C++ 강좌 13회 : 오버라이딩

김성엽 카페매니저 + 구독 1:1 채팅

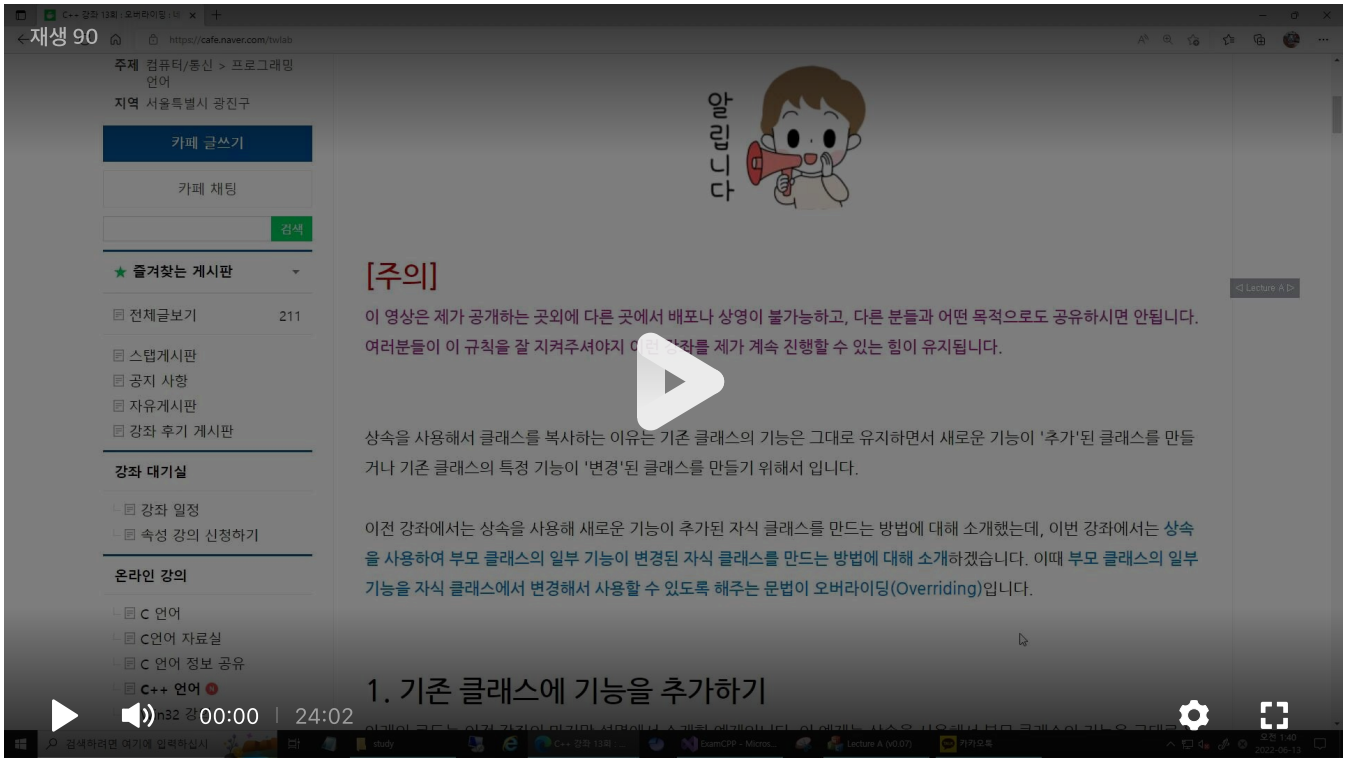
2022.06.12. 22:16 조회 241

[주의]

이 영상은 제가 공개하는 곳외에 다른 곳에서 배포나 상영이 불가능하고, 다른 분들과 어떤 목적으로도 공유하시면 안됩니다. 여러분들이 이 규칙을 잘 지켜주셔야지 이런 강좌를 제가 계속 진행할 수 있는 힘이 유지됩니다.

상속을 사용해서 클래스를 복사하는 이유는 기존 클래스의 기능은 그대로 유지하면서 새로운 기능이 '추가'된 클래스를 만들거나 기존 클래스의 특정 기능이 '변경'된 클래스를 만들기 위해서 입니다.

이전 강좌에서는 상속을 사용해 새로운 기능이 추가된 자식 클래스를 만드는 방법에 대해 소개했는데, 이번 강좌에서는 상속을 사용하여 부모 클래스의 일부 기능이 변경된 자식 클래스를 만드는 방법에 대해 소개하겠습니다. 이때 부모 클래스의 일부 기능은 자식 클래스에서 변경해서 사용할 수 있도록 해주는 문법이 오버라이딩(Overriding)입니다.



C++ 언어 13회 강좌. 오버라이딩

1. 기존 클래스에 기능은 추가하기

아래의 코드는 이전 강좌의 마지막 설명에서 소개한 예제입니다. 이 예제는 상속을 사용해서 부모 클래스의 기능은 그대로 유지하며 새로운 기능이 추가된 자식 클래스를 선언하는 방법을 소개하려고 만든 것입니다. 여기서 기능을 추가한다는 뜻은 새로운 멤버 함수를 추가한다는 뜻이기 때문에 Tipsware 클래스를 상속해서 만든 Tipssoft 클래스에 ShowMemberCount 함수를 추가한 것입니다. 이렇게 하면 Tipssoft 클래스는 Tipsware 클래스의 기능에 추가해서 SetMemberCount 함수도 사용할 수 있습니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipsware
{
protected:
    int m_member_count;  // 멤버의 수를 저장할 변수

public:
    Tipsware()  // 생성자
    {
        m_member_count = 0;
    }

    void SetMemberCount(int a_count)  // 멤버의 수를 저장할 때 사용할 함수
    {
        m_member_count = a_count;
    }

    int GetMemberCount()  // 멤버의 수를 얻을 때 사용할 함수
    {
        return m_member_count;
    }
};

// Tipsware 클래스의 내용을 Tipssoft 클래스에 그대로 복사해달라는 뜻!
class Tipssoft : public Tipsware
{
public:
    void ShowMemberCount()  // 멤버의 수를 출력할 때 사용할 함수
    {
        printf("Member Count : %d\n", m_member_count);  // 부모 클래스의 protected 멤버는 직접 사용
    }
};

int main()
{
    Tipssoft data;

    data.SetMemberCount(31);  // 객체에 멤버의 수를 지정한다.
    data.ShowMemberCount();  // 객체에 저장된 멤버의 수를 출력한다.
    return 0;
}
```

2. 기존 클래스의 일부 기능만 변경한 새로운 클래스 만듣기

위 예제에서 Tipsware 클래스의 SetMemberCount 함수에 전달된 값은 아무런 제한 없이 m_member_count 변수에 그대로 저장됩니다. 그런데 이 값이 50보다 작은 경우에만 대입하고 범위를 벗어나는 값은 무시해야 한다는 조건이 추가되었다

고 합시다.

그래서 아래와 같이 Tipsware 클래스의 SetMemberCount 함수에 조건문을 바로 추가하려고 했는데, Tipsware 클래스를 사용하는 다른 코드에서는 50보다 값이 커도 그대로 대입해야 하는 경우도 있어서 Tipsware 클래스의 코드를 그대로 유지해야 한다면 이 상황을 어떻게 해결해야 할까요?

```
void SetMemberCount(int a_count)  // 멤버의 수를 저장할 때 사용할 함수
{
    if(a_count < 50) m_member_count = a_count;
}
```

먼저 아래와 같이 SetMemberCount 함수는 그대로 두고 SetMemberCountEx 함수를 추가하여 값에 제한이 필요한 경우에만 SetMemberCountEx 함수를 사용해서 이 문제를 해결할 수 있습니다. 하지만 이런 방식은 객체를 만들어서 지속적으로 사용하는 경우 m_member_count 변수에 값을 대입할 때마다 지금 상황이 SetMemberCount 함수를 사용해야 하는 상황인지 아니면 SetMemberCountEx 함수를 사용해야 하는 상황인지를 개발자가 계속 체크해야 하는 불편함이 있습니다. 그리고 개발자의 실수로 SetMemberCountEx 함수를 호출해야 하는 상황인데 SetMemberCount 함수를 호출하면 이 문제를 해결하기가 쉽지 않을 것입니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipsware
{
protected:
    int m_member_count;  // 멤버의 수를 저장할 변수

public:
    Tipsware()  // 생성자
    {
        m_member_count = 0;
    }

    void SetMemberCount(int a_count)  // 멤버의 수를 저장할 때 사용할 함수
    {
        m_member_count = a_count;
    }

    void SetMemberCountEx(int a_count)  // 멤버의 수를 저장할 때 사용할 함수 (50 제한)
    {
        if(a_count < 50) m_member_count = a_count;
    }

    int GetMemberCount()  // 멤버의 수를 얻을 때 사용할 함수
    {
        return m_member_count;
    }
};
```

그래서 객체를 만들어서 지속적으로 사용하는 상황이라면 아래와 같이 Tipssoft 클래스를 추가로 만들어서 사용하는 것이 더 유리합니다. 왜냐하면 **개발자 입장에서는 클래스를 사용해서 객체를 만든 때만 Tipsware 클래스를 사용할 것인지 Tipssoft 클래스를 사용할 것인지를 결정**하면 되기 때문입니다. 즉, 두 객체는 동일한 이름의 SetMemberCount 함수를 사용하지만 Tipsware 클래스로 만든 객체는 전달된 값을 그대로 사용하고 Tipssoft 클래스로 만든 객체는 값이 50보다 작은 경우에만 값을 대입해서 사용할 것입니다.

```cpp
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipsware
{
protected:
    int m_member_count;  // 멤버의 수를 저장할 변수

public:
    Tipsware()  // 생성자
    {
        m_member_count = 0;
    }

    void SetMemberCount(int a_count)  // 멤버의 수를 저장할 때 사용할 함수
    {
        m_member_count = a_count;
    }

    int GetMemberCount()  // 멤버의 수를 얻을 때 사용할 함수
    {
        return m_member_count;
    }
};

class Tipssoft
{
protected:
    int m_member_count;  // 멤버의 수를 저장할 변수

public:
    Tipssoft()  // 생성자
    {
        m_member_count = 0;
    }

    void SetMemberCount(int a_count)  // 멤버의 수를 저장할 때 사용할 함수
    {
        // 50보다 큰 경우에만 대입
        if(a_count < 50) m_member_count = a_count;
    }

    int GetMemberCount()  // 멤버의 수를 얻을 때 사용할 함수
    {
        return m_member_count;
    }
};

int main()
{
    Tipsware temp;
    temp.SetMemberCount(60);  // 객체에 멤버의 수를 지정
    printf("[Tipsware] Member Count : %d\n", temp.GetMemberCount());
```

```
    Tipssoft data;
    data.SetMemberCount(60);  // 객체에 멤버의 수를 지정
    printf("[Tipssoft] Member Count : %d\n", data.GetMemberCount());
    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
[Tipsware] Member Count : 60
[Tipssoft] Member Count : 0

이 창을 닫으려면 아무 키나 누르세요...
```

3. 오버라이딩

그런데 위에서 소개한 Tipsware 클래스와 Tipssoft 클래스는 SetMemberCount 함수에 사용된 조건문 코드만 다른 뿐 나머지 코드는 거의 동일하기 때문에 명백한 중복 코드입니다. 그래서 상속을 사용해서 중복 코드를 생략하고 싶지만 두 클래스가 동일한 이름을 가진 SetMemberCount 함수를 가지고 있기 때문에 상속을 표현하는데 문제가 있습니다.

왜냐하면 아래와 같이 코드를 구성하면 자식 클래스인 Tipssoft 클래스에는 SetMemberCount 함수가 두 개 존재하게 됩니다. 즉, Tipssoft 클래스에는 Tipsware 클래스에서 상속받은 SetMemberCount 함수도 있고 Tipssoft 클래스에 새로 추가한 SetMemberCount 함수도 있다는 뜻입니다. 물론 C++ 언어에서는 함수 오버로딩이 제공되어 함수의 이름이 같아도 매개 변수의 자료형이나 개수가 다르면 문제가 되지 않지만, 이 경우에는 두 함수의 원형이 동일하여 문제가 될 것입니다.

```
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipsware
{
protected:
    int m_member_count;  // 멤버의 수를 저장할 변수

public:
    Tipsware()  // 생성자
    {
        m_member_count = 0;
    }

    void SetMemberCount(int a_count)  // 멤버의 수를 저장할 때 사용할 함수
    {
        m_member_count = a_count;
    }

    int GetMemberCount()  // 멤버의 수를 얻을 때 사용할 함수
    {
        return m_member_count;
    }
};

// Tipsware 클래스의 내용을 Tipssoft 클래스에 그대로 복사해달라는 뜻!
class Tipssoft : public Tipsware
{
public:
    void SetMemberCount(int a_count)  // 멤버의 수를 저장할 때 사용할 함수 (기능 변경)
    {
        // 50보다 큰 경우에만 대입
        if(a_count < 50) m_member_count = a_count;
    }
};

int main()
{
    Tipsware temp;
    temp.SetMemberCount(60);  // 객체에 멤버의 수를 지정
    printf("[Tipsware] Member Count : %d\n", temp.GetMemberCount());

    Tipssoft data;
    data.SetMemberCount(60);  // 객체에 멤버의 수를 지정
    printf("[Tipssoft] Member Count : %d\n", data.GetMemberCount());
    return 0;
}
```

그런데 위 코드는 오류가 발생하지 않습니다. 왜냐하면 C++ 언어는 자식 클래스에서 부모 클래스의 함수를 상속 받을 때 부모 클래스의 이름을 함께 포함해서 기록하기 때문입니다. 따라서 함수 원형까지 동일한 SetMemberCount 함수가 두 개 있다고 해도 부모 클래스에서 상속받은 SetMemberCount 함수는 내부적으로 Tipsware::SetMemberCount로 되어 있고

자식 클래스에 추가된 SetMemberCount 함수는 Tipssoft::SetMemberCount로 되어 있습니다. 그래서 두 함수가 함수 원형까지 동일해도 네임스페이스가 다르기 때문에 오류가 아니라서 위와 같이 상속 코드를 구성해도 됩니다.

그리고 Tipssoft 클래스로 만든 data 객체의 SetMemberCount 함수를 사용하면, data 객체의 네임스페이스가 Tipssoft이기 때문에 내부적으로 가지고 있는 두 개의 SetMemberCount 함수 중에 Tipssoft::SetMemberCount 함수가 호출됩니다. 따라서 Tipssoft 클래스에 새로 추가된 함수가 호출되어 50이 넘는 값은 저장되지 않습니다.

이처럼 부모 클래스에서 제공하는 함수의 기능을 변경하기 위해 자식 클래스에 부모 클래스의 함수를 함수 원형까지 동일하게 추가해서 함수의 내용은 변경하는 문법을 오버라이딩(overriding)이라고 합니다.

그리고 Tipssoft 클래스로 만든 data 객체에는 Tipsware 클래스에서 상속받는 SetMemberCount 함수도 가지고 있기 때문에 아래와 같이 Tipsware 네임스페이스를 직접 지정하면 부모 클래스의 SetMemberCount 함수를 강제로 호출할 수 도 있습니다. 즉, 아래와 같이 호출하면 60이 data 객체의 m_member_count 변수에 저장됩니다.

```
Tipssoft data;
data.Tipsware::SetMemberCount(60);  // 부모 클래스의 함수가 호출되어 60이 저장
```

그리고 오버라이딩이라는 기술은 부모 함수의 코드를 수정하는 기능이기 때문에 부모 함수의 코드 일부가 자식 함수에도 동일하게 사용됩니다. 예를 들어, Tipsware 클래스의 SetMemberCount 함수에 조건문을 추가하기 위해 Tipssoft 클래스에 SetMemberCount 함수를 재정의 했기 때문에 두 함수에 m_member_count = a_count; 코드가 중복된 것을 볼수 있습니다. 결국 오버라이딩 작업으로 만들어진 함수들은 부모 클래스가 제공하는 함수의 내용을 제거하는 경우보다, 지금처럼 부모 클래스가 제공하는 함수의 코드를 유지하면서 약간의 코드가 추가되는 경우가 더 많습니다.

이렇게 발생하는 중복코드는 아래와 같이 부모 클래스의 네임스페이스를 직접 적는 방법을 사용하여 제거할 수 있습니다. 즉, 위에서 소개한 예시에는 m_member_count = a_count; 코드가 부모 클래스와 자식 클래스 양쪽에 모두 있기 때문에 이 코드에 변경이 생기면 양쪽 코드를 모두 수정해야 합니다.

하지만 아래와 같이 코드를 작성하면 Tipsware 클래스쪽에만 코드의 원본이 남아있고 Tipssoft 클래스 쪽에서는 Tipsware 클래스에서 상속받은 함수를 사용하겠다고 되어 있기 때문에 기능이 변경되어도 Tipsware 클래스의 SetMemberCount 함수에서만 코드를 수정하면 됩니다.

```cpp
#include <stdio.h>    // printf 함수를 사용하기 위해

class Tipsware
{
protected:
    int m_member_count;  // 멤버의 수를 저장할 변수

public:
    Tipsware()  // 생성자
    {
        m_member_count = 0;
    }

    void SetMemberCount(int a_count)  // 멤버의 수를 저장할 때 사용할 함수
    {
        m_member_count = a_count;
    }

    int GetMemberCount()  // 멤버의 수를 얻을 때 사용할 함수
    {
        return m_member_count;
    }
};

// Tipsware 클래스의 내용을 Tipssoft 클래스에 그대로 복사해달라는 뜻!
class Tipssoft : public Tipsware
{
public:
    void SetMemberCount(int a_count)  // 멤버의 수를 저장할 때 사용할 함수 (기능 변경)
    {
        // 50보다 큰 경우에만 대입
        if(a_count < 50) Tipsware::SetMemberCount(a_count); // 부모 함수 재활용
    }
};

int main()
{
    Tipsware temp;
    temp.SetMemberCount(60);  // 객체에 멤버의 수를 지정
    printf("[Tipsware] Member Count : %d\n", temp.GetMemberCount());

    Tipssoft data;
    data.SetMemberCount(60);  // 객체에 멤버의 수를 지정
    printf("[Tipssoft] Member Count : %d\n", data.GetMemberCount());
    return 0;
}
```

결국 C++ 언어로 작성된 프로그램에서 **상속은 사용해 클래스를 복사하는 이유는 기존 클래스의 기능은 그대로 유지하면서 새로운 기능이 추가된 클래스를 만든거나 기존 클래스의 일부 기능이 수정된 클래스를 만든기 위해**서 입니다.

댓글 등록순 최신순

관심글 댓글 알림

Zermi 3

지금까지 상속받은 부모 클래스의 함수를 덮어쓴다라고 배웠었는데,
사실은 부모 클래스의 함수와 자식 클래스의 함수 둘 다 기억하고 네임스페이스로 구분하는 거였네요..
알고있던 내용이어서 부담 없이 들었지만
이번 강의에선 스승의 중요성을.. 뼈저리게 느낍니다!

2022.09.12. 13:26 답글쓰기

김성엽 M 작성자

ㅎㅎ 사소한 차이긴하지만 확실하게 알아두면 더 좋습니다 ㅎ 그래서 C++ 강좌는 알아도 계속 보시는것이 좋습니다~ :)

2022.09.12. 16:11 답글쓰기

티모 3

오버라이딩 이해를 잘 못하고 있었는데 부모 자식 관계에서 사용 방법 이해가 잘 되었어요!!
매번 함수 새로 만들었었는데...

2022.11.21. 16:14 답글쓰기

김성엽 M 작성자

2022.11.21. 16:27 답글쓰기

카일 3

재정의라는 개념이 어렵지 않아서 그냥 그런가보다 하고 넘어갔었는데 모르던 내용 배웠습니다.

2022.11.25. 21:47 답글쓰기

김성엽 M 작성자

오늘은 진도가 많이 나가는군요 ㅎ

2022.11.25. 21:53 답글쓰기

카일 3

김성엽 이번주에 C++ 완강 목표입니다!

2022.11.26. 12:29 답글쓰기

dh221009

댓글을 남겨보세요

등록