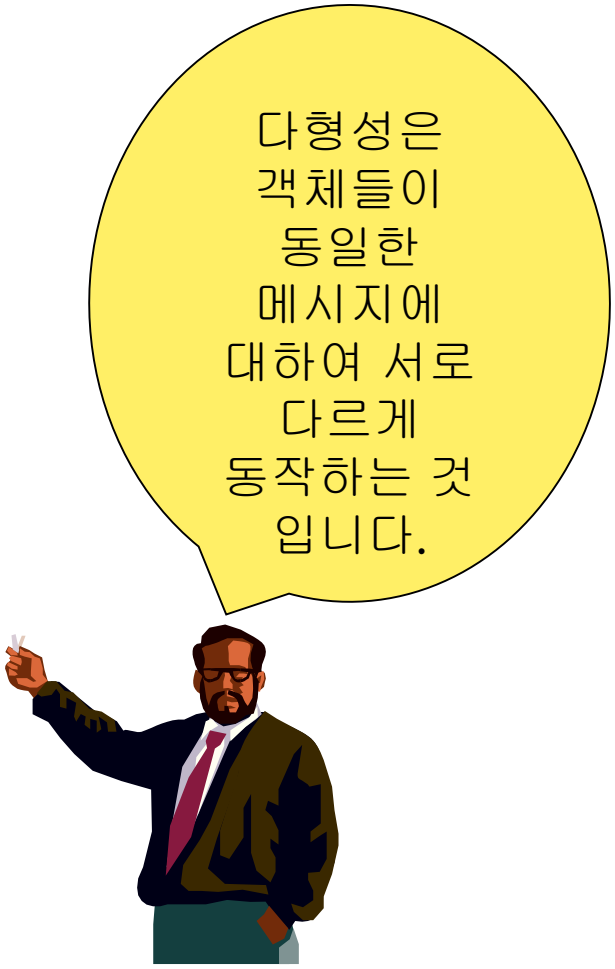
Power C++

제14장 다형성

C / C++ 전부 다형성 가능
C++는 C의 함수 포인터로
이루어진 다형성에서 나옴.

이번 장에서 학습할 내용

- 다형성
- 가상 함수
- 순수 가상 함수

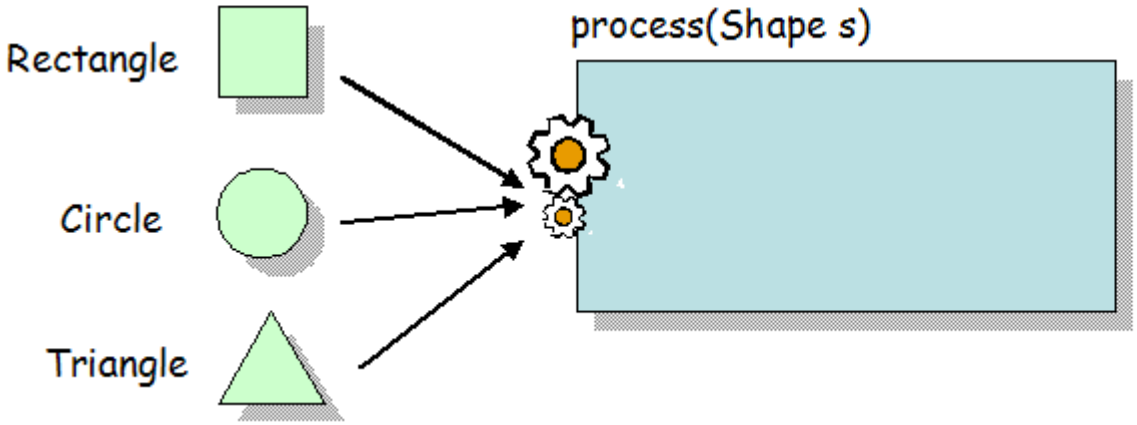


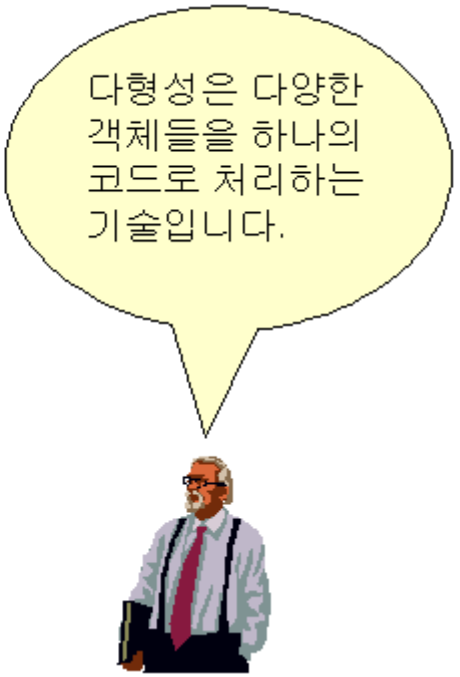
다형성이란?

- 다형성(polymorphism)이란 객체들의 타입이 다르면 똑같은 메시지가 전달되더라도 서로 다른 동작을 하는 것
- 다형성은 객체 지향 기법에서 하나의 코드로 다양한 타입의 객체를 처리하는 중요한 기술이다.

자신의 형태에 맞도록 function이 돌아가는 것

다형성이란?





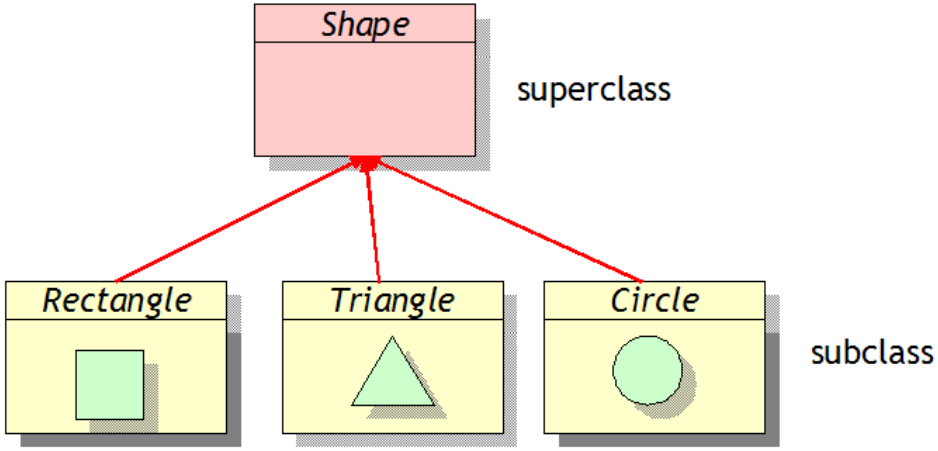
객체 간의 형변환

- 먼저 객체 간의 형변환을 살펴보자.

객체 간의 형변환

- 상향 형변환(upcasting):
  자식 클래스 타입을 부모 클래스타입으로 변환
- 하향 형변환(downcasting):
  부모 클래스 타입을 자식 클래스타입으로 변환

상속과 객체 참조



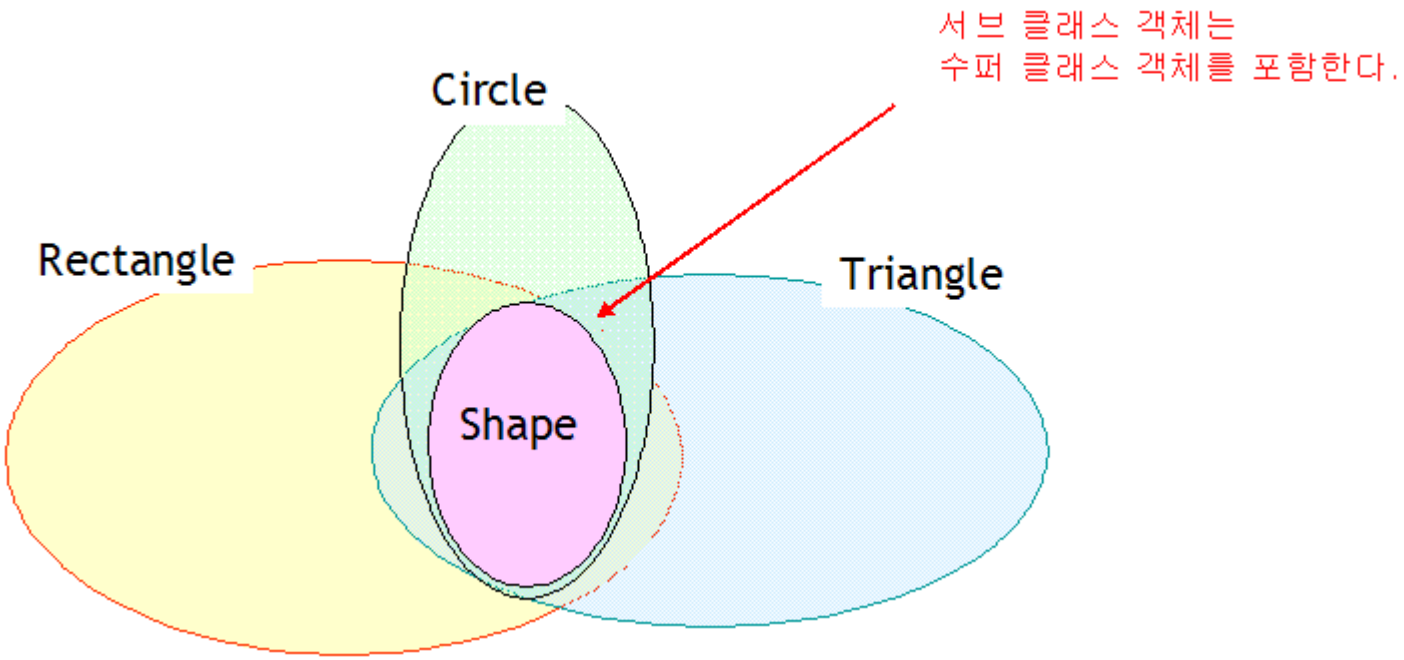
Shape 타입 포인터로 Rectangle 객체를 참조하니 틀린 거 같지만 올바른 문장!!

```
Shape *ps = new Rectangle(); // OK!
```

관리는 최상위에서!!!!!!!!
*ps : ps로 가면 Shape이 있다는 뜻
거기에 Rectangle이 있다는 것은 아직 모름

왜 그럴까?

- 서브 클래스 객체는 수퍼 클래스 객체를 포함하고 있기 때문이다.



도형 예제

```
class Shape {
protected:
	int x, y;

public:
	void setOrigin(int x, int y){
		this->x = x;
		this->y = y;
	}
	void draw() {
		cout <<"Shape Draw";
	}
};
```

도형 예제

```
class Rectangle : public Shape {
private:
    int width, height;
public:
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }
    void draw() {
        cout << "Rectangle Draw";
    }
};
```

상향 형변환

관리는 최상위 클래스에서 구현은 하위 클래스에서!!!!!!!!!!!!!!!

- Shape *ps = new Rectangle(); // OK!
- ps->setOrigin(10, 10); // OK!

상향 형변환

*ps 는 그쪽으로 가면 Shape이 있다는 것만을 알려주기 때문에, x, y 변수 / setOrigin(), draw() 는 접근 가능. Rectangle()로 만들었음에도 불구하고 setWidth()는 접근 X

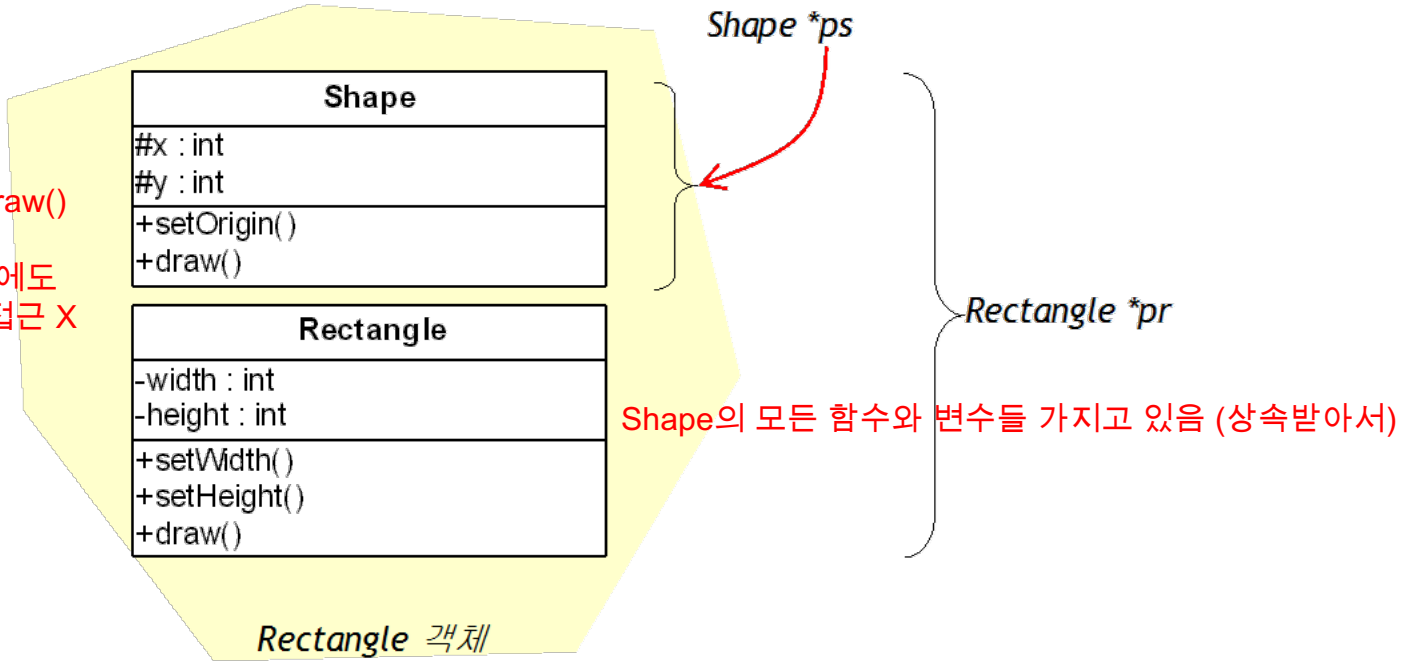


그림 14.4 Rectangle 객체를 Shape 포인터로 가리키면 Shape에 정의된 부분밖에 가리키지 못한다.

하향 형변환

- Shape *ps = new Rectangle();
- 여기서 ps를 통하여 Rectangle의 멤버에 접근하려면?

1. Rectangle *pr = (Rectangle *) ps;
   pr->setWidth(100);

ps가 130번지였으면 pr도 130
Shape*은 130 ~137까지 접근할 수 있음
Rectangle* 로 바꿨기 때문에 130~ 145번지까지 접근할 수 있음 -> 포인터 type의 중요성
pr의 번지수가 Rectangle의 끝까지 볼 수 있도록 바꼈기 때문에 setWidth()해도 에러가 안남
down casting / up casting 모두 가능

2. ((Rectangle *) ps)->setWidth(100);

하향 형변환

예 제

```
#include <iostream>
using namespace std;

class Shape {                          // 일반적인 도형을 나타내는 부모 클래스
protected:
        int x, y;

public:
        void draw() {
                cout <<"Shape Draw" << endl;
        }
        void setOrigin(int x, int y){
                this->x = x;
                this->y = y;
        }
};
```

예제

```
class Rectangle : public Shape {
private:
	int width, height;
public:
	void setWidth(int w) {
		width = w;
	}
	void setHeight(int h) {
		height = h;
	}
	void draw() {
		cout << "Rectangle Draw"<< endl;
	}
};
```

예제

```
class Circle : public Shape {
private:
	int radius;

public:
	void setRadius(int r) {
		radius = r;
	}
	void draw() {
		cout << "Circle Draw"<< endl;
	}
};
```

예제

```
int main()
{
    Shape *ps = new Rectangle();                // OK!
    ps->setOrigin(10, 10);
    ps->draw();
    ((Rectangle *)ps)->setWidth(100);           // Rectangle의 setWidth() 호출
    delete ps;

    Circle c;
    Shape &s = c;                               // OK!
    s.setOrigin(10, 10);
    s.draw();
    ((Circle *)ps)->setRadius(5);               // Circle의 setRadius() 호출
}
```

```
Shape Draw
Shape Draw
계속하려면 아무 키나 누르십시오 . . .
```

함수의 매개 변수

- 이와 같은 형변환 규칙은 함수 호출시에도 그대로 적용된다. 따라서 함수의 매개 변수는 자식 클래스보다는 부모 클래스 타입으로 선언하는 것이 좋다.

```
void move(Shape& s, int sx, int sy)
{
    s.setOrigin(sx, sy);
}

int main()
{
    Rectangle r;
    move(r, 0, 0);

    Circle c;
    move(c, 10, 10);

    return 0;
}
```

모든 도형을 받을 수 있다.

중간 점검 문제

1. 부모 클래스 포인터 변수는 자식 클래스 객체를 참조할 수 있는가? 역은 성립하는가?

2. 부모 클래스 포인터로 자식 클래스에만 정의된 함수를 호출할 수 있는가?

가상 함수

- 단순히 자식 클래스 객체를 부모 클래스 객체로 취급하는 것이 어디에 쓸모가 있을까?
- 다음과 같은 상속 계층도를 가정하여 보자.

circle -> shape -> rectangle로 캐스팅을 할 경우, 캐스팅엔 문제가 없지만 circle에서 정의하지 않은변수나 함수를 부르면 프로그램이 죽음

Shape라는 상위 클래스가 실체를 가질 수 있을 순 없으니까 virtual 사용.
Shape의 draw()는 함수의 포인터를 가지고 있음.
Rectangle, Triangle, Circle 모두의 포인터를 가지고 있어야 하기 때문에 이것이 virtual table에 저장되어 있음

virtual draw()함수를 Shape, Rectangle, Triangle만 짰을 때 Circle의 draw를 호출하면 Shape의 draw() 함수를 호출함. 만약에 Shape와 Circle 둘 다 draw() 구현체가 없으면 컴파일 에러.

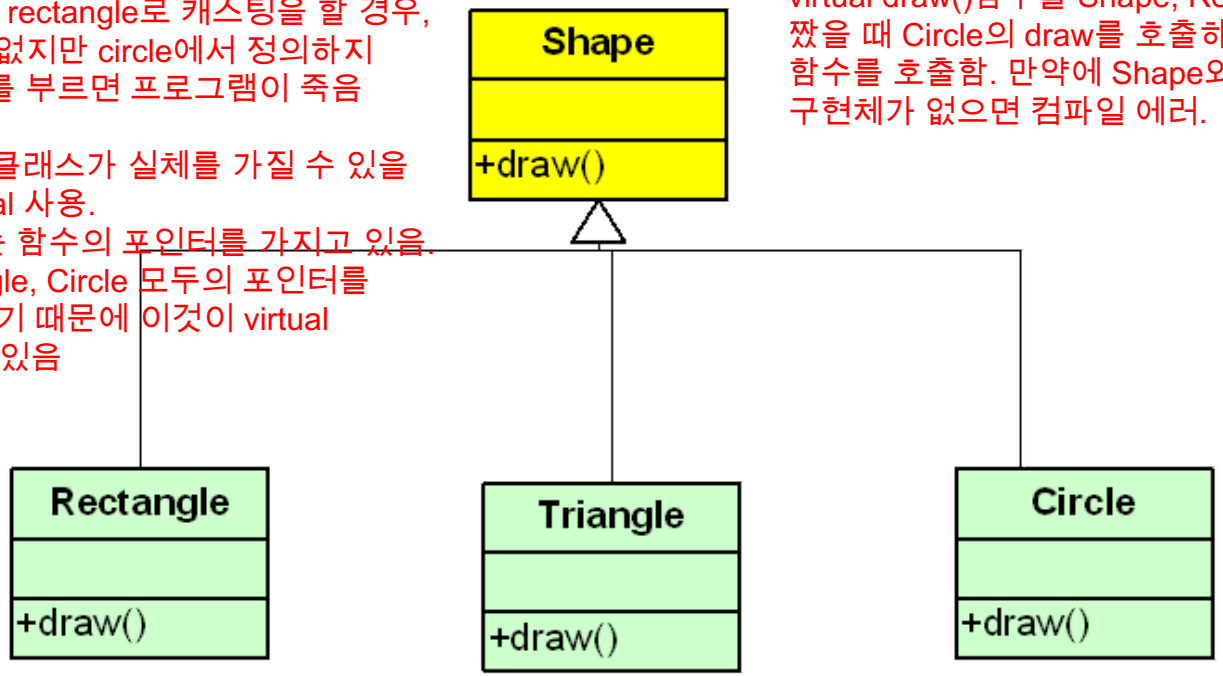


그림 14.6 도형의 UML

예제

```
class Shape {
    ...
}
class Rectangle : public Shape {
    ...
}
int main()
{
    Shape *ps = new Rectangle();        // OK!
    ps->draw();                         // 어떤 draw()가 호출되는가?
}
```

Shape
포인터이기
때문에
Shape의
draw()가 호출

Shape Draw

가상 함수

- 만약 Shape 포인터를 통하여 멤버 함수를 호출하더라도 도형의 종류에 따라서 서로 다른 draw()가 호출된다면 상당히 유용할 것이다.

- 즉 사각형인 경우에는 사각형을 그리는 draw()가 호출되고 원의 경우에는 원을 그리는 draw()가 호출된다면 좋을 것이다.

-> draw()를 가상 함수로 작성하면 가능

가상 함수

```
#include <iostream>
#include <string>
using namespace std;

class Shape {
protected:
	int x, y;

public:
	void setOrigin(int x, int y){
		this->x = x;
		this->y = y;
	}
	virtual void draw() {
		cout <<"Shape Draw" << endl;
	}
};
```

가상 함수 정의

예제

```
int main()
{
    Shape *ps = new Rectangle();          // OK!
    ps->draw();                           // Rectangle의 draw()가 호출된다.
    delete ps;

    Circle c;
    Shape &s = c;                 // OK!
    s.draw();                             // Circle의 draw()가 호출된다.
    return 0;
}
```

```
Rectangle Draw
Circle Draw
계속하려면 아무 키나 누르십시오 . . .
```

동적 바인딩

- 컴파일 단계에서 모든 바인딩이 완료되는 것을 정적 바인딩(static binding)이라고 한다.
- 반대로 바인딩이 실행 시까지 연기되고 실행 시간에 실제 호출되는 함수를 결정하는 것을 동적 바인딩(dynamic binding), 또는 지연 바인딩(late binding)이라고 한다.

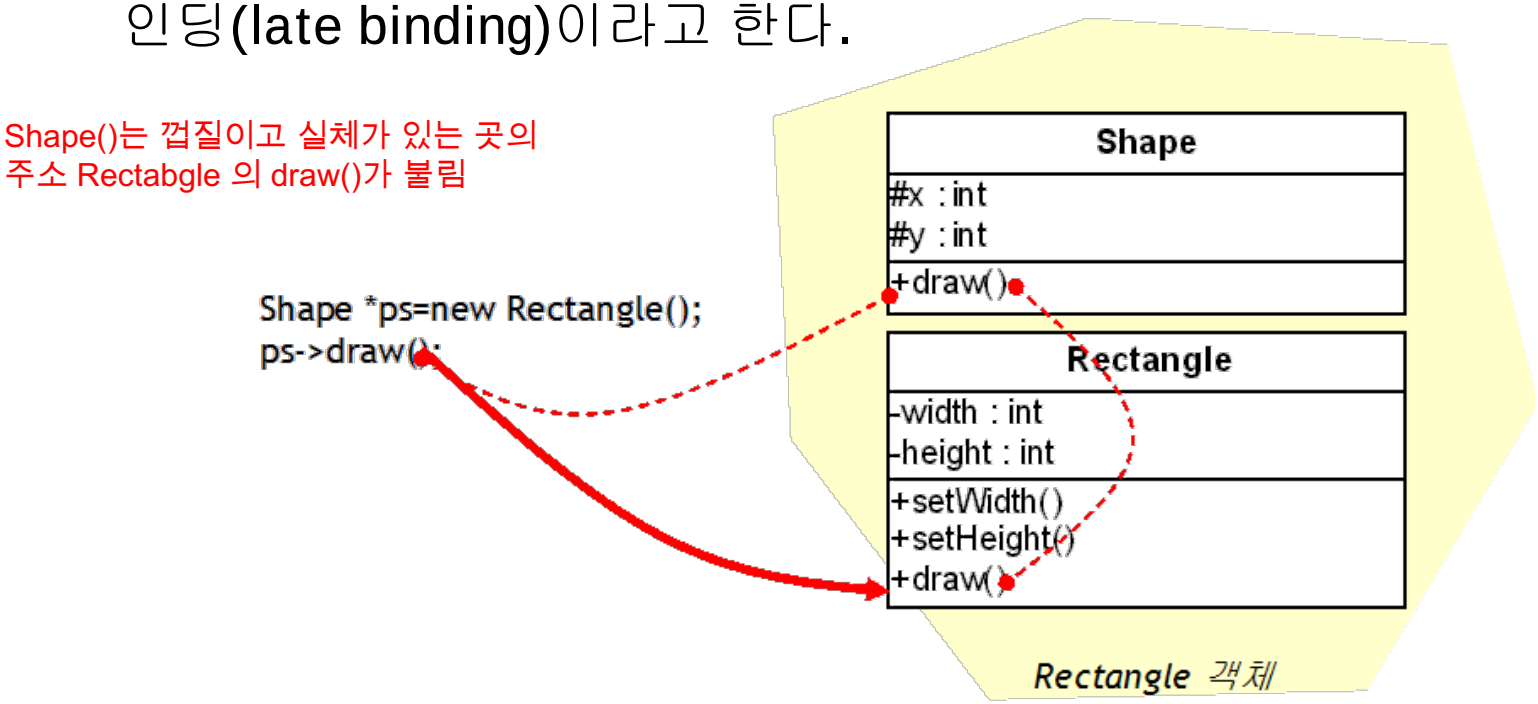


그림 14.8 동적 바인딩

정적 바인딩과 동적 바인딩

바이딩의 종류	특징	속도	대상
정적 바인딩 (dynamic binding)	컴파일 시간에 호출 함수가 결정된다.	빠르다	일반 함수
동적 바인딩 (static binding)	실행 시간에 호출 함수가 결정된다.	늦다	가상 함수

가상 함수의 구현

- V-table을 사용한다.

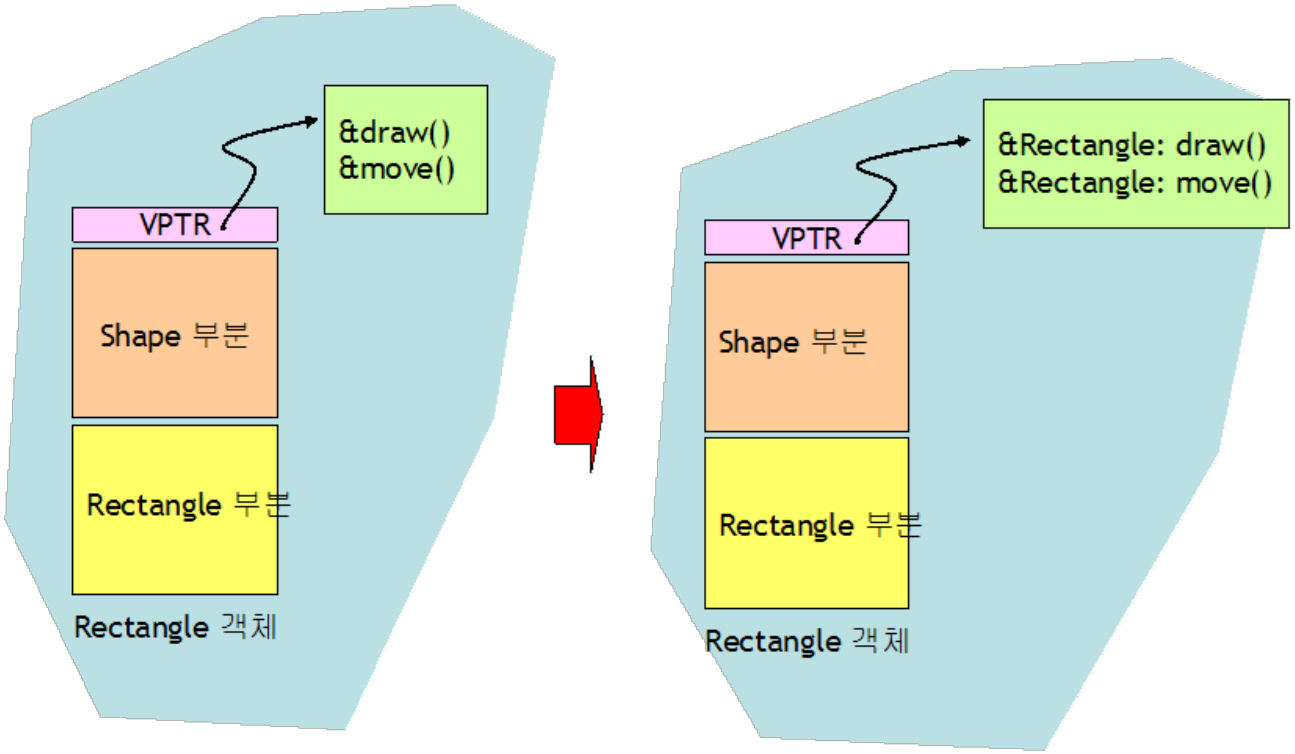


그림 14.9 가상 함수의 구현

예제

```
#include <iostream>
using namespace std;

class Shape {
protected:
	int x, y;

public:
	virtual void draw() {
		cout <<"Shape Draw";
	}
	void setOrigin(int x, int y){
		this->x = x;
		this->y = y;
	}
};
```

예제

```
class Rectangle : public Shape {
private:
	int width, height;

public:
	void setWidth(int w) {
		width = w;
	}

	void setHeight(int h) {
		height = h;
	}

	void draw() {
		cout << "Rectangle Draw" << endl;
	}
};
```

예제

```
class Triangle: public Shape {
private:
	int base, height;
public:
	void draw() {
		cout << "Triangle Draw" << endl;
	}
};

int main()
{
	Shape *arrayOfShapes[3];

	arrayOfShapes[0] = new Rectangle();
	arrayOfShapes[1] = new Triangle();
	arrayOfShapes[2] = new Circle();
	for (int i = 0; i < 3; i++) {
		arrayOfShapes[i]->draw();
	}
}
```

예 제

Rectangle Draw
Triangle Draw
Circle Draw

다형성의 장점

- 새로운 도형이 추가되어도 main()의 루프는 변경할 필요가 없다.

```
class Parallelogram extends Shape
{
public:
    void draw(){
            cout << "Parallelogram Draw" << endl;
    }
};
```

예제

```
#include <iostream>
using namespace std;

class Animal
{
public:
    Animal() { cout <<"Animal 생성자" << endl; }
    ~Animal() { cout <<"Animal 소멸자" << endl; }
    virtual void speak() { cout <<"Animal speak()" << endl; }
};

class Dog : public Animal
{
public:
    Dog() { cout <<"Dog 생성자" << endl; }
    ~Dog() { cout <<"Dog 소멸자" << endl; }
    void speak() { cout <<"멍멍" << endl; }
};
```

예제

```
class Cat : public Animal
{
public:
    Cat() { cout <<"Cat 생성자" << endl; }
    ~Cat() { cout <<"Cat 소멸자" << endl; }
    void speak() { cout <<"야옹" << endl; }
};

int main()
{
    Animal *a1 = new Dog();
    a1->speak();

    Animal *a2 = new Cat();
    a2->speak();
    return 0;
}
```

예제

```
Animal 생성자
Dog 생성자
멍멍
Animal 소멸자
Animal 생성자
Cat 생성자
야옹
Animal 소멸자
```

소멸자 문제

- 다형성을 사용하는 과정에서 소멸자를 virtual로 해주지 않으면 문제가 발생한다.

- 문자열을 나타내는 String 클래스를 작성하여 보자. String 클래스는 내부에는 문자열을 저장하기 위하여 char 배열을 동적으로 생성한다. 따라서 소멸자에서는 반드시 동적 생성된 배열을 삭제하여야 한다.

소멸자 문제

```
#include <iostream>
using namespace std;

class String {
	char *s;
public:
	String(char *p){
		s = new char[strlen(p)+1];
		strcpy(s, p);
	}
	~String(){
		cout << "String() 소멸자" << endl;
		delete[] s;
	}
};
```

소멸자 문제

```
class MyString : public String {
    char *header;
public:
    MyString(char *h, char *p) : String(p){
        header = new char[strlen(h)+1];
        strcpy(header, h);
    }
    ~MyString(){
        cout << "MyString() 소멸자" << endl;
        delete[] header;
    }
};
```

소멸자 문제

```
int main()
{
	cout << "자식 클래스 포인터 이용"<< endl;
	MyString *s1 = new MyString("//////", "Hello World!");
	delete s1;
	cout << endl;

	cout << "부모 클래스 포인터 이용"<< endl;
	String *s2 = new MyString("*****", "Hello World!");
	delete s2;

	return 0;
}
```

```
자식 클래스 포인터 이용
MyString() 소멸자
String() 소멸자

부모 클래스 포인터 이용
String() 소멸자
계속하려면 아무 키나 누르십시오 . . .
```

MyString의 소멸자가 호출되지 않음

virtual 사용

소멸자 문제

```
class String {
        char *s;
public:
        String(char *p){
                s = new char[strlen(p)+1];
                strcpy(s, p);
        }
        virtual ~String(){
                cout << "String() 소멸자" << endl;
                delete[] s;
        }
};
class MyString : public String {
        ...// 앞과 동일
};
```

자식 클래스 포인터 이용
MyString() 소멸자
String() 소멸자

부모 클래스 포인터 이용
MyString() 소멸자
String() 소멸자

중간 점검 문제

1. 가상 함수가 필요한 이유는 무엇인가?

2. 어떤 경우에 부모 클래스의 소멸자에 **virtual**을 붙여야 하는가?

순수 가상 함수

- 순수 가상 함수(pure virtual function): 함수 헤더만 존재하고 함수의 몸체는 없는 함수

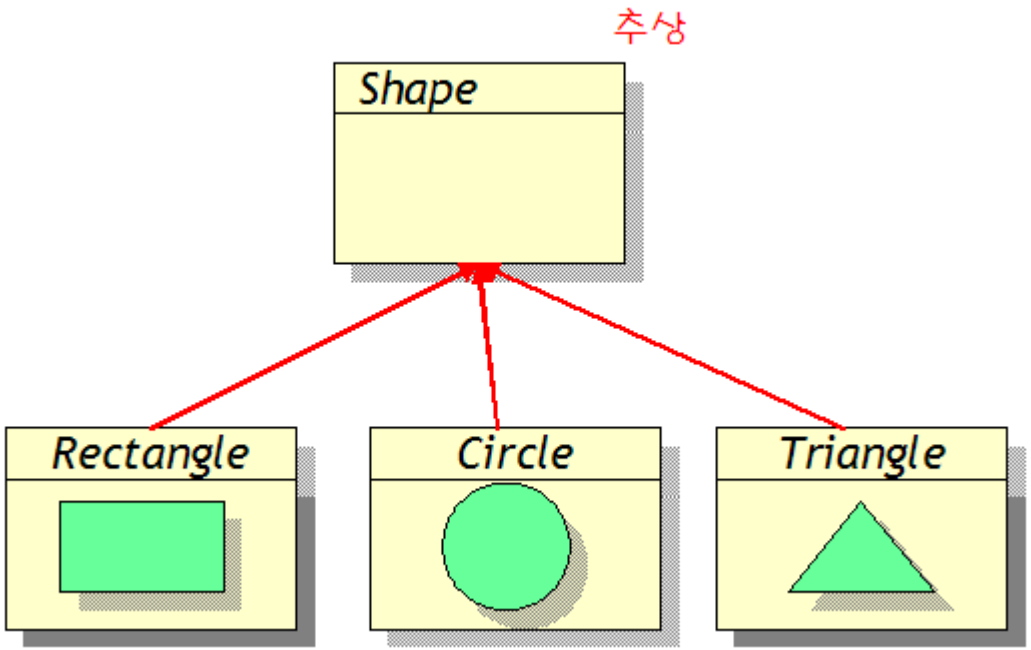
```
virtual     반환형        함수이름(매개변수  리스트) = 0;
```

(abstract function)

- (예) virtual void draw() = 0;

- 추상 클래스(abstract class): 순수 가상 함수를 하나라도 가지고 있는 클래스

추상 클래스의 예



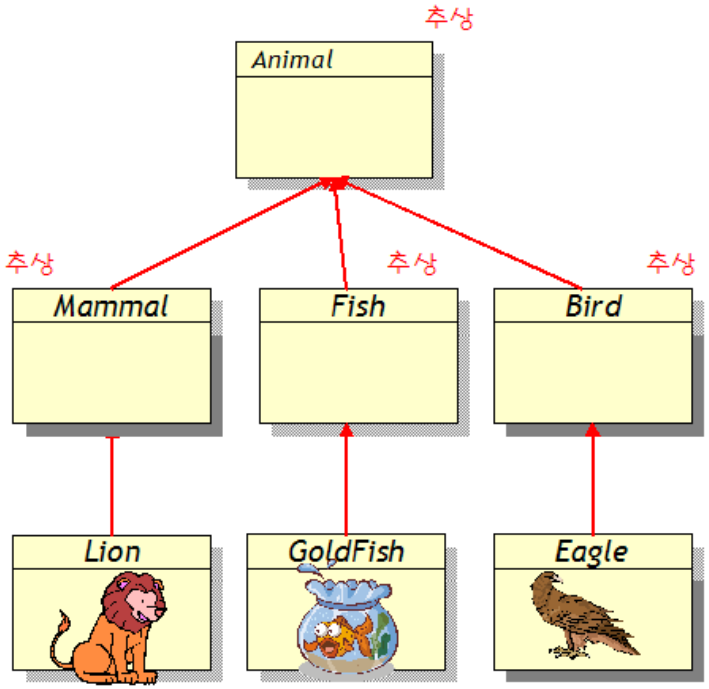
추상 클래스의 예

```
class Shape {
protected:
        int x, y;

public:
        ...
        virtual void draw() = 0;
};

class Rectangle : public Shape {
private:
        int width, height;

public:
        void draw() {
                cout << "Rectangle Draw" << endl;
        }
};
```

자바에서 implement니까 반드시 구현을 해줘야 함

추상 클래스

- 추상 클래스(abstract class): 순수 가상 함수를 가지고 있는 클래스
- 추상 클래스는 추상적인 개념을 표현하는데 적당하다.

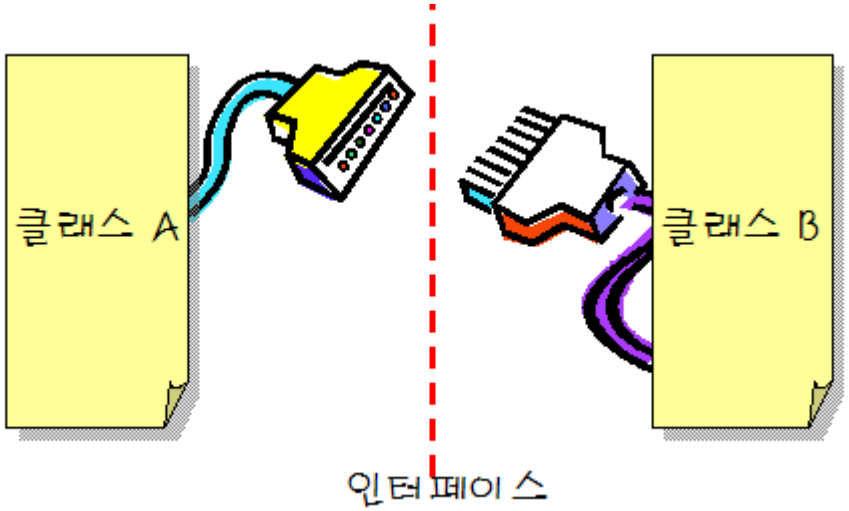


예제

```
class Animal {
    virtual void move() = 0;
    virtual void eat() = 0;
    virtual void speak() = 0;
};
class Lion : public Animal {
    void move(){
        cout << "사자의 move() << endl;
    }
    void eat(){
        cout << "사자의 eat() << endl;
    }
    void speak(){
        cout << "사자의 speak() << endl;
    }
};
```

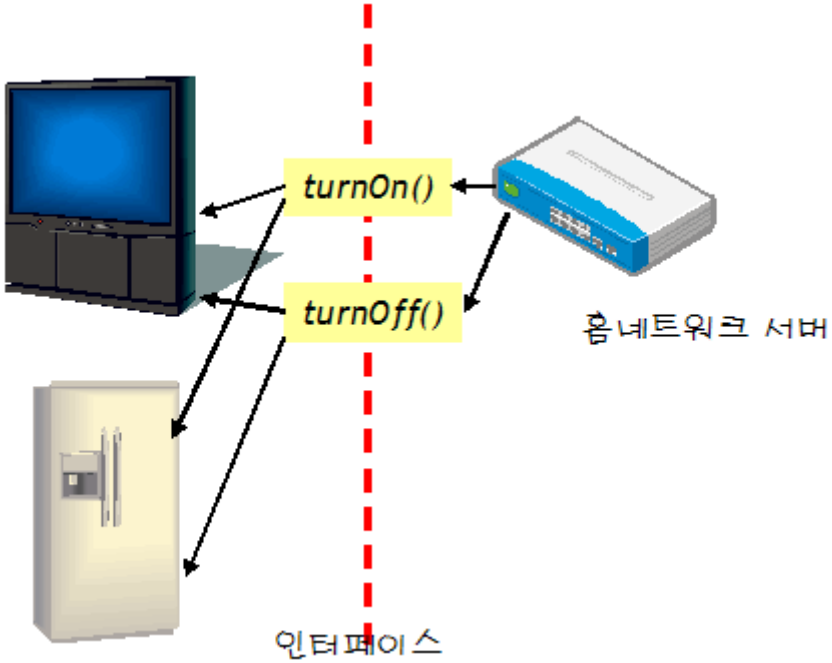
추상 클래스를 인터페이스로

- 추상 클래스는 객체들 사이에 상호 작용하기 위한 인터페이스를 정의하는 용도로 사용할 수 있다.



인터페이스의 예

- 홈 네트워킹 예제



예제

```
class RemoteControl {
      // 순수 가상 함수 정의
      virtual void turnON() = 0;       // 가전 제품을 켠다.
      virtual void turnOFF() = 0;      // 가전 제품을 끈다.
}

class Television : public RemoteControl {
      void turnON()
      {
            // 실제로 TV의 전원을 켜기 위한 코드가 들어 간다.
            …
      }
      void turnOFF()
      {
            // 실제로 TV의 전원을 끄기 위한 코드가 들어 간다.
            …
      }
}
```

예제

```
int main()
{

    Television *pt = new Television();
    pt->turnOn();
    pt->turnOff();

    Refrigerator *pr = new Refrigerator();
    pr->turnOn();
    pr->turnOff();

    delete pt;
    delete pr;
    return 0;
}
```

중간 점검 문제

1. 순수 가상 함수의 용도는?

2. 모든 순수 가상 함수를 구현하여야 하는가?

Q & A