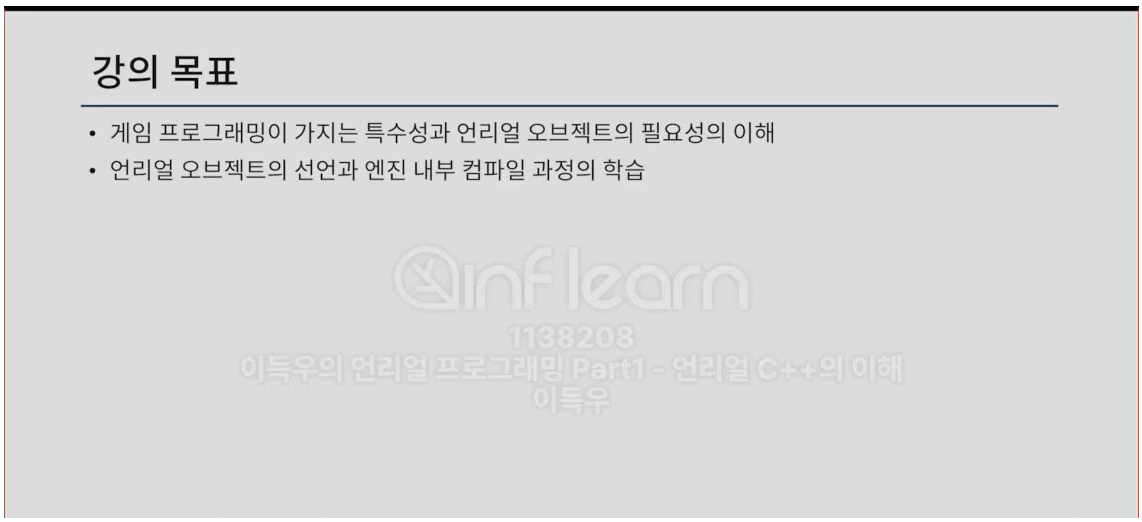
언리얼 프로그래밍 Part1-4

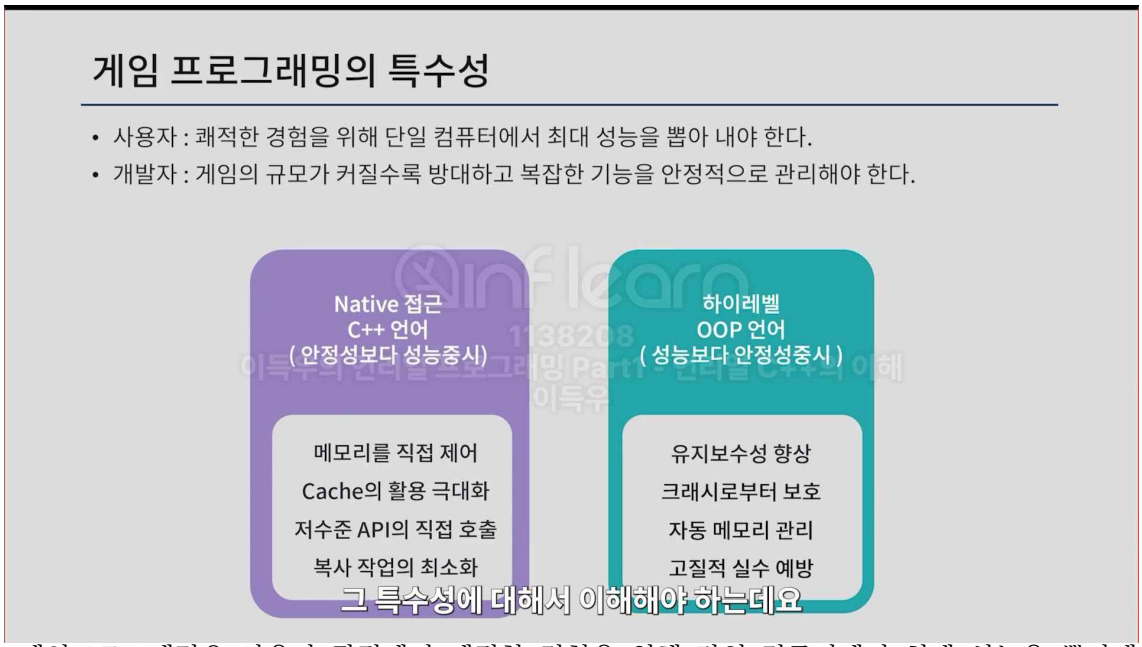
제목:언리얼 오브젝트 소개

**강의 내용 : 언리얼 오브젝트의 소개와 선언 방법

**강의 목표



**게임 프로그래밍의 특수성

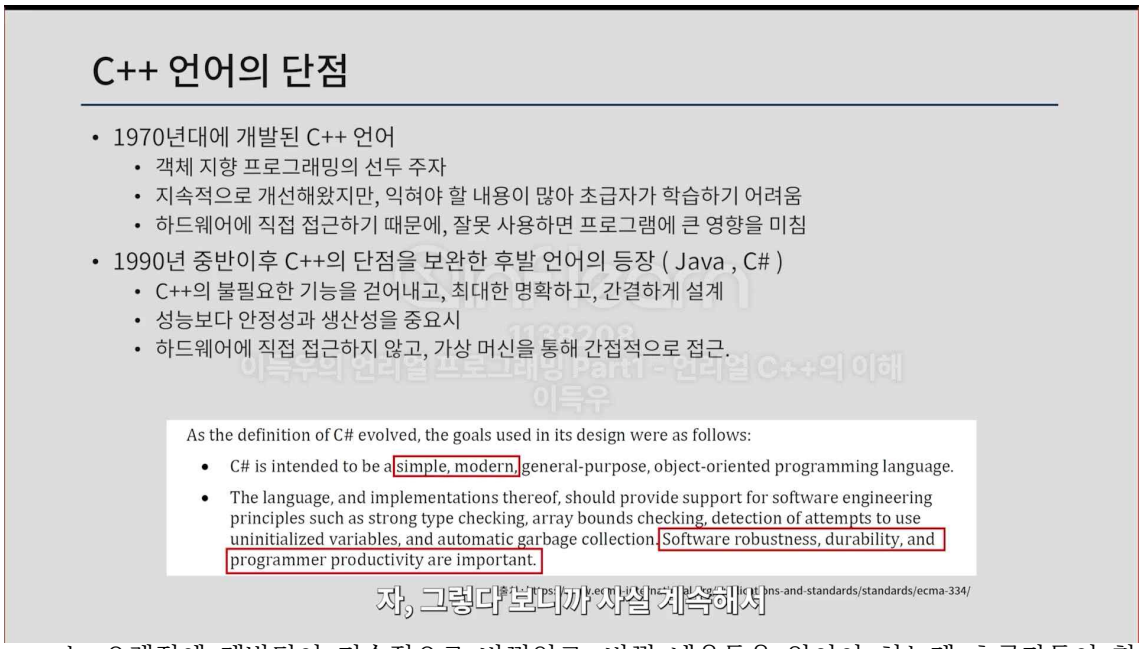


-게임프로그래밍은 사용자 관점에서 쾌적한 경험을 위해 단일 컴퓨터에서 최대 성능을 뽑아내야 한다.

-개발자는 규모가 커질수록 방대하고 복잡한 기능을 안정적으로 관리할 수 있어야 한다.

-개발자는 성능과 안전성이라는 두 마리 토끼를 잡아야 하지만 우선순위를 꼽으라면 사용자 관점에서 사용자를 위한 성능을 택해야 한다.
-이를 위해서 개발자는 하드웨어에 직접 접근하여 성능을 높일 수 있는 c++언어를 선택한다.
-c++는 안전성보다는 성능을 중시함으로써 메모리를 직접 제어하고 캐시의 활용성을 극대화 시킨다.
-또한 저수준에서 하드웨어 및 운영체제에 직접 접근이 가능하기 때문에 높은 성능 발휘가 가능하고, 자료구조를 다룰때도 복사작업을 최소화하여 성능을 향상시킬 수 있다.
-자바나 c#의 경우 성능보단 안전성을 중요시 여기기 때문에 유지보수가 쉽고, 자동으로 메모리를 관리하기 때문에 프로그래머들의 실수로부터 안전하게 프로그램을 지켜준다.

**c++언어의 단점

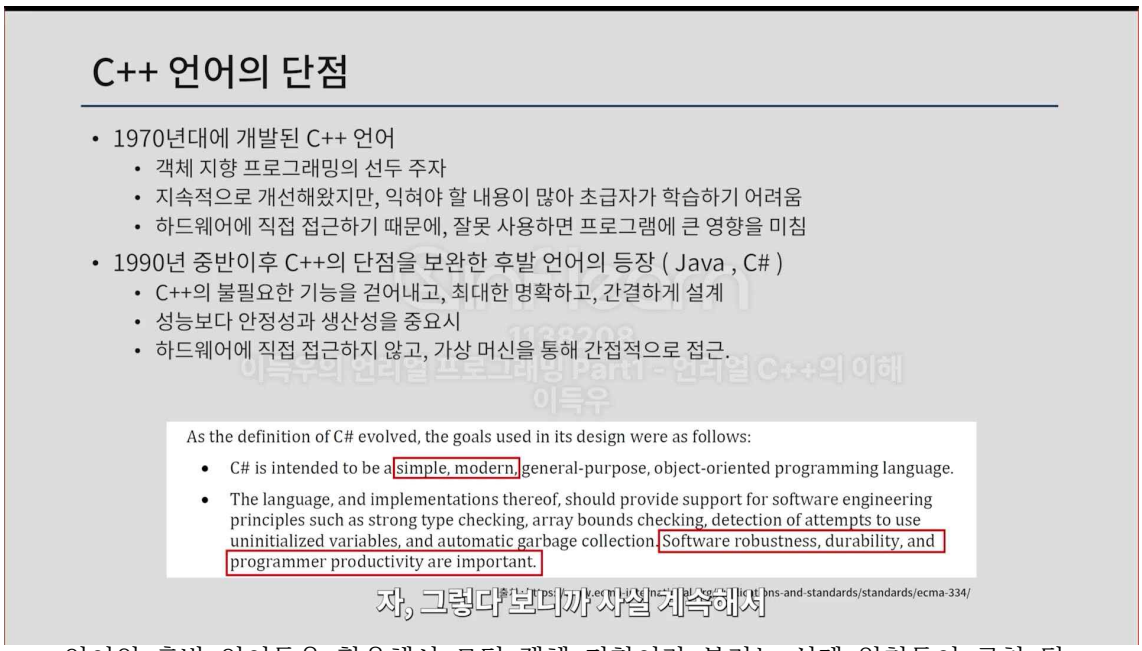


-c++는 오래전에 개발되어 지속적으로 바뀌었고, 바뀐 내용들을 알아야 하는데 초급자들이 학습하기에 어려운 언어이다.

-하드웨어에 직접 접근하는 특징 때문에 실수를 하게 되면 전체적으로 영향을 끼치게 된다.
-오랫동안 c++언어를 사용하다가 등장한 자바를 필두로 c#과 같은 언어가 등장하게 되었다.

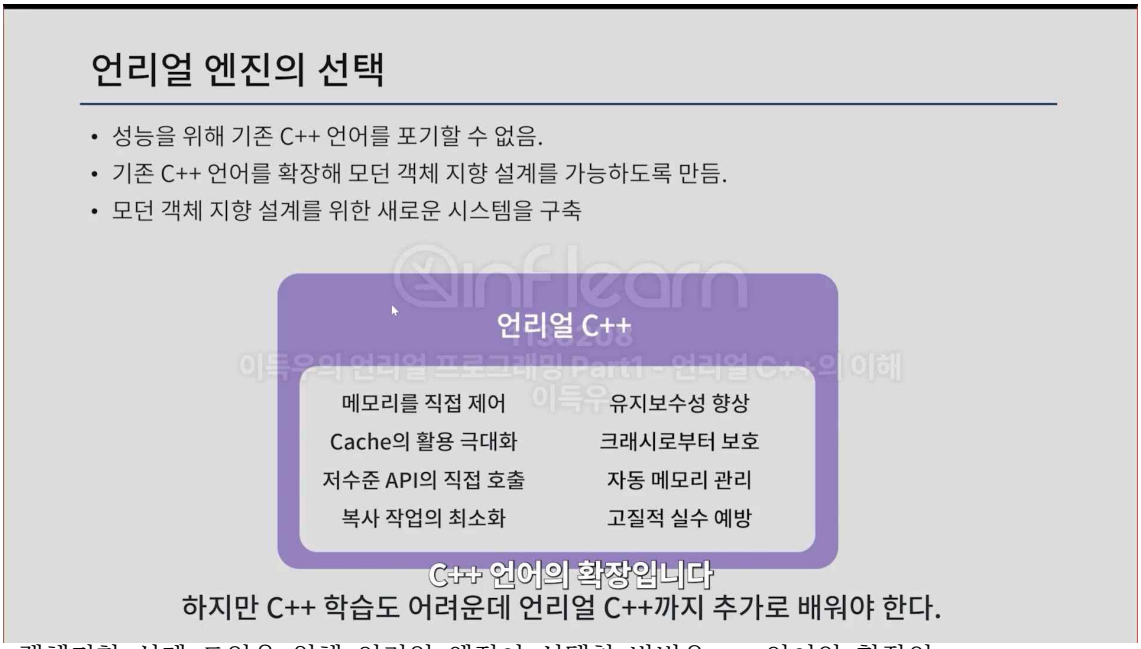
-자바나 c#은 c++에서 사용된 불필요한 기능을 걷어내고, 명확하고 간결하게 설계됨.
-특히 자바나 c#은 성능보다는 안전성과 생산성을 중요시 여기고, 하드웨어에 직접 접근하는 게 아닌 가상머신을 통해 간접적으로 접근한다.

**모던 객체 지향 설계 원칙



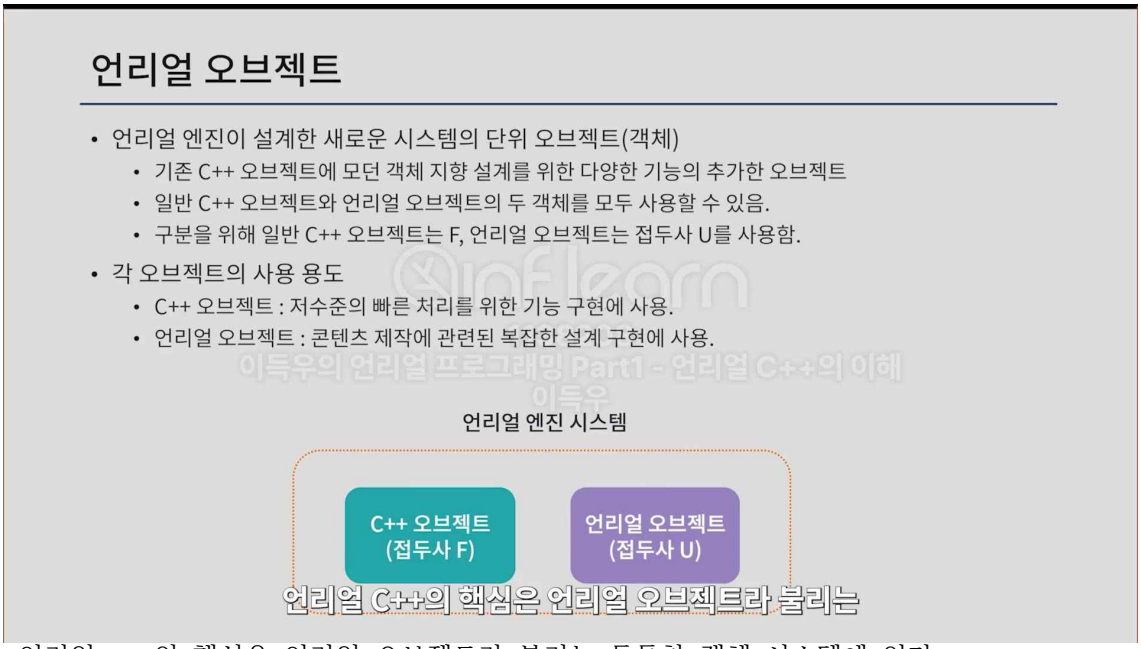
-c++언어의 후발 언어들을 활용해서 모던 객체 지향이라 불리는 설계 원칙들이 구현 됨
-이것도 만들어진지 20년이 지난시점에서 현재의 '모던'을 뜻하는게아님.
-이때부터 안정적인 객체지향 프로그래밍을 위한 원칙이 고안되고 실무에서 통용되고 있다.
-각각 SOLID라는 다섯가지 원칙이 있다.
-후발 언어들은 객체지향 원칙을 보다 편리하게 구현할수 있도록 c++가 가지고 있지 않은 인터페이스, 리플렉션, 델리게이트와 같은 기능을 만들어 모던 객체지향을 보다 편리하게 할 수 있게끔 지원한다.
- 이런 소프트웨어 안전성을 위해 고안된 기능들은 게임 규모가 대형화 될수록 c++언어가 주 언어로 사용하는 게임 제작에서도 이러한 객체지향 설계 도입이 필요하다고 공감대를 형성하게 됨.

**언리얼 엔진의 선택



-객체지향 설계 도입을 위해 언리얼 엔진이 선택한 방법은 c++언어의 확장임.
-성능을 위해 c++를 버릴수는 없기 때문에 c++의 매크로 기능을 활용하여 모던 객체 지향 언어들이 가지고 있는 기능을 언리얼 엔진이 자체적으로 구현해 버림.
-언리얼c++라는 새로운 시스템은 성능과 유지보수 라는 두 마리 토끼를 잡을 수 있게 됨.
-다만 일반적인 c++도 어려운데 언리얼 c++는 더 어려움

**언리얼 오브젝트



-언리얼 c++의 핵심은 언리얼 오브젝트라 불리는 독특한 객체 시스템에 있다.
-언리얼 엔진은 c++를 토대로 제작되었기 때문에 일반 c++에서 사용했던 객체 시스템은 그대로 사용 가능하지만, 언리얼 오브젝트라는 새로운 언리얼만의 객체 규약을 사용하여 객체 설계가 가능하다.
-언리얼 오브젝트를 사용하면 자바나 c# 같은 후발 언어가 가진 장점을 사용해서 생산성을 높일 수가 있다.
-이 둘을 구분하기 위해 c++오브젝트는 F, 언리얼 오브젝트는 U라는 접두사를 사용함
-두 오브젝트 모두 언리얼 프로그래밍에 사용 가능하며 c++는 저수준 빠른처리가 필요한 기능에, 언리얼 오브젝트는 콘텐츠 제작에 관련된 복잡한 설계에 주로 사용한다.

**언리얼 오브젝트 공식문서
-링크 : https://dev.epicgames.com/documentation/ko-kr/unreal-engine/fname-in-unreal-engine?application_version=5.1

-언리얼에는 게임 오브젝트 처리를 위한 탄탄한 시스템(UObject)가 존재함.
-UObject는 가장 원초적은 객체이며, UObject 클래스를 상속받아 어떤 클래스를 생성하면 그 클래스는 언리얼 오브젝트가 된다. 그리고 이클래스는 언리얼 오브젝트로 지정하기 위해 UCLASS라는 매크로를 사용하여 태그를 해줘야 한다.
-UCLASS 매크로가 있는 경우에는 내부적인 기능등일 자동으로 만들어 진다.
-프로퍼티, 함수세트 CDO는 다음 강좌에서 알아본다.
-언링러 오브젝트를 생설할땐 c++의 new와 다르게 NewObject라고 하는 API 및 기타 다른 API를 활용한다.
-언리얼 엔진 특징으로 후발 언어가 제공하는 특징들을 매크로 선언만으로 바로 활용이 가능함.

(1)가비지 컬렉션
(2)레퍼런스 업데이트
(3)리플렉션
(4)시리얼라이제이션
(5)디폴트 프로퍼티 변경사항 자동 업데이트
(6)자동 프로퍼티 초기화
(7)자동 에디터 통합
(8)실행시간에 유형 정보 사용가능
(9)네트워크 리플리케이션

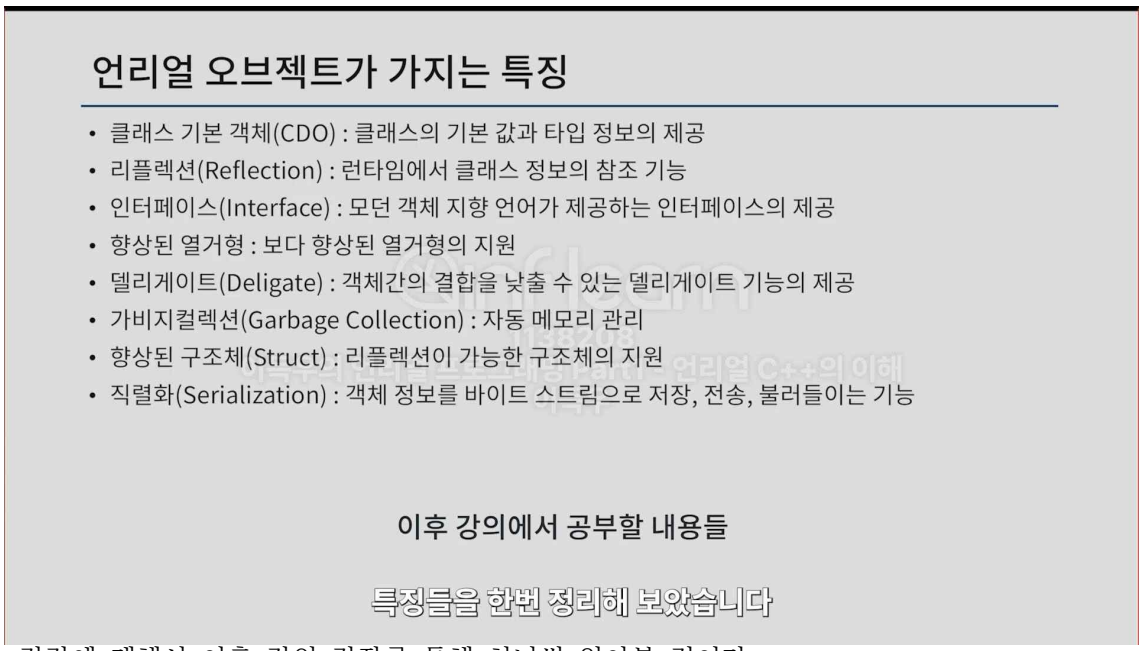
-언리얼 오브젝트를 처리하기 위해서 컴파일 과정에서 특수한 과정이 들어가는데 이를 언리얼 헤더 툴이라고 한다.

-언리얼 오브젝트 구조

```
#pragma once
	#include 'Object.h'
	#include 'MyObject.generated.h'
	/**
	 *
	 */
	UCLASS()
	class MYPROJECT_API UMyObject : public UObject
	{
		GENERATED_BODY()
	};
```

(1)UCLASS()와 같은 매크로와 #include 'MyObject.generated.h'와 같은 독특한 헤더파일들을 항상 가지게 된다.

**강의자께서 정리하신 언리얼 오브젝트가 가지는 특징



-각각에 대해서 이후 강의 강좌를 통해 하나씩 알아볼 것이다.

**언리얼 오브젝트의 선언

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

#pragma once

#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h"
#include "MyObject.generated.h"

/**
 *
 */
UCLASS()
class UNREALOBJECT_API UMyObject : public UObject
{
	GENERATED_BODY()

};
```

-언리얼 오브젝트가 되기 위해서는 다음과 같은 헤더가 필수로 포함된다.

```
#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h“
```

-MyObject.generated.h 같은 경우는 다른 폴더에 포함되어 있다.
-UObject를 상속받는다.

-클래스 이름 MyObject 앞에는 접두사 U를 붙이지만 탐색기에 있는 파일 이름 앞에서는 붙이지 않는다.
언리얼 오브젝트 선언임을 명시하기 위해 UCLASS()라는 매크로를 클래스 전에 명시해준다.

-그리고 나서 UNREALOBJECT_API의 경우 언리얼 오브젝트라는 프로젝트 이름이며 프로젝트 이름_API라고 지정함으로써 MyObject라고 하는 언리얼 오브젝트가 다른 DLL이나 다른 모듈로 불리는 언리얼에서의 라이브러리 단위에서 즉 다른곳에서도 사용할 수 있게 개방해준다는 것을 의미한다.
-UNREALOBJECT_API 이거 없애버리면 언리얼 오브젝트라고 하는 모듈내에서 밖에 사용할 수가 없게 된다.

-GENERATED_BODY()의 선언부를 보면 매크로의 다수인자가 BODY_MACRO_COMBINE 로 변경되서 진행되고 BODY_MACRO_COMBINE 다시 BODY_MACRO_COMBINE_INNER라는 매크로로 변경이 되는데 이때는 매크로가 가지고 있는 토큰(##) 들을 사용하여 A,B,C,D와 같은 인자를 하나로 묶은 긴 단어를 만들어 낸다.

-GENERATED_BODY() 매크로는 BODY_MACRO_COMBINE의 CURRENT_FILE_ID 라고하는 정

보와 언더바(_) 정보 그리고 현재 라인줄에 대한 정보 그리고 GENERATED_BODY라고 하는 이 정보 네가지를 묶어서 하나의 긴문자를 만든다고 보면 된다.

-A##B##C##D 가 어디에 사용되는지 알기 위해서는 MyObject.generated.h 여기로 가야하는데 우리 프로젝트 파일을 탐색기로 열어야 한다.(E:\HighProject\UnrealObject\Intermediate\Build\Win64\UnrealEditor\Inc\UnrealObject\UHT)
-복잡한 파일 경로에 헤더파일을 만든 경우는 해당 헤더파일을 건들지 말라는 뜻으로 이해해주는게 좋다.

- 95 번째줄에 CURRENT_FILE_ID라고 긴 문장으로 재정의 되어 있다. (#define CURRENT_FILE_ID FID_HighProject_UnrealObject_Source_UnrealObject_MyObject_h)

(1)CURRENT_FILE_ID는 앞선 오브젝트 매크로(ObjectMacros.h)에 있는 BODY_MACRO_COMBINE(CURRENT_FILE_ID,_,__LINE__,_GENERATED_BODY)에서 CURRENT_FILE_ID 부분을 담당한다. (언더바가 같이 묶이게 됨),

(2)GENERATED_BODY가 코딩된 라인 정보가 언더바가 붙어서 이어서 __LINE__으로 들어가게 된다.

(3) 그뒤에 또 _GENERATED_BODY라는 단어가 붙는데 generated.h의 80번째 줄에 있는 #define FID_HighProject_UnrealObject_Source_UnrealObject_MyObject_h_15_GENERATED_BODY 처럼 이렇게 합쳐진 긴 단어가만들어진다.

-결국 정리하자면 GENERATED_BODY라고 하는 것은 generated.h에 있는 헤더파일에 있는
```
#define FID_HighProject_UnrealObject_Source_UnrealObject_MyObject_h_15_GENERATED_BODY \
PRAGMA_DISABLE_DEPRECATION_WARNINGS \
public: \
	FID_HighProject_UnrealObject_Source_UnrealObject_MyObject_h_15_SPARSE_DATA \

FID_HighProject_UnrealObject_Source_UnrealObject_MyObject_h_15_RPC_WRAPPERS_NO_PURE_DECLS \
	FID_HighProject_UnrealObject_Source_UnrealObject_MyObject_h_15_ACCESSORS \
	FID_HighProject_UnrealObject_Source_UnrealObject_MyObject_h_15_INCLASS_NO_PURE_DECLS \
	FID_HighProject_UnrealObject_Source_UnrealObject_MyObject_h_15_ENHANCED_CONSTRUCTORS \
private: \
PRAGMA_ENABLE_DEPRECATION_WARNINGS
```
이부분을 사용할것임을 말한다. 또 각각의 부분들이 복잡하게 define된 매크로들이 한줄씩 각각 선언되어 있는 것을 볼 수 있다. 또한 이 부분들이 위쪽(17번째 줄)에 보면 define이 되어 있는 것을 알 수 있다.

-이러한 것들이 결국

```
private: \
        static void StaticRegisterNativesUMyObject(); \
        friend struct Z_Construct_UClass_UMyObject_Statics; \
public: \
        DECLARE_CLASS(UMyObject, UObject, COMPILED_IN_FLAGS(0), CASTCLASS_None, TEXT("/Script/UnrealObject"), NO_API) \
        DECLARE_SERIALIZER(UMyObject)
```

다음과 같은 코드들을 만들어 내는데 사용된다.

-이러한 코드들은 또 무슨 역할을 하는지 궁금할 수도 있는데, 몰라도 무방하다. 여기에 관련된 내용들은 자바나 c#에서 제공하는 객체지향 설계를 위해 제공되는 여러 가지 좋은 기능들을 언리얼 엔진이 자동으로 생성해 지원하는 기능이라고 이해해도 크게 지장이 없다.

-이렇게 깊숙한 곳에 꽁꽁 숨겨놓은 이유도 우리가 이것을 이해할 필요가 없을을 의미한다.

-만약에 정말 이게 궁금하다면, 이러한것들을 정리한 웹사이트를 보는 것이 편리할 것이다.

-에러 예시

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

#pragma once

#include "CoreMinimal.h"
#include "UObject/NoExportTypes.h"
#include "MyObject.generated.h"
(여기 8번째 줄인데 한칸 더띄어쓰기 해서 두줄 공백 만들면 에러생김)
/**
 *
 */
UCLASS()
class UNREALOBJECT_API UMyObject : public UObject
{
        GENERATED_BODY()

};
```

여기에서 헤더 밑에 한줄이라도 더 띄어쓰기 하게 되면 라인 정보가 변경되어 헤더부터 다시 만들어줘야한다.

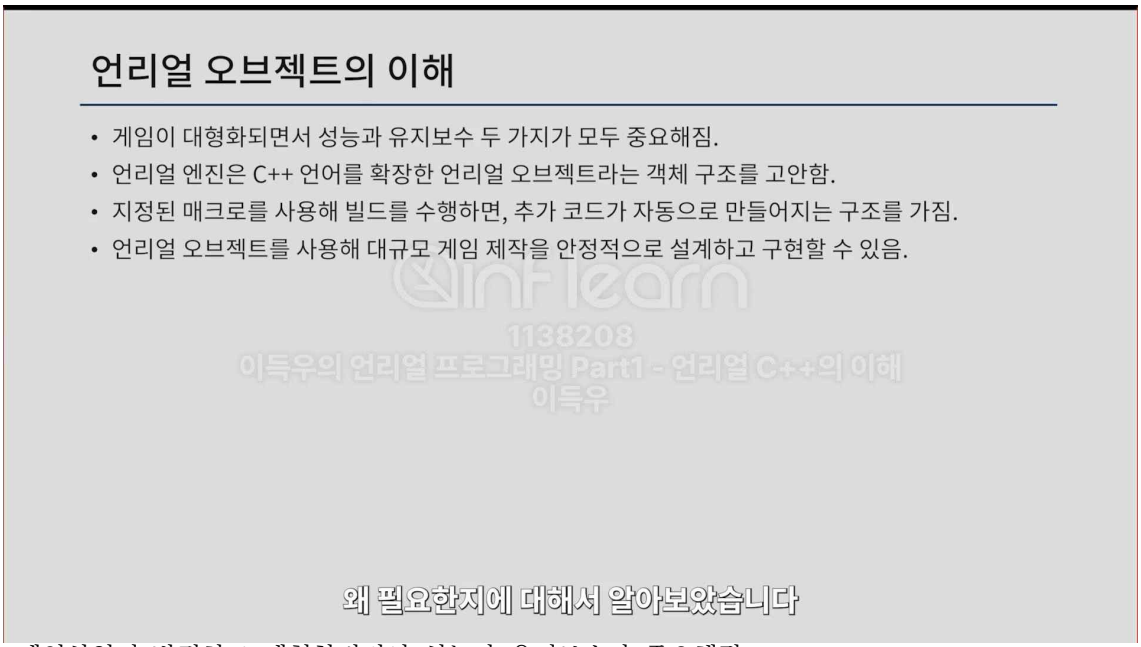
-에디터를 끄고 다시 빌드하고 재컴파일 하면 generated.h 파일에 15->16으로 변경됨을 볼 수 있다.

-우리가 헤더파일을 직접 고칠일은 없고, 우리가 수정사항을 반영해서 빌드를 수행하면 언리얼 엔진의 빌드 시스템이 알아서 generated.h파일을 자동으로 생성해준다. 심지어 삭제하더라도 빌드를 다시 하면 재생성해준다.

-삭제 하더라도 빌드하면 재생성되는이유
(1) 로그 기록을 보면 Parsing headers for UnrealOnjectEditor라고 되어있는데 헤더 파일은 Parsing한다고 되어 있음.
(2)UnrealObject라는 프로젝트의 에디터용 빌드에 있는 헤더를 Parsing한다고 이해하자
(3) 여기 UHT라고 하는 내부적인 UnrealHeaderTool 라고 하는 실행하겠다고 되어있음
(4) UnrealHeaderTool이 실행된다면 헤더로 선언된것들을 보고 이런 매크로들을 참조하여 자기가 언리얼 오브젝트에 관련된 generated.h파일을 여기에 있는 UnrealHeaderTool이라는 프로그램이 자동으로 생성시켜준다.
(5) 만일 generated.h가 없다면 얘는 include가 안되기 때문에 제대로된 c++ 프로젝트가 아니게 된다.
(6)따라서 UnrealHeaderTool을 사용하여 먼저 파일을 생성해주면 그다음부턴 include 될 수 있기 때문에 UnrealHeaderTool에서 선언된 클래스들을 포함해서 완벽한 소스가 만들어지게됨.
(7)이제 만들어진 소스를 컴파일 하게됨

-정리
(1) 언리얼 오브젝트에 코드를 분석하는 단계가 한가지가 있어서 바로 컴파일이 아닌 UnrealHeaderTool에 의해서 소스코드를 자동으로 생성하고 자동으로 생성된 코드를 포함하여 마지막 최종 빌드를 진행하는 두단계 과정으로 진행한다.

**이번 강의 정리



-게임산업이 발전하고 대형화되면서 성능과 유지보수가 중요해짐
-두마리 토끼를 가지기 위해 언리얼 오브젝트라는 객체구조를 고안함
-언리얼 오브젝트는 많은 기능을 제공하는데 개발자가 입력하기엔 복잡하여 자동으로 관련된 코드를 생성하도록 언리얼 헤더 툴이라는 도구를 제공함
-엔리얼 헤더툴은 자동으로 추가 코드를 만들어내서 언리얼 오브젝트가 자동으로 다양한 기능을 제공하도록 도와준다.

-개발자는 단순히 지정된 매크로만 입력하면 되고, 언리얼 헤더 툴이 자동으로 생성되는 헤더파일에는 접근할 필요가 없다.

-이러한 언리얼 오브젝트는 대규모 게임 제작을 안정적으로 설계하고 구현하는데 큰 도움을 주기 때문에 언리얼 엔진 기술의 핵심이라고 볼 수 있다.