# 언리얼 오브젝트 소개

## (Unreal Object Introduction)

# 강의 내용

## 언리얼 오브젝트의 소개와 선언 방법

# 강의 목표

- 게임 프로그래밍이 가지는 특수성과 언리얼 오브젝트의 필요성의 이해
- 언리얼 오브젝트의 선언과 엔진 내부 컴파일 과정의 학습

# 언리얼 오브젝트 소개

# 게임 프로그래밍의 특수성

- 사용자 : 쾌적한 경험을 위해 단일 컴퓨터에서 최대 성능을 뽑아 내야 한다.
- 개발자 : 게임의 규모가 커질수록 방대하고 복잡한 기능을 안정적으로 관리해야 한다.

**Native 접근**
**C++ 언어**
**( 안정성보다 성능중시)**

메모리를 직접 제어
Cache의 활용 극대화
저수준 API의 직접 호출
복사 작업의 최소화

하이레벨
**OOP 언어**
**( 성능보다 안정성중시 )**

유지보수성 향상
크래시로부터 보호
자동 메모리 관리
고질적 실수 예방

# C++ 언어의 단점

- 1970년대에 개발된 C++ 언어
  - 객체 지향 프로그래밍의 선두 주자
  - 지속적으로 개선해왔지만, 익혀야 할 내용이 많아 초급자가 학습하기 어려움
  - 하드웨어에 직접 접근하기 때문에, 잘못 사용하면 프로그램에 큰 영향을 미침
- 1990년 중반이후 C++의 단점을 보완한 후발 언어의 등장 ( Java , C# )
  - C++의 불필요한 기능을 걷어내고, 최대한 명확하고, 간결하게 설계
  - 성능보다 안정성과 생산성을 중요시
  - 하드웨어에 직접 접근하지 않고, 가상 머신을 통해 간접적으로 접근.

As the definition of C# evolved, the goals used in its design were as follows:

- C# is intended to be a simple, modern, general-purpose, object-oriented programming language.
- The language, and implementations thereof, should provide support for software engineering principles such as strong type checking, array bounds checking, detection of attempts to use uninitialized variables, and automatic garbage collection. Software robustness, durability, and programmer productivity are important.

출처 : https://www.ecma-international.org/publications-and-standards/standards/ecma-334/

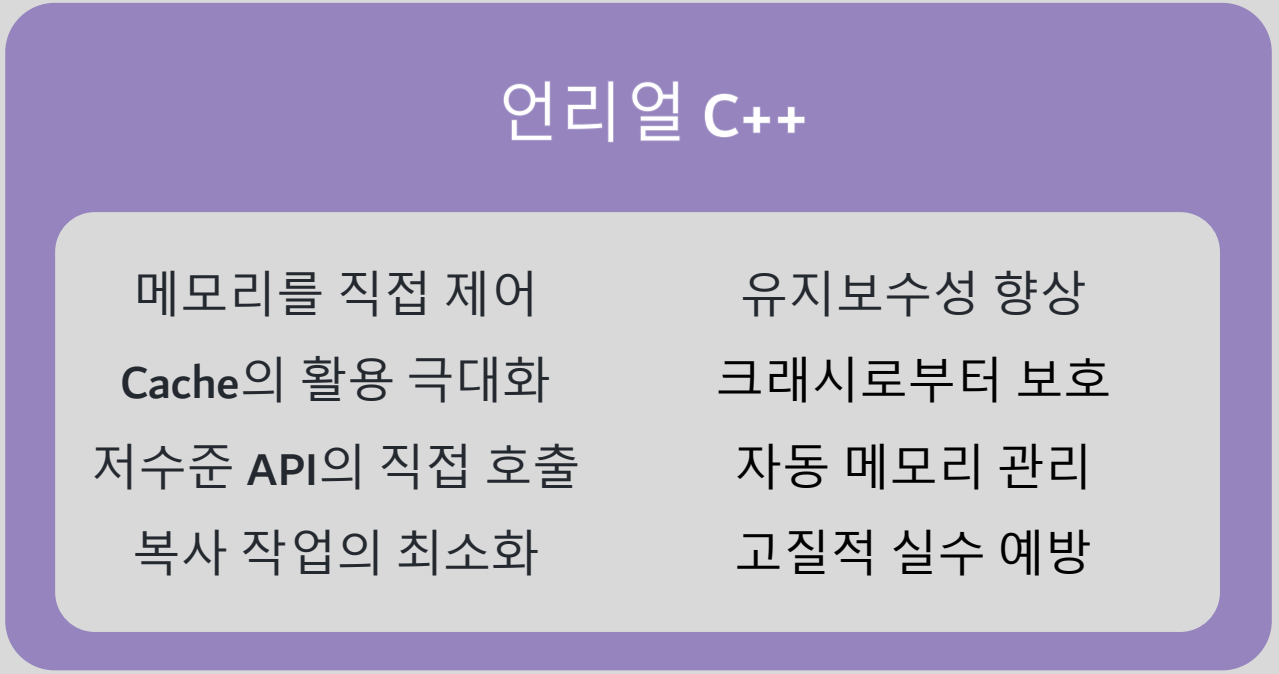
# 모던 객체 지향 설계 원칙

- 디자인 패턴을 필두로 안정적인 설계 방법이 연구됨
- 현재 시점에서 모던(Modern)하다는 뜻은 아님.
- 유지보수와 유연함, 확장성 향상을 위한 객체 지향 프로그래밍 원칙 ( SOLID )
  - Single responsibility principle : 하나의 클래스는 하나의 책임만 가져야 한다.
  - Open/closed principle : 클래스 설계를 변경하지 않고 동작을 확장할 수 있어야 한다.
  - Liskov substitution principle : 자식 클래스는 부모 클래스를 대체 사용할 수 있어야 한다.
  - Interface segregation principle : 작고 명확한 인터페이스들로 분리해 관리해야 한다.
  - Dependency inversion principle : 구현을 배제시킨 상위 정책을 바라보며 설계해야 한다.
- 후발 언어(C#, Java)등이 보완한 새로운 기능
  - 인터페이스(Interface) : 객체 설계의 틀을 제공하는 추상 클래스
  - 리플렉션(Reflection) : 런타임에서 객체의 구조를 파악하고 객체에 메타데이터를 부여
  - 델리게이트(Delegate) : 프로그램에서 발생한 이벤트를 다수의 객체에 효과적으로 전달하는데 활용

게임 규모가 대형화되면서 모던 객체 지향 설계 도입이 필요해짐

# 언리얼 엔진의 선택

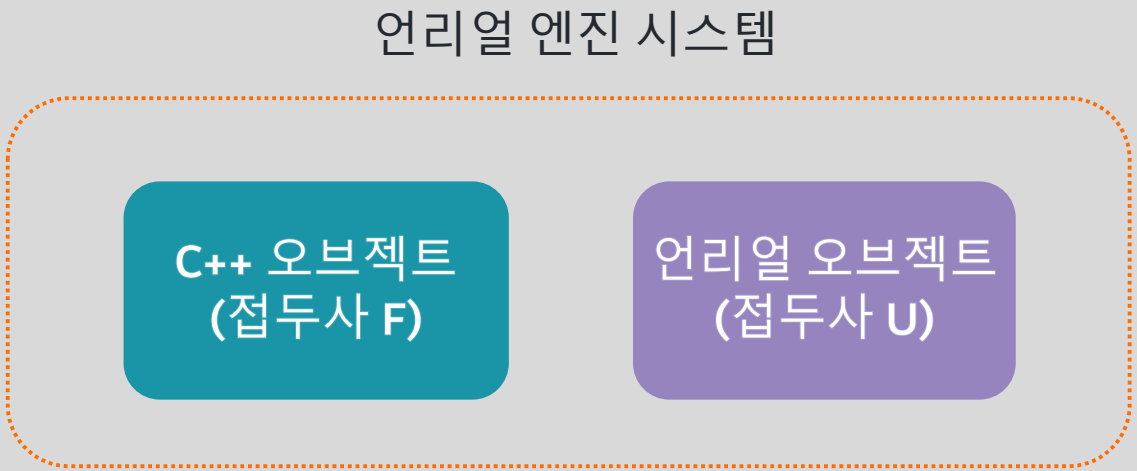
- 성능을 위해 기존 C++ 언어를 포기할 수 없음.
- 기존 C++ 언어를 확장해 모던 객체 지향 설계를 가능하도록 만듬.
- 모던 객체 지향 설계를 위한 새로운 시스템을 구축




하지만 C++ 학습도 어려운데 언리얼 C++까지 추가로 배워야 한다.

# 언리얼 오브젝트

- 언리얼 엔진이 설계한 새로운 시스템의 단위 오브젝트(객체)
  - 기존 C++ 오브젝트에 모던 객체 지향 설계를 위한 다양한 기능의 추가한 오브젝트
  - 일반 C++ 오브젝트와 언리얼 오브젝트의 두 객체를 모두 사용할 수 있음.
  - 구분을 위해 일반 C++ 오브젝트는 F, 언리얼 오브젝트는 접두사 U를 사용함.
- 각 오브젝트의 사용 용도
  - C++ 오브젝트 : 저수준의 빠른 처리를 위한 기능 구현에 사용.
  - 언리얼 오브젝트 : 콘텐츠 제작에 관련된 복잡한 설계 구현에 사용.



# 언리얼 오브젝트

**https://bit.ly/ueobjectkr**

https://docs.unrealengine.com/5.1/ko/objects-in-unreal-engine/

# 언리얼 오브젝트가 가지는 특징

- 클래스 기본 객체(CDO) : 클래스의 기본 값과 타입 정보의 제공
- 리플렉션(Reflection) : 런타임에서 클래스 정보의 참조 기능
- 인터페이스(Interface) : 모던 객체 지향 언어가 제공하는 인터페이스의 제공
- 향상된 열거형 : 보다 향상된 열거형의 지원
- 델리게이트(Deligate) : 객체간의 결합을 낮출 수 있는 델리게이트 기능의 제공
- 가비지컬렉션(Garbage Collection) : 자동 메모리 관리
- 향상된 구조체(Struct) : 리플렉션이 가능한 구조체의 지원
- 직렬화(Serialization) : 객체 정보를 바이트 스트림으로 저장, 전송, 불러들이는 기능

이후 강의에서 공부할 내용들

# 언리얼 오브젝트의 선언

# 정리

# 언리얼 오브젝트의 이해

- 게임이 대형화되면서 성능과 유지보수 두 가지가 모두 중요해짐.
- 언리얼 엔진은 C++ 언어를 확장한 언리얼 오브젝트라는 객체 구조를 고안함.
- 지정된 매크로를 사용해 빌드를 수행하면, 추가 코드가 자동으로 만들어지는 구조를 가짐.
- 언리얼 오브젝트를 사용해 대규모 게임 제작을 안정적으로 설계하고 구현할 수 있음.