SEOIL UNIVERSITY

다형성 / 가상함수

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 기술

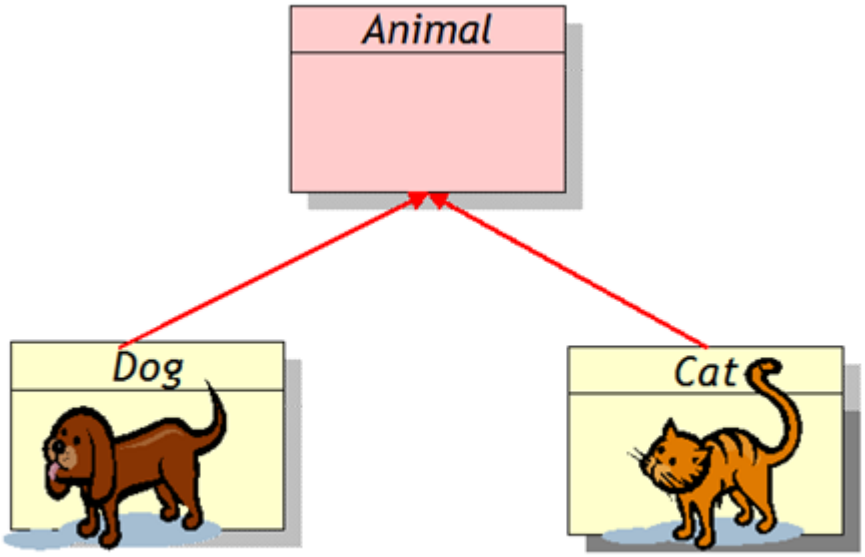
객체 포인터의 형변환

객체 포인터의 형변환

객체 포인터의 형변환

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

상속과 객체 포인터



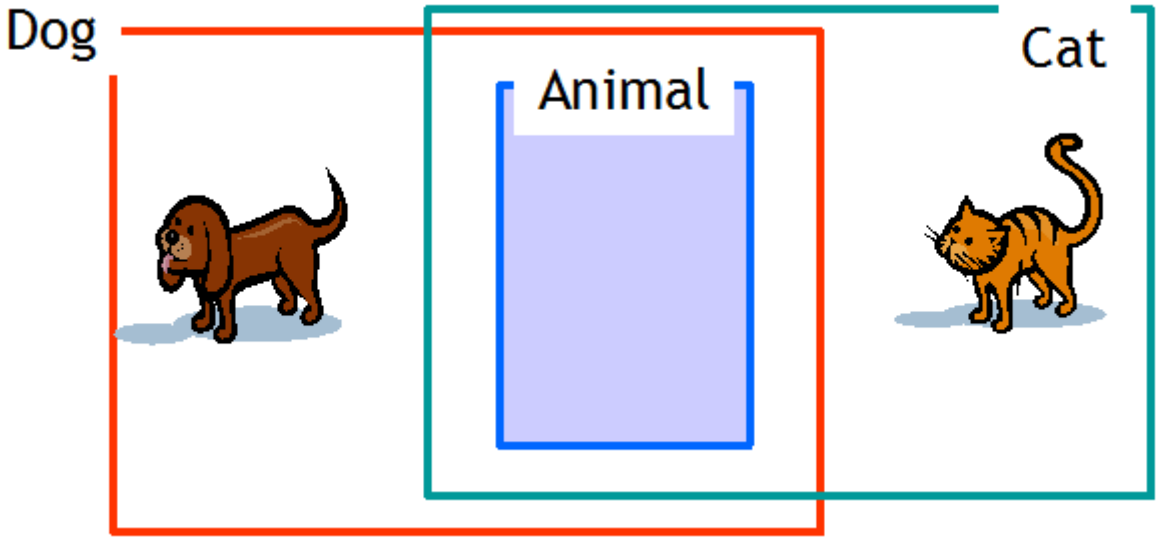
상속 계층도

```
Animal  *pa = new Dog();// 상향 형변환
Animal  *pb = new Cat();// 상향 형변환
```

Animal 타입 포인터로 Dog, Cat 객체를 참조

상속과 객체 포인터

- 자식 클래스 객체는 부모 클래스 객체를 포함하고 있기 때문이다.



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";
        }
};
```

예제

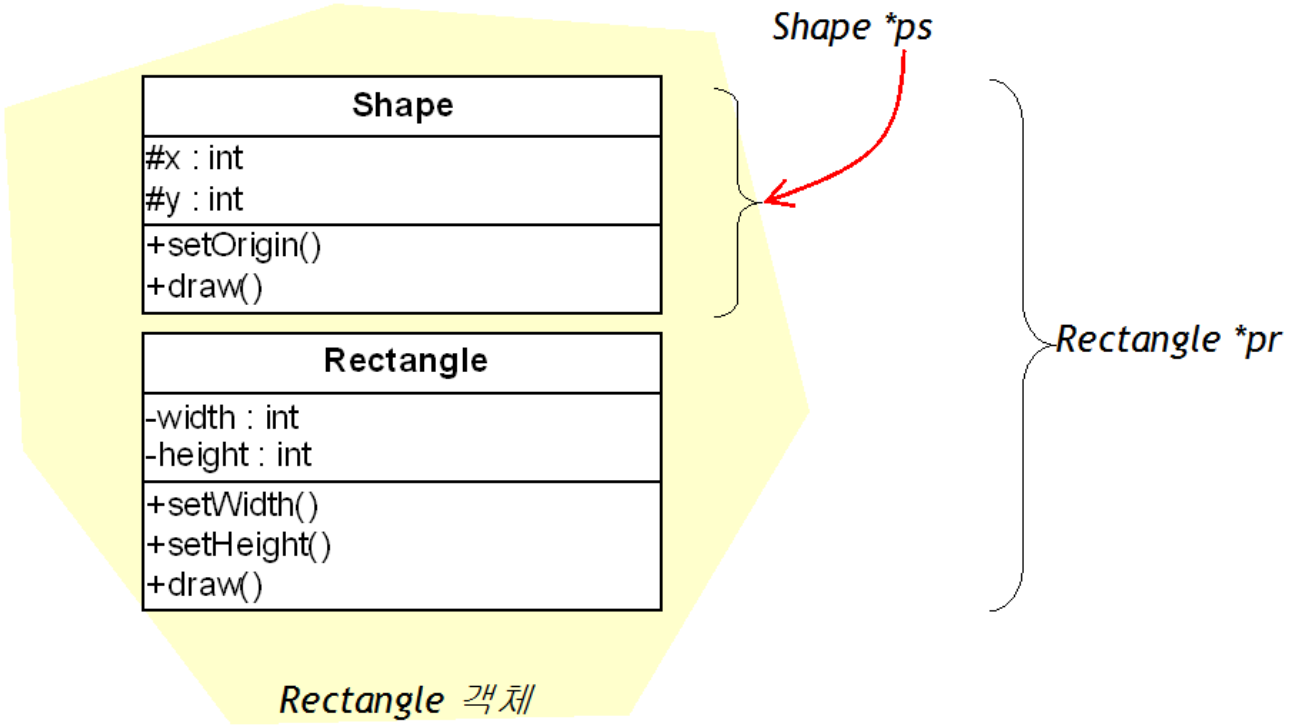

```
class Rectangle : public Shape {
private:
        int width, height;
public:
        void setWidth(int w) {
                width = w;
        }
        void setHeight(int h) {
                height = h;
        }
        void draw() {
                cout << "Rectangle Draw";
        }
};
```

예제


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

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

상향 형변환

- Shape *ps = new Rectangle(); // 상향 형 변환
- Ps->setOrigin(10, 10);



Rectangle 객체를 Shape 포인터로 가리키면 Shape에 정의된 부분밖에 가리키지 못한다.

하향 형변환

- Shape *ps = new Rectangle();
- 여기서 ps를 통하여 Rectangle의 멤버에 접근
- 하향형변환 두가지 방법

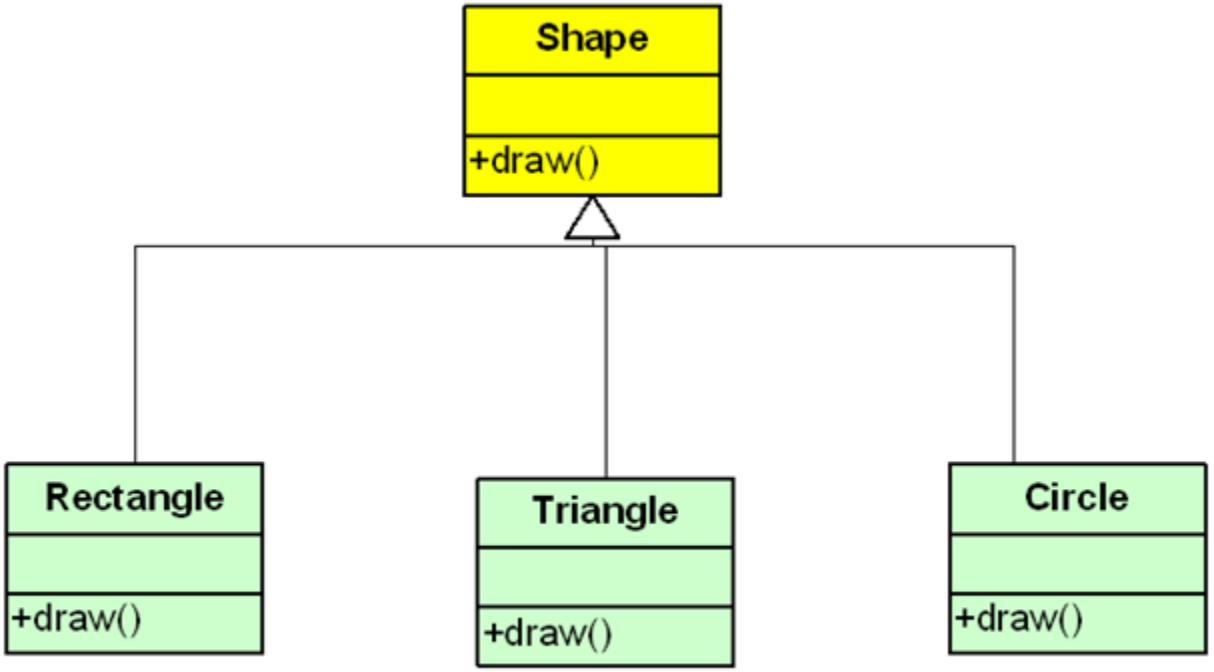
```
1. Rectangle *pr = (Rectangle *) ps;     // 하향형변환
   pr->setWidth(100);

2. ((Rectangle*)ps)->draw();  // 하향형변환
```

형변환 예제


```
int main()
{
        Shape *ps = new Rectangle();          // 상향 형변환

        ps->setOrigin(10, 10);
        ps->draw();
        ((Rectangle *)ps)-> draw();           // 하향 형변환


        delete ps;
}
```

가상함수

- 단순히 자식 클래스 객체를 부모 클래스 객체로 취급하는 것 ???
- 다음과 같은 상속 계층도를 가정



도형의 UML

예제


```
class Shape {
            ...
}
class Rectangle : public Shape {
            ...
}
int main()
{
            Shape *ps = new Rectangle();  // 상향 형변환
            ps->draw();                   // 누가 호출 ?
}
```

Shape 포인터이기 때문에 Shape의 draw()가 호출

가상함수

- 만약 Shape 포인터를 통하여 멤버 함수를 호출하더라도 도형의 종류에 따라서 서로 다른 draw()가 호출
- 사각형인 경우에는 사각형을 그리는 draw()가 호출되고 원의 경우에는 원을 그리는 draw()가 호출

-> draw()를 가상 함수로 작성하면 가능

가상함수 예제


```
#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;
        }
};

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

가상 함수 정의

재정의

가상함수 예제

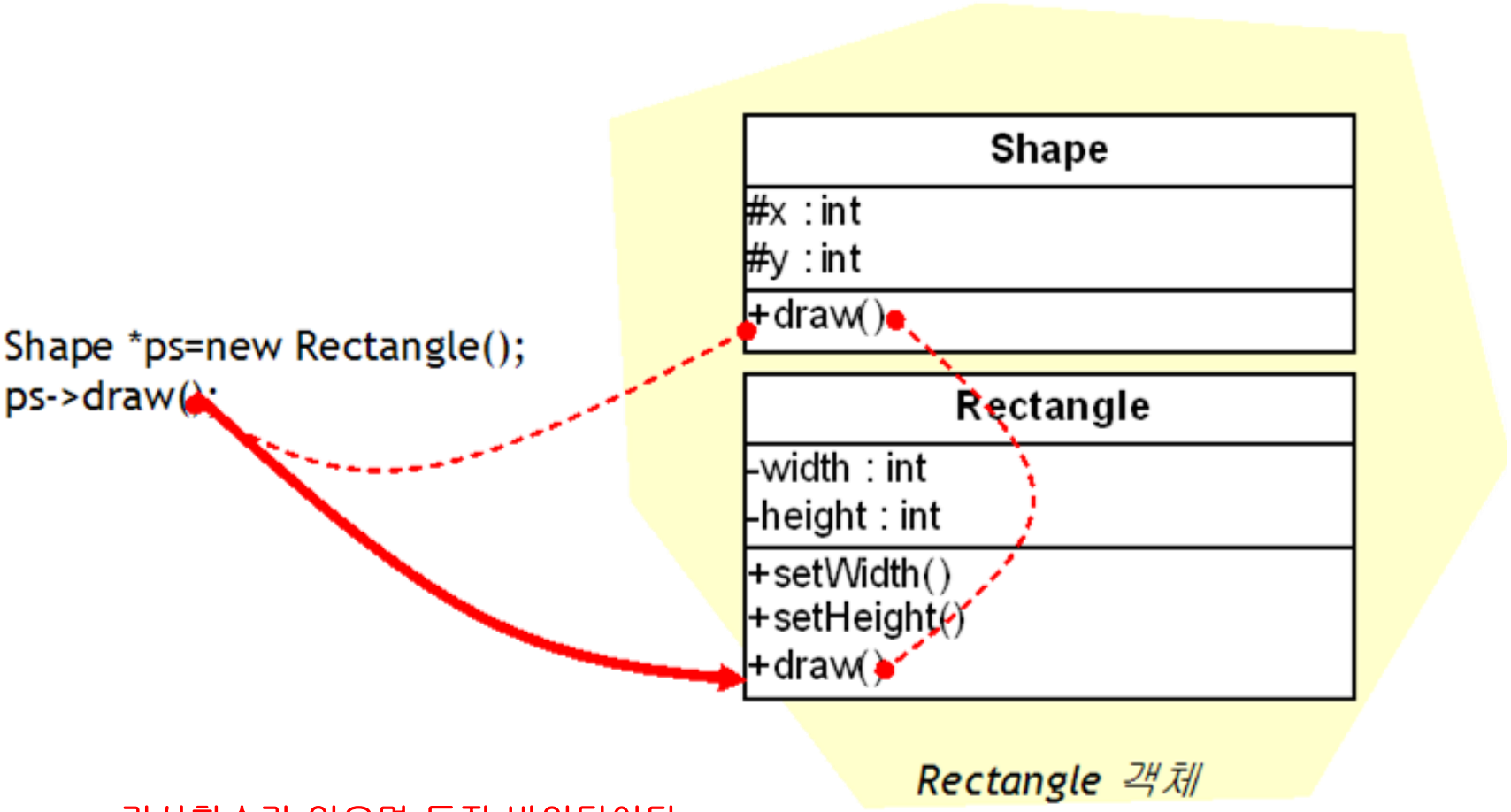

```
int main()
{
        Shape *ps = new Rectangle();
        ps->draw();
        delete ps;

        Shape *ps1 = new Circle();
        ps1->draw();
        delete ps1;
        return 0;
}
```

동적 바인딩

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

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

동적 바인딩



가상함수가 있으면 동적 바인딩이다.

동적 바인딩


```
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();
        }
}
```

참조자와 가상함수

- 참조자도 포인터와 마찬가지로 모든 것이 동일하게 적용
- 부모 클래스의 참조자로 자식 클래스를 가리킬 수 있으며 가상 함수의 동작도 동일

참조자와 가상함수 예제


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

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

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

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

참조자와 가상함수 예제


```
int main()
{
        Dog d;
        Animal &a1 = d;
        a1.speak();

        Cat c;
        Animal &a2 = c;
        a2.speak();
        return 0;
}
```

```
Rectangle r1;
Shape& s1 = r1;
s1.draw();
```

```
Dog *d = new Dog();
Animal& a1 = *d; // 포인터가 가리키는 값을 참조자에 대입
a1.speak();
```

가상 소멸자

- 다형성을 사용하는 과정에서 소멸자를 virtual로 해주지 않으면 문제가 발생

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

class Parent
{
public:
        ~Parent() { cout << "Parent 소멸자" << endl; }
};
class Child : public Parent
{
public:
        ~Child() { cout << "Child 소멸자" << endl; }
};

int main()
{
        Parent* p = new Child();      // 상향 형변환
        delete p;
}
```

Parent() 소멸자 동작
Child() 소멸자 동작 안함

가상 소멸자

해결 방법

```
class Parent
{
public:
        virtual ~Parent() { cout << "Parent 소멸자" << endl; }
};
```

생성자: "객체를 만들 때는 설계도(클래스)가 이미 정해져 있어야 하므로, 가상 선택 불가."

소멸자: "객체를 없앨 때는 부모 포인터를 통해 호출될 수 있으므로, 실제 타입에 맞는 소멸자가 필요 → 가상 가능."

순수 가상 함수

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

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

- (예) virtual void draw() = 0;

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

순수 가상함수 예제

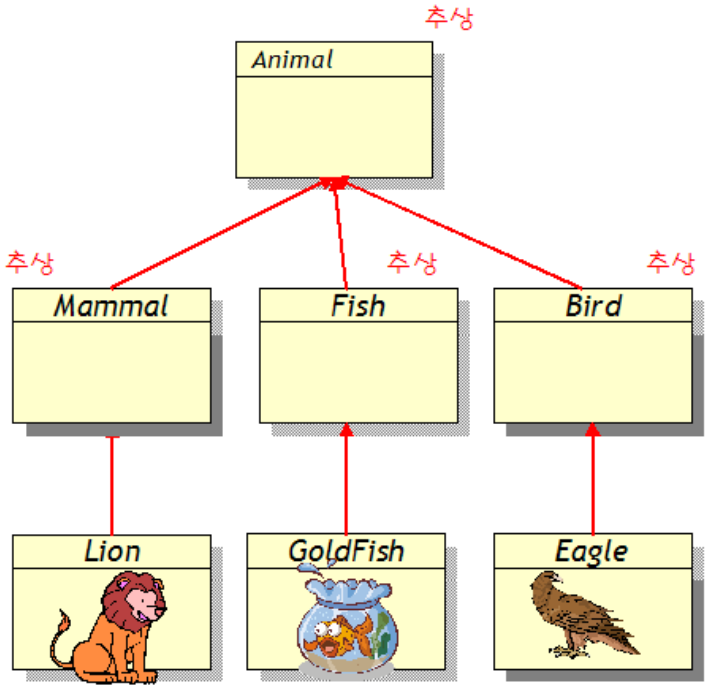
SEOIL UNIVERSITY

```
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;
        }
};

int main()
{

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

        return 0;
}
```

순수가상함수

추상 클래스

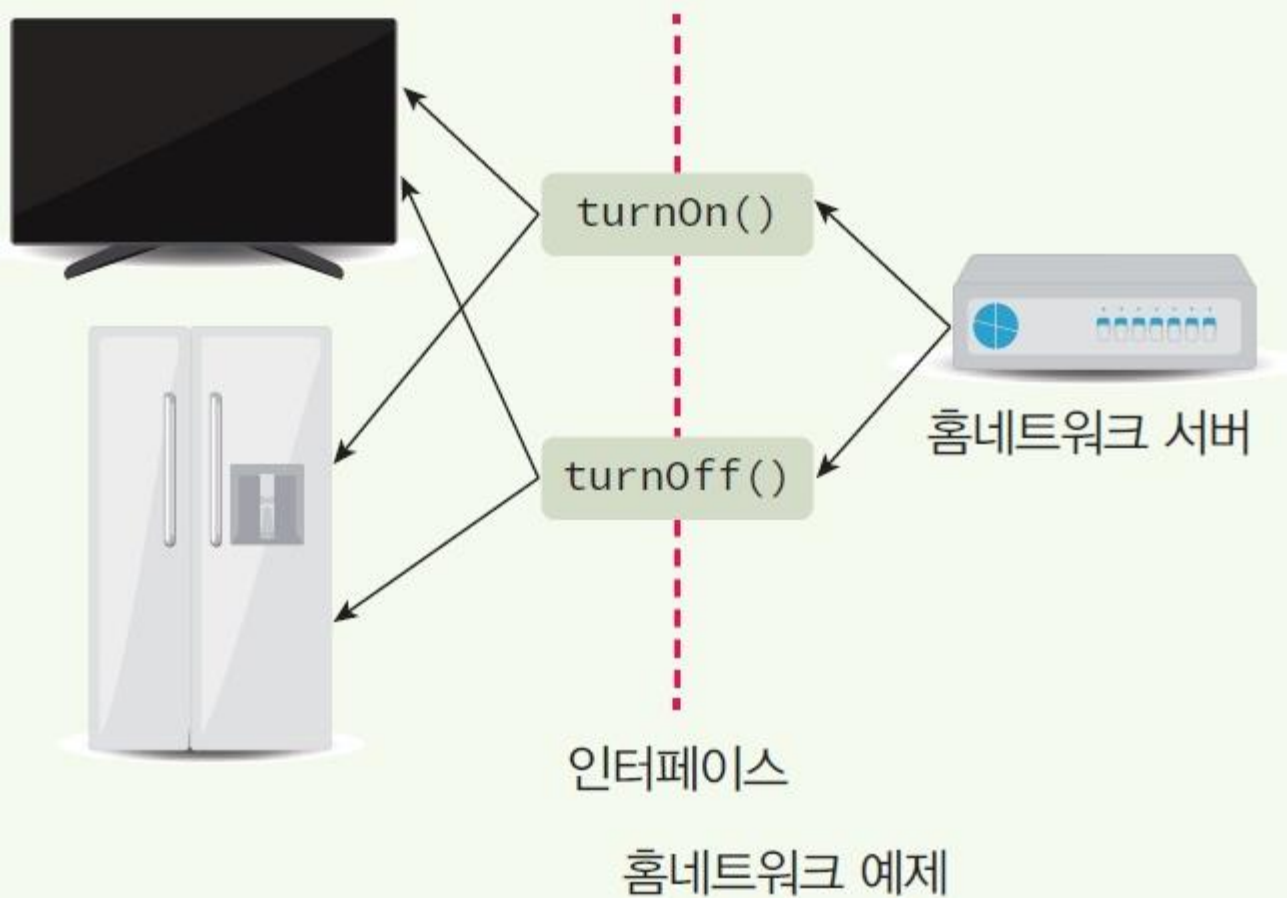
- 추상 클래스(abstract class): 순수 가상 함수를 가지고 있는 클래스
- 추상 클래스는 추상적인 개념을 표현하는데 적당하다.



추상 클래스 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 클래스 예제



추상 클래스 예제


```
class RemoteControl {
        // 순수가상함수정의
        virtual void turnON() = 0;      // 가전제품을켠다.
        virtual void turnOFF() = 0;     // 가전제품을끈다.
}

class Television : public RemoteControl {
        void turnON()
        {
                // 실제로TV의전원을켜기위한코드가들어간다.
                ...
        }
        void turnOFF()
        {
                // 실제로TV의전원을끄기위한코드가들어간다.
                ...
        }
}
```

추상 클래스 예제


```
int main()
{

        RemoteControl* r1 = new Television();
        r1->turnOn();

        RemoteControl* r2 = new Radio();
        r2->turnOn();


        ((Television*)r1)->volCh();
        ((Radio*)r2)->volAMFM();

        delete r1;
        delete r2;
        return 0;
}
```

감사합니다.