23장. 다형성과 가상 함수

01_ 가상 함수를 사용한 다형성의 구현

02_ 오버라이딩

Shape 클래스와 자식 클래스들(1)

- 가상 함수가 없으면 문제가 발생하는 경우를 알아보자.

```
// 일반적인 '도형'을 상징하는 클래
class Shape
{
public:
          void Move(double x, double y);
          void Draw() const;

          Shape();
          Shape(double x, double y);

protected:
          double _x, _y;
};

void Shape::Draw() const
{
          cout << "[Shape] Position = ( " << _x << ", " << _y << ")\n";
}
```

Shape 클래스와 자식 클래스들(2)

- 가상 함수가 없으면 문제가 발생하는 경우를 알아보자.

```
// 사각형을 상징하는 클래스
class Rectangle : public Shape
{
public:
          void Draw() const;
          void Resize(double width, double height);

          Rectangle();
          Rectangle(double x, double y, double width, double height);

protected:
          double _width;
          double _height;

};

void Rectangle::Draw() const
{
          cout << "[Rectangle] Position = ( " << _x << ", " << _y << ") "
                    "Size = ( " << _width << ", " << _height << ")\n";
}
```

Shape 클래스와 자식 클래스들(3)

- 가상 함수가 없으면 문제가 발생하는 경우를 알아보자.

```
// 원을 상징하는 클래스
class Circle : public Shape
{
public:
          void Draw() const;
          void SetRadius(double radius);

          Circle();
          Circle(double x, double y, double radius);

protected:
          double _radius;
};

void Circle::Draw() const
{
          cout << "[Circle] Position = ( " << _x << ", " << _y << ") "
                    "Radius = " << _radius << "\n";
}
```

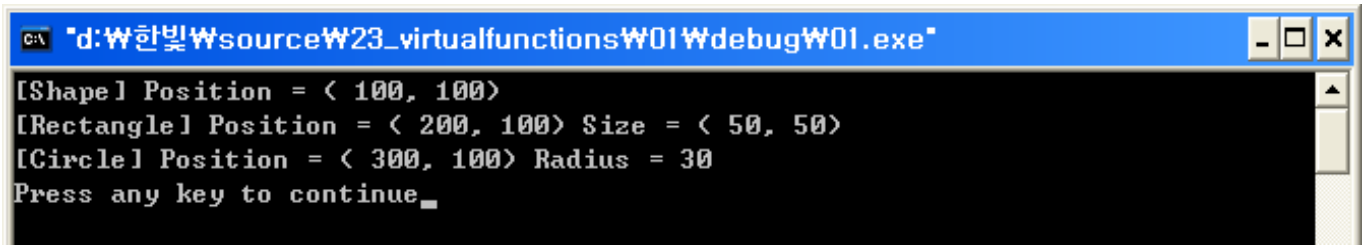
Shape 클래스와 자식 클래스들(4)

- 가상 함수가 없으면 문제가 발생하는 경우를 알아보자.

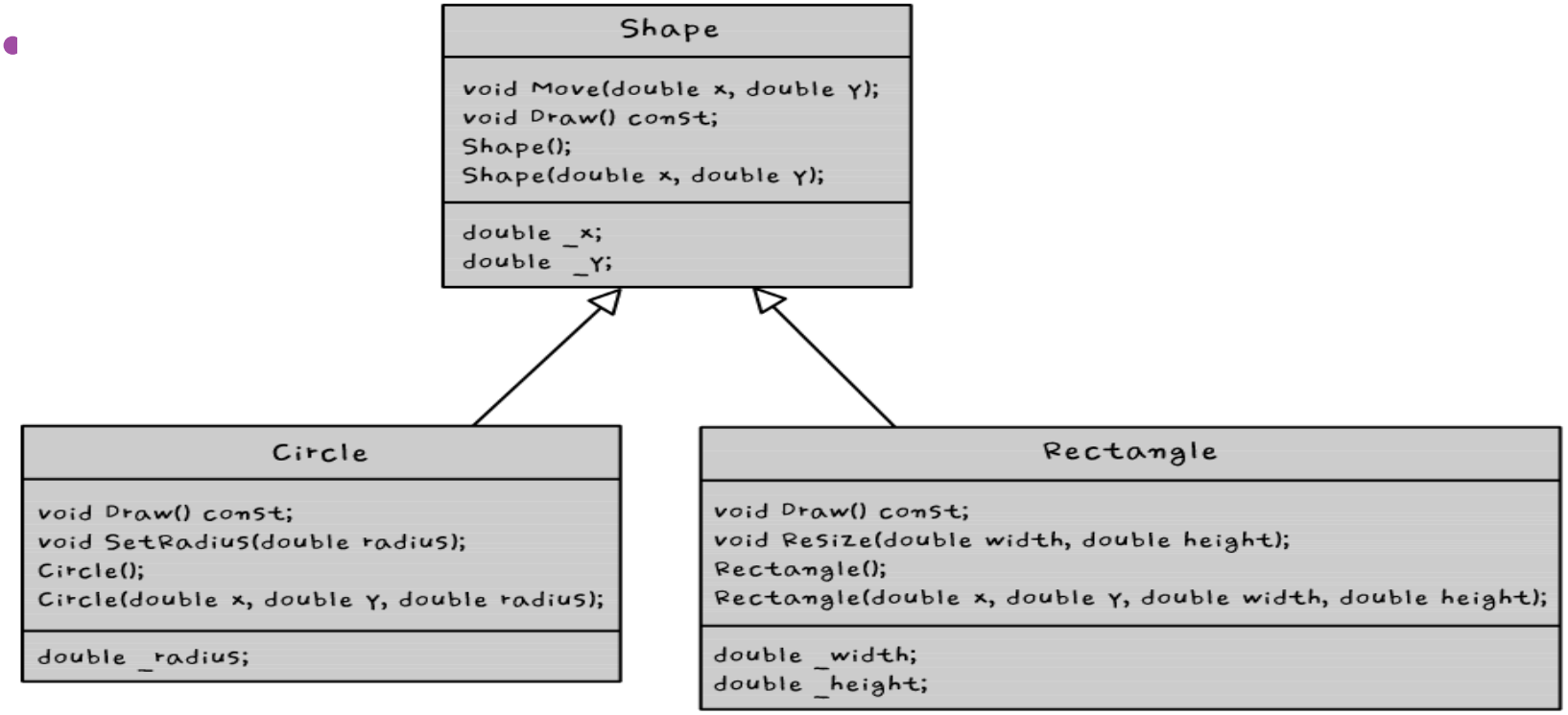
```
int main()
{

        // 도형 객체 생성 및 그리기
        Shape s;
        s.Move(100, 100);
        s.Draw();

        // 사각형 객체 생성 및 그리기
        Rectangle r;
        r.Move( 200, 100);
        r.Resize( 50, 50);
        r.Draw();

        // 원 객체 생성 및 그리기
        Circle c;
        c.Move( 300, 100);
        c.SetRadius( 30);
        c.Draw();

        return 0;
}
```

Shape 클래스와 자식 클래스들(5)

- 실행 결과



-

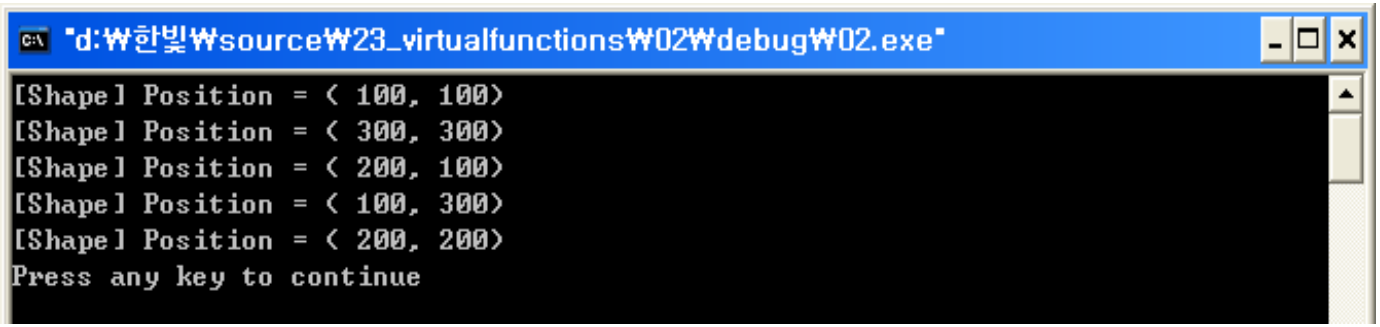


다양한 클래스의 객체를 배열에 담기(1)

- 도형 클래스의 객체들을 배열에 담아서 사용하는 예

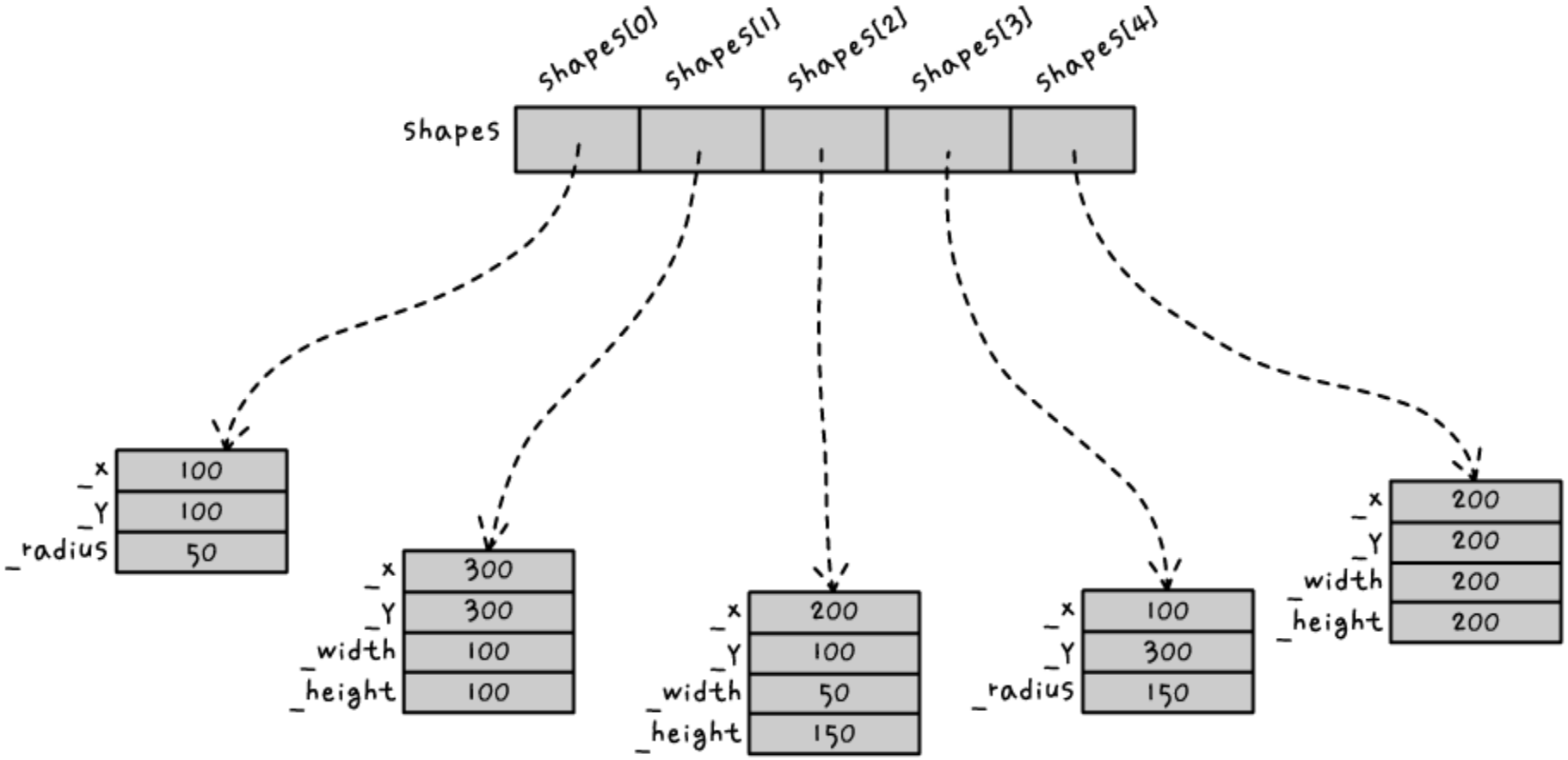
```
Shape* shapes[5] = {NULL};

shapes[0] = new Circle( 100, 100, 50);
shapes[1] = new Rectangle( 300, 300, 100, 100);
shapes[2] = new Rectangle( 200, 100, 50, 150);
shapes[3] = new Circle(100, 300, 150);
shapes[4] = new Rectangle( 200, 200, 200, 200);

for (int i = 0; i < 5; ++i)
          shapes[i]->Draw();

for (i = 0; i < 5; ++i)
{
          delete shapes[i];
          shapes[i] = NULL;
}
```

- 실행 결과 – 항상 Shape::Draw() 가 호출되는 문제가 있다.



다양한 클래스의 객체를 배열에 담기(2)

- 배열과 객체들의 메모리 구조

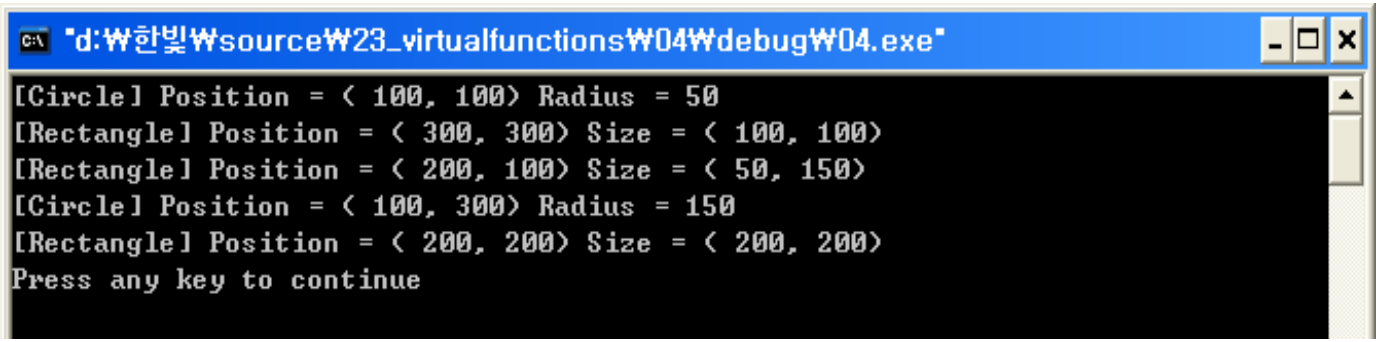


가상 함수로 문제점 해결하기

- Draw() 함수를 가상 함수로 만들어서 문제를 해결해보자.

```
class Shape
{
public:
          void Move(double x, double y);
          virtual void Draw() const;

          Shape();
          Shape(double x, double y);

protected:
          double _x, _y;
};
```

- 실행 결과 – 알맞은 클래스의 Draw() 함수가 호출된다.



다형성(Polymorphism)과 가상함수(1)

- 다른 분야에서의 다형성의 의미

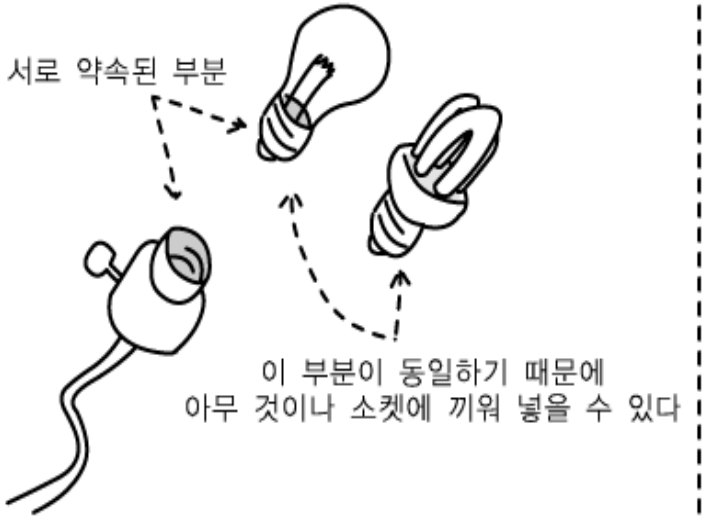
생물학	동일종의 생물이면서 형태나 성질이 다르게 보이는 다양성. 생물은 본래 동일종이라도 완전히 일치하는 개체는 거의 없으므로, 이 말은 상대적으로 현저한 차이가 있을 경우에 한해서 사용한다. 다만, 암수의 성별에 따른 2차 성징(二次性徵)의 차이에는 쓰지 않는다.
화학	화학조성이 같은 물질로써 결정구조를 달리하는 것. 다형(多形), 동질이형(同質異形), 동질다상(同質多像), 동질이정(同質異晶)이라고도 한다. 동질다상은 결정구조의 수에 따라 동질이상(同質二像), 동질삼상(同質三像) 등으로 나뉜다.

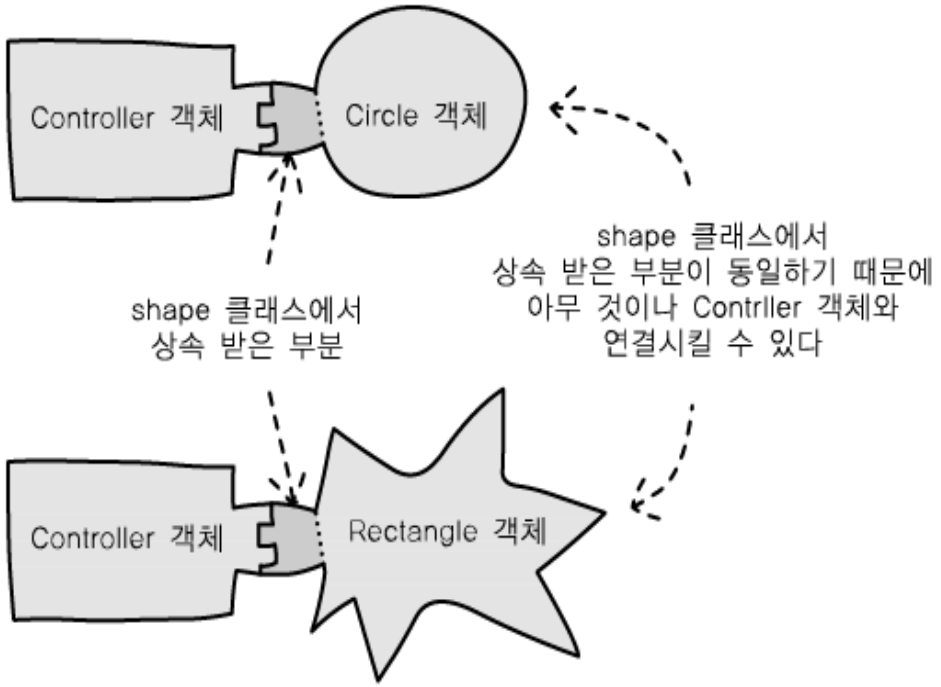
- 객체지향 프로그래밍에서의 다형성이란 타입에 관계없이 동일한 방법으로 다룰 수 있는 능력을 말한다.
  - 예) Circle이나 Rectangle 객체들을 각각의 타입에 상관없이 Shape 객체처럼 다룰 수 있는 능력

다형성(Polymorphism)과 가상함수(2)

- 다형성은 객체간의 결합(Coupling)을 약하게 만들어서, 객체 간의 연결을 유연하게 해준다.

```
// 도형을 원점으로 이동하는 함수
void Controller::MoveToOrigin( Shape* p )
{
          p->Move( 0, 0 );
          p->Draw();
}
```



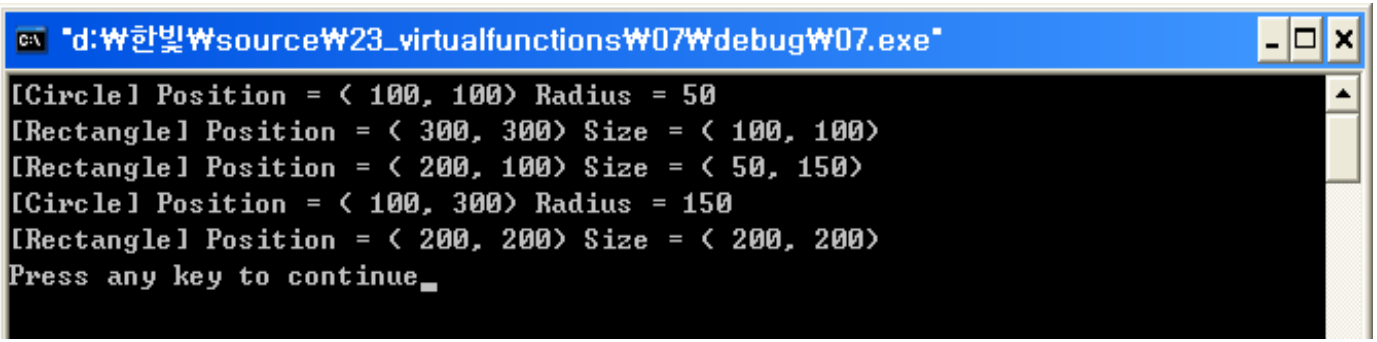


순수 가상 함수(Pure Virtual Functions)

- Shape::Draw() 함수를 순수 가상 함수로 만들어보자.

```
class Shape
{
public:
          void Move(double x, double y);
          virtual void Draw() const = 0;

// 중간 생략

/* 함수의 정의를 지워도 컴파일 오류가 발생하지 않는다.
void Shape::Draw() const
{
          cout << "[Shape] Position = ( " << _x << ", " << _y << ")\n";
}
*/
```

- 실행 결과 (이전 버전과 다른 점이 없다.)



순수 가상 함수의 의미

- 하나 이상의 순수 가상 함수를 포함하고 있는 클래스를 추상 클래스(Abstract Class)라고 부른다.
  - 추상 클래스 타입의 의 객체를 생성하는 것은 불가능하고, 오로지 부모 클래스로서만 사용할 수 있다.

- Shape::Draw() 함수를 순수 가상 함수로 만드는 것의 효과
  - Shape 클래스를 추상 클래스로 만들기 때문에 Shape 클래스의 객체를 만들 수 없다.
    - 컴파일러나 다른 개발자들에게 Shape 클래스는 오로지 부모 클래스로만 사용할 것이라는 점을 알리는 역할을 한다.

  - Shape::Draw() 함수를 구현하지 않아도 컴파일 오류가 발생하지 않는다.
    - 컴파일러나 다른 개발자들에게 Shape::Draw() 함수가 자식 클래스에 의해서 오버라이딩 될 것이며, 다형성을 통해서만 호출할 것이라는 점을 알리는 역할을 한다.

다양한 종류의 멤버 함수

- 상속과 관련해서 지금까지 살펴본 멤버 함수의 종류
  - 일반적인 멤버 함수
  - 가상 함수
  - 순수 가상 함수

- 어떤 종류의 멤버 함수를 사용할 지에 대한 가이드 라인
  - 처음엔 그냥 멤버 함수로 만든다.
  - 다형성을 이용해야 하는 경우라면 가상 함수로 만든다.
  - 다형성을 위해서 함수의 원형만 필요한 경우라면 순수 가상 함수로 만든다.

오버로딩과 오버라이딩

- 부모 클래스에서 오버로드된 함수 중에서 어느 것 하나라도 오버라이드 하면 나머지 다른 함수들도 모두 사용할 수 없다.

```
class Pet
{
public:
          void Eat();
          void Eat(const string& it);

          string name;
};

class Dog : public Pet
{
public:
          void Eat();
};

int main()
{
          // 강아지 생성
          Dog dog1;
          dog1.name = "Patrasche";

          // 두 가지 Eat() 함수를 호출한다.
          dog1.Eat();
          dog1.Eat( "milk" );           // Error!!

          return 0;
}
```

Q&A