제10장 다형성

이번 장에서 학습할 내용

- 다형성
- 가상 함수
- 순수 가상 함수

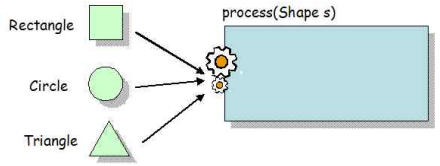
다형성은 객체들이 동일한 메시지에 대하여 서로 다르게 동작하는 것입니다.

다형성이란?

- 다형성(polymorphism)이란 객체들의 타입이 다르면 똑같은 메시지가 전달되더라도 서로 다른 동작을 하는 것
  - Dog bato;
  - Cat nabi;
  - bato.speak()
  - nabi.speak()

- 다형성은 객체 지향 기법에서 하나의 코드로 다양한 타입의 객체를 처리하는 중요한 기술이다.
  - Overloading, Overriding
  - Binding
    - Static binding
    - Dynamic binding

다형성이란?



다형성은 다양한 객체들을 하나의 코드로 처리하는 기술입니다.

객체 포인터의 형변환

- 먼저 객체 포인터의 형변환을 살펴보자.

객체 포인터의 형변환

상향 형변환(upcasting): OK
자식 클래스 타입을 부모 클래스타입으로 변환

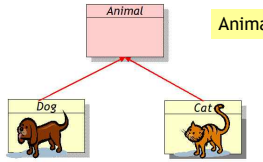


그림 9.2 상속 계층도

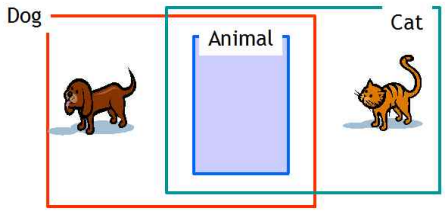
```
Animal  *pa = new Dog();// OK!
```

```
Dog  *pd = new Animal();// No!
```

하향 형변환(downcasting): No
부모 클래스 타입을 자식 클래스타입으로 변환

왜 그럴까?

- 자식 클래스 객체는 부모 클래스 객체를 포함하고 있기 때문이다.




```
Animal  *pa = new Dog();// OK!
```

```
Dog  *pd = new Animal();// No!
```


도형 예제

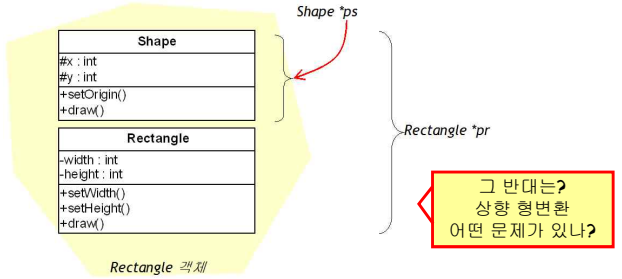
```
class Shape {
protected:
   int x, y;

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

   }
};
```

```
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

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

상향 형변환



Rectangle 객체

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);

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

하향 형변환

예제

```
class Shape {
protected:
   int x, y;
public:
   void setOrigin(int a, int b){ x = a; y = b;}
   void draw() { cout <<"Shape Draw"; }
};
```

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

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

재정의

예제

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

   ps->setOrigin(10, 10);
   ps->draw();

   ((Rectangle *)ps)->setWidth(100);   // Rectangle의setWidth()호출

   delete ps;
}
```

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


함수의 매개 변수

- 함수의 매개 변수는 자식 클래스보다는 부모 클래스 타입으로 선언하는 것이 좋다.

```
void move(Shape& s, int sx, int sy)
{
        s.setOrigin(sx, sy);
}
void main()
{
        Rectangle r;
        move(r, 0, 0);

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

모든 도형을 받을 수 있다.

중간 점검 문제

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

2. 다형성은 어떤 경우에 유용한가?

3. 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.

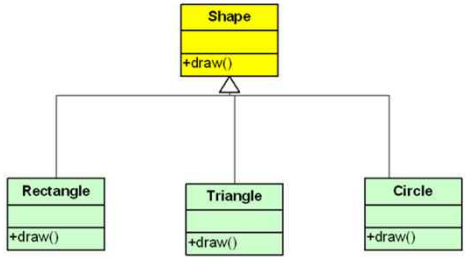


그림 14.6 도형의 UML

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

예제

```
class Shape {
protected:
   int x, y;
public:
   void setOrigin(int a, int b){ x = a; y = b;}
   virtual void draw(){ cout <<"Shape Draw"; }
};
```

가상 함수 정의

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

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

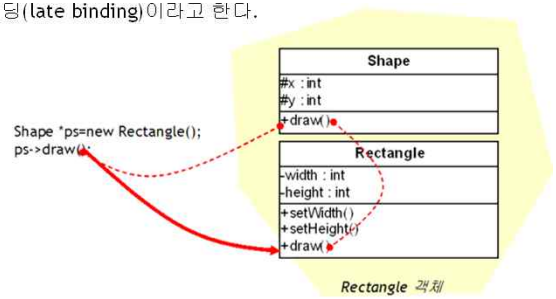

예제

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

        Shape *ps1 = new Circle();        // OK!
        ps1->draw();
        delete ps1;
        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.



정적 바인딩과 동적 바인딩

바인딩의 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.

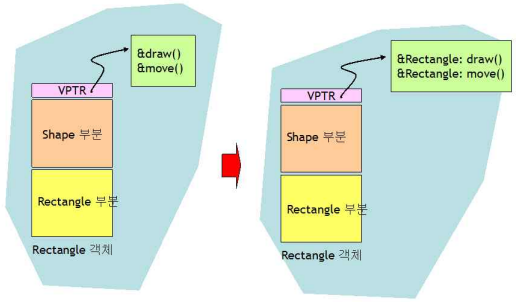


그림 9.9 가상 함수의 구현

예제

```
class Shape {
protected:
   int x, y;
public:
   void setOrigin(int a, int b){ x = a; y = b;}
   virtual void draw(){ cout <<"Shape Draw"; }
};
```

가상 함수 정의

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

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


예제

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

void 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();
        }
}
```

```
Rectangle Draw
Triangle Draw
Circle Draw
```


다형성의 장점

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

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

예제

```
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; }
};

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

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

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

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


중간 점검 문제

1. *가상 함수를 사용하면 어떤 장점이 있는가?*
2. *동적 바인딩과 정적 바인딩을 비교하라.*

참조자와 가상함수

- 참조자인 경우에는 다형성이 동작될 것인가?
  - 참조자도 포인터와 마찬가지로 모든 것이 동일하게 적용된다.

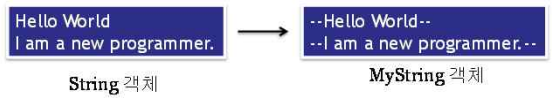
```
class Animal  {
public:
    virtual void speak() { cout <<"Animal speak()" << endl; }
};
class Dog : public Animal  {
public:
    void speak() { cout <<"멍멍" << endl; }
};
class Cat : public Animal  {
public:
    void speak() { cout <<"야옹" << endl; }
};
```

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

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

```
멍멍
야옹
```


가상 소멸자

- 다형성을 사용하는 과정에서 소멸자를 virtual로 해주지 않으면 문제가 발생한다.
- (예제) String 클래스를 상속받아서 각 줄의 앞에 헤더를 붙이는 MyString 이라는 클래스를 정의하여 보자.



String 객체 MyString 객체

소멸자 문제

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

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


소멸자 문제

```
void main()
{
   String *p = new MyString("----", "Hello World!"); // OK!
   p->display();
   delete p;
}
```

```
String() 생성자
MyString() 생성자
----Hello World!----
String() 소멸자
```

MyString의 소멸자가 호출되지 않음

가상 소멸자

- 그렇다면 어떻게 해야 MyString 소멸자도 호출되게 할 수 있는가?
- String 클래스의 소멸자를 virtual로 선언하면 된다.

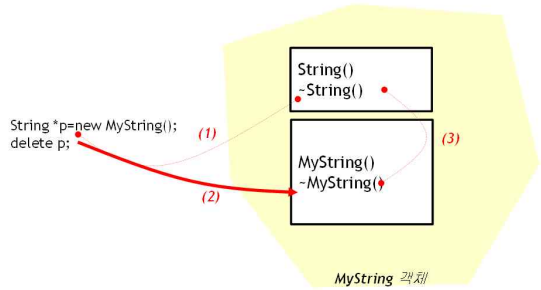


그림 9.10 가상 소멸자

가상 소멸자

```
class String {
    char *s;
public:
    ... // 앞과동일

    virtual ~String(){
        cout << "String() 소멸자"
         << endl;
        delete[] s;
    }
};
class MyString : public String { };

void main() {
        ...// 앞과동일
}
```

```
String() 생성자
MyString() 생성자
----Hello World!----
MyString() 소멸자
String() 소멸자
```

MyString의 소멸자가 호출되고, 이에 따라 부모 클래스의 소멸자도 순서대로 호출됨

중간 점검 문제

1. 가상 함수가 필요한 이유는 무엇인가?

2. 어떤 경우에 부모 클래스의 소멸자에 virtual을 붙여야 하는가?

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 가상 함수의 예

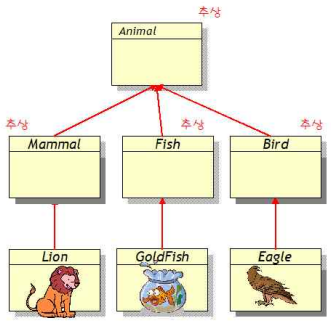
```
void main() {
   Shape *ps = new Rectangle();
   ps->draw();
   delete ps;
}
```

Rectangle Draw

```
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;
        }
};
```


추상 클래스

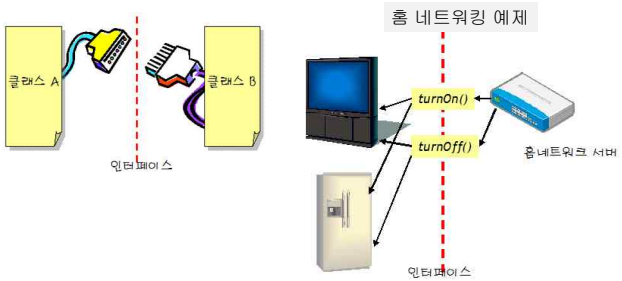
- 추상 클래스(abstract class): 순수 가상 함수를 가지고 있는 클래스
- 추상 클래스는 추상적인 개념을 표현하는데 적당하다.




```
class Animal {
   virtual void move() = 0;
   virtual void eat() = 0;
   virtual void speak() = 0;
};

class Lion : public Animal {
   void move(){
        cout << "사자의 move()\n";
   }
   void eat(){
        cout << "사자의 eat()\n";
   }
   void speak(){
        cout << "사자의 speak()\n";
   }
};
```

추상 클래스를 인터페이스로

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



예제

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

class Television
   : public RemoteControl {
   void turnON() {
        // 실제로TV의 전원을
        // 켜기위한코드가들어간다.
        ...
   }
   void turnOFF() {
        // 실제로TV의 전원을
        // 끄기위한코드가들어간다.
        ...
   }
}
```

```
void main() {
  RemoteControl *pt, *pr;

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

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

  delete pt;
  delete pr;
}
```


중간 점검 문제

1. 순수 가상 함수의 용도는?

2. 모든 순수 가상 함수를 구현하여야 하는가?

Q & A