

C++ 강좌 14회 : 다형성과 가상 함수



김성엽 카페매니저



+ 구독

1:1 채팅

2022.06.14. 03:22 조회 393



댓글 40

URL 복사



[주의]

이 영상은 제가 공개하는 곳 외에 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.

이번 강좌에서는 상속으로 유사한 클래스들을 묶어 부모 클래스로 객체를 통합 관리하는 다형성(polymorphism) 기술에 대해 소개하고 다형성을 사용할 때 발생하는 참조 문제를 해결하는 가상 함수(virtual function)에 대해서도 설명하겠습니다.

주제 컴퓨터/통신 > 프로그래밍 언어

지역 서울특별시 광진구

카페 글쓰기

카페 채팅

검색

★ 즐겨찾는 게시판

- 전체글보기 213
- 스텝게시판
- 공지 사항
- 자유게시판
- 강좌 후기 게시판

강좌 대기실

- 강좌 일정
- 속성 강의 신청하기

온라인 강의

- C 언어
- C 언어 자료실
- C 언어 정보 공유
- C++ 언어

[주의]

이 영상은 제가 공개하는 곳 외에 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.

이번 강좌에서는 상속으로 유사한 클래스들을 묶어 부모 클래스로 객체를 통합 관리하는 다형성(polymorphism) 기술에 대해 소개하고 다형성을 사용할 때 발생하는 참조 문제를 해결하는 가상 함수(virtual function)에 대해서도 설명하겠습니다.

1. 직사각형과 타원의 넓이를 구하는 클래스 만들기

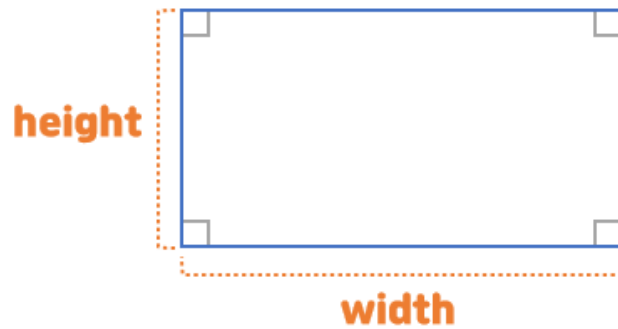
직사각형과 타원은 아래와 같이 동일한 기준으로 값을 입력받아서 도형의 넓이를 계산할 수 있습니다.

00:00 | 52:03

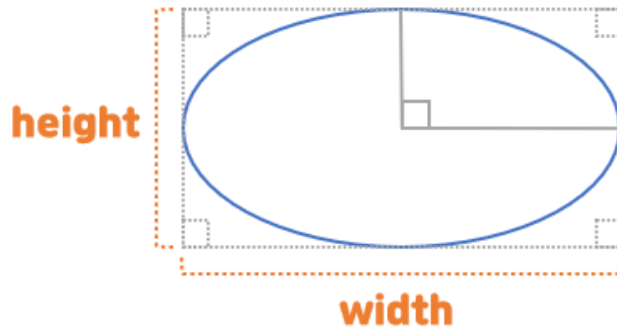
C++ 강좌 14회 : 다형성과 가상 함수

1. 직사각형과 타원의 넓이를 구하는 클래스 만들기

직사각형과 타원은 아래와 같이 동일한 기준으로 값을 입력받아서 도형의 넓이를 계산할 수 있습니다.



$$\text{직사각형 넓이} = \text{width} * \text{height}$$



$$\text{타원 넓이} = 3.141592 * \text{width}/2 * \text{height}/2$$

그래서 위 공식을 사용하여 도형의 넓이를 계산하는 클래스를 간단하게 만들어보면 다음과 같습니다.


```

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

class Rectangle
{
protected:
    int m_width, m_height;

public:
    Rectangle()    // 생성자
    {
        m_width = m_height = 1;
    }

    void SetSize(int a_width, int a_height) // 크기를 지정하는 함수
    {
        m_width = a_width;
        m_height = a_height;
    }

    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return m_width*m_height;
    }
};

class Ellipse
{
protected:
    int m_width, m_height;

public:
    Ellipse()    // 생성자
    {
        m_width = m_height = 1;
    }

    void SetSize(int a_width, int a_height) // 크기를 지정하는 함수
    {
        m_width = a_width;
        m_height = a_height;
    }

    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return 3.141592 * m_width/2 * m_height/2;
    }
};

int main()
{
    Rectangle r;
    r.SetSize(60, 30);
}

```

```

    Ellipse e;
    e.SetSize(60, 30);

    printf("Rectangle: %g, Ellipse: %g\n", r.GetArea(), e.GetArea());
    return 0;
}

```

위 예제는 아래와 같이 출력됩니다.

```

Rectangle: 1800, Ellipse: 1413.72

```

이 창을 닫으려면 아무 키나 누르세요...

그런데 위 예제에서 Rectangle 클래스와 Ellipse 클래스는 중복 코드가 많기 때문에 상속을 사용해서 중복 코드를 제거해야 하지만 **두 클래스의 의미가 너무 달라서 어떤 클래스를 부모 클래스로 선택해야 할지 결정하기 어려울 것**입니다. 따라서 이런 경우에는 **두 클래스의 의미를 포함하는 새로운 클래스를 추가해서 두 클래스의 부모로 사용**하는 것이 좋습니다.

예를 들어, Rectangle과 Ellipse는 도형이라는 공통점이 있기 때문에 도형을 의미하는 Figure 클래스를 아래와 같이 추가합니다. 그리고 Figure 클래스에서 Rectangle 클래스와 Ellipse 클래스를 상속받는 구조로 만들면 기존 소스 코드에서 중복된 코드를 제거할 수 있습니다. 즉, Rectangle 클래스와 Ellipse 클래스에 중복되었던 코드는 Figure 클래스에만 존재하기 때문에 이 코드에 기능 변경이 발생하면 Figure 클래스에서만 수정하면 됩니다.


```

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

class Figure
{
protected:
    int m_width, m_height;

public:
    Figure()    // 생성자
    {
        m_width = m_height = 1;
    }

    void SetSize(int a_width, int a_height) // 크기를 지정하는 함수
    {
        m_width = a_width;
        m_height = a_height;
    }

    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return 0.0;
    }
};

class Rectangle : public Figure
{
public:
    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return m_width*m_height;
    }
};

class Ellipse : public Figure
{
public:
    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return 3.141592 * m_width/2 * m_height/2;
    }
};

int main()
{
    Rectangle r;
    r.SetSize(60, 30);

    Ellipse e;
    e.SetSize(60, 30);

    printf("Rectangle: %g, Ellipse: %g\n", r.GetArea(), e.GetArea());
}

```

```
return 0;
}
```

위 예제는 아래와 같이 출력됩니다.

```
Rectangle: 1800, Ellipse: 1413.72
```

이 창을 닫으려면 아무 키나 누르세요...

2. 클래스가 많아지면 코드 관리가 어려워진다.

소스 코드에서 클래스는 한 번만 정의되지만 이 클래스를 사용하는 코드는 계속 반복해서 나타납니다. 따라서 **소스 코드의 복잡도나 유지 보수에 대한 평가는 클래스를 정의하는 곳이 아니라 클래스를 사용하는 곳에서 평가**하는 것이 일반적입니다. 예를 들어, 위 예제 소스에서 Figure, Rectangle 그리고 Ellipse 클래스를 정의한 코드는 한 번만 나오겠지만 main 함수에 사용된 코드는 프로그램의 기능이 많아질수록 소스 코드의 여러 곳에서 사용될 것입니다.

위 예제의 main 함수에서는 사각형 객체 한 개와 타원 객체 한 개를 만들어서 각각의 넓이를 출력하도록 했지만 아래와 같이 객체를 배열 형식으로 만들어서 전체 도형의 크기를 합산하는 코드가 사용될 수도 있습니다. 그런데 아래의 코드를 보면 **Rectangle 객체와 Ellipse 객체가 서로 다른 객체이기 때문에 한 개의 배열로 선언하지 못했습니다. 그래서 넓이를 합산할 때 두 객체 목록을 서로 다른 반복문으로 분리해서 처리했습니다.** 이처럼 프로그램을 구성하는 소스 코드에 클래스의 종류가 많아지면 해당 클래스를 사용하는 코드가 분리되어 나열되기 때문에 **중복 코드는 아닌데 묘하게 중복 코드의 느낌을 주는 코드가 많아지게 됩니다.**

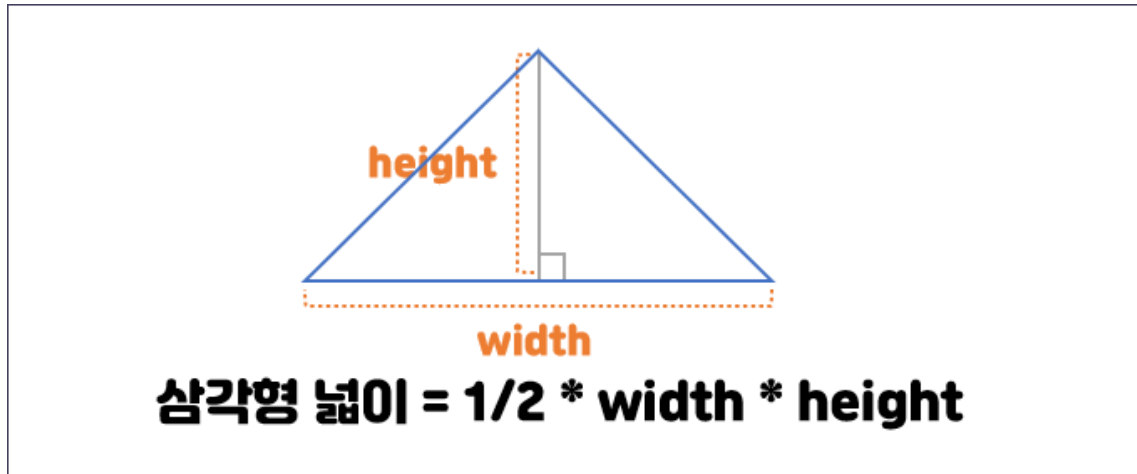
```
int main()
{
    Rectangle r[2];
    r[0].SetSize(60, 30);
    r[1].SetSize(80, 50);

    Ellipse e[3];
    e[0].SetSize(60, 30);
    e[1].SetSize(50, 25);
    e[2].SetSize(40, 60);

    double total_area = 0.0;
    for (int i = 0; i < 2; i++) total_area += r[i].GetArea();
    for (int i = 0; i < 3; i++) total_area += e[i].GetArea();

    printf("Total Area: %g\n", total_area);
    return 0;
}
```


이제 코드의 복잡도를 높이기 위해 아래와 같이 삼각형의 넓이를 계산하는 공식을 사용하는 Triangle 클래스를 추가하겠습니다. 이 클래스도 도형을 의미하는 Figure 클래스에서 상속했습니다.



```
class Triangle : public Figure
{
public:
    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return m_width * m_height * 0.5;
    }
};
```

main 함수에 Triangle 클래스를 사용하는 코드도 아래와 같이 추가해 보겠습니다. 새로운 객체가 추가되었기 때문에 **이 객체를 나열하기 위한 배열이 추가**되고 **이 객체를 사용하기 위한 반복문도 추가**되었습니다. 즉, 상속 문법을 사용해서 클래스 간에 발생하는 중복 코드는 효과적으로 제거할 수 있지만, **이 클래스들을 사용하는 코드에는 중복된 패턴의 코드가 증가하여 코드의 유지 보수는 점점 더 나빠지게 됩니다.**

```

int main()
{
    Rectangle r[2];
    r[0].SetSize(60, 30);
    r[1].SetSize(80, 50);

    Ellipse e[3];
    e[0].SetSize(60, 30);
    e[1].SetSize(50, 25);
    e[2].SetSize(40, 60);

    Triangle t[4];
    t[0].SetSize(60, 30);
    t[1].SetSize(20, 25);
    t[2].SetSize(30, 30);
    t[3].SetSize(50, 60);

    double total_area = 0.0;
    for (int i = 0; i < 2; i++) total_area += r[i].GetArea();
    for (int i = 0; i < 3; i++) total_area += e[i].GetArea();
    for (int i = 0; i < 4; i++) total_area += t[i].GetArea();

    printf("Total Area: %g\n", total_area);
    return 0;
}

```

그리고 위 코드에서 각 도형의 넓이를 합산하는 기능을 새로운 함수로 분리해 보면 클래스가 많아지는 것이 얼마나 코드의 복잡도를 증가시키는지 확실하게 알 수 있습니다. 만약, 아래와 같이 코드를 구성했는데 새로운 도형 클래스가 추가되면 GetTotalArea 함수에는 반복문만 한 개 더 추가되는 것이 아니라 해당 도형으로 선언된 배열과 배열에 저장된 데이터의 갯수를 넘겨받기위해 매개 변수도 두 개 더 추가되어야 합니다. 그리고 main 함수에서 GetTotalArea 함수를 호출하는 코드에도 인자를 추가하기 위해 코드를 수정해야 합니다.

결국 이런 형식으로 프로그램 소스를 구성하면 클래스가 새로 추가될 때마다 많은 수정이 발생되기 때문에 개발자는 점점 더 소스 코드를 유지 보수하기가 어려워질 것입니다.

```

double GetTotalArea(Rectangle a_r[], int a_r_count, Ellipse a_e[], int a_e_count,
                    Triangle a_t[], int a_t_count)
{
    double total_area = 0.0;
    for (int i = 0; i < a_r_count; i++) total_area += a_r[i].GetArea();
    for (int i = 0; i < a_e_count; i++) total_area += a_e[i].GetArea();
    for (int i = 0; i < a_t_count; i++) total_area += a_t[i].GetArea();

    return total_area;
}

int main()
{
    Rectangle r[2];
    r[0].SetSize(60, 30);
    r[1].SetSize(80, 50);

    Ellipse e[3];
    e[0].SetSize(60, 30);
    e[1].SetSize(50, 25);
    e[2].SetSize(40, 60);

    Triangle t[4];
    t[0].SetSize(60, 30);
    t[1].SetSize(20, 25);
    t[2].SetSize(30, 30);
    t[3].SetSize(50, 60);

    double total_area = GetTotalArea(r, 2, e, 3, t, 4);

    printf("Total Area: %g\n", total_area);
    return 0;
}

```

3. 다형성과 가상 함수

Rectangle 클래스와 Ellipse 클래스를 사용해서 r, e 객체를 선언하고 이 객체의 주소를 포인터 변수에 저장해서 사용하려면 아래와 같이 **Rectangle *로 선언된 포인터 변수와 Ellipse *로 선언된 포인터 변수에 각 객체의 주소를 저장해서 사용**하면 됩니다.

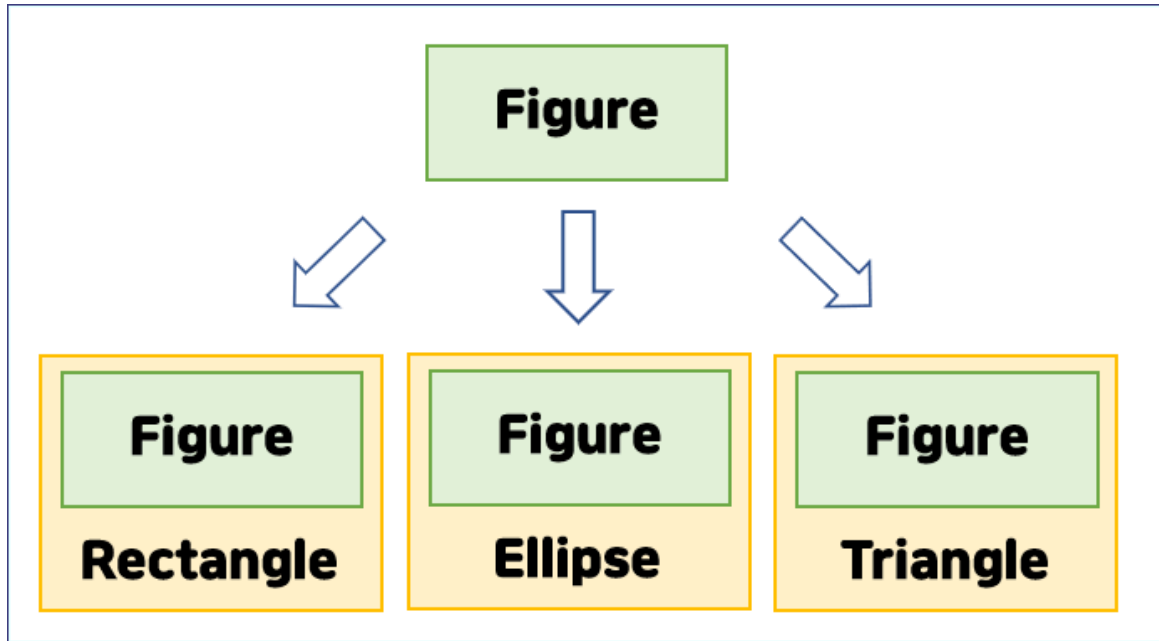
```

Rectangle r;
Rectangle *p_r = &r;

Ellipse e;
Ellipse *p_e = &e;

```

그런데 우리가 여기서 기억해야 할 것은 아래의 그림처럼 **상속 문법이 클래스 복사 기능을 사용하기 때문에 Rectangle 클래스와 Ellipse 클래스는 Figure 클래스를 그대로 가지고 있다는 점**입니다.



따라서 r 객체와 e 객체를 Rectangle *와 Ellipse *로 선언된 포인터 대신 아래와 같이 **Figure *로 선언된 포인터 변수에 주소를 저장해서 사용하는 것도 가능**합니다. 이처럼 **부모 클래스의 포인터로 자식 객체의 주소를 저장해서 자식 객체를 간접적으로 사용하는 문법이 다형성(polymorphism)**입니다.

```
Rectangle r;
Figure *p_r = &r;
p_r->SetSize(30, 20); // r 객체의 폭을 30, 높이를 20으로 설정한다.

Ellipse e;
Figure *p_e = &e;
p_e->SetSize(30, 20); // r 객체의 폭을 30, 높이를 20으로 설정한다.
```

하지만 위 예제에서 사용된 **SetSize 함수는 Figure 클래스에 정의된 함수이기 때문에 정상적으로 수행**되지만 아래와 같이 **p_r->GetArea** 형식으로 함수를 호출하면 Rectangle 클래스에 오버라이딩된 GetArea 함수가 호출되는 것이 아니라 **Figure 클래스의 GetArea 함수가 호출되어 사각형 넓이가 0으로 출력**될 것입니다.

이 현상은 컴파일러 입장에서 봤을 때 **p_r 변수의 자료형이 Figure *라서 함수 호출시 Figure 네임스페이스가 우선 처리되기 때문에** 그런 것입니다. 결국 Rectangle 객체에는 Rectangle::GetArea 함수와 Figure::GetArea 함수가 공존합니다. 그리고 평소에는 Rectangle 클래스로 선언된 객체를 사용하기 때문에 Rectangle 네임스페이스가 우선 처리되어 Rectangle::GetArea 함수가 정상적으로 호출되지만 지금처럼 **다형성 문법을 사용하면 Figure 네임스페이스가 우선 처리되어 Figure::GetArea 함수가 호출**되는 것입니다.

```

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

class Figure
{
protected:
    int m_width, m_height;

public:
    Figure()    // 생성자
    {
        m_width = m_height = 1;
    }

    void SetSize(int a_width, int a_height) // 크기를 지정하는 함수
    {
        m_width = a_width;
        m_height = a_height;
    }

    double GetArea()    // 도형의 넓이를 계산하는 함수
    {
        return 0.0;
    }
};

class Rectangle : public Figure
{
public:
    double GetArea()    // 도형의 넓이를 계산하는 함수
    {
        return m_width*m_height;
    }
};

int main()
{
    Rectangle r;
    Figure *p_r = &r;
    p_r->SetSize(30, 20);

    printf("r.GetArea: %g, p_r->GetArea: %g\n", r.GetArea(), p_r->GetArea());
    return 0;
}

```

위 예제는 아래와 같이 출력됩니다.

```
r.GetArea: 600, p_r->GetArea: 0
```

이 창을 닫으려면 아무 키나 누르세요...

따라서 다형성 문법을 사용할 때, 자식 클래스에 오버라이딩된 함수를 정상적으로 호출하고 싶다면 아래와 같이 Figure 클래스에서 오버라이딩될 GetArea 함수 앞에 virtual 키워드를 추가하면 됩니다. 이렇게 함수 앞에 virtual 키워드를 추가하면 컴파일러가 소스를 해석할 때 이 함수가 오버라이딩되면 관련된 함수의 주소 테이블을 별도로 만들어서 관리합니다. 따라서 Figure *로 GetArea 함수를 호출하더라도 해당 주소에 있는 객체가 Rectangle 클래스라면 Rectangle::GetArea 함수가 호출되도록 처리해줍니다.

이렇게 다형성 문법을 사용할 때 자식 클래스에 오버라이딩된 함수를 정상적으로 호출하기 위해 virtual 키워드를 붙인 함수를 가상 함수(virtual function)라고 합니다.

```

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

class Figure
{
protected:
    int m_width, m_height;

public:
    Figure()    // 생성자
    {
        m_width = m_height = 1;
    }

    void SetSize(int a_width, int a_height) // 크기를 지정하는 함수
    {
        m_width = a_width;
        m_height = a_height;
    }

    virtual double GetArea()    // 도형의 넓이를 계산하는 함수 (가상 함수)
    {
        return 0.0;
    }
};

class Rectangle : public Figure
{
public:
    double GetArea()    // 도형의 넓이를 계산하는 함수
    {
        return m_width*m_height;
    }
};

int main()
{
    Rectangle r;
    Figure *p_r = &r;
    p_r->SetSize(30, 20);

    printf("r.GetArea: %g, p_r->GetArea: %g\n", r.GetArea(), p_r->GetArea());
    return 0;
}

```

위 예제는 아래와 같이 출력됩니다. 즉, Figure 클래스의 GetArea 함수가 가상 함수로 선언되었기 때문에 p_r->GetArea처럼 함수가 호출되어도 정상적으로 사각형 넓이가 계산됩니다.

```
r.GetArea: 600, p_r->GetArea: 600
```

이 창을 닫으려면 아무 키나 누르세요...

4. 다형성을 사용해서 코드 수정하기

다형성을 사용하면 코드가 얼마나 개선되는지 비교하기 위해 2번 항목에서 소개했던 코드를 다시 보여드리겠습니다. 이 코드는 Rectangle 클래스로 선언된 객체와 Ellipse 클래스로 선언된 객체의 넓이를 모두 합산하는 코드입니다.

```
int main()
{
    Rectangle r[2];
    r[0].SetSize(60, 30);
    r[1].SetSize(80, 50);

    Ellipse e[3];
    e[0].SetSize(60, 30);
    e[1].SetSize(50, 25);
    e[2].SetSize(40, 60);

    double total_area = 0.0;
    for (int i = 0; i < 2; i++) total_area += r[i].GetArea();
    for (int i = 0; i < 3; i++) total_area += e[i].GetArea();

    printf("Total Area: %g\n", total_area);
    return 0;
}
```

위 코드에서는 사각형 객체와 타원 객체가 서로 다른 클래스라는 관점에서 작업했기 때문에 두 개의 배열로 객체를 분리해서 관리했습니다. 하지만 다형성 관점에서 생각하면 Rectangle 클래스로 만든 객체와 Ellipse 클래스로 만든 객체는 부모 클래스의 포인터 형식인 Figure *로 한 번에 관리가 가능합니다. 따라서 **f *형식의 배열을 선언하고 r 객체 배열과 e 객체 배열의 주소를 대입해서 사용하면 두 개의 반복문을 한 개의 반복문으로 합칠 수 있습니다.**


```

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

class Figure
{
protected:
    int m_width, m_height;

public:
    Figure()    // 생성자
    {
        m_width = m_height = 1;
    }

    void SetSize(int a_width, int a_height) // 크기를 지정하는 함수
    {
        m_width = a_width;
        m_height = a_height;
    }

    virtual double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return 0.0;
    }
};

class Rectangle : public Figure
{
public:
    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return m_width*m_height;
    }
};

class Ellipse : public Figure
{
public:
    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return 3.141592 * m_width/2 * m_height/2;
    }
};

int main()
{
    Rectangle r[2];
    r[0].SetSize(60, 30);
    r[1].SetSize(80, 50);

    Ellipse e[3];
    e[0].SetSize(60, 30);
    e[1].SetSize(50, 25);

```

```

e[2].SetSize(40, 60);

// Figure * 형식의 포인터를 항목으로 가지는 배열에 r, e 배열의 주소를 대입
Figure *p_f[5] = { r, r + 1, e, e + 1, e + 2 };

double total_area = 0.0;
for (int i = 0; i < 5; i++) total_area += p_f[i]->GetArea();

printf("Total Area: %g\n", total_area);
return 0;
}

```

위 예제는 아래와 같이 출력됩니다.

```
Total Area: 10080.4
```

이 창을 닫으려면 아무 키나 누르세요...

위와 같이 배열을 선언하고 다시 포인터로 주소를 합치는 것이 불편해 보인다면 아래와 같이 main 함수의 코드를 동적 메모리 할당을 사용해서 작업해도 됩니다.

```

int main()
{
    // Figure * 형식의 포인터를 항목으로 가지는 배열에 사각형 객체 두 개와 타원 객체 3개를 할당
    Figure *p_f[5] = { new Rectangle, new Rectangle, new Ellipse, new Ellipse, new Ellipse };

    p_f[0]->SetSize(60, 30); // 사각형
    p_f[1]->SetSize(80, 50); // 사각형
    p_f[2]->SetSize(60, 30); // 타원
    p_f[3]->SetSize(50, 25); // 타원
    p_f[4]->SetSize(40, 60); // 타원

    double total_area = 0.0;
    for (int i = 0; i < 5; i++) total_area += p_f[i]->GetArea();

    printf("Total Area: %g\n", total_area);

    // 할당된 메모리를 해제한다.
    for (int i = 0; i < 5; i++) delete p_f[i];
    return 0;
}

```

그리고 객체를 동적 할당한 다음 SetSize 함수로 폭과 높이를 지정하는 것이 불편해 보인다면 아래와 같이 **각 클래스에 도형의 크기를 지정할 수 있는 객체 생성자를 추가하여 단순화 시키는 것도 가능합니다**. 이제 main 함수의 코드를 보면 클래스를 사용하는 코드가 많이 단순해 졌음을 확인할 수 있습니다.


```

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

class Figure
{
protected:
    int m_width, m_height;

public:
    Figure()    // 생성자
    {
        m_width = m_height = 1;
    }

    Figure(int a_width, int a_height)    // 정숫값 두 개를 가진 생성자
    {
        SetSize(a_width, a_height);
    }

    void SetSize(int a_width, int a_height)    // 크기를 지정하는 함수
    {
        m_width = a_width;
        m_height = a_height;
    }

    virtual double GetArea()    // 도형의 넓이를 계산하는 함수
    {
        return 0.0;
    }
};

class Rectangle : public Figure
{
public:
    // 정숫값 두 개를 가진 생성자. 부모 객체 생성자가 두 개라서
    // : Figure(a_width, a_height) 형식으로 원하는 생성자를 호출하면 됩니다.
    Rectangle(int a_width, int a_height) : Figure(a_width, a_height)
    {
        // 부모 생성자에서 값을 대입하기 때문에 별도의 코드가 필요 없습니다.
    }

    double GetArea()    // 도형의 넓이를 계산하는 함수
    {
        return m_width*m_height;
    }
};

class Ellipse : public Figure
{
public:
    // 정숫값 두 개를 가진 생성자. 부모 객체 생성자가 두 개라서
    // : Figure(a_width, a_height) 형식으로 원하는 생성자를 호출하면 됩니다.
    Ellipse(int a_width, int a_height) : Figure(a_width, a_height)

```

```

{
    // 부모 생성자에서 값을 대입하기 때문에 별도의 코드가 필요 없습니다.
}

double GetArea() // 도형의 넓이를 계산하는 함수
{
    return 3.141592 * m_width/2 * m_height/2;
}

};

#define MAX_FIGURE_COUNT 5 // 사용할 도형의 개수

int main()
{
    // Figure * 형식의 포인터를 항목으로 가지는 배열에 사각형 객체 두 개와 타원 객체 3개를 할당
    Figure *p_f[MAX_FIGURE_COUNT] = {
        new Rectangle(60, 30), new Rectangle(80, 50),
        new Ellipse(60, 30), new Ellipse(50, 25), new Ellipse(40, 60)
    };

    double total_area = 0.0;
    for (int i = 0; i < MAX_FIGURE_COUNT; i++) total_area += p_f[i]->GetArea();

    printf("Total Area: %g\n", total_area);

    // 할당된 메모리를 해제한다.
    for (int i = 0; i < MAX_FIGURE_COUNT; i++) delete p_f[i];
    return 0;
}

```

위 예제는 아래와 같이 출력됩니다.

```
Total Area: 10080.4
```

이 창을 닫으려면 아무 키나 누르세요...

그리고 이렇게 코드를 구성하면 삼각형 도형을 구현한 Triangle 클래스를 추가해도 main 함수의 코드는 객체를 생성하는 쪽 코드와 도형의 개수만 수정하면 됩니다. 즉, 이렇게 코드를 구성하면 클래스를 정의하는 코드 뿐만 아니라 클래스를 사용하는 코드에도 유사 패턴이 사라져서 소스 코드의 유지 보수가 편해집니다.


```

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

class Figure
{
protected:
    int m_width, m_height;

public:
    Figure()    // 생성자
    {
        m_width = m_height = 1;
    }

    Figure(int a_width, int a_height)    // 정숫값 두 개를 가진 생성자
    {
        SetSize(a_width, a_height);
    }

    void SetSize(int a_width, int a_height)    // 크기를 지정하는 함수
    {
        m_width = a_width;
        m_height = a_height;
    }

    virtual double GetArea()    // 도형의 넓이를 계산하는 함수
    {
        return 0.0;
    }
};

class Rectangle : public Figure
{
public:
    // 정숫값 두 개를 가진 생성자. 부모 객체 생성자가 두 개라서
    // : Figure(a_width, a_height) 형식으로 원하는 생성자를 호출하면 됩니다.
    Rectangle(int a_width, int a_height) : Figure(a_width, a_height)
    {
        // 부모 생성자에서 값을 대입하기 때문에 별도의 코드가 필요 없습니다.
    }

    double GetArea()    // 도형의 넓이를 계산하는 함수
    {
        return m_width*m_height;
    }
};

class Ellipse : public Figure
{
public:
    // 정숫값 두 개를 가진 생성자. 부모 객체 생성자가 두 개라서
    // : Figure(a_width, a_height) 형식으로 원하는 생성자를 호출하면 됩니다.
    Ellipse(int a_width, int a_height) : Figure(a_width, a_height)

```



```

{
    // 부모 생성자에서 값을 대입하기 때문에 별도의 코드가 필요 없습니다.
}

double GetArea() // 도형의 넓이를 계산하는 함수
{
    return 3.141592 * m_width/2 * m_height/2;
}

};

class Triangle : public Figure
{
public:
    // 정숫값 두 개를 가진 생성자. 부모 객체 생성자가 두 개라서
    // : Figure(a_width, a_height) 형식으로 원하는 생성자를 호출하면 됩니다.
    Triangle(int a_width, int a_height) : Figure(a_width, a_height)
    {
        // 부모 생성자에서 값을 대입하기 때문에 별도의 코드가 필요 없습니다.
    }

    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return m_width * m_height * 0.5;
    }

};

#define MAX_FIGURE_COUNT 9 // 사용할 도형의 개수

int main()
{
    // Figure * 형식의 포인터를 항목으로 가지는 배열에 사각형 객체 두 개와 타원 객체 3개를 할당
    Figure *p_f[MAX_FIGURE_COUNT] = {
        new Rectangle(60, 30), new Rectangle(80, 50),
        new Ellipse(60, 30), new Ellipse(50, 25), new Ellipse(40, 60),
        new Triangle(60, 30), new Triangle(20, 25), new Triangle(30, 30), new Triangle(50, 60)
    };

    double total_area = 0.0;
    for (int i = 0; i < MAX_FIGURE_COUNT; i++) total_area += p_f[i]->GetArea();

    printf("Total Area: %g\n", total_area);

    // 할당된 메모리를 해제한다.
    for (int i = 0; i < MAX_FIGURE_COUNT; i++) delete p_f[i];
    return 0;
}

```

위 예제는 아래와 같이 출력됩니다.


Total Area: 13180.4

이 창을 닫으려면 아무 키나 누르세요...

5. 다형성을 사용해야 하는 이유


C++ 언어는 **클래스를 기반으로하는 소스 전개 방식을** 사용합니다. 따라서 클래스가 많이 만들어지는 것은 당연한 현상입니다. 하지만 이렇게 **클래스가 많아지면 클래스를 사용하는 코드가 객체 종류에 따라 나열되기 때문에 코드를 유지 보수하는 입장에서 보면 C 언어보다 소스 코드 관리가 더 불편**합니다. 그래서 C++ 언어에서 클래스와 상속만 사용해서 작업하면 코드를 유지 보수하는 것이 좋아지지 않기 때문에 프로그램의 기능이 많아질수록 개발자는 작업이 점점 더 힘들어집니다.

따라서 C++ 언어에서 코드에 대한 유지 보수를 잘 하려면 **다형성을 사용하여 유사한 기능을 가진 클래스들을 하나의 부모 클래스로 통합해서 사용**해야 합니다. 즉, 클래스 사용 코드에서는 객체를 생성하는 코드를 제외한 나머지 코드에서 가능하면 부모 클래스만 사용하도록 코드를 작성하는 것이 좋습니다. 이렇게만 소스 코드를 작성할수 있다면 프로그램에 기능 변경이 생기거나 클래스가 추가되더라도 클래스를 사용하는 코드에는 거의 변화가 생기지 않게 코드를 작성할 수 있습니다.

 클린봇이 악성 댓글을 감지합니다.

 설정

댓글 등록순 최신순 

 관심글 댓글 알림 ☐



해적왕곰팅 3

다형성을 이해하기 쉽게 설명해 주셔서 1시간 동안 즐겁고 재미있게 강의를 들었네요. ㅎㅎ

2022.06.21. 00:53 답글쓰기



김성엽 M 작성자

으흐흐~ 다행입니다 ㅎㅎ



수고하셨습니다

2022.06.21. 01:06 답글쓰기



bac 2

mythread 카페에 있는 그림판 만들기 강의에 쓰인 다형성 기법들을 복습할 때 참고할 수 있겠네요. 감사합니다.

2022.07.01. 15:53 답글쓰기



김성엽 M 작성자

회원님은 열심회원이라 아래의 자료를 확인할수 있으니, 이 소스로 복습하면 됩니다.
<https://cafe.naver.com/mythread/558>

2022.07.01. 16:35 답글쓰기



bac 2

김성엽 이 카페에서 C++ 복습하느라 열심회원인 걸 잠시 깜박했네요. 감사합니다.

2022.07.01. 18:38 답글쓰기

a12345 2



버추얼 키워드가 유용해보인다고 난사하면 컴파일러가 가상함수 테이블에서 함수찾느라 더 고생하는군요 감사합니다!!

2022.07.24. 20:04 답글쓰기



a12345

delete 도 각각 따로해야하는것도 배워갑니다!!

2022.07.24. 20:07 답글쓰기



김성엽

작성자

ㅎㅎ 뭐든 과하면 부작용이 생깁니다ㅎㅎ 개발자가 적절하게 잘 사용하는것이 좋습니다~ :)

2022.07.24. 20:08 답글쓰기



김성엽

작성자

a12345



수고하셨습니다

2022.07.24. 20:08 답글쓰기



조민희

가상 함수가 생각보다 쉬운 개념이었군요 ! 좋은 설명 감사드립니다 ~

2022.08.03. 22:17 답글쓰기



김성엽

작성자



수고하셨습니다

2022.08.03. 23:18 답글쓰기



조민희

```
Rectangle(int a_width, int a_height) : Figure(a_width, a_height)
{
```

```
}
를
```

```
Rectangle(int a_width, int a_height)
{
    Figure(a_width, a_height);
}
```

과 같이 표현하면 컴파일 오류가 발생하나요 ?

2022.08.06. 20:27 답글쓰기



김성엽

작성자

:를 사용해서 적은 함수는 부모 클래스에 존재하는 생성자 중에 원하는 생성자를 선택하는 표현이라서 Rectangle 함수 자체의 형식과는 무관합니다. 따라서 : 를 사용한 뒤에 적은 Figure를 제외하면 두 Rectangle 함수의 인자가 두 개이고 모두 int라서 동일 함수로 체크되어 오류가 발생합니다.

2022.08.07. 02:47 답글쓰기



조민희

김성엽 이것이 생성자 초기화 문법 (initializer list)인가요 ?

2022.08.07. 09:15 답글쓰기



김성엽

작성자

조민희 Rectangle 자체는 생성자를 정의하는 문법이고 그 뒤에 : Figure는 자신의 부모 클래스나 자신에게 포함된 객체의 생성자를 선택하는 문법입니다.

2022.08.07. 12:25 답글쓰기



김성엽

작성자



조민희 생성자의 선택적 호출에 대해서는 강좌에서 잠깐 언급만했어 그럴수 있습니다. 제대로 강의하는건 20회차에서 할 예정인데 아직 나가지 못했네요. 일단 아래에 링크한 글을 읽어보면 개념을 익힐수 있을테니, 아래에 링크한 강좌를 참고하세요.

<https://blog.naver.com/tipsware/221080780153>

2022.08.07. 12:28 답글쓰기



조민희 3

김성엽 글을 읽어보니 흐름이 이해가 갑니다 ! 20회차 강의가 기다려 지네요 ㅎㅎ 친절한 답변 감사드립니다 !

2022.08.07. 15:26 답글쓰기



김성엽 M 작성자

조민희



2022.08.07. 15:35 답글쓰기



티모 3

이해가 되는 듯 하면서 혼자 코딩하면 어려운 내용인듯.. 역시 선생님 강의를 반복해서 듣고 따라하면서 몸에 익혀야겠네요



2022.11.21. 17:29 답글쓰기



김성엽 M 작성자

좋은 기술일수록 개념을 익히는데 시간이 좀 걸리는 법입니다. 제대로 배우고 익혀서 개발에 잘 활용하세요~ :)



2022.11.21. 18:05 답글쓰기



카일 3

1기 온라인 강좌에 이어 다른 예제와 함께 설명해 주시니 좀 더 이해의 폭이 넓어진 듯 합니다. 자유자재로 쓸수 있을 때까지 복습해야겠습니다. 강의 잘 들었습니다.

2022.11.25. 22:44 답글쓰기



김성엽 M 작성자



2022.11.25. 23:46 답글쓰기



조민희 3

선생님 ! 반복해서 보면서 갑자기 궁금증이 생겨서 질문드립니다

1. C만으로 C++의 상속, 다형성, 가상함수 기능을 구현 할 수 있나요 ? (C 위에서 C++이 만들어져서 당연히 가능할 것이라고 생각하지만 혹시나 하는 생각에 질문 드립니다 ㅎㅎ)
2. C로만으로도 구현(상속, 다형성, 가상함수 등)이 가능하다면 혹시 (의도적으로) C++을 사용하지 않으시고 C만으로 프로그램을 만드시는 경우도 있으신가요 ?

2022.12.07. 22:14 답글쓰기



김성엽 M 작성자



C언어만 가지고 다형성 구현할때는 다형성 문법이 없어서 함수 포인터 사용해서 구현합니다. 나중에 함수 포인터 사용하는 것과 테이블링 기법들 익숙해지면 다형성과 동일한 구조를 구현할 수 있습니다. 그래서 유연한 구조를 가지면서 극강의 수행 능력을 가져야하는 경우에는 함수 포인터 가지고 직접 작업하기도 합니다. 하지만 개발자가 고려해야할 부분이 많아서 비용을 많이 주는 경우에만 작업합니다 ㅎㅎ

2022.12.07. 23:10 답글쓰기



조민희 3

김성엽 답변 감사드립니다,, C언어만 가지고도 다형성을 구현할 수 있을 만큼의 실력있는 개발자가 되기 위해 노력해야 겠네요 ㅎㅎ

2022.12.08. 22:02 답글쓰기



황금잉어가물치 3

위에서 Figure* p.f[2] = { new Rectangle, new Rectangle};로 할 시 E0291에러가 발생합니다. 해당 에러가 발생하는 이유가 Rectangle(int a_width, int a_height) : Figure(a_width, a_height){};가 추가되면서 발생하는게 맞는지요. 그전 예제를 보면 Rectangle(int a_width, int a_height) : Figure(a_width, a_height){};가 없으면 E0291에러가 발생하지 않고 NEW 인스턴스가 됩니다. 맞는지요?

```
class Rectangle : public Figure
{
public:
    // 동적인 두 개를 가진 생성자, 부모 객체 생성자가 두 개라서
    // : Figure(a_width, a_height) 형식으로 불리는 생성자를 호출하면 됩니다.
    Rectangle(int a_width, int a_height) : Figure(a_width, a_height)
    {
        // 부모 생성자에서 값을 대입하기 때문에 별도의 코드가 필요 없습니다.
    }

    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return m_width*m_height;
    }
};
```

2022.12.30. 20:38 답글쓰기



황금잉어가물치 3

new 객체 인스턴스 할 시 부모클래스 Figure클래스의 Figure생성자보다 Rectangle(int a_width, int a_height) : Figure(a_width, a_height){};가 먼저 호출됨에 따라 new Rectangle 호출 시 에러가 발생하는게 맞는지요?

2022.12.30. 20:46 답글쓰기



김성엽 M 작성자

new Rectangle 이라고 하면 인자가 없는 생성자가 호출되어야 하는데 Rectangle 클래스에는 인자가 없는 생성자가 없습니다. 그래서 오류가 발생하는 것입니다. 따라서 new Rectangle 처럼 인자 없이 사용하고 싶다면 Rectangle 클래스에 아래와 같이 인자가 없는 Rectangle 함수를 추가해주면 됩니다. 그리고 인자가 없는 생성자는 기본적으로 인자가 없는 부모 생성자를 호출하기 때문에 : Figure()를 사용하지 않아도 됩니다.

```
Rectangle()
{
}

}
```

2022.12.30. 21:01 답글쓰기



김성엽 M 작성자

황금잉어가물치 호출 순서의 오류가 아니라 단순히 원하는 생성자가 없어서 발생하는 오류입니다.

2022.12.30. 21:02 답글쓰기



황금잉어가물치 3

김성엽 여기에는 Rectangle(){}; 생성자가 없지만 new Rectangle을 해도 오류가 발생하지 않습니다. 부모클래스인 Figure()에서 Figure()생성자가 호출하는 것인지요? 이해가 잘 안됩니다.

```
class Rectangle : public Figure
{
public:
    double GetArea() // 도형의 넓이를 계산하는 함수
    {
        return m_width*m_height;
    }
};
```

2022.12.30. 21:07 답글쓰기



김성엽 M 작성자

황금잉어가물치 생성자가 없는 상태에서는 그냥 오류 없이 사용되도록 처리해주는 것입니다. 클래스가 반드시 생성자를 가져야 하는 것은 아니기 때문에 그냥 없으니 무시해주는 것입니다.

2022.12.30. 21:14 답글쓰기



김성엽 M 작성자

황금잉어가물치 그리고 Rectangle 함수는 무시된 상태로 생성자 호출이 되지만 부모쪽에 인자 없는 기본 생성자가 있으면 그건 찾아서 호출해주고 부모쪽도 생성자가 없으면 전체다 무시합니다.



김성엽 M 작성자

황금잉어가물치 그래서 부모 클래스에 기본 생성자(인자 없는 생성자) 없이 인자 있는 생성자만 있는 경우, 자식 클래스에 생성자가 하나도 없으면 자식 클래스를 인자 없이 new 하면 오류가 발생합니다.

2022.12.30. 21:17 답글쓰기



김성엽 M 작성자

황금잉어가물치 결론적으로 자식 클래스에 생성자가 없는 경우, new 클래스 표현은 생성자를 별도로 호출하지 않게 처리하지만 부모 클래스에 생성자가 있으면 기본 생성자(인자 없는 생성자) 호출을 시도합니다. 따라서 부모 클래스에 기본 생성자 없이 인자 있는 생성자만 있다면 자식 클래스에 기본 생성자 추가하고 : 부모클래스(인자) 와 같은 표현을 추가해야지 오류가 발생하지 않습니다.

2022.12.30. 21:19 답글쓰기



황금잉어가물치 3

김성엽 그럼 기본적으로 class안에는 인자가 없는 기본 생성자를 만들고 시작하는게 맞는지요? 구조체랑 같다고 하지만 역시 뭔가 다른것 같아서..... class를 만들때는 기본 생성자를 만드록 시작한다라는 개념을 갖는게 좋은지요?

2022.12.30. 21:22 답글쓰기



김성엽 M 작성자

황금잉어가물치 필요가 없는 경우 안만들어도 되는데 제대로된 클래스라면 대부분 멤버 변수를 가지고 있고 동적할당도 사용하기 때문에 생성자와 파괴자를 가지는 경우가 많습니다.

2022.12.30. 21:23 답글쓰기



황금잉어가물치 3

김성엽 감사합니다.

2022.12.30. 21:27 답글쓰기



모리또 2

제가 c++에서 가장 이해하기 어려웠던 부분이 이 다형성과 오버라이딩이라 강의 듣기 전에 긴장을 좀 했는데 비었던 퍼즐 조각 하나가 맞춰진 느낌입니다.

책에서는 업 / 다운 캐스팅이라는 용어와 함께 나와 몇 시간씩 붙들고 이해하고 이해되지 않은 부분은 쓰는 방법을 외워서 더듬더듬 써 왔었습니다. 강의에서 부모 클래스의 포인터로 자식 클래스의 객체를 가리켰을 때 왜 자식 클래스의 함수를 불러오지 않는지, 그러한 점 때문에 가상 함수가 제공되며, 다형성이 없는 클래스의 경우 가상 함수를 쓰지 않아도 된다 까지. 너무나 명쾌한 정리였습니다. 스스로가 이해하지 못했다고 생각한 파트라 문제들을 접할 때에도 이게 맞나 싶었는데 무언가 숨이 트인 듯이 이제 확실한 가이드 라인이 생긴 것 같은 느낌입니다 ㅎㅎ

2024.01.30. 23:36 답글쓰기



김성엽 M 작성자

다형성이 C++ 언어 공부에서 제일 중요한 부분이라 제일 어렵기도 하죠 :) 그래도 제 강좌가 도움이 되듯해서 기쁘네요! 그리고 다형성 같은 강좌는 최소 5회 정도는 보세요. 안다고 생각해도 반복해서 보면서 익숙해지는것이 중요합니다. 그래야지 상황이 닥쳤을때 사용할수 있거든요 ㅎㅎ 파이팅입니다 :)



화이팅!

2024.01.31. 11:49 답글쓰기



모리또 2

김성엽 님 여러번 반복해서 보겠습니다!!

2024.01.31. 20:52 답글쓰기

dh221009

댓글을 남겨보세요



등록