# 언리얼 빌드 시스템

## (Unreal Build System)

# 강의 내용

언리얼 에디터의 동작 방식을 이해하고,
언리얼 엔진의 독특한 모듈 시스템을 기반으로
소스코드와 플러그인 구조를 직접 제작하기

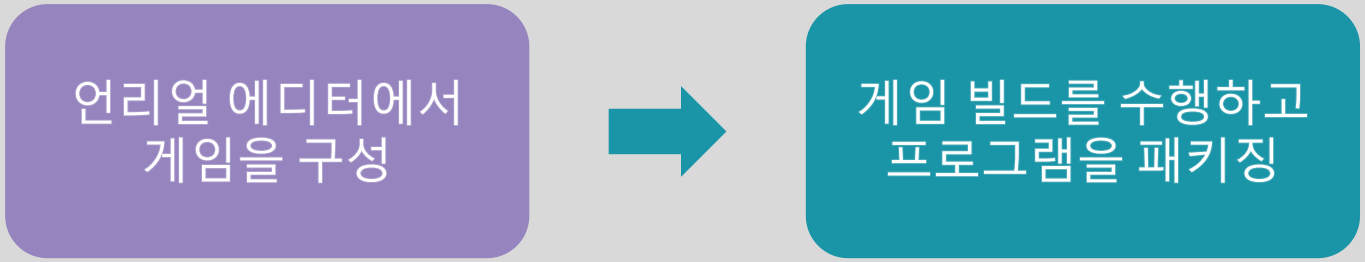
# 강의 목표

- 언리얼 엔진의 프로젝트 구성과 에디터 동작 방식의 이해
- 언리얼 엔진의 모듈 시스템을 기반으로 소스 코드를 구성하는 방법의 학습
- 언리얼 플러그인 시스템을 활용한 효과적인 모듈 구성의 학습

# 언리얼 에디터 프로젝트 구성

# 언리얼 에디터 구성

- 게임 제작을 위해 에픽 게임즈가 제공하는 저작 도구
- 언리얼 엔진의 구성
  - 에디터 : 게임 제작을 위해 제공되는 응용 프로그램 ( 일반적으로 인식하는 언리얼 엔진 )
  - 게임 빌드 : EXE 파일과 리소스로 이루어진 독립적으로 동작하는 게임 클라이언트
- 언리얼 에디터의 특징
  - 게임 개발 작업을 위해 다양한 폴더와 파일 이름 규칙이 미리 설정되어 있다.
  - 정해진 규칙을 잘 파악하고 프로젝트 폴더와 파일을 설정해야 함.
- 에디터에서 기획과 개발을 완료한 후, 게임 빌드를 통해 최종 게임 빌드를 제작하도록 설정

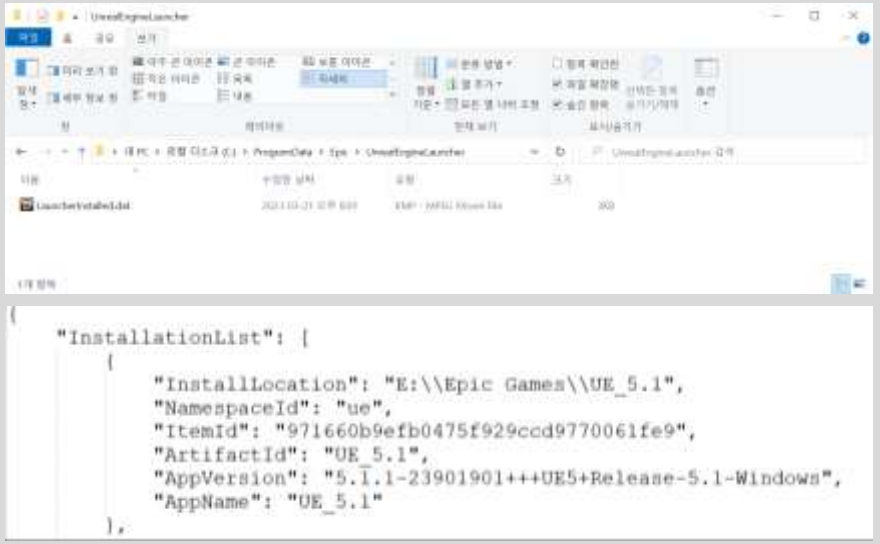


# 언리얼 에디터의 동작

- 프로젝트 폴더의 uproject 확장자를 더블클릭하면 에디터가 트리거 됨.
- 에디터의 실행 방식
  - uproject 확장자는 윈도우 레지스트리에 등록되어 있음.
    - 등록이 안되어 있다면 런처를 실행해 등록
  - UnrealVersionSelector 프로그램으로 프로젝트 정보가 넘겨짐
  - UnrealVersionSelector는 런처가 저장한 에디터 정보로부터 버전에 맞는 에디터를 실행함
- UnrealVersionSelector 소스는 에픽게임즈 GitHub에서 확인 가능
  https://github.com/EpicGames/UnrealEngine/tree/release/Engine/Source/Programs/UnrealVersionSelector

# 에디터 버전 정보의 파악

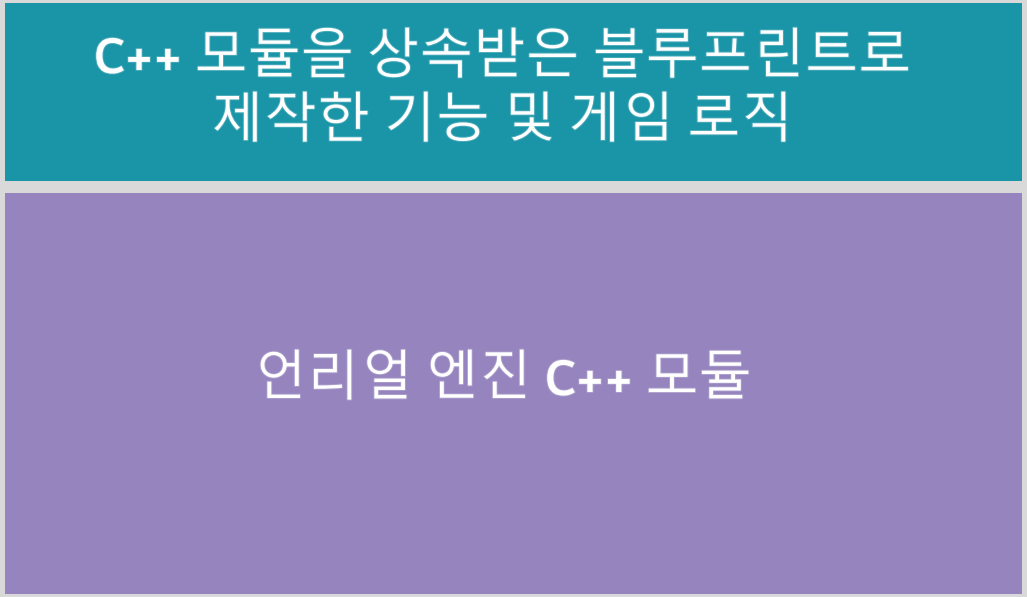
- 프로젝트.uproject 텍스트 파일에 정해져있음.
- .uproject 확장자는 에디터를 띄우기 위한 명세서 역할을 함
- 버젼 내용은 JSON 형식으로 구성되어 있음.
- 파일에 기록된 버전 정보를 바탕으로 에픽 런처가 지정한 정보를 찾아 에디터를 실행함.
  - ProgramData/Epic/UnrealLauncher 폴더에 관련 정보가 있음.
  - 이 역시 JSON 형식으로 설치된 언리얼 버전 정보가 기록되어 있음.

# 언리얼 에디터 실행 실습

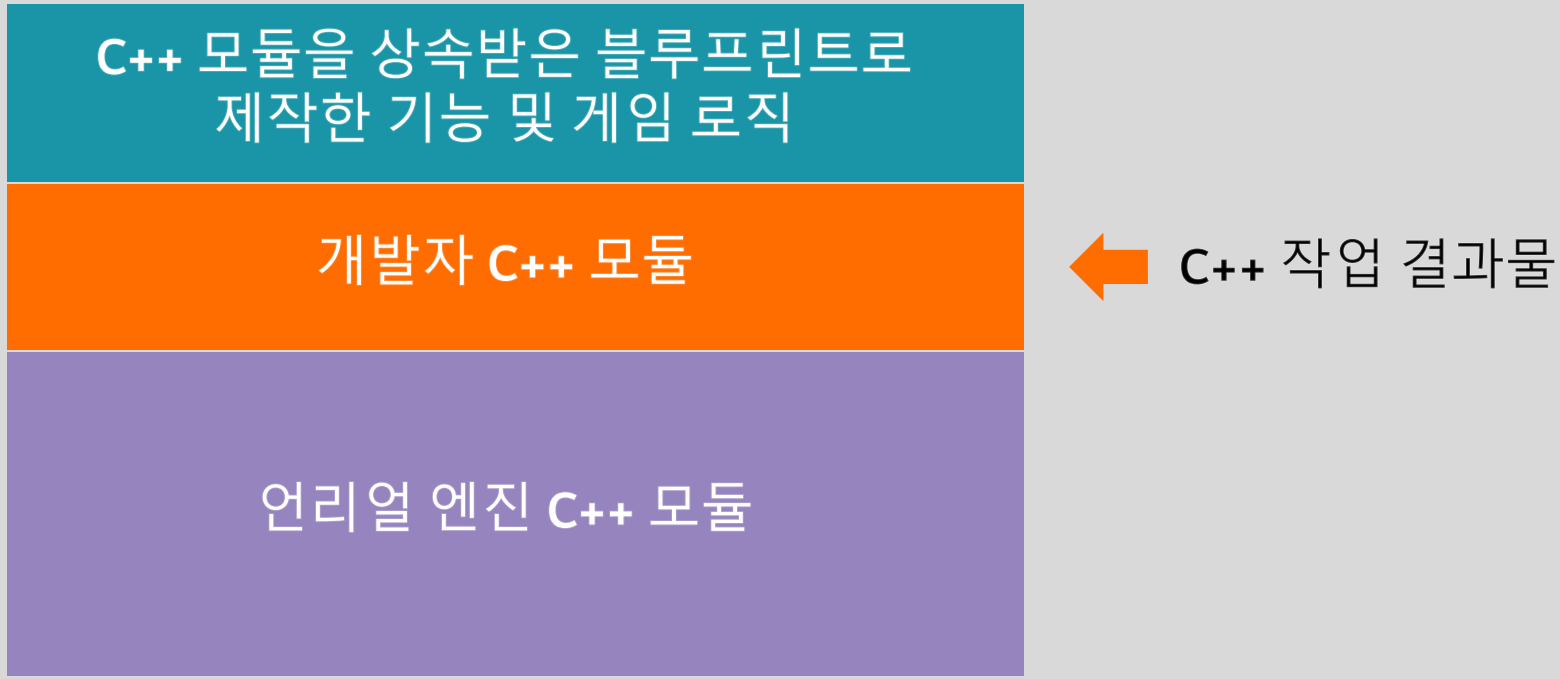
# 블루프린트 프로젝트

- C++ 코드가 없는 언리얼 프로젝트를 의미함.
- 언리얼 엔진이 제공하는 기본 기능을 활용해 게임을 제작하는 프로젝트
- 언리얼 엔진은 게임 제작에 필요한 기능을 모듈이라는 단위로 제공하고 있음.
- 언리얼 엔진의 모듈을 상속받아 블루프린트를 활용해 모든 기능과 로직을 구현하는 방식



# 언리얼 C++ 프로젝트

- 언리얼 엔진 C++ 모듈에 개발자가 추가로 자신만의 C++ 모듈을 추가할 수 있음.
- 언리얼 엔진 모듈과 개발자 모듈을 함께 사용하는 프로젝트



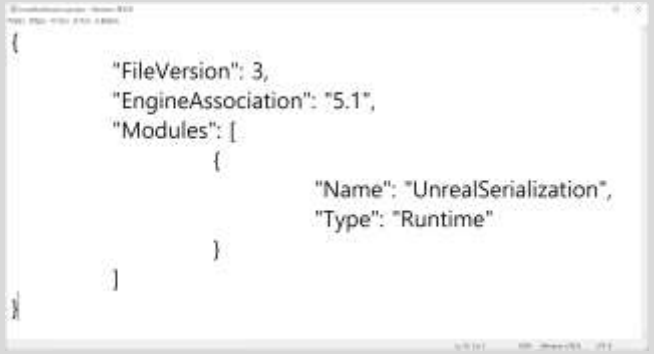
# 언리얼 C++ 모듈

- 언리얼 엔진의 소스코드는 모두 모듈(Module) 단위로 구성되어 있음.
- 모듈을 컴파일함으로서 에디터 및 게임에 우리가 제작한 로직을 공급할 수 있음.
- 모듈 단위로 구성된 C++ 소스 코드를 컴파일한 결과물
  - 에디터 용으로 DLL 동적라이브러리
  - 게임 용으로는 정적 라이브러리
- 에디터 용 모듈은 언제나 UnrealEditor-{모듈이름}.DLL 이름 규칙을 가지고 있음.


언리얼 엔진 설치 폴더에 가면 수많은 에디터 모듈이 설치된 것을 볼 수 있음

# 언리얼 C++ 모듈의 추가

- 기본 언리얼 모듈에 우리가 제작한 C++ 모듈을 추가해 에디터를 띄우고 싶은 경우.
- 우리가 만든 에디터 모듈(DLL 동적라이브러리)을 빌드 폴더에 넣어주어야 함.
  - Windows의 경우 Binaries/Win64 폴더에 해당 DLL을 넣어야 함
  - 빌드된 모듈 목록이 있는 UnrealEditor.modules 파일도 같은 폴더에 넣어주어야 인식됨.
- uproject 명세서에 모듈 이름을 지정하고 에디터를 실행.

# 언리얼 에디터 모듈 추가 실습

# 모듈 C++ 코드의 관리

- 언리얼 프로젝트가 소스 코드를 관리하는 규칙에 따라 소스 코드 구조를 구성해야 함.
- 소스 코드는 멀티 플랫폼 빌드 시스템을 지원하기 위해 특정 프로그램에 종속되어 있지 않음.
- 실제 빌드를 진행하는 주체 : Unreal Build Tool 이라는 C# 프로그램
- Source 폴더에 지정된 규칙대로 소스를 넣으면 플랫폼에 맞춰서 알아서 컴파일을 진행

# Source 폴더의 구조

- Source 폴더
  - 타겟 설정 파일
  - 모듈 폴더 ( 보통은 프로젝트 이름으로 모듈 이름을 지정 )
    - 모듈 설정 파일
    - 소스 코드 파일 ( .h 및 .cpp 파일들 )
- 타겟 설정 파일 : 전체 솔루션이 다룰 빌드 대상을 지정함.
  - {프로젝트이름}.Target.cs : 게임 빌드 설정
  - {프로젝트이름}Editor.Target.cs : 에디터 빌드 설정
- 모듈 설정 파일 : 모듈을 빌드하기 위한 C++ 프로젝트 설정 정보
  - {모듈이름}.Build.cs : 모듈을 빌드하기 위한 환경 설정

**C#**이 가진 유연한 기능**(compile on-the-fly)**을 활용해

런타임에 **cs** 파일을 읽어 빌드 환경을 구축하고 컴파일을 진행함

# 게임 프로젝트의 소스

- 내가 만든 소스가 게임 프로젝트의 C++ 모듈이 되기 위해 필요한 것.
- 모듈(Module)을 구현한 선언한 헤더와 소스 파일이 있어야 함.
  - 주로 {모듈이름}.h와 {모듈이름}.cpp로 지정함
- 모듈의 뼈대를 제작
  - 매크로를 통해 기본 뼈대 구조를 제작
    - IMPLEMENT_MODULE : 일반 모듈
    - IMPLEMENT_GAME_MODULE : 게임 모듈
    - IMPLEMENT_PRIMARY_GAME_MODULE : 주 게임 모듈
- 일반적으로 게임 프로젝트는 주 게임 모듈을 하나 선언해야 함.

모든 준비가 완료되면 **Generate Visual Studio project files.** 메뉴를 선택

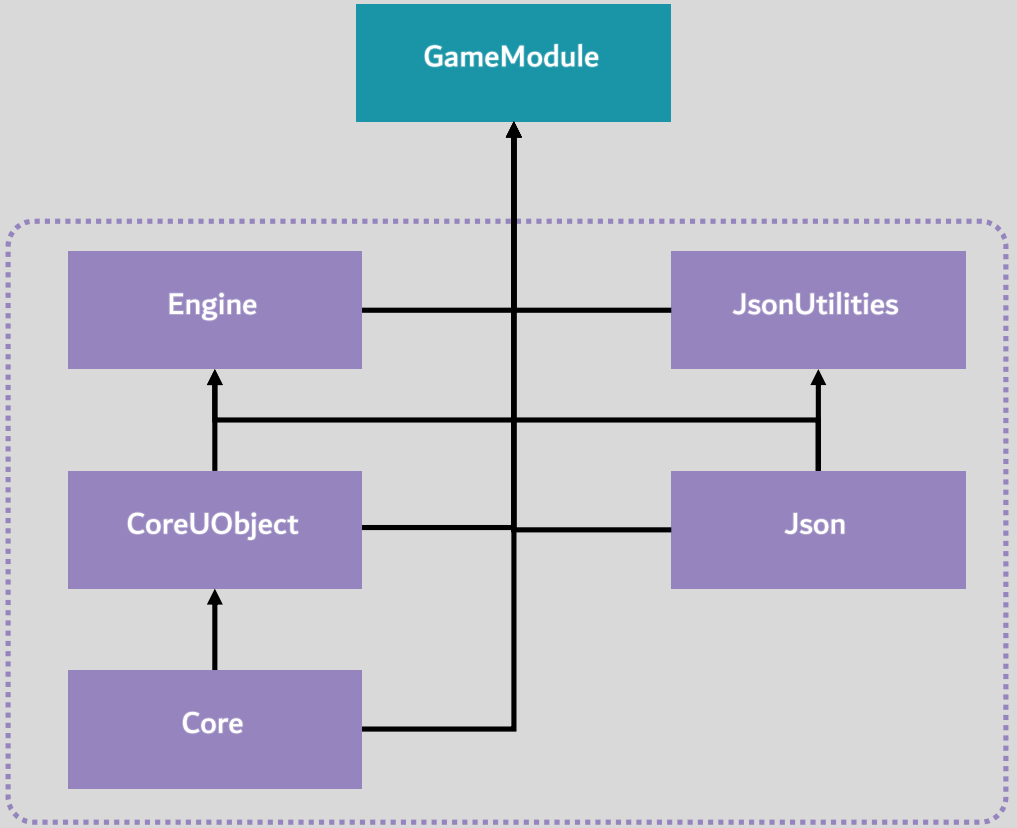
**Intermediate** 폴더에 프로젝트 관련 파일이 자동으로 생성됨.

**Source** 폴더를 규칙에 맞게 구성하면 **Intermediate** 폴더는 언제든지 재생성이 가능함

# 언리얼 프로젝트 생성 실습

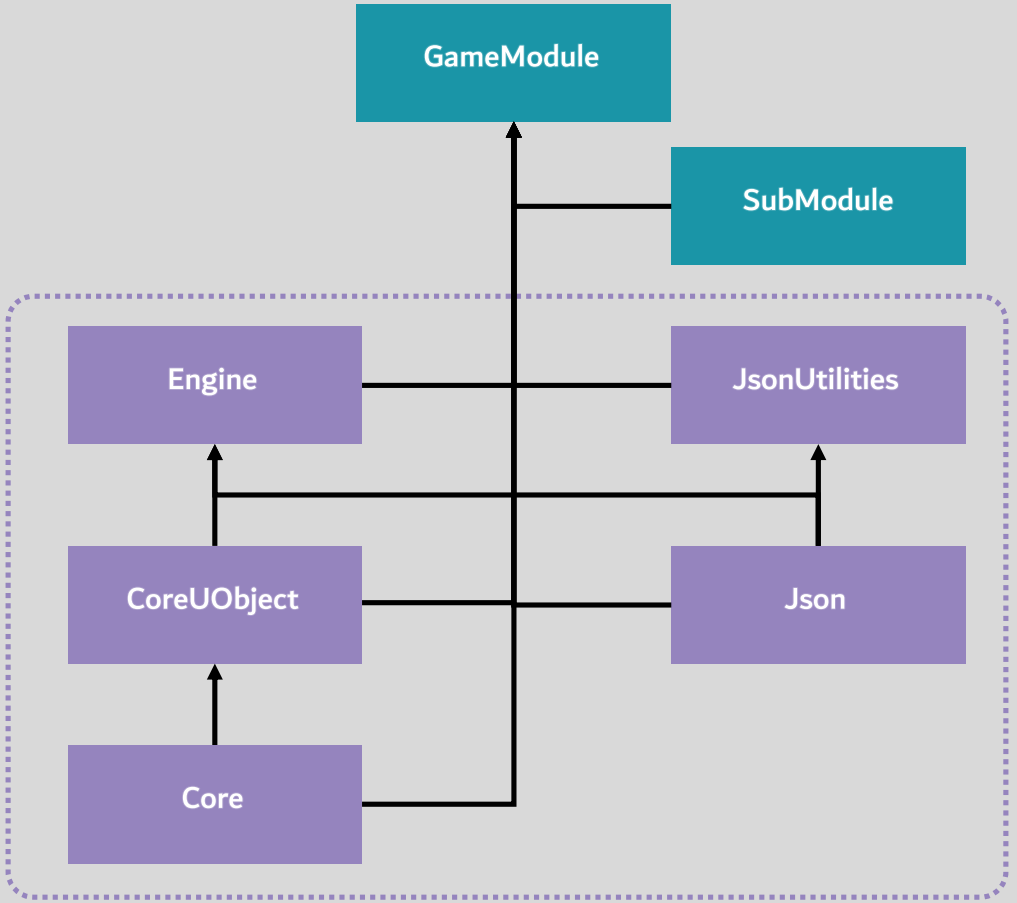
# 모듈간의 종속 관계

- 모듈 사이에 종속 관계를 설정해 다양한 기능을 구현할 수 있다.
- 우리가 만드는 게임 모듈도 언리얼 엔진이 만든 모듈을 활용해야 한다.
- 언리얼 엔진이 제공하는 모듈 사이에도 종속 관계가 있음.



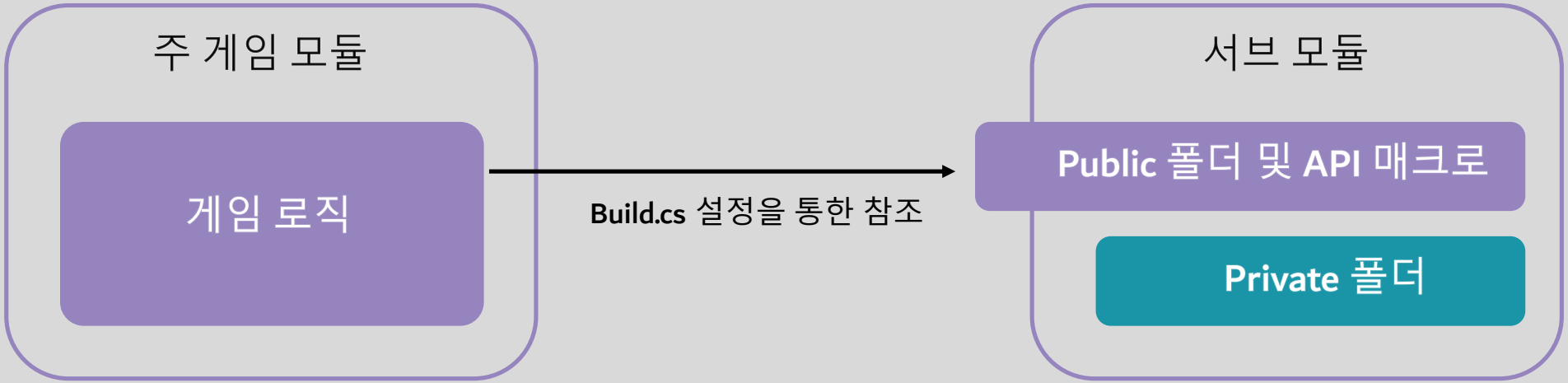
# 새로운 모듈의 추가

- 하나의 모듈에 너무 많은 코드가 들어가면 언리얼 엔진은 빌드 방식을 변경함.
- 그렇기에 프로젝트가 커질 수록 모듈을 나누어서 관리하는 것이 유리.



# 모듈의 공개와 참조

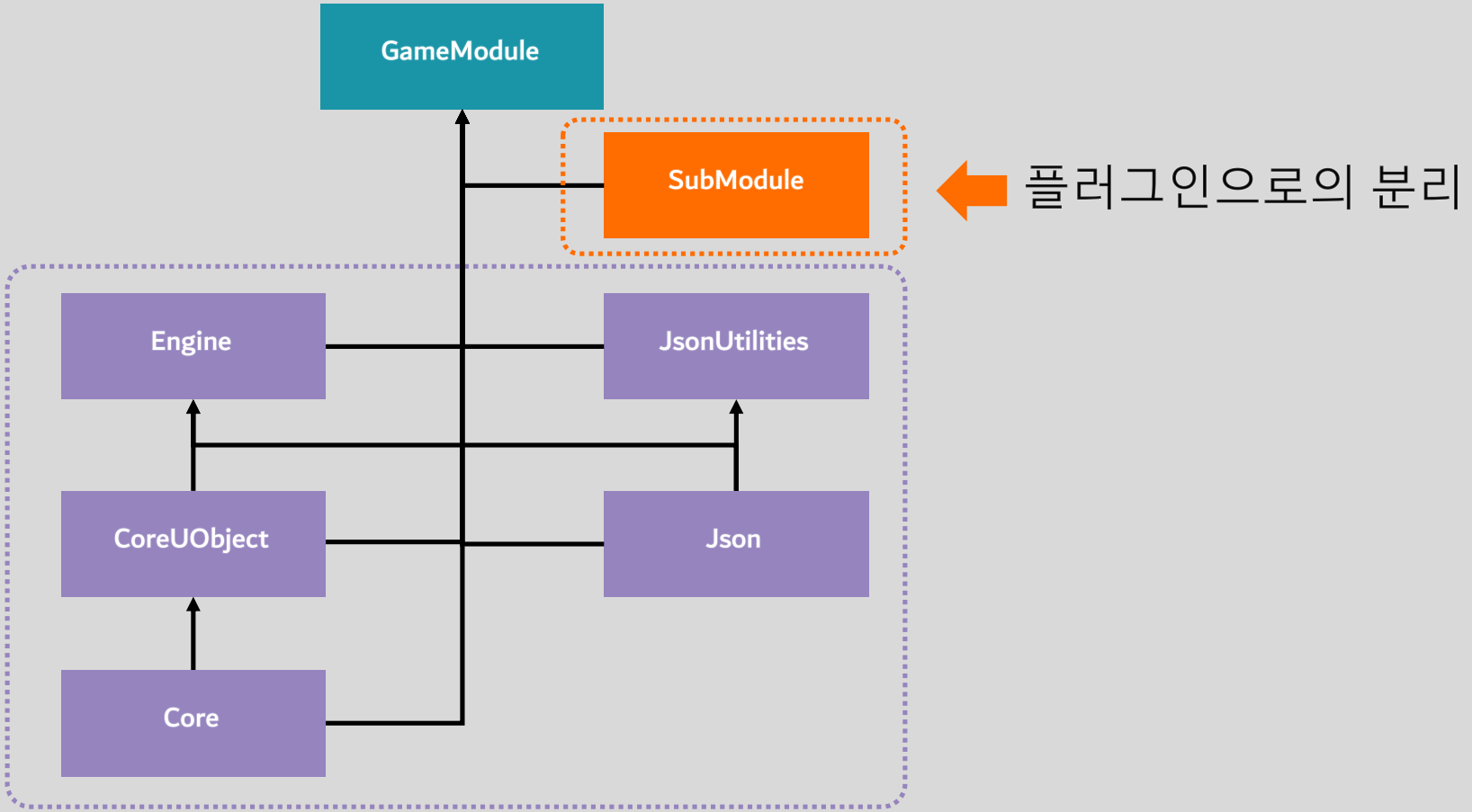
- 모듈 내 소스를 필요한 만큼만 공개해야  모듈 간 의존성을 줄이고 컴파일 타임을 최소화 할 수 있음.
- 공개할 파일은 모두 Public 폴더로
  - 예외) 예전 언리얼 엔진은 Classes 폴더가 있어 Public 폴더 역할을 하면서 언리얼 오브젝트를 관리했음
- 숨길 파일은 모두 Private 폴더로
- 외부로 공개할 클래스 선언에는 {모듈이름}_DLL 매크로를 붙일 것.
- 게임 모듈에서는 Build.cs 설정을 통해 참조할 모듈을 지정할 수 있음.



# 언리얼 소스 코드의 모듈 둘러보기

# 플러그인 시스템

- 게임 프로젝트 소스에 모듈을 추가하는 방법은 분업이 어렵다는 단점이 있음.
- 모듈만 독립적으로 동작하는 플러그인 구조를 만들어 분업화하는 것이 바람직함



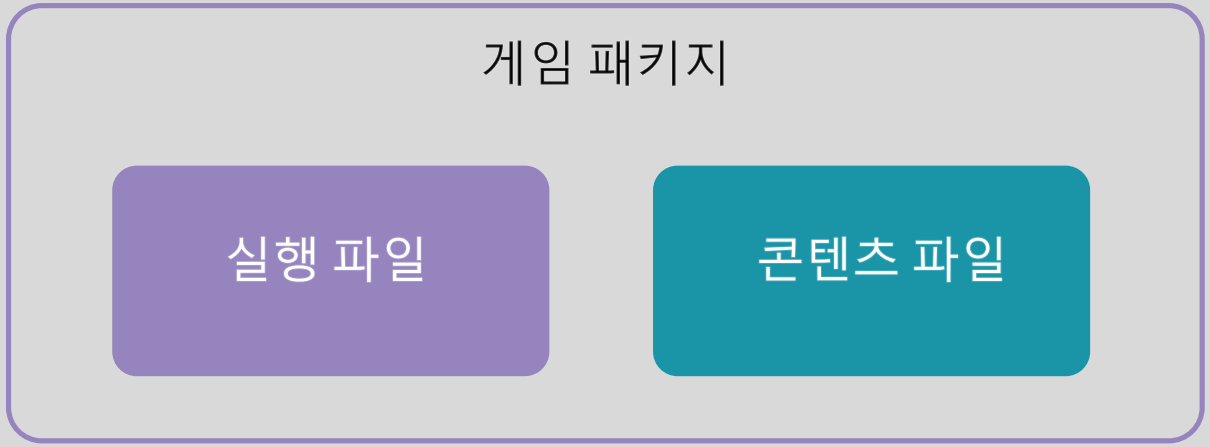
# 플러그인 구조

- 플러그인은 다수의 모듈과 게임 콘텐츠를 포함하는 포장 단위
- 에디터 설정을 통해 유연하게 플러그인을 추가하거나 삭제할 수 있음
- 플러그인 구조
  - 플러그인 명세서 ( uplugin 파일 )
  - 플러그인 리소스 ( Resource  폴더, 에디터 메뉴용 아이콘 )
  - 콘텐츠
  - 모듈 폴더
- 이러한 플러그인은 마켓 플레이스 판매로도 이어질 수 있도록 여러 설정을 추가할 수 있음.

# 언리얼 플러그인 모듈 실습

# 게임 빌드

- 게임 타겟 설정을 추가하면 게임 빌드 옵션이 추가됨.
- 게임 타겟으로 빌드된 모듈은 정적 라이브러리로 실행 파일에 포함됨.
- 게임이 실행되기 위해서는 실행 파일과 콘텐츠 애셋이 함께 있어야 함.
- 빌드 : 실행 파일을 생성하기 위한 컴파일
- 쿠킹 : 지정한 플랫폼에 맞춰 콘텐츠 애셋을 변환하는 작업
- 패키징 : 이들을 모두 모아서 하나의 프로그램으로 만드는 작업



# 정리

# 언리얼 빌드 시스템

1. uproject 명세서를 사용한 언리얼 에디터 동작 원리
2. 언리얼 엔진의 모듈 시스템과 소스 코드 관리 방법
3. 모듈 작업 분리를 위한 플러그인 시스템
4. 언리얼 소스코드의 구조
5. 게임 빌드의 설정과 게임 패키징 과정

# Part1 : 언리얼 C++의 이해

1. 헬로 언리얼!
2. 언리얼 C++ 코딩 규칙
3. 언리얼 C++ 기본 타입과 문자열
4. 언리얼 오브젝트 소개
5. 언리얼 오브젝트 리플렉션 시스템 I
6. 언리얼 오브젝트 리플렉션 시스템 II
7. 언리얼 C++ 설계 I – 인터페이스
8. 언리얼 C++ 설계 II – 컴포지션
9. 언리얼 C++ 설계 III – 델리게이트
10. 언리얼 컨테이너 라이브러리 I – Array와 Set
11. 언리얼 컨테이너 라이브러리 II – 구조체와 Map
12. 언리얼 엔진의 메모리 관리
13. 언리얼 오브젝트 관리 I – 직렬화
14. 언리얼 오브젝트 관리 II – 패키지
15. 언리얼 빌드 시스템