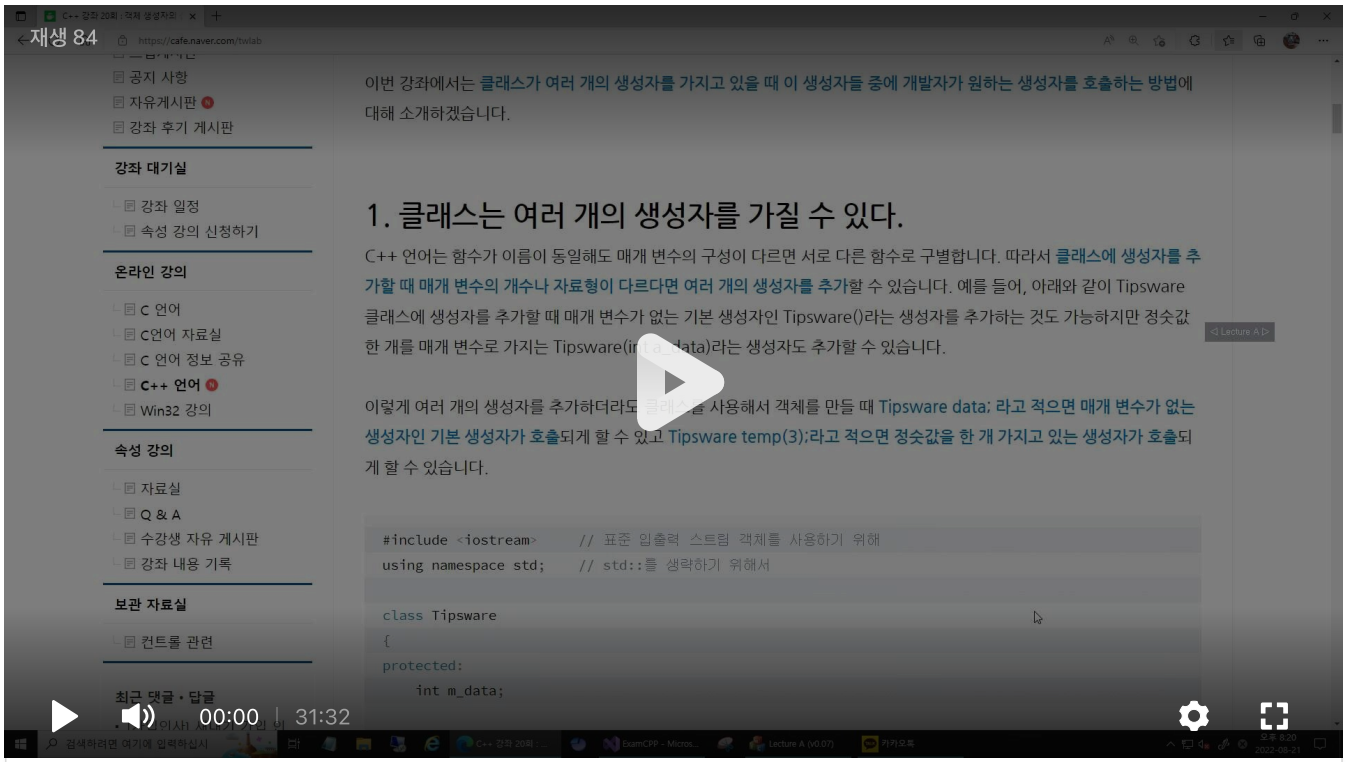
C++ 강좌 20회 : 객체 생성자의 선택적 호출

김성엽 카페매니저 + 구독 1:1 채팅
2022.08.21. 14:19 조회 251

[주의]

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

이번 강좌에서는 클래스가 여러 개의 생성자를 가지고 있을 때 이 생성자들 중에 개발자가 원하는 생성자를 호출하는 방법에 대해 소개하겠습니다.



20회차. 객체 생성자의 선택적 호출

1. 클래스는 여러 개의 생성자를 가질 수 있다.

C++ 언어는 함수가 이름이 동일해도 매개 변수의 구성이 다르면 서로 다든 함수로 구별합니다. 따라서 **클래스에 생성자를 추가할 때 매개 변수의 개수나 자료형이 다르다면 여러 개의 생성자를 추가**할 수 있습니다. 예를 들어, 아래와 같이 Tipsware 클래스에 생성자를 추가할 때 매개 변수가 없는 기본 생성자인 Tipsware()라는 생성자를 추가하는 것도 가능하지만 정숫값 한 개를 매개 변수로 가지는 Tipsware(int a_data)라는 생성자도 추가할 수 있습니다.

이렇게 여러 개의 생성자를 추가하더라도 클래스를 사용해서 객체를 만들 때 **Tipsware data; 라고 적으면 매개 변수가 없는 생성자인 기본 생성자가 호출**되게 할 수 있고 **Tipsware temp(3);라고 적으면 정숫값은 한 개 가지고 있는 생성자가 호출**되게 할 수 있습니다.

```cpp
#include <iostream>      // 표준 입출력 스트림 객체를 사용하기 위해
using namespace std;     // std::를 생략하기 위해서

class Tipsware
{
protected:
    int m_data;

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

    Tipsware(int a_data)  // 정숫값을 인자로 가지는 생성자
    {
        m_data = a_data;
    }

    // 클래스 외부에서 멤버 변수에 값을 대입할 때 사용하는 함수
    void SetData(int a_data)
    {
        m_data = a_data;
    }

    // 클래스 외부에서 멤버 변수 값을 얻을 때 사용하는 함수
    int GetData()
    {
        return m_data;
    }
};

int main()
{
    Tipsware data;      // data.Tipsware();   기본 생성자 호출
    Tipsware temp(3);  // temp.Tipsware(3);   정수 인자가 한개인 생성자 호출

    // data, temp 객체에 저장된 값을 출력한다.
    cout << "data: " << data.GetData() << endl;
    cout << "temp: " << temp.GetData() << endl;

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
data: 0
temp: 3

이 창을 닫으려면 아무 키나 누르세요...
```

그런데 매개 변수가 없는 기본 생성자를 아래와 같이 호출하면 오듀가 납니다. 즉, 매개 변수가 없으니 괄호만 사용하면 동일할 것이라고 생각하겠지만 아래와 같이 적으면 컴파일러는 data를 객체가 아닌 함수 이름으로 해석하게 됩니다. 따라서 Tipsware 자료형은 반환하는 data라는 이름의 함수 원형(function prototype)은 선언한 것으로 처리하기 때문에 아래의 코드는 오류가 발생됩니다.

```
Tipsware data();  // 기본 생성자를 호출하고 싶어서 사용했으나, 오류 처리 됨!
```

2. 상속구조에서 부모 클래스가 여러 개의 생성자를 가지고 있는 경우

정숫값을 저장하는 MyValue 클래스가 있습니다. 그런데 이 클래스는 정숫값을 양수와 음수에 상관없이 그대로 저장하기 때문에 음수를 양수로 변경해서 저장하는 기능은 추가하기 위해 MyValue 클래스에서 상속받아 AbsValue라는 클래스를 아래와 같이 만들었습니다. 그래서 MyValue 클래스와 AbsValue 클래스는 부모와 자식 관계가 되었습니다.

그런데 부모 클래스인 MyValue 클래스에 생성자가 두 개 있기 때문에 자식 클래스인 AbsValue 클래스의 생성자를 추가할 때 부모 클래스의 생성자 중에 어떤 생성자를 사용할 것인지를 함께 적어야 합니다. 예를 들어, AbsValue 클래스에 정숫값 한개를 매개 변수로 가지는 AbsValue(int a_num) 생성자를 추가할 때, 이 생성자가 부모 클래스의 MyValue() 생성자를 사용할 것인지 아니면 MyValue(int a_num) 생성자를 사용할 것인지를 지정해야 합니다. 만약, 부모 클래스의 MyValue() 생성자를 호출하고 싶다면 AbsValue(int a_num) : MyValue() 라고 코드를 구성하면 되고 부모 클래스의 MyValue(int a_num) 생성자를 호출하고 싶다면 AbsValue(int a_num) : MyValue(a_num)이라고 호출하면 됩니다.

그리고 부모 클래스에 매개 변수가 없는 기본 생성자가 있다면 부모 클래스의 기본 생성자를 호출하는 코드는 생략이 가능합니다. 즉, AbsValue(int a_num) : MyValue()코드는 AbsValue(int a_num) 이라고만 적어도 됩니다. 그래서 아래의 코드에서 AbsValue 클래스의 두 개의 생성자에서 기본 생성자는 AbsValue() : MyValue()라는 표현을 생략해서 AbsValue() 라고만 적은 것입니다. 그리고 정숫값을 가지는 생성자는 a_num 값을 부모 클래스의 생성자로 전달하기 위해 부모 생성자 함수를 호출하는 코드를 생략하지 않고 AbsValue(int a_num) : MyValue(a_num)라고 적은 것입니다.

```cpp
#include <iostream>      // 표준 입출력 스트림 객체를 사용하기 위해
using namespace std;     // std::를 생략하기 위해서

class MyValue  // 부모 클래스
{
protected:
    int m_num;

public:
    MyValue()  // 기본 생성자
    {
        m_num = 0;
    }

    MyValue(int a_num)  // 정숫값을 인자로 가지는 생성자
    {
        m_num = a_num;
    }

    // 클래스 외부에서 멤버 변수에 값을 대입할 때 사용하는 함수
    void SetValue(int a_num)
    {
        m_num = a_num;
    }

    // 클래스 외부에서 멤버 변수 값을 얻을 때 사용하는 함수
    int GetValue()
    {
        return m_num;
    }
};

class AbsValue : public MyValue  // 자식 클래스
{
public:
    AbsValue() // : MyValue()은 생략해도 됨
    {
    }

    AbsValue(int a_num) : MyValue(a_num)
    {
        // 음수면 양수로 변경한다.
        if (m_num < 0) m_num = -m_num;
    }

    // 클래스 외부에서 멤버 변수에 값을 대입할 때 사용하는 함수
    void SetValue(int a_num)
    {
        // 음수면 양수로 변경한다.
        if (a_num < 0) a_num = -a_num;
        // 변경된 값을 대입한다.
        MyValue::SetValue(a_num);
```

```
    }
};

int main()
{
    // data.AbsValue() 함수가 먼저 호출되지만 다른 코드를 수행하기 전에
    // 부모 클래스의 생성자인 data.MyValue::MyValue() 함수가 호출되어 수행되고
    // data.AbsValue() 함수 내부에 있는 코드가 수행된다.
    AbsValue data;
    // temp.AbsValue(-3) 함수가 먼저 호출되지만 다른 코드를 수행하기 전에
    // 부모 클래스의 생성자인 temp.MyValue::MyValue(-3) 함수가 호출되어 수행되고
    // temp.AbsValue(-3) 함수 내부에 있는 코드가 수행된다.
    AbsValue temp(-3);

    // data, temp 객체에 저장된 값을 출력한다.
    cout << "data: " << data.GetValue() << endl;
    cout << "temp: " << temp.GetValue() << endl;

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
data: 0
temp: 3

이 창을 닫으려면 아무 키나 누르세요...
```

만약 아래와 같이 **부모 클래스에 기본 생성자가 없고 정숫값 한개를 가지는 생성자만 있다면, 위 코드처럼 AbsValue 클래스의 기본 생성자를 AbsValue() 라고만 적으면 오류가 발생**합니다. 그래서 AbsValue() : MyValue(0) 이라고 명시적으로 부모 클래스의 생성자를 사용하는 코드를 추가해야 합니다. 이렇게 하면 AbsValue 클래스를 가지고 객체를 만들기 위해 AbsValue data;라고 코드를 적으면 부모 클래스의 생성자로 0 값이 전달되어 m_num 변수는 0으로 초기화 됩니다.

```cpp
#include <iostream>      // 표준 입출력 스트림 객체를 사용하기 위해
using namespace std;     // std::를 생략하기 위해서

class MyValue  // 부모 클래스
{
protected:
    int m_num;

public:
    MyValue(int a_num)  // 정숫값을 인자로 가지는 생성자만 있음
    {
        m_num = a_num;
    }

    // ... 나머지 코드 생략 ...
};

class AbsValue : public MyValue  // 자식 클래스
{
public:
    AbsValue() : MyValue(0) // MyValue(0)을 생략하면 오류 발생!
    {
    }

    AbsValue(int a_num) : MyValue(a_num)
    {
        // 음수면 양수로 변경한다.
        if (m_num < 0) m_num = -m_num;
    }

    // ... 나머지 코드 생략 ...
};

int main()
{
    // ... 나머지 코드 생략 ...
    return 0;
}
```

3. 멤버 변수가 여러 개의 생성자를 가지고 있는 경우 (nested 상속)

클래스의 멤버 변수에는 기본 자료형(char, int, ...)을 가진 변수만 사용할 수 있는 것이 아니라 객체도 사용이 가능합니다. 그래서 이렇게 **클래스의 멤버 변수로 객체가 사용되면 이것도 일종의 상속으로 간주하기 때문에 nested 상속**이라고 부릅니다. 그리고 객체에 생성자가 여러 개 있는 경우, 객체를 선언할 때 MyValue m_value; 또는 MyValue m_value(-3); 처럼 어떤 종류의 생성자를 사용할 것인지를 함수가 호출되는 것처럼 적어야 하는데 객체를 멤버 변수로 선언할 때는 아래와 같은 표현은 오류 처리됩니다.

```
class AbsValue
{
private:
    MyValue m_value(-3);    // 오류!! 객체를 멤버 변수로 사용할 때는 생성자 선택 표현을 사용할 수 없음
    // ... 나머지 코드 생략 ...
};
```

따라서 객체를 멤버 변수로 선언할 때는 아래와 같이 기본 생성자를 사용하겠다는 표현을 사용해야 합니다. 하지만 이 표현이 무조건 기본 생성자를 사용하겠다는 뜻은 아닙니다. 왜냐하면 **아래의 표현은 MyValue 클래스를 사용해서 m_value 객체를 생성하겠다는 표현일 뿐이고 m_value 객체가 어떤 생성자를 사용할 것인지는 포함 되어 있지 않습니다.** 즉, **nested 상속도 상속이기 때문에 일반 상속에서 부모 클래스의 생성자를 선택하는 것처럼 동일한 표현을 사용**합니다. 다만 **일반 상속은 부모 클래스의 이름을 사용**하고 **nested 상속은 해당 객체의 이름을 사용**한다는 것이 다릅니다. 따라서 아래와 같이 AbsValue 클래스의 기본 생성자에서 m_value 객체의 기본 생성자를 호출하고 싶다면 AbsValue() : m_value()라고 적으면 됩니다. 물론 이때 m_value() 기본 생성자 표현이기 때문에 생략이 가능해서 AbsValue()라고만 적어도 됩니다. 그리고 정숫값을 가지는 m_value 객체의 생성자를 사용하고 싶다면 AbsValue() : m_value(5)라고 적으면 됩니다.

```
class MyValue  // 부모 클래스
{
protected:
    int m_num;

public:
    MyValue()  // 기본 생성자
    {
        m_num = 0;
    }

    MyValue(int a_num)  // 정숫값을 인자로 가지는 생성자
    {
        m_num = a_num;
    }

    // ... 나머지 코드 생략 ...
}

class AbsValue
{
private:
    MyValue m_value;

public:
    AbsValue() : m_value() // : m_value()은 기본 생성자라 생략해도 됨
    {
    }

    // ... 나머지 코드 생략 ...
};
```

실습을 할 수 있도록 위에 설명한 내용을 전체 코드로 구성해보면 다음과 같습니다.

```cpp
#include <iostream>      // 표준 입출력 스트림 객체를 사용하기 위해
using namespace std;     // std::를 생략하기 위해서

class MyValue  // 부모 클래스
{
protected:
    int m_num;

public:
    MyValue()  // 기본 생성자
    {
        m_num = 0;
    }

    MyValue(int a_num)  // 정숫값을 인자로 가지는 생성자
    {
        m_num = a_num;
    }

    // 클래스 외부에서 멤버 변수에 값을 대입할 때 사용하는 함수
    void SetValue(int a_num)
    {
        m_num = a_num;
    }

    // 클래스 외부에서 멤버 변수 값을 얻을 때 사용하는 함수
    int GetValue()
    {
        return m_num;
    }
};

class AbsValue
{
private:
    MyValue m_value;

public:
    AbsValue() : m_value() // : m_value()은 생략해도 됨
    {
    }

    // a_num 값은 절댓값이 아니기 때문에 : m_value(a_num)라고 저장하는 것은
    // 의미 없는 작업이라서 그냥 m_value 객체의 기본 생성자를 사용합니다.
    // 그래서 : m_value()라고 적어야 하는데 이코드는 생략이 가능해서 생략했습니다.
    AbsValue(int a_num)
    {
        // 음수면 양수로 변경한다.
        if (a_num < 0) a_num = -a_num;
        // 절댓값을 m_value  변수에 저장한다.
        m_value.SetValue(a_num);
    }
```

```
    // 클래스 외부에서 멤버 변수에 값을 대입할 때 사용하는 함수
    void SetValue(int a_num)
    {
        // 음수면 양수로 변경한다.
        if (a_num < 0) a_num = -a_num;
        // 변경된 값을 대입한다.
        m_value.SetValue(a_num);
    }

    // 클래스 외부에서 멤버 변수 값을 얻을 때 사용하는 함수
    int GetValue()
    {
        return m_value.GetValue();
    }
};

int main()
{
    AbsValue data;
    AbsValue temp(-3);

    // data, temp 객체에 저장된 값을 출력한다.
    cout << "data: " << data.GetValue() << endl;
    cout << "temp: " << temp.GetValue() << endl;

    return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
data: 0
temp: 3

이 창을 닫으려면 아무 키나 누르세요...
```

위 예제 코드를 보면 알겠지만 nested 상속 구조를 가지게 되면 대부분 멤버 변수로 사용되는 객체는 기본 생성자를 사용할 확률이 높습니다. 그래서 보통 생략하기 때문에 생각보다 AbsValue() : m_value()와 같은 표현을 모르는 분들이 많습니다. 그래서 아래와 같이 MyValue 클래스에 기본 생성자가 없는 경우에 발생하는 오류를 해결하지 못하는 분들도 있습니다.

```
class MyValue  // 부모 클래스
{
protected:
    int m_num;

public:
    MyValue(int a_num)  // 정숫값을 인자로 가지는 생성자
    {
        m_num = a_num;
    }

     // ... 나머지 코드 생략 ...
};

class AbsValue
{
private:
    MyValue m_value;

public:
    AbsValue()   // 오류 발생
    {
    }

    AbsValue(int a_num)  // 오류 발생
    {
        // 음수면 양수로 변경한다.
        if (a_num < 0) a_num = -a_num;
        // 절댓값을 m_value  변수에 저장한다.
        m_value.SetValue(a_num);
    }

    // ... 나머지 코드 생략 ...
};
```

즉, MyValue 클래스에 기본 생성자가 있을 때는 AbsValue 클래스의 생성자에 오듀가 발생하지 않다가 MyValue 클래스에 기본 생성자를 지우면 AbsValue 클래스의 생성자에 오류가 받생하는 것입니다. 그러면 대부분 다시 MyValue 클래스에 기본 생성자를 추가하는 경우가 많은데, 이런 경우 기본 생성자를 추가하지 않고 아래와 같이 m_value 객체의 생성자를 직접 지정하는 방식으로 코드를 구성하면 오류를 해결할 수 있습니다.

```
class MyValue  // 부모 클래스
{
protected:
    int m_num;

public:
    MyValue(int a_num)  // 정숫값을 인자로 가지는 생성자
    {
        m_num = a_num;
    }

     // ... 나머지 코드 생략 ...
};

class AbsValue
{
private:
    MyValue m_value;

public:
    AbsValue() : m_value(0)  // 정숫값을 사용하는 m_value 객체의 생성자를 호출
    {
    }

    AbsValue(int a_num) : m_value(0)  // 정숫값을 사용하는 m_value 객체의 생성자를 호출
    {
        // 음수면 양수로 변경한다.
        if (a_num < 0) a_num = -a_num;
        // 절댓값을 m_value  변수에 저장한다.
        m_value.SetValue(a_num);
    }

    // ... 나머지 코드 생략 ...
};
```

결국 기본 생성자는 있으면 생략 표현을 사용할 수 있기 때문에 편리하지만 오듀 해결을 위해 반드시 필요한 것은 아닙니다. 따라서 기본 생성자가 필요없다면 필요한 생성자만 남겨두시고 사용할 때 직접 원하는 생성자를 지정하는 표현을 사용하면 됩니다.

2022.11.26. 23:02 답글쓰기

모자꾸기 2

안녕하십니까!! 자식클래스는 부모클래스가 가지고 있는 것들이 복제됨 + 알파일텐데, 자식클래스가 만들어지면 자식클래스의 생성자에서 부모클래스의 생성자까지 불려진다는건 자식클래스 객체가 만들어질 때 부모클래스 객체도 만들어진다는 것 같은데 앞서 말씀드린대로 자식클래스 = 부모클래스 + 알파로 이미 충분한데 부모클래스 객체가 왜 또 만들어지는건가요?!!

2024.02.04. 23:31 답글쓰기

김성엽 M 작성자

부모 객체가 만들어지는 것이 아니라 생성자 함수만 호출되어진다는 뜻입니다. 생성자 호출은 객체가 만들어지는 것과 상관없이 그냥 함수 개념으로 호출될수 있습니다 :)

2024.02.04. 23:47 답글쓰기

김성엽 M 작성자

생성자가 호출되어 객체가 생성되는 것이 아니라 객체가 만들어지고 초기화를 위해 생성자를 호출하는 개념입니다. 따라서 상속받은 자식 객체가 생성되었을때 자신의 영역은 자신이 초기화 하면 되지만 부모 영역은 부모 생성자를 호출해서 초기화 하는 것이 중복코드를 줄이는 방법이기 때문입니다.

2024.02.04. 23:50 답글쓰기

모자꾸기 2

김성엽 명확히 이해했습니다 .감사합니다!

2024.02.04. 23:50 답글쓰기

김성엽 M 작성자

모자꾸기

2024.02.04. 23:51 답글쓰기

dh221009

댓글을 남겨보세요

등록