

언리얼 프로그래밍 Part1-7

제목:언리얼 C++ 설계 I - 인터페이스

**강의 내용 : 언리얼 C++ 인터페이스의 선언과 활용

**강의 목표 : 언리얼 C++ 인터페이스 클래스를 사용해 보다 안정적으로 클래스를 설계하는 기법의 학습

**언리얼 C++인터페이스

언리얼 C++ 인터페이스

- 인터페이스란?
 - 객체가 반드시 구현해야 할 행동을 지정하는데 활용되는 타입
 - 다형성(Polymorphism)의 구현, 의존성이 분리(Decouple)된 설계에 유용하게 활용.
- 언리얼 엔진에서 게임 콘텐츠를 구성하는 오브젝트의 설계 예시
 - 월드에 배치되는 모든 오브젝트, 안 움직이는 오브젝트를 포함 (Actor)
 - 움직이는 오브젝트 (Pawn)
 - 길찾기 시스템을 반드시 사용하면서 움직이는 오브젝트 (INavAgentInterface 인터페이스를 구현한 Pawn)



-인터페이스는 모든 객체지향 설계에서 유용하게 사용되는 타입이다.

-언리얼 엔진도 인터페이스를 지원한다.

-액터는 움직이는 물체와 안움직이는 물체 모두를 통틀은 상위 개념이다.

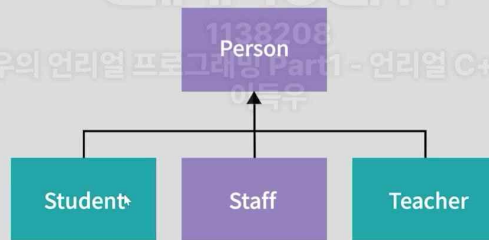
-움직이는 오브젝트는 Actor를 상속받은 Pawn을 적용한다.

-실제 Pawn 클래스 설계시 INavAgentInterface를 구현하도록 설계 되어 있다.

**예제를 위한 클래스 다이어그램

예제를 위한 클래스 다이어그램

- 수업에 참여하는 사람과 참여하지 않는 사람의 구분
 - 수업에 반드시 참여해야 하는 학교 구성원 : 학생, 선생
 - 수업에 참여하지 않는 학교 구성원 : 교직원
 - 수업 행동에 관련된 인터페이스 : ILessonInterface



자, 지금 기존의 인물에서

-기존의 인물에서 학생과 선생님만 있었는데 이젠 '교직원'도 추가됨

-교직원은 수업에 참여하면 안된다.

-같은 인물클래스에서 상속받지만 반드시 구현해야 되는 기능이 명시된 경우에는 이것을 위한 인터페이스를 따로 설계하고 분리하여 Student와 Teacher클래스에 각각 부여하는 것이 좋다.

**언리얼 C++인터페이스 특

언리얼 C++ 인터페이스 특징

- 인터페이스를 생성하면 두 개의 클래스가 생성됨
 - U로 시작하는 타입 클래스
 - I로 시작하는 인터페이스 클래스
- 객체를 설계할 때 I 인터페이스 클래스를 사용
 - U타입 클래스 정보는 런타임에서 인터페이스 구현 여부를 파악하는 용도로 사용됨.
 - 실제로 U타입 클래스에서 작업할 일은 없음.
 - 인터페이스에 관련된 구성 및 구현은 I 인터페이스 클래스에서 진행
- C++ 인터페이스의 특징
 - 추상 타입으로만 선언할 수 있는 Java, C#과 달리 언리얼은 인터페이스에도 구현이 가능함.

UInterface

IInterface

언리얼 엔진에서 인터페이스를 생성하게 되면
클래스 타입정보의 제공 실질적인 설계 및 구현

-인터페이스 생성시 U로 시작하는 타입 클래스와 I로 시작하는 인터페이스클래스가 항상 같이 생긴다.(하나의 인터페이스에 두 개의 클래스가 생성됨.)

-U로 시작하는 클래스는 클래스 타입정보를 제공

-I로 시작하는 클래스는 실질적인 설계를 구현하는 곳임.

-자바와 c#언어는 인터페이스를 추상타입으로만 가능하다. 하지만 언리얼은 내부적으로 c++ 클래스를 사용하기 때문에 추상으로 강제할 방법이 없어 기본 로직에 구현이 가능하다.

****예시**

-이전 프로젝트(ObjectRefletion)의 MyGameInstance.h와 MyGameInstance.cpp를 복사하여 현재프로젝트(UnrealInterface)의 source->UnrealInterface 안에 넣고 리프레시 시키면 사용할수 있을까?

--> 사용 못한다. 왜 일까?

(1) **OBJECTREFLECTION_API**은 외부 모듈이 현재 언리얼 인터페이스라는 모듈내의 클래스인 UMyGameInstance에 접근할수 있는지 지시하는 키워드이다. 하지만 이 키워드가 아직 정의되어 있지 않아 이대로 쓰면 컴파일 에러가 발생한다.

(2) 현재 프로젝트에 맞는 키워드로 바꿔야 하는데 GameModeBase.h에 들어가면 현재 프로젝트의 키워드(**UNREALINTERFACE_API**) 확인이 가능하니 복사해서 붙여넣자.

(3) 컴파일을 진행하기전에 컴파일이 되도록 CPP파일로 변경해야한다.

(4) 변경 된 MygameInstance 소스코드파일

4.1 MygameInstance.cpp

```
#include "MyGameInstance.h"
```

```
//여기 헤더선언 부분도 주의해야할것이 언리얼 오브젝트에 선언된 MynameInstance가 가자 위에 있어야 한다.
```

```
UMyGameInstance::UMyGameInstance()
```

```
{
```

```
    SchoolName = TEXT("기본학교");
```

```
    //이 기본값은 CDO라고 하는 템플릿 객체에 저장이 되어있음
```

```
}
```

```
void UMyGameInstance::Init()
```

```
{
```

```
    Super::Init();
```

```
}
```

4.2 MyGameInstance.h

```
// Fill out your copyright notice in the Description page of Project Settings.
```

```
#pragma once
```

```
#include "CoreMinimal.h"
```

```
#include "Engine/GameInstance.h"
```

```
#include "MyGameInstance.generated.h"
```

```
/**
```

```
*
```

```

*/
UCLASS()
class UNREALINTERFACE_API UMyGameInstance : public UGameInstance
{
    GENERATED_BODY()

public:
    UMyGameInstance();
    virtual void Init() override;

private:
    UPROPERTY()
    FString SchoolName;
};

```

(5) staff, Teacher, Student에 각각 생성자를 통해 이름을 초기화 한 후 MyGameInstance.cpp를 다음과 같이 변경후에 실행한다.


```

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

    UE_LOG(LogTemp, Log, TEXT("====="));
    TArray<UPerson*> Persons = { NewObject<UStudent>(), NewObject<UTeacher>(),
    NewObject<UStaff>()};

    //언리얼 엔진이 제공하는 라이브러리로 TArray가 있는데 이후 강좌에서 소개할 예정임.
    //동적 가변배열에 집어넣은 변수들은 'ranged for'문을 통해서 이터레이트(순회) 할 수
    있다.
    for (const auto Person : Persons) //포인터 경우엔 쓰기 불편하니 auto로 통침. ,
    persons에 있는 모든 요소를 순회한다는 의미임.
    {
        UE_LOG(LogTemp, Log, TEXT("구성원 이름 : %s"), *Person->GetName());
    }
    UE_LOG(LogTemp, Log, TEXT("====="));
}

```



```

LogTemp: =====
LogTemp: 구성원 이름 : 이학생
LogTemp: 구성원 이름 : 이선생
LogTemp: 구성원 이름 : 이직원
LogTemp: =====

```

(6) 인터페이스 추가

6.1 **UINTERFACE**(MinimalAPI)는 인터페이스와 관련된 정보를 저장하기 위함이다.

6.2 타입 정보 관리를 위한 U인터페이스에는 우리가 딱히 할 것은 없다.

6.3 **virtual void** DoLesson() = 0; 과 같이 abstract 가상함수('=0' ==> 이게 abstract 가상함수선언 부분임)를 선언하게 되면 이 인터페이스를 상속받는 클래스들은 반드시 이 DoLesson 함수를 구현해 줘야 한다.

6.4 상속 관계를 사용하여 특정 클래스에게 구현을 강제할 수 있다.

6.5 Teacher와 Student에게 구현을 강제 시키기

만약 Teacher에서 ILessonInterface를 상속받고 DoLesson()을 구현하지 않게될 경우 가상함수로 인터페이스를 선언했기 때문에 컴파일 할 수 없다는 빌드 또는 컴파일 에러가 나오게 된다.(구현 강제시킴)

6.6 언리얼 c++ 인터페이스 특징으로 DoLesson()을 abstract 상태로 유지하는게 좋지만, 꼭 그렇게 할 필요는 없다. 즉 우리는 DoLesson() 에다가 코드를 넣을수도 있다.

-LessonInterface.h의 **virtual void** DoLesson() = 0;을 다음과 같이 변경

virtual void DoLesson()

```
{
    UE_LOG(LogTemp, Log, TEXT("수업에 입장합니다."));
}
```

이렇게 될 경우 Student나 Teacher에서 DoLesson()을 추상 클래스가 아니기 때문에 강제로 구현할 필요가 없어진다. (선택하기 나름)

6.7 6.6처럼 변경하게 되면 원래 모던 객체지향에서 추구하는 방식과는 거리가 멀지만 언리얼 소스코드를 보면 다음과 같이 활용한것들을 볼수 있다. 엄격히 따르는것보다는 우리에게 편리한 형태를 구현해보자.

6.8 인터페이스에 구현된 로그도 출력하고 Student에 로그도 같이 출력하고 싶을 때 상위 클래스의 DoLesson을 호출해야 하는데 이때는 Super를 사용할 수없다 왜냐하면 UStudent의 Super는 UPerson을 지정하기 때문이다. (클래스 정보에 대해서는 단일 상속만 지원함.) ==>따라서 직접 입력해야 한다. **ILessonInterface::DoLesson()**;

6.9 Staff의 경우 수업을 받지 않는데 수업을 받는 인원과 안받는 인원을 구분하기 위해서는 그 구성원이 LessonInterface라는 상속 받았는지 체크해주면 된다. 이때 유용한 것이 형변환(casting 연산자) 이다. 언리얼 엔진은 안정적인 캐스팅이 가능하기 때문에 만약 형변환에 실패하면 null을 반환하여 구현했는지 안구현했는지 파악할 수 있다.

```
LogTemp: ...
LogTemp: 구성원 이름 : 이학생
LogTemp: 구성원 이름 : 이선생
LogTemp: 구성원 이름 : 이직원
LogTemp: ...
LogTemp: 이학생님은 수업에 참여할 수 있습니다
LogTemp: 이선생님은 수업에 참여할 수 있습니다
LogTemp: 이직원님은 수업에 참여할 수 없습니다
LogTemp: ...
```

6.10 예시코드가 이제부터 길어지기 때문에 파일을 직접 열어보자.

****정리**

언리얼 C++ 인터페이스

1. 클래스가 반드시 구현해야 하는 기능을 지정하는데 사용함.
2. C++은 기본적으로 다중상속을 지원하지 않지만, 언리얼 C++의 인터페이스를 사용해 가급적 축소된 다중상속의 형태로 구현하는 것이 향후 유지보수에 도움된다.
3. 언리얼 C++ 인터페이스는 두 개의 클래스를 생성한다.
4. 언리얼 C++ 인터페이스는 추상 타입으로 강제되지 않고, 내부에 기본 함수를 구현할 수 있다.

1138208
이득우의 언리얼 프로그래밍 Part1 - 언리얼 C++의 이해

언리얼 C++ 인터페이스를 사용하면,
클래스가 수행해야 할 의무를 명시적으로 지정할 수 있어
좋은 객체 설계를 만드는데 도움을 줄 수 있다.

이번 강의에서는 언리얼 C++ 인터페이스의 선언과

-언리얼 c++ 인터페이스는 두 개의 클래스를 생성하고 언리얼 c++는 c++언어에서 구현하다보니 다른 언어들과 다르게 추상 타입으로 강제되지 않고 내부에 기본 함수를 구현 할 수 있는 특징을 가지게 된다.