11. 가상함수와 추상클래스

11.1 클래스 형 변환 규칙

11.2 가상함수

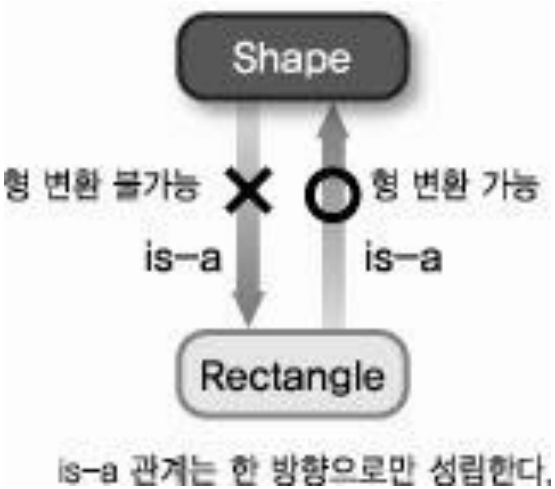
11.3 추상클래스와 인터페이스 상속

11.1 클래스 형 변환 규칙

클래스 형 변환

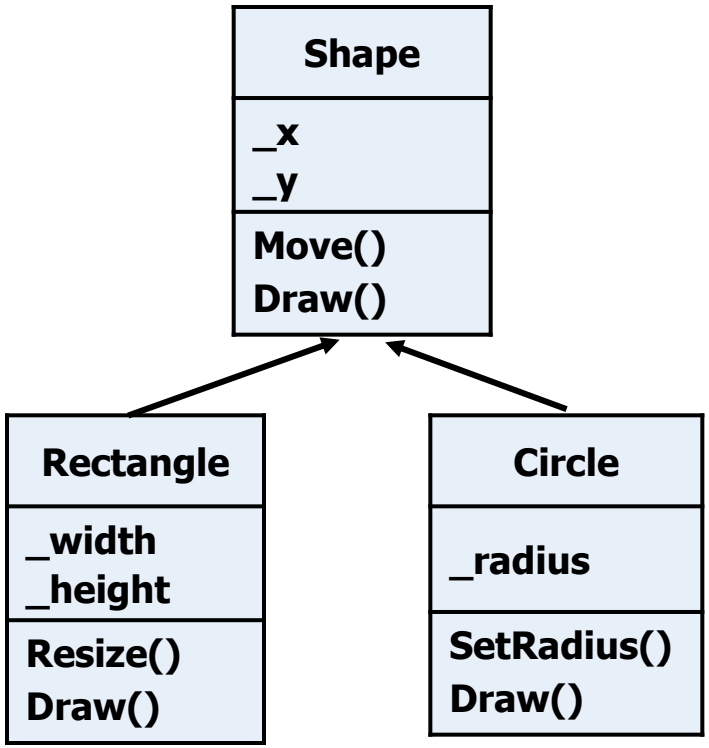
- 클래스 형 변환
  - 클래스의 형 변환은 상속 관계에 놓인 클래스 간에만 가능
  - 파생 클래스에서 기본 클래스쪽으로의 형 변환
  - is-a 관계가 성립하는 방향으로만 일어남

- 클래스 형 변환 규칙
  - 파생 클래스의 객체는
    기본 클래스의 객체로 형 변환 가능
  - 파생 클래스의 포인터는
    기본 클래스의 포인터로 형 변환 가능
  - 파생 클래스의 레퍼런스는
    기본 클래스의 레퍼런스로 형 변환 가능



도형 그리기 프로그램의 예

- 기본 클래스 : Shape 클래스
  - 직사각형과 원의 공통된 특징 제공
- 파생클래스 : Rectangle 및 Circle 클래스
  - 기본 클래스가 제공해주는 공통된 특징 외에 각각 직사각형과 원의 구체적인 특징 가짐



파생 클래스의 기본 클래스로의 형 변환(1)

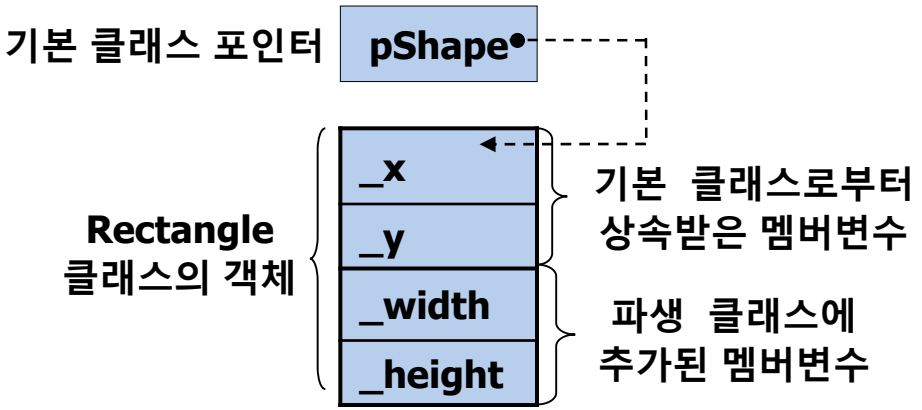
is-a 관계가 성립하는 쪽으로의 형 변환

- Rectangle 또는 Circle 클래스는 Shape 클래스의 파생클래스 이므로 is-a 관계가 성립함
- Rectangle 클래스로 만든 객체나 Circle 클래스로 만든 객체는 Shape 클래스로 만든 객체로 형 변환 가능
- 하지만 반대의 경우는 성립하지 않음
- 객체에 대한 포인터도 마찬가지임

```
Shape *pShape1 = new Rectangle;        // OK
Shape *pShape2 = new Circle;           // OK
Rectangle* pRect = new Shape;          // 컴파일 에러
Circle *pCircle = new Shape;           // 컴파일 에러
```

파생 클래스의 기본 클래스로의 형 변환(2)

- 파생 클래스 포인터를 기본 클래스 포인터로 형 변환한다는 것의 의미
  - 파생 클래스가 가진 기능 중 기본 클래스로부터 상속받은 기능에만 접근할 수 있다




```
Shape *pShape = new Rectangle;          // 자동 형 변환
pShape->Move(10, 10);                   // Shape::Move 호출
pShape->Resize(100,100);                // 컴파일 에러 이유는?
pShape->Draw();                         // Shape::Draw 호출
delete pShape;
```

함수의 인자 전달에서의 클래스 형 변환

- 함수의 인자 전달
  - 기본 클래스에 대한 포인터나 레퍼런스를 인자로 갖는 함수에 파생 클래스의 객체를 전달하는 것이 가능함.
  - 서로 다른 클래스가 동일한 함수의 인자로 사용될 수 있음
    - ⇒ 기본 클래스 포인터나 레퍼런스는 파생 클래스의 객체를 같은 방법으로 관리 할 수 있는 방법을 제공

```
void f(Shape& s) {
   s.Draw();
}

int main() {
    Rectangle r1;
    f(r1);              // Shape& s = r1;의 의미

    Circle e1;
    f(e1);              // Shape& s = e1;의 의미
}
```

클래스 형 변환 정리

- 파생 클래스 객체를 기본 클래스 포인터로 접근하면
  - **기본 클래스로부터 상속받은 멤버에만 접근가능**
  - 객체에 대한 포인터 변수의 데이터 형에 의해서 호출될 함수가 **컴파일 시** 결정됨.
  - 기본 클래스 포인터로는 상속받는 멤버 함수는 호출할 수 있지만 파생 클래스에 새로 추가된 멤버 함수는 호출할 수 없음.
  - 기본 클래스 포인터로 파생 클래스에 재정의된 멤버 함수를 호출하면 기본 클래스에서 상속 받은 함수가 호출됨(재정의된 함수가 호출되지 않음)

파생클래스의 재정의된 함수가 호출되게 하려면?

11.2 가상함수

가상함수(virtual function)

- 기본 클래스 포인터로 호출하더라도 파생 클래스에 재정의된 함수를 호출하도록 만들려면?
  - 기본 클래스의 멤버 함수를 가상 함수로 선언하면 끝.

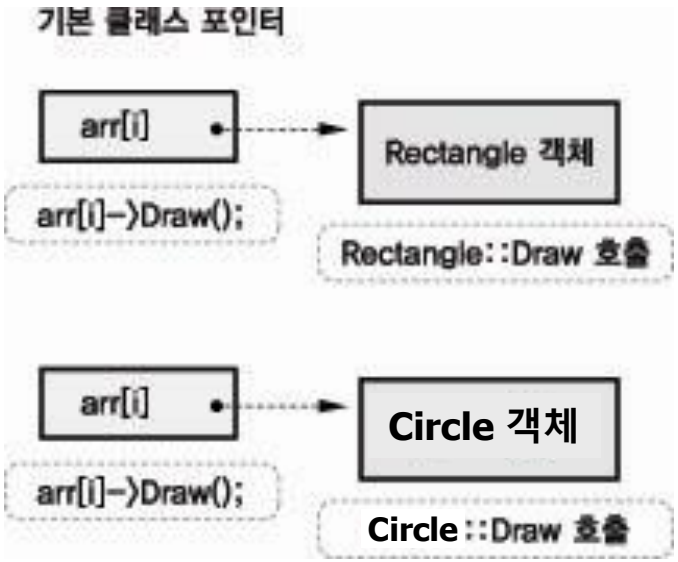
- 가상함수로 만들려면?
  - virtual 키워드를 기본 클래스의 멤버 함수에 지정
  - virtual 키워드의 의미
    - 동적 바인딩 지시어
    - 컴파일러에게 함수에 대한 호출 바인딩을 실행 시간까지 미루도록 지시
  - 함수 선언에만 지정하고 함수 정의에는 쓰지 않음.
  - 기본 클래스의 멤버 함수를 가상 함수로 지정하면 파생 클래스에서 재정의되는 함수는 virtual 키워드를 지정하지 않아도 자동으로 가상 함수가 됨

```
class Shape  {
    ...
    virtual void Draw() const;              // Draw()는 가상함수
};
```

가상함수를 통한 다형성 구현 예

- 함수 오버라이딩(function overriding)
  - 함수의 재정의
  - 파생 클래스에서 기본 클래스의 가상 함수와 동일한 이름의 함수 선언

```
void main() {

      Shape* arr[2] = {new Rectangle(),
                       new Circle()};

      for (int i=0; i<2; i++) {
            arr[i]→Draw();
        // 파생클래스의 멤버함수가 호출됨
       }
}
```



동적 바인딩(Dynamic Binding)

- **실행 시간**에 호출될 함수를 결정
  - 기본 클래스의 포인터나 레퍼런스로 **가상 함수**를 호출하면 기본 클래스 포인터나 레퍼런스가 가리키는 곳에 실제로 어떤 클래스의 객체가 있는지에 따라서 호출될 함수가 결정됨.

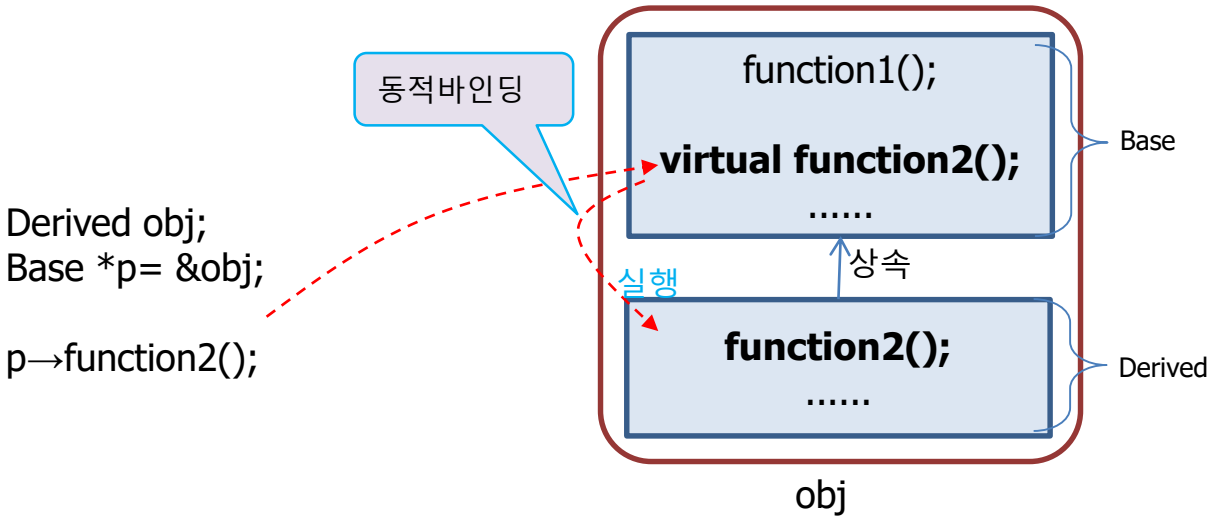
- 동적 바인딩 vs 정적 바인딩

특징	정적바인딩	동적바인딩
바인딩 시기	컴파일 시간	실행시간
구분	일반 함수	가상함수
단점	프로그램의 융통성이 적다	프로그램의 융통성이 큼
장점	동적 바인딩에 비해 처리속도가 빠르다	정적 바인딩에 비해 처리속도가 늦다

동적 바인딩

동적 바인딩

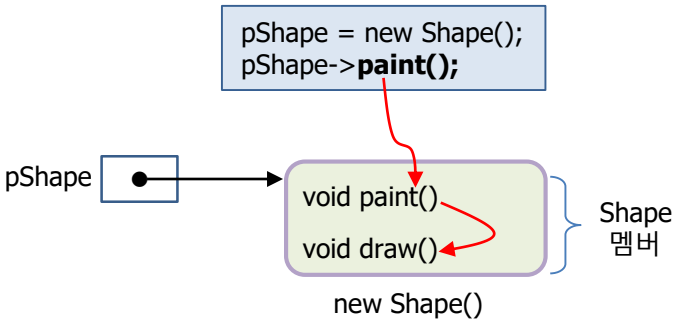
- 파생 클래스에 대해, 기본 클래스에 대한 포인터로 가상 함수를 호출하는 경우
- 객체 내에 오버라이딩한 파생 클래스의 함수를 찾아 실행
  - 실행 중에 이루어짐 : 실행시간 바인딩, 런타임 바인딩, 늦은 바인딩으로 불림



동적 바인딩

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

class Shape {
public:
    void paint() {
        draw();
    }
    virtual void draw() {
        cout << "Shape::draw() called" << endl;
    }
};

int main() {
    Shape *pShape = new Shape();
    pShape->paint();
    delete pShape;
}
```

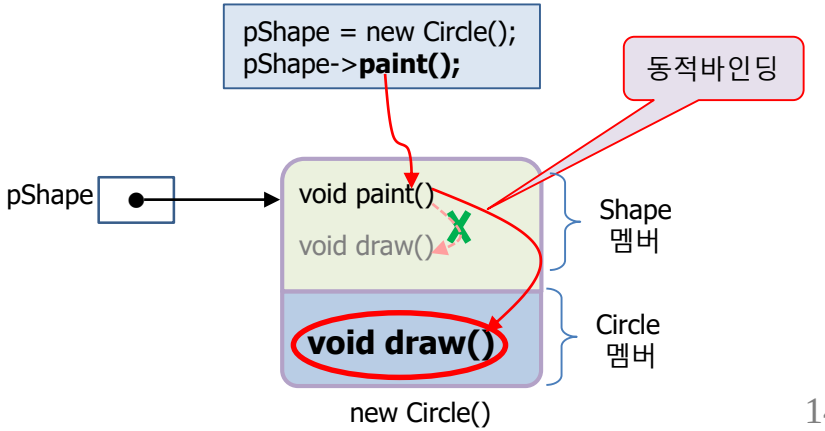
Shape::draw() called




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

class Shape {
public:
    void paint() {
        draw();
    }
    virtual void draw() {
        cout << "Shape::draw() called" << endl;
    }
};

class Circle : public Shape {
public:
    virtual void draw() {
        cout << "Circle::draw() called" << endl;
    }
};

int main() {
    Shape *pShape = new Circle();
    pShape->paint();
    delete pShape;
}
```

기본 클래스에서 파생 클래스의 함수를 호출하게 되는 사례

Circle::draw() called



가상 소멸자

- 소멸자도 가상함수로 선언해야 하는 경우
  - 클래스에 가상함수가 있고, 클래스가 반드시 소멸자를 사용해야 할 때

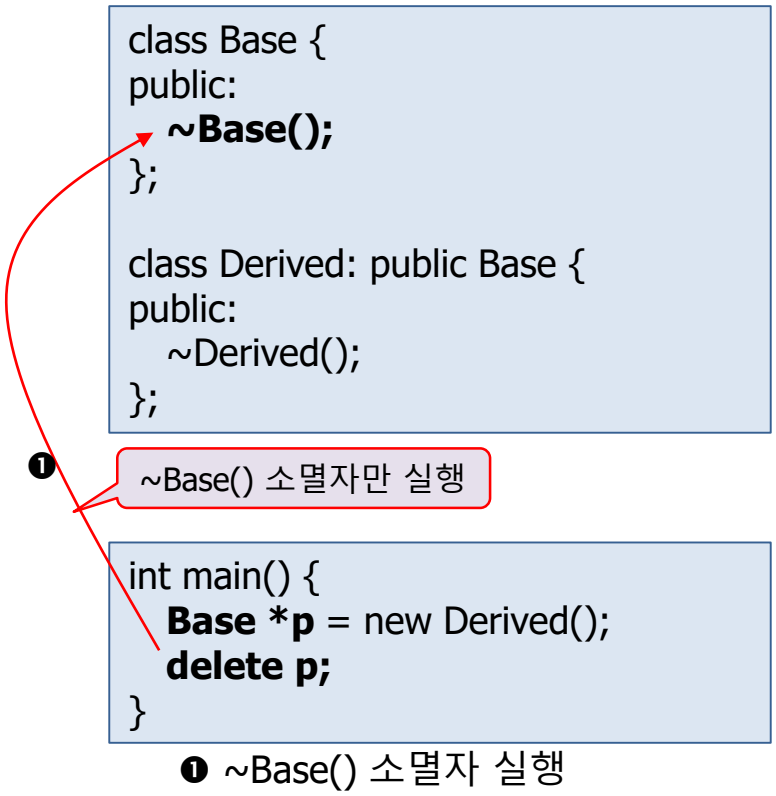
    예) 동적으로 생성한 파생 클래스의 객체를 기본 클래스 포인터로 가리킬 때

    → 기본 클래스 포인터로 delete하더라도 파생 클래스의 소멸자가 호출되도록 해야 함

```
Shape* pShape = new Rectangle;
delete pShape;        // Rectangle 소멸자가 호출되도록 하려면
                      // Shape 소멸자로 가상함수로 만든다.
```

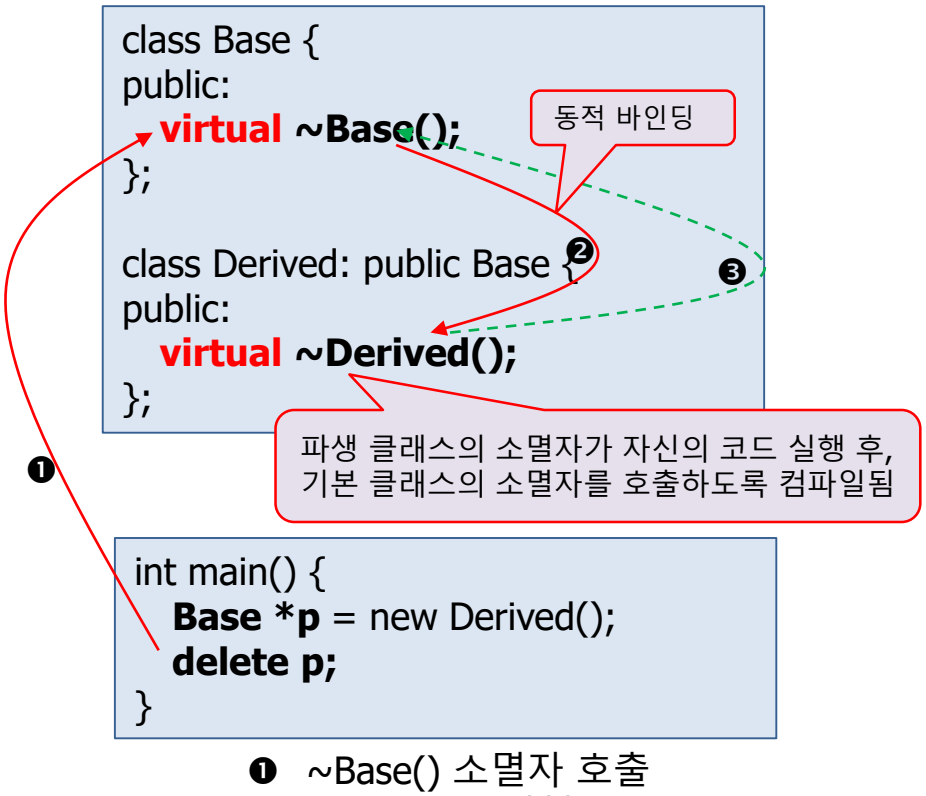
```
class Shape {
    ...
    virtual void Draw() const;
    ...
    virtual ~Shape();
};
```

가상 소멸자



❶ ~Base() 소멸자 실행

소멸자가 가상 함수가 아닌 경우

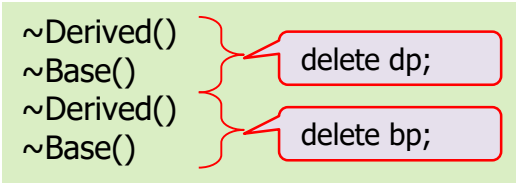


❶ ~Base() 소멸자 호출
❷ ~Derived() 실행
❸ ~Base() 실행

가상 소멸자 경우

가상 소멸자

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

class Base {
public:
   virtual ~Base() { cout << "~Base()" << endl; }
};

class Derived: public Base {
public:
   virtual ~Derived() { cout << "~Derived()" << endl; }
};

int main() {
   Derived *dp  = new Derived();
   Base *bp = new Derived();

   delete dp; // Derived의 포인터로 소멸
   delete bp; // Base의 포인터로 소멸
}
```



오버로딩과 오버라이딩 비교

비교 요소	오버로딩	오버라이딩
정의	매개 변수 타입이나 개수가 다르지만, 이름이 같은 함수들이 중복 작성되는 것	기본 클래스에 선언된 가상 함수를 파생 클래스에서 이름, 매개 변수 타입, 매개 변수 개수, 리턴 타입까지 완벽히 같은 원형으로 재작성하는 것
존재	외부 함수들 사이. 한 클래스의 멤버들. 상속 관계	상속 관계. 가상 함수에서만 적용
목적	이름이 같은 여러 개의 함수를 중복 작성하여 사용의 편의성 향상	기본 클래스에 구현된 가상 함수를 무시하고, 파생 클래스에서 새로운 기능으로 재정의하고자 함
바인딩	정적 바인딩. 컴파일 시에 중복된 함수들의 호출 구분	동적 바인딩. 실행 시간에 오버라이딩된 함수를 찾아 실행
관련 객체 지향 특성	다형성	다형성

11.3 추상클래스와 인터페이스 상속

순수 가상함수

기본 클래스의 가상 함수 목적

- 파생 클래스에서 재정의할 함수를 알려주는 역할
  - 실행할 코드를 작성할 목적이 아님
- ***기본 클래스의 가상 함수를 굳이 구현할 필요가 있을까?***

순수 가상함수

- 함수의 **코드가 없고** 선언만 있는 **가상 멤버** 함수
- 파생 클래스에서 재정의될 함수에 대해서 미리 원형이 필요하기는 하지만 기본 클래스에서는 아직 구현할 필요가 없을 때 사용
- 순수 가상 함수는 "파생 클래스에서 재정의하도록 원형만 미리 준비해두는 것"이라는 의미가 되며, 파생 클래스는 상속받은 순수 가상 함수를 **반드시 재정의**해야 한다.

순수 가상 함수 선언 방법

- 함수 선언의 끝 부분에 "= 0"을 적어줌

```
class Shape {
public:
   virtual void draw() const= 0; // 순수 가상 함수
};
```

추상 클래스

- 추상 클래스(abstract class)
  - 최소한 하나의 순수 가상 함수를 갖는 클래스

```
class Shape { // Shape은 추상 클래스
 protected :
     int _x;
     int _y;
 public:
     void paint();
     virtual void draw() const= 0; // 순수 가상 함수
};

void Shape::paint() {
   draw(); // 순수 가상 함수라도 호출은 할 수 있다.
}
```

추상 클래스

추상 클래스 특징

- **온전한 클래스가 아니므로 추상 클래스는 객체를 생성할 수 없다.**

```
Shape shape; // 컴파일 오류
Shape *p = new Shape(); // 컴파일 오류
```

- **추상 클래스의 포인터 변수나 레퍼런스 변수는 정의할 수 있다.**
  - 파생 클래스 객체를 가리키는 용도로 사용
  - 파생 클래스 객체에 접근하는 인터페이스 역할을 제공

```
Shape *p;
```

추상 클래스의 목적

- 추상 클래스의 목적
  - 추상 클래스의 인스턴스를 생성할 목적이 아님
  - 상속에서 기본 클래스의 역할을 하기 위함
    - 순수 가상 함수를 통해 파생 클래스에서 구현할 함수의 형태(원형)을 보여주는 인터페이스 역할

추상 클래스의 상속과 구현

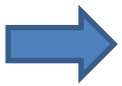
- 추상 클래스의 상속
  - 추상 클래스를 단순 상속하면 자동 추상 클래스
- 추상 클래스의 구현
  - 추상 클래스를 상속받아 순수 가상 함수를 오버라이딩
    - 파생 클래스는 추상 클래스가 아님

Shape은 추상 클래스

Circle도 추상 클래스

```
class Shape {
public:
   virtual void draw() = 0;
};

class Circle : public Shape {
public:
   string toString() {
      return "Circle 객체";
   }
};


Shape shape; // 객체 생성 오류
Circle waffle; // 객체 생성 오류
```

추상 클래스의 단순 상속



```
class Shape {
public:
   virtual void draw() = 0;
};

class Circle : public Shape {
public:
   virtual void draw() {
      cout << "Circle";
   }
   string toString() {
      return "Circle 객체";
   }
};

Shape shape; // 객체 생성 오류
Circle waffle; // 정상적인 객체 생성
```

Shape은 추상 클래스

Circle은 추상 클래스 아님

순수 가상 함수 오버라이딩

추상 클래스의 구현

구현 상속과 인터페이스 상속

- 구현상속
  - 기본 클래스가 해당 함수의 구현을 제공
  - 클래스의 멤버 함수가 일반 함수라면 파생 클래스에서는 이 함수를 재정의할 필요가 없다.
- 디폴트 구현 상속 + 인터페이스 상속
  - 파생 클래스가 해당 함수를 재정의할지 여부를 선택
  - 클래스의 멤버 함수가 가상 함수라면 파생 클래스에서는 이 함수를 재정의할 수도 있고, 재정의하지 않을 수도 있다.
- 인터페이스 상속
  - 클래스의 멤버 함수가 순수 가상 함수라면 파생 클래스에서는 이 함수를 반드시 재정의해야 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.

오버로드(overroad) 된 멤버 함수의 오버라이드(override)

- 오버로드(overroad) 된 멤버 함수
  - 기본 클래스에서 오버로드 된 함수 중에서 어느 것 하나라도 오버라이드(override, 재정의) 하면 나머지 다른 함수들도 모두 사용할 수 없다.

Dog1.cpp

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

        string name;
};

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

void Dog::Eat() {
        cout << name << "says, 'Growl~'□n";
}
```

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

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

        return 0;
}
```

정리

- 파생 클래스의 포인터나 레퍼런스는 기본 클래스의 포인터나 레퍼런스로 형 변환 가능하다. 반대로 기본 클래스의 포인터나 레퍼런스를 파생 클래스의 포인터나 레퍼런스로 형 변환하는 것은 불가능하다.
- 기본 클래스 포인터나 레퍼런스로 파생 클래스 객체를 가리키고 있을 때, 기본 클래스 포인터나 레퍼런스로 가상 함수를 호출하면 파생 클래스에 재정의된 함수가 호출된다.
- 가상 함수처럼 실행 시간에 호출될 함수를 결정하는 것을 동적 바인딩이라고 한다.
- 만일 파생 클래스에서 재정의될 함수에 대한 원형만 필요하다면 클래스의 멤버 함수를 순수 가상 함수로 선언할 수 있다. 순수 가상 함수는 선언만 있고 정의는 없는 가상 함수를 말한다.
- 순수 가상 함수를 갖는 클래스를 추상 클래스라고 하며 추상 클래스는 객체를 생성할 수 없다.

다음 수업

- 템플릿과 STL
  - 1_ 클래스 템플릿
  - 2_ STL