


# 1단계\_09. C++설계 #3 델리게이트

📅 수강일	@2023/04/02
➦ 이름	 전영재
🔍 멘토	Min-Kang Song, 현웅 최
👥 멘티	
✨ 작성 상태	Not started
📁 단계	
☑ 강의 시청 여부	<input checked="" type="checkbox"/>
☑ 이수 여부	<input type="checkbox"/>

## Contents



C++설계 #3 델리게이트

[0n. C++설계 #3 델리게이트 강의 과제](#)

아직 노트 작성을 시작하지 않았으며, 그저 강의를 듣고 의식의 흐름대로 적은것들입니다.

## C++설계 #3 델리게이트



### 강의 목표

- 느슨한 결합의 장점과 이를 편리하게 구현하도록 도와주는 델리게이트의 이해
- 발행-구독 디자인 패턴의 이해
- 언리얼 델리게이트를 활용한 클래스 간의 느슨한 결합 설계와 구현 학습

### 느슨한 결합?

강한결합 = 클래스들이 서로 의존성을 가지는 경우.

느슨한 결합 = 실물에 의존하지 않고 추상적 설계에 의존

나중에 Person이 소유한 Card의 내용을 바꾸려면 ICard만 건들면 되는게 인터페이스의 장점이었다.

그런데 함수 자체를 오브젝트처럼 관리한다면 어떨까? = 델리게이트

### 델리게이트 선언방법

### 델리게이트 변종매크로.

1대1 대응이나, 1대다냐, 블루프린트랑 연동할것이나 와 같은 추가옵션을 지정가능

```
DECLARE_MULTICAST_DELEGATE...
DECLARE_DYNAMIC_DELEGATE...
DECLARE_DYNAMIC_MULTICAST_DELEGATE...
```

```
DECLARE_{델리게이트 유형}DELE
```

### 델리게이트 바인딩하기

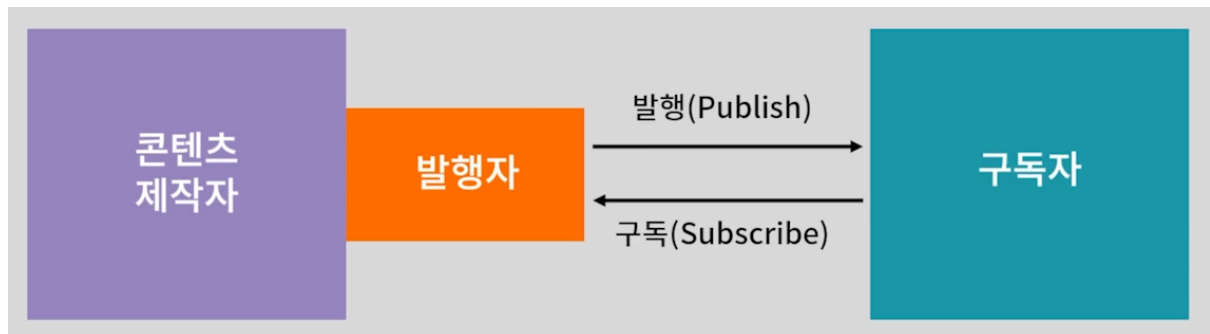
```
Bind()
```

```
BindStatic()
```

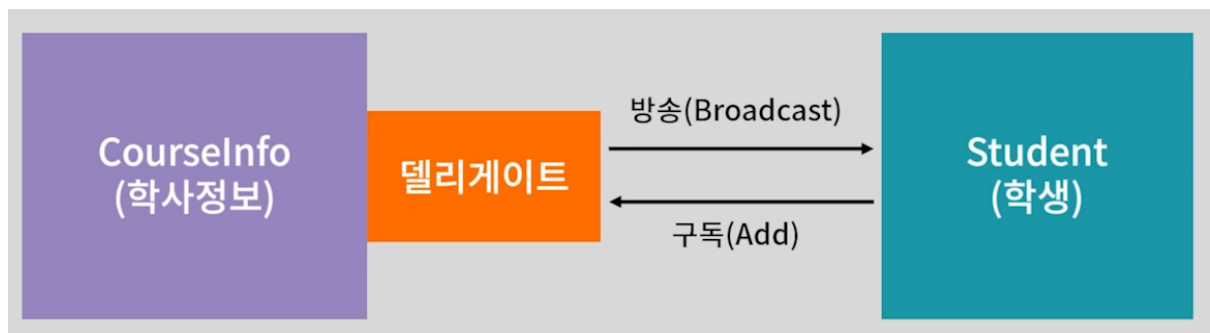
```
BindUObject() 이런거. 나중에 document보고 다시 정리하자
```

//다큐먼트의 사용예제 너무 이해 안되게 써놨다.

### 발행-구독 디자인 패턴



컨텐츠 제작자는 컨텐츠 제작에만, 구독자는 컨텐츠를 사용하는데에만 집중할 수 있다.



사용할 예제의 구조

### 델리게이트 선언시 고려사항

- 어떤 데이터를 전달하고 받을 것인가? 인자의 수와 타입을 설계
- 프로그래밍 환경설정
  - c++에만 쓸건지, 블루프린트에도 쓸건지.
- 어떤 함수와 연결할 것인지.

## 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

이부분 정리하면 좋겠다.

```
CourseInfo.h

DECLARE_MULTICAST_DELEGATE_TwoParams(FCourseInfoOnChangedSignature, const FString&, const FString&)
UCLASS()
class ...API ...
{
Public:

    FCourseInfoOnChangedSignature OnChanged;

    void ChangeCourseInfo(const FString& InSchoolName, const FString& InNewContents);
}
```

```
CourseInfo.h

void UCourseInfo::ChangeCourseInfo(const FString& InSchoolName, const FString& InNewContents)
[
    UE_LOG(LogTemp, Log, TEXT("[CourseInfo] 학사정보가 변경되어 알림을 발송합니다."));
    Contents = InNewContents;
    OnChanged.Broadcast(InSchoolName, Contents); //OnChanged에 연결된 모든 함수들에게 방송
]
```

```
Student.h
UCLASS()
{
...
public:
    void GetNotification(const FString& School, const FString& NewCourseInfo)
}
```

```
Student.cpp

void UStudent::GetNotification(const FString& School, const FString& NewCourseInfo)
{
    UE_LOG(LogTemp, Log, TEXT("[Student] %s로부터 메시지 수신: %s"), *School, *NewCourseInfo);
}
```

```
MyGameInstance.h
UCLASS()
{
    ...
private:
    UPROPERTY()
    TObjectPtr<class UCourseInfo> CourseInfo;
}
```

```
MyGameInstance.cpp
#include "CourseInfo.h"
...
//사실 CDO에 생성해도 되지만, 지금은 필요시에 호출함을 보일수 있게 Init에다 구현한다.
void UMyGameInstance::Init()
{
    Super::Init();
    CourseInfo = NewObject<UCourseInfo>(this);
    //CourseInfo의 아우터를 MyGameInstance로 지정해줌으로써, 컴포지션 관계를 만든다
    //(이렇게 해야 메모리에 안전하게 남길수있는듯하다?)

    UStudent* Student1 = NewObject<UStudent>();
    UStudent* Student2 = NewObject<UStudent>();
    // 애는 어찌피 Init()내에서 실행되고 사라질 것이기에 아우터 설정안해줌
    CourseInfo->OnChanged.AddUObject(Student1, &UStudent::GetNotification) //CourseInfo의 OnChanged라는 함수에 Student객체들을 연결
    CourseInfo->OnChanged.AddUObject(Student2, &UStudent::GetNotification)
    CourseInfo->ChangeCourseInfo(SchoolName, TEXT("변경된 학사 정보"));
}
```

위 코드 진행 시, 다음과 같은 결과가 나온다.

[CourseInfo] 학사정보가 변경되어 알림을 발송합니다.

[Student] 학교로부터 메시지 수신: 변경된 학사 정보 //참고로 여긴 Student2

[Student] 학교로부터 메시지 수신: 변경된 학사 정보 //여긴 Student1에서 나온 텍스트다. 바인드의 순서는 스택형식이라고 예상할 수 있다.

위 예제를 통해, CourseInfo와 Student는 서로를 include 하지 않았음에도 상호작용 할 수 있었다.

이것이 곧 느슨한 결합을 의미하는 것이다.

## Summary

- 느슨한 결합의 장점
  - 향후 변경사항에 대해 손쉽게 대처할 수 있다.
- 느슨한 결합으로 구현된 발행-구독 모델의 장점

- 클래스는 자신의 작업에만 집중할 수 있다.
- 외부의 변경사항에 대해 영향받지 않는다.
- 자신의 변경사항이 외부에 영향을 주지 않는다.
- 언리얼 C++ 델리게이트의 선언 방법과 활용
  - 몇개의 인자를 가지는지? (Param 갯수)
  - 어떤 방식으로 동작하는지? (MULTICAST 사용 유무)
  - 언리얼 에디터와 연동할 것인지? (DYNAMIC 사용 유무)
  - 위를 조합해 적합한 매크로를 선택하면 된다.

⇒ 델리게이트는 데이터 기반의 디자인 패턴을 설계할 때 유용하다.

## 0n. C++설계 #3 델리게이트 강의 과제

**?** Q1. 언리얼 델리게이트를 활용한 예제를 고안하고 직접 구현해보자.



Reference