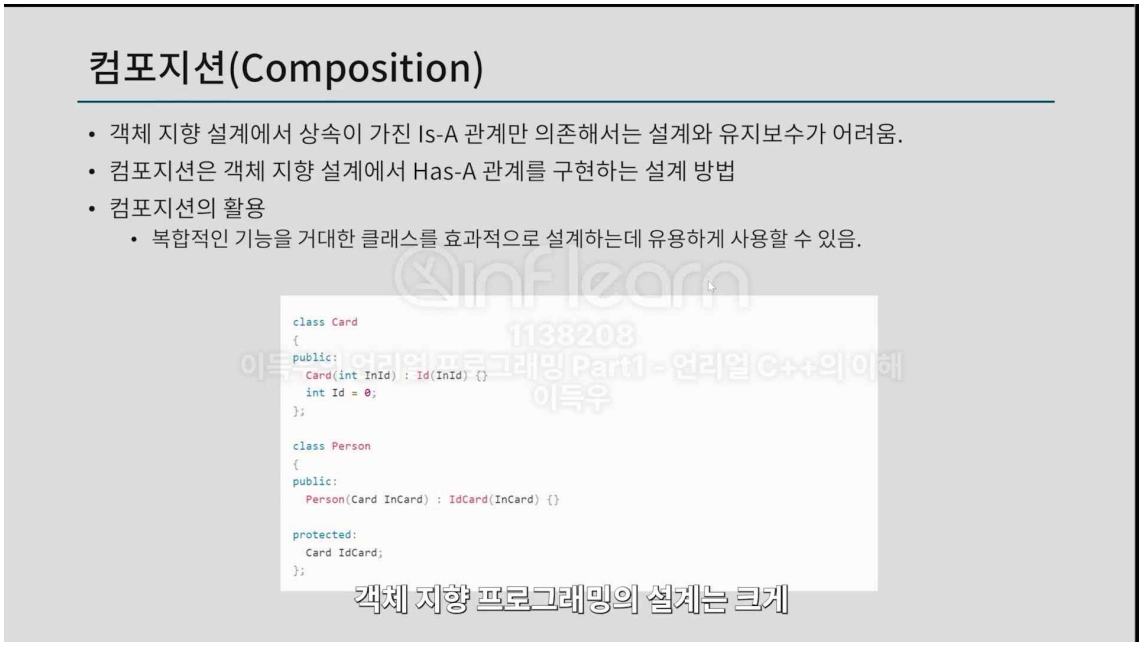
**강의 내용 : 언리얼 C++만의 컴포지션 기법을 사용해 복잡한 언링러 오브젝트를 효과적으로 생성하기

**강의 목차



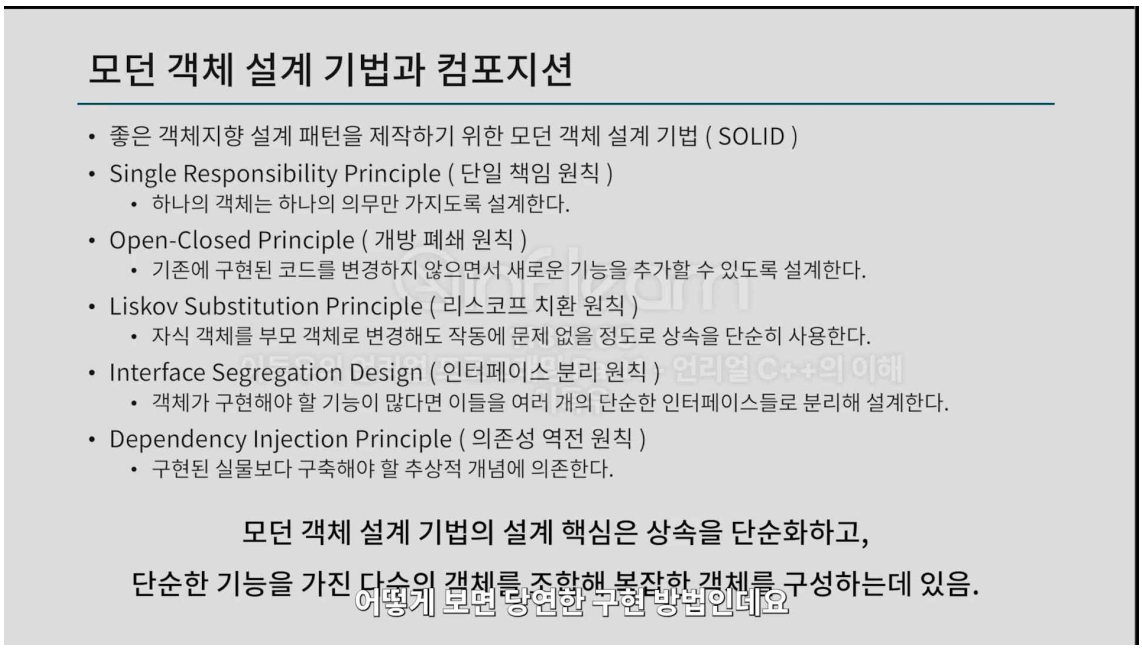
언리얼 오브젝트의 컴포지션

**컴포지션



-객체지향프로그래밍의 설계는 크게 상속과 컴포지션의 활용으로 요약이 가능하다
-상속은 성질이 같은 부모클래스와 자식클래스의 관계를 의미하는 IS-A관계
-컴포지션은 성질이 다른 두 객체에서 어떤 객체가 다른 객체를 소유하는 Has -A 관계로 요약할 수 있다.
-컴포지션을 사용하면 복잡한 기능을 가진 거대한 클래스를 효과적으로 설계하는데 유용하게 사용할 수가 있다.
-위 코드는 어떤 사람이 카드를 소유하는 기능을 컴포지션을 사용해 구현한 예시이다.

**모던 객체 설계 기법과 컴포지션



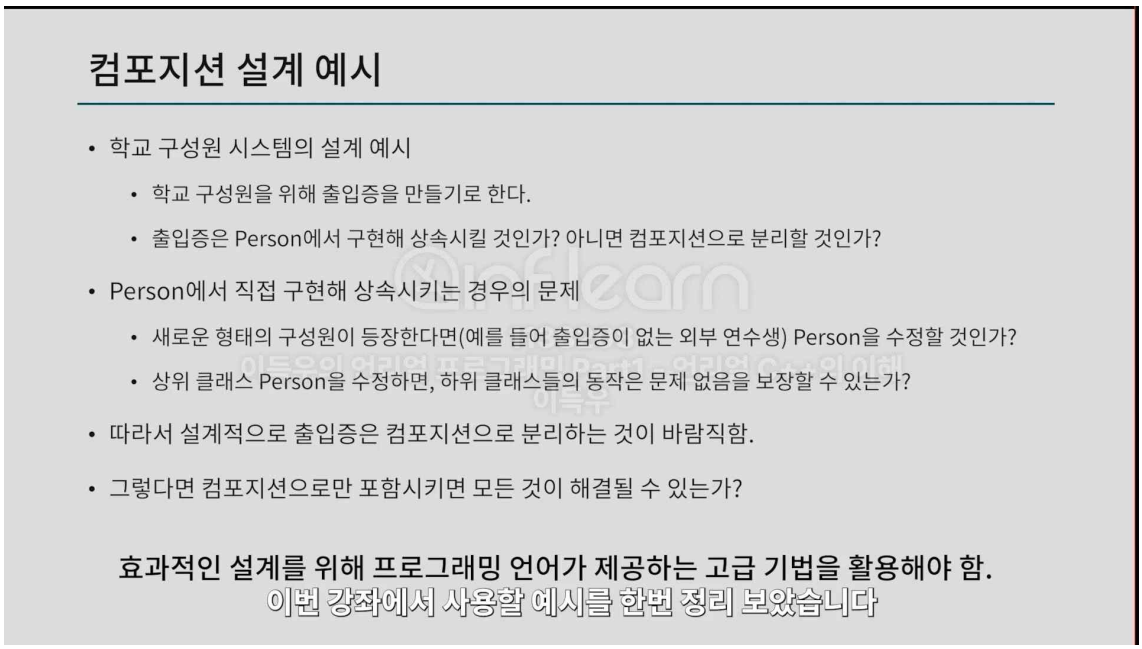
-앞선 강의에서 언급한 모던 객체 설계 기법에서 이러한 컴포지션을 효과적으로 설계하는 것을 중요시 여긴다

-좋은 객체 지향 설계 패턴을 모아놓은 모던 설계 기법을 SOLID라고 이야기 하는데 위 사진과 같이 다섯가지 원칙이 존재함.

-예를 들어 우리와 함께 일하는 기획자가 게임에서 사용할 캐릭터의 실물을 디자인 해서 가져왔는데 우리는 이 실물을 그대로 구현하기 보다는 해당 캐릭터의 기획 의도를 파악하여 보다 추상적인 상위 개념을 기획하고 이를 목표로 설계 해야 한다. 이렇게 할 경우 기획자가 캐릭터 디자인을 변경하거나 새로운 캐릭터가 추가 될 때 유연하게 대처가 가능하다.

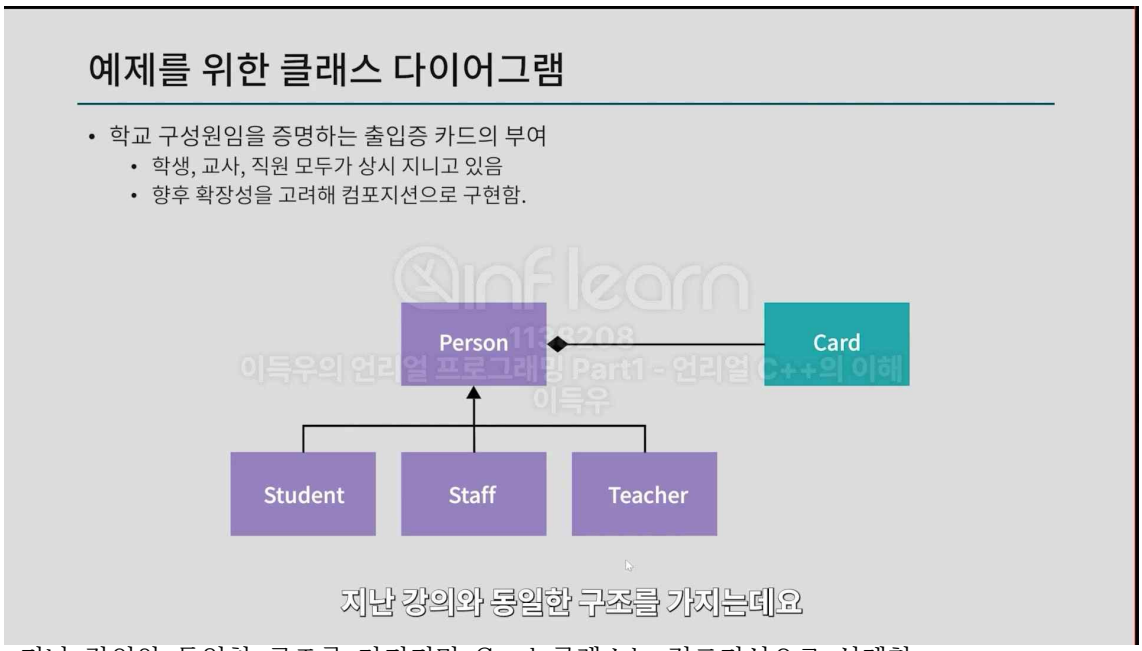
-이러한 원칙들을 정리하자면 상속을 단순화 하고, 컴포지션을 적극 활용하여 복잡한 객체를 구성하라는데에 있다.

**컴포지션 설계 예시



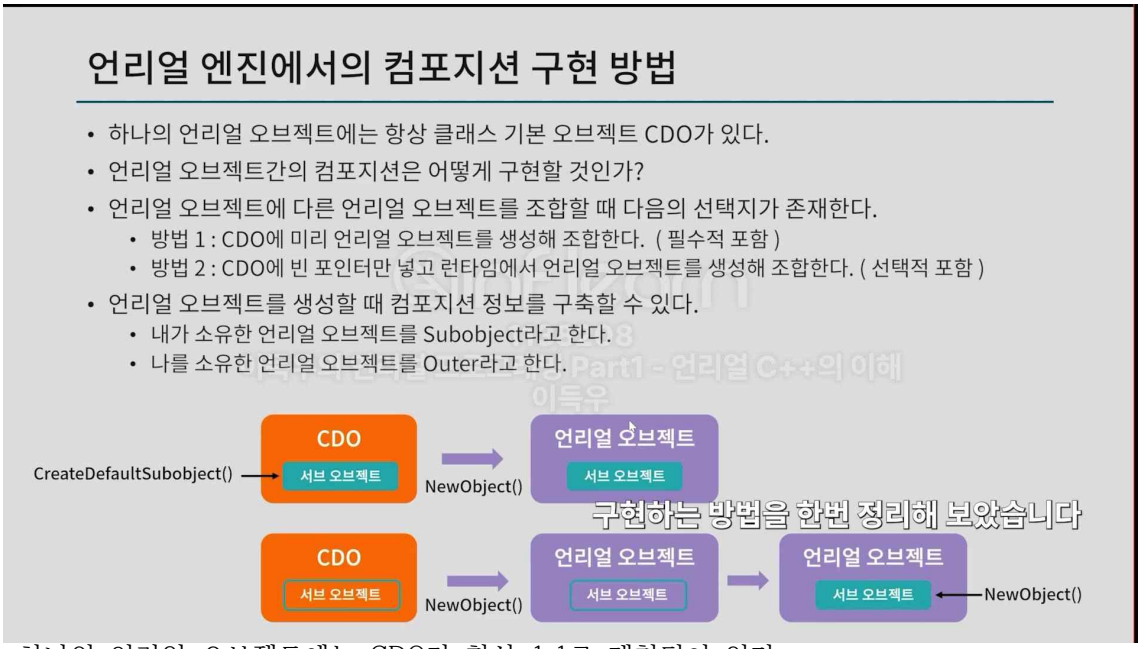
-출입증 기능을 Person에서 구현하게 된다면 학생 ,교사, 직원은 자동으로 카드를 받게됨.
-외부인의 경우에는 Person에서 구현된 기능은 이를 고려하여 수정해야함
-이렇게 수정될 경우 하위클래스들이 문제가 없을까?
-소프트웨어는 기획 변경이 매우 잦기 때문에 향후 기획 변경으로 인한 유지보수를 생각한다면 출입증에 대한 기능은 부모클래스에서 구현하는 것이 아니라 컴포지션으로 분리하는 것이 바람직하다.
-컴포지션으로 구현만 하면 모든 것이 해결될까? -> 그냥 구현하면 안되고 모덴 객체지향이 제공하는 고급 기법을 활용해야 한다.

**예제를 위한 클래스 다이어그램



-지난 강의와 동일한 구조를 가지지만 Card 클래스는 컴포지션으로 설계함

**언리얼 엔진에서의 컴포지션 구현 방법



-하나의 언리얼 오브젝트에는 CDO가 항상 1:1로 매칭되어 있다.

-방법1에 CDO에 미리 언리얼 오브젝트를 생성해 조합할 때 조합하는 객체는 언제나 필수적으로 언리얼 오브젝트에 포함이 된다는 설계를 기준으로 해서 작업하는 방식이다.

-방법2의 경우 내가 필요할 때 생성해서 부착한다는 선택적인 설계를 할 때 이방법을 사용한다.

-이 두가지 방법을 구현할 때 사용하는 코드의 위치도 다르며, 방법1은 CreateDefaultSubobject()라는 API , 방법2는 NewObject() API를 사용한다. 즉 사용하는 API도 다르다.

-이러한 API들로 컴포지션을 구현할 수 있는데 언리얼오브젝트는 생성할 때 컴포지션 정보를 자동으로 구축해준다.

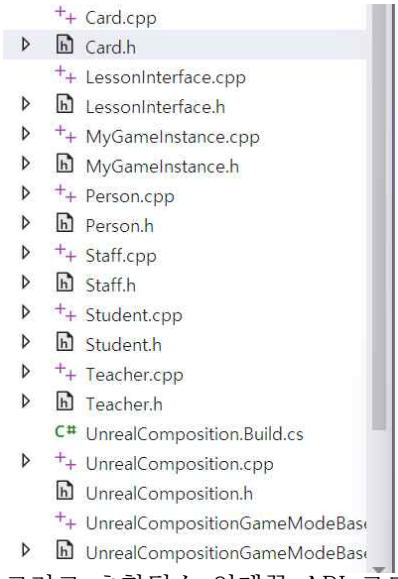
-어떠한 언리얼 오브젝트가 소유한 다른 언리얼 오브젝트를 서브오브젝트라고 이야기하며 서브오브젝트 관점에서 나를 소유한 오브젝트를 아웃터라고 한다.

-정리하자면 서브오브젝트가 필수적인지 아니면 선택적 포함인지에 따라서 우리가 필요한 방법을 지정해주면 된다.

-이번 예제에서는 첫 번째 방법을 사용할것임.

**예시

-기존 UnrealInterface의 작업했던 코드를 모두 복사하여 옮긴후 Card오브젝트 만들기



그리고 호환될수 있게끔 API 코드 복사 해서 붙여넣자.

-uint32 Id는 이번 강의에서 사용하지는 않는다.

- 다음과 같이 지정하는 방식은 c++방식의 열거형

```
enum class ECardType : uint8 //열거형 타입은 항상 접두사E를 붙이고, 열거형이 가지는 기본 타입은 보통 uint8과 같이 바이트 형태로 해준다.
{
	Student = 1,
	Teacher,
	Staff,
	Invalid //이렇게 지정하는건 언리얼c++이 아닌 일반 c++방식이며 언리얼 c++에서는 열거형 타입을 확장해서 다른 무언가를 지정할 수 있다.
};
```

-언리얼 방식의 확장형 열거형 방식

```
UENUM() //이 매크로를 선언하면 열거형 객체에 대한 정보를 언리얼 엔진이 파악하여 우리가 유용한 정보를 가져올 수 있다.
enum class ECardType : uint8 //열거형 타입은 항상 접두사E를 붙이고, 열거형이 가지는 기본 타입은 보통 uint8과 같이 바이트 형태로 해준다.
{
	Student = 1 UMETA(DisplaName = "For Student"), //언리얼 방식에는
	Teacher UMETA(DisplaName = "For Teacher"),
	Staff UMETA(DisplaName = "For Staff"),
	Invalid //이렇게 열거형의 각 데이터마다 메타정보들을 집어 넣을 수 있고 코드에서 사용할수 있는데 이렇게 열거형이 지정되면 ECardType을 만들어줘야한다.
};
```

- 카드 객체를 소유하도록 지정하기
(1)
Card* Card;
--> 우리가 보통 이렇게 카드에 대한 객체를 선언하기 위해서는 헤더를 위에 선언해야 한다.
하지만 우리가 컴포지션 관계에 있을 때는 선언 해서 전방 선언을 하는 것이 좋다.
전방 선언을 하면 헤더를 포함하지 않고 오브젝트는 포인터로 관리하기 때문에
우리가 정확한 구현부는 알 수 없지만 포인터 크기를 가지기 때문에 전방 선언을 통해 의존성을 없앨 수 있다.

(2) 이부분부터 전방선언
UPROPERTY()
class UCard* Card;
-->이렇게 언리얼 오브젝트를 선언하여 컴포지션 관계를 구축 함.

!!!!***하지만 주의점***!!!!
이버전이 4버전까지는 정석이었지만 5버전부터는 다른 방식으로 선언하라고 새로운 표준을 들고옴
언리얼 엔진 5 마이그레이션 가이드 (4->5 바꿀때 주의할점) 를 보면 c++ 오브젝트 포인터 프로퍼티 라는 항목을 보면 기존에 사용하던 원시 포인터 UPROPERTY변수에 원시포인터가 있었던 선언들을 TObjectPtr을 사용하여 변경하라는 내용이 있다.
그래서 선택 사항이지만, 모두 포인터로 선언된 것들을 TObjectPtr이라는 템플릿으로 감싸서 선언하라고 되어 있다. 따라서 선언에 대한 부분만 이렇게 TObjectPtr을 사용하고 구현부에서는 포인터를 사용해도 큰 문제가 없다고 명시한다.

(3) 진짜 선언부
```
UPROPERTY()
TObjectPtr<class UCard> Card;
```
--> 이 방식이 맞다.

(4)Person.h
```
UPerson();
FORCEINLINE const FString& GetName() const { return Name; }
```
4.1 GetName() 뒤에 const지시자를 붙여주는게 좋지만 쓰게 된다면 문제가 생기는데 왜냐하면 변경하지 않겠다고 const를 설정했는데 리턴값을 레퍼런스(&)로 받고 있다.
4.2 리턴값을 레퍼런스로 받으면 받은측에서 변경이 가능하기 때문에 const지시자가 없다. 따라서 레퍼런스로 반환할때 const로 지시하면 FString 앞에도 const로 반환해주어야 한다.

(5) Preson.cpp
```
Card = CreateDefaultSubobject<UCard>(TEXT("NAME_Card"));
```
5.1 생성자 Card에 대해서 CDO에서 구현할땐 CreateDefaultSubobject라는 API를 사용해야 한다.
5.2 인자에는 FName이라는 식별자를 넣어줘야 하는데 고유한 Name이면 된다. 따라서 FName임을 명시적으로 알려줄려면 NAME_이라는 접두사를 써주는게 좋다.

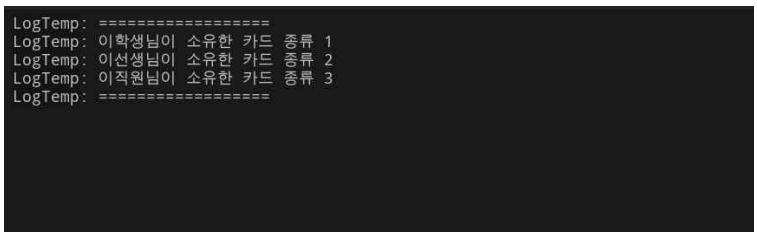
(6)Teacher.cpp

6.1 Teacher의 생성자의 경우에는 부모클래스의 생성자가 호출된 이후에 이코드가 실행된. ==>CreateDefaultSubobject를 할필요가 없다. 하게되면 중복이 됨.

(7) MyGameInstance.cpp

```
void UMyGameInstance::Init()
{
	Super::Init();

	UE_LOG(LogTemp, Log, TEXT("=================="));
	TArray<UPerson*> Persons = { NewObject<UStudent>(), NewObject<UTeacher>(), NewObject<UStaff>()};
	for (const auto Person : Persons)
	{
		const UCard* OwnCard =Person->GetCard();
		check(OwnCard);
		ECardType CardType = OwnCard->GetCardType();
		UE_LOG(LogTemp, Log, TEXT("%s님이 소유한 카드 종류 %d"), *Person->GetName(), CardType)

	}

	UE_LOG(LogTemp, Log, TEXT("=================="));
}
```

==> 실행결과



(8) 열거형 번호가 아닌 메타정보인 DisplayName이라는 문자열을 출력해보기 (FindObject라는 API 사용)

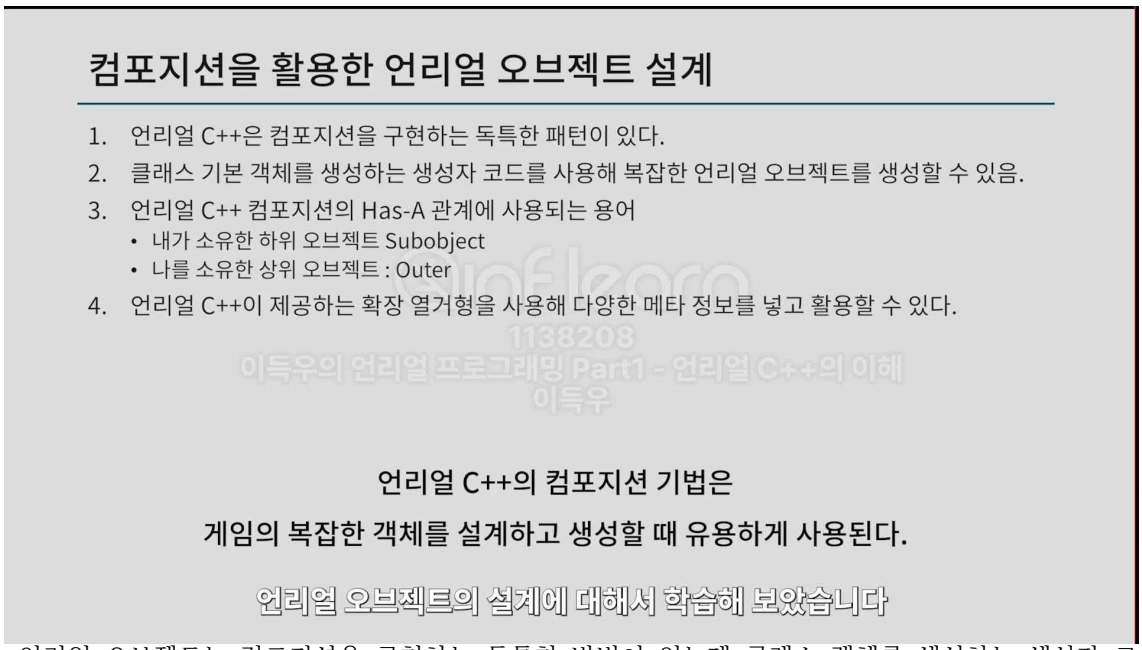
```
void UMyGameInstance::Init()
{
	Super::Init();

	UE_LOG(LogTemp, Log, TEXT("=================="));
	TArray<UPerson*> Persons = { NewObject<UStudent>(), NewObject<UTeacher>(), NewObject<UStaff>()};
	for (const auto Person : Persons)
	{
		const UCard* OwnCard =Person->GetCard();
		check(OwnCard);
		ECardType CardType = OwnCard->GetCardType();
		//UE_LOG(LogTemp, Log, TEXT("%s님이 소유한 카드 종류 %d"), *Person->GetName(), CardType)


		const UEnum* CardEnumType= FindObject<UEnum>(nullptr, TEXT("/Script/UnrealComposition.ECardType"));
		//열거형 타입을 가져오는데 첫번째 인자는 null값 두번째 인자로는 TEXT를 넣어준다
		//TEXT에 들어가는 절대 주소값을 사용하여 우리가 원하는 타입 정보를 얻어올 수 있다.
		//c++에 생성된 언리얼 객체들은 Script라고 하는 절대 주소를 가지는데 우리는 언리얼 컴포지션이라고 하는 모듈이름 을 써준다.
		//이때 프로젝트 이름이 모듈 이름이 된다.
		if (CardEnumType)
		{
			FString CardMetaData = CardEnumType->GetDisplayNameTextByValue((int64)CardType).ToString();
			//GetDisplayNameTextByValue는 int64만 받기 때문에 CardType을 형변환 해준다.
			//그다음에 다시 String으로 형변환해주는데 GetDisplayNameTextByValue는 FText로 반환되기 때문에 다국어 지원 문자열이라서 출력할땐 String으롤 변환해주어야 한다.
			UE_LOG(LogTemp, Log, TEXT("%s님이 소유한 카드 종류 %s"), *Person->GetName(), *CardMetaData)
		}
	}

	UE_LOG(LogTemp, Log, TEXT("=================="));
}
```

==>실행 결과

```
LogTemp: ==================
LogTemp: 이학생님이 소유한 카드 종류 Student
LogTemp: 이선생님이 소유한 카드 종류 Teacher
LogTemp: 이직원님이 소유한 카드 종류 Staff
LogTemp: ==================
```

1,2,3이 문자열로 바뀐걸 볼 수 있다.

**정리



-언리얼 오브젝트는 컴포지션을 구현하는 독특한 방법이 있는데 클래스 객체를 생성하는 생성자 코드를 사용하여 복잡한 언리얼 오브젝트를 구축하고 한번에 생성이 가능하다. 이를 사용하여 컴포지션 구현이 가능하다.
-컴포지션 기법은 복잡한 객체를 설계하고 생성할 때 유용하다.