언리얼C++ 설계 III - 델리게이트

(Unreal C++ Design III - Delegate)

강의 내용

언리얼 델리게이트를 사용해
클래스 간의 느슨한 결합을 구현하기

강의 목표

- 느슨한 결합의 장점과 이를 편리하게 구현하도록 도와주는 델리게이트의 이해
- 발행 구독 디자인 패턴의 이해
- 언리얼 델리게이트를 활용한 느슨한 결합의 설계와 구현의 학습

느슨한 결합(Loose Coupling)

강한 결합과 느슨한 결합

- 강한 결합(Tight Coupling)
  - 클래스들이 서로 의존성을 가지는 경우를 의미한다.
  - 아래 예시에서 Card가 없는 경우 Person이 만들어질 수 없다.
  - 이 때 Person은 Card에 대한 의존성을 가진다고 한다.
  - 핸드폰에서도 인증할 수 있는 새로운 카드가 도입된다면?

- 느슨한 결합(Loose Coupling)
  - 실물에 의존하지 말고 추상적 설계에 의존하라. (DIP 원칙)
  - 왜 Person은 Card가 필요한가? 출입을 확인해야 하기 때문
  - 출입에 관련된 추상적인 설계에 의존하자.
  - ICheck를 상속받은 새로운 카드 인터페이스를 선언해 해결
  - 이러한 느슨한 결합 구조는 유지 보수를 손쉽게 만들어줌.

```
class ICheck
{
public:
  virtual bool check() = 0;
};

class Card : public ICheck
{
public:
  Card(int InId) : Id(InId) {}

  virtual bool check() { return true; }

private:
  int Id = 0;
};

class Person
{
public:
  Person(ICheck* InCheck) : Check(InCheck) {}

protected:
  ICheck* Check;
};
```

느슨한 결합의 간편한 구현 - 델리게이트(Delegate)

- 그렇다면 함수를 오브젝트처럼 관리하면 어떨까?
- 함수를 다루는 방법
  - 함수 포인터를 활용한 콜백(callback) 함수의 구현
  - 가능은 하나 이를 정의하고 사용하는 과정이 꽤나 복잡함.
  - 안정성을 스스로 검증해주어야 함
  - C++ 17 구약의 std::bind와 std::function 활용은 느림
- C#의 델리게이트(delegate) 키워드
  - 함수를 마치 객체처럼 다룰 수 있음
  - 안정적이고 간편한 선언
- 언리얼 C++도 델리게이트를 지원함.
  - 느슨한 결합 구조를 간편하고 안정적으로 구현할 수 있음.

```cpp
class ICheck
{
public:
  virtual bool check() = 0;
};

class Card : public ICheck
{
public:
  Card(int InId) : Id(InId) {}

  virtual bool check() { return true; }

private:
  int Id = 0;
};

class Person
{
public:
  Person(ICheck* InCheck) : Check(InCheck) {}

protected:
  ICheck* Check;
};
```

```cpp
public class Card
{
    public int Id;
    public bool CardCheck() { return true; }
}

public delegate bool CheckDelegate();
public class Person
{
    Person(CheckDelegate InCheckDelegate)
    {
        this.Check = InCheckDelegate;
    }

    public CheckDelegate Check;
}
```

언리얼 델리게이트

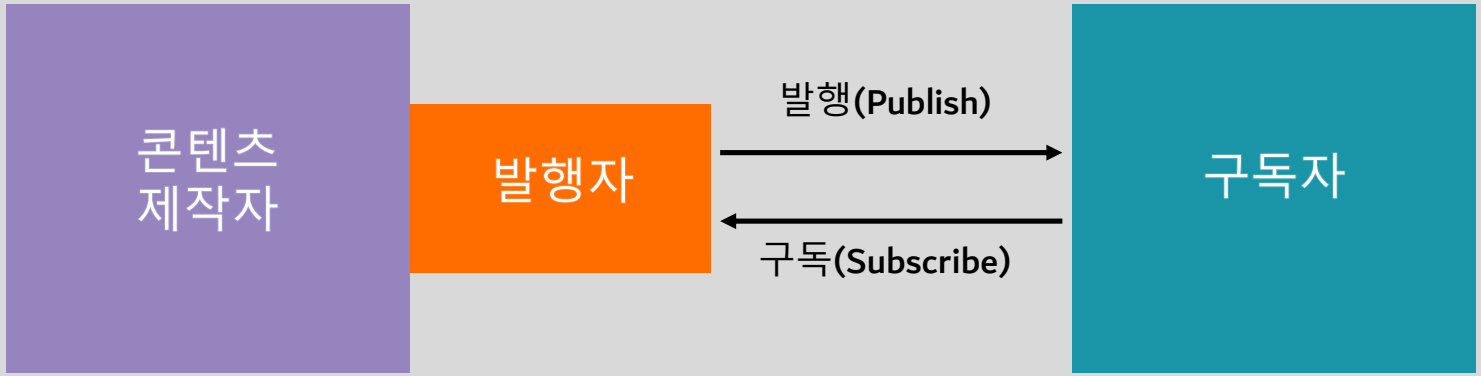
https://bit.ly/uedelegatekr

https://docs.unrealengine.com/5.1/ko/delegates-and-lamba-functions-in-unreal-engine/

발행 구독 디자인 패턴

발행 구독 디자인 패턴

- 푸시(Push)형태의 알림(Notification)을 구현하는데 적합한 디자인 패턴
- 발행자(Publisher)와 구독자(Subscriber)로 구분된다.
  - 콘텐츠 제작자는 콘텐츠를 생산한다.
  - 발행자는 콘텐츠를 배포한다.
  - 구독자는 배포된 콘텐츠를 받아 소비한다.
  - 제작자와 구독자가 서로를 몰라도, 발행자를 통해 콘텐츠를 생산하고 전달할 수 있다. (느슨한 결합)
- 발행 구독 디자인 패턴의 장점
  - 제작자와 구독자는 서로를 모르기 때문에 느슨한 결합으로 구성된다.
  - 유지 보수(Maintenance)가 쉽고, 유연하게 활용될 수 있으며(Flexibility), 테스트가 쉬워진다.
  - 시스템 스케일을 유연하게 조절할 수 있으며(Scalability), 기능 확장(Extensibility)이 용이하다

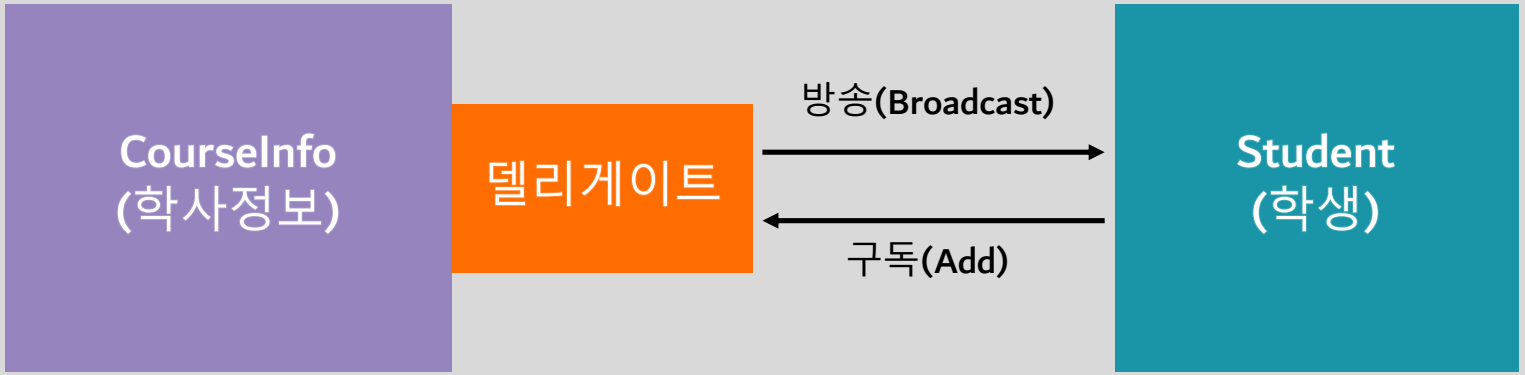


예제를 위한 클래스 다이어그램과 시나리오

- 학교에서 진행하는 온라인 수업 활동 예시
- 학사정보(CourseInfo)와 학생(Student)
  - 학교는 학사 정보를 관리한다.
  - 학사 정보가 변경되면 자동으로 학생에게 알려준다.
  - 학생은 학사 정보의 알림 구독을 해지할 수 있다.
- 시나리오
  1. 학사 정보와 3명의 학생이 있다.
  2. 시스템에서 학사 정보를 변경한다.
  3. 학사 정보가 변경되면 알림 구독한 학생들에게 변경 내용을 자동으로 전달한다.

언리얼 델리게이트(Delegate)

- 언리얼 엔진은 발행 구독 패턴 구현을 위해 델리게이트 기능을 제공함.
- 델리게이트의 사전적 의미는 대리자.
  - 학사정보의 구독과 알림을 대리해주는 객체
- 시나리오 구현을 위한 설계
  - 학사 정보는 구독과 알림을 대행할 델리게이트를 선언.
  - 학생은 학사 정보의 델리게이트를 통해 알림을 구독.
  - 학사 정보는 내용 변경시 델리게이트를 사용해 등록한 학생들에게 알림.



언리얼 델리게이트의 선언

언리얼 델리게이트 선언시 고려사항

- 델리게이트를 설계하기 위한 고려 사항
  - 어떤 데이터를 전달하고 받을 것인가? 인자의 수와 각각의 타입을 설계
    - 몇 개의 인자를 전달할 것인가?
    - 어떤 방식으로 전달할 것인가?
    - 일대일로 전달
    - 일대다로 전달
  - 프로그래밍 환경 설정
    - C++ 프로그래밍에서만 사용할 것인가?
    - UFUNCTION으로 지정된 블루프린트 함수와 사용할 것인가?
  - 어떤 함수와 연결할 것인가?
    - 클래스 외부에 설계된 C++ 함수와 연결
    - 전역에 설계된 정적 함수와 연결
    - 언리얼 오브젝트의 멤버 함수와 연결 (대부분의 경우에 이 방식을 사용)

언리얼 델리게이트 선언 매크로

DECLARE_{델리게이트유형}DELEGATE{함수정보}

- 델리게이트 유형 : 어떤 유형의 델리게이트인지 구상한다
  - 일대일 형태로 C++만 지원한다면 유형은 공란으로 둔다.
    DECLARE_DELEGATE
  - 일대다 형태로 C++만 지원한다면 MULTICAST를 선언한다.
    DECLARE_MULTICAST
  - 일대일 형태로 블루프린트를 지원한다면 DYNAMIC을 선언한다.
    DECLARE_DYNAMIC
  - 일대다 형태로 블루프린트를 지원한다면 DYNAMIC과 MULTICAST를 조합한다.
    DECLARE_DYNAMIC_MULTICAST
- 함수 정보 : 연동 될 함수 형태를 지정한다
  - 인자가 없고 반환값도 없으면 공란으로 둔다. 예) DECLARE_DELEGATE
  - 인자가 하나고 반환값이 없으면 OneParam으로 지정한다. 예) DECLARE_DELEGATE_OneParam
  - 인자가 세 개고 반환값이 있으면 RetVal_ThreeParams로 지정한다.
    예) DECLARE_DELEGATE_RetVal_ThreeParams (MULTICAST는 반환값을 지원하지 않음)
  - 최대 9개까지 지원함

언리얼 델리게이트 매크로 선정 예시

- 학사 정보가 변경되면 알림 주체와 내용을 학생에게 전달한다.
  - 두 개의 인자를 가짐
- 변경된 학사 정보는 다수 인원을 대상으로 발송한다.
  - MULTICAST를 사용
- 오직 C++ 프로그래밍에서만 사용한다.
  - DYNAMIC은 사용하지 않음

DECLARE_MULTICAST_DELEGATE_TwoParams 매크로 사용

언리얼 델리게이트의 설계

- 학사 정보 클래스와 학생 클래스의 상호 의존성을 최대한 없앤다.
  - 하나의 클래스는 하나의 작업에만 집중하도록 설계
  - 학사 정보 클래스는 델리게이트를 선언하고 알림에만 집중
  - 학생 클래스는 알림을 수신하는데만 집중
  - 직원도 알림을 받을 수 있도록 유연하게 설계
  - 학사 정보와 학생은 서로 헤더를 참조하지 않도록 신경쓸 것.
- 이를 위해 발행과 구독을 컨트롤하는 주체를 설정
  - 학사 정보에서 선언한 델리게이트를 중심으로 구독과 알림을 컨트롤하는 주체 설정

정리

언리얼 C++ 델리게이트

1. 느슨한 결합(Loose Coupling)이 가지는 장점
   - 향후 시스템 변경 사항에 대해 손쉽게 대처할 수 있음.
2. 느슨한 결합(Loose Coupling)으로 구현된 발행 구독 모델의 장점
   - 클래스는 자신이 해야 할 작업에만 집중할 수 있음.
   - 외부에서 발생한 변경 사항에 대해 영향받지 않음.
   - 자신의 기능을 확장하더라도 다른 모듈에 영향을 주지 않음.
3. 언리얼 C++의 델리게이트의 선언 방법과 활용
   - 몇 개의 인자를 가지는가?
   - 어떤 방식으로 동작하는가? (MULTICAST 사용 유무 결정)
   - 언리얼 에디터와 함께 연동할 것인가? (DYNAMIC 사용 유무 결정)
   - 이를 조합해 적합한 매크로 선택

데이터 기반의 디자인 패턴을 설계할 때 유용하게 사용