

다형성 Polymorphism (가상함수 Virtual Function)

2023

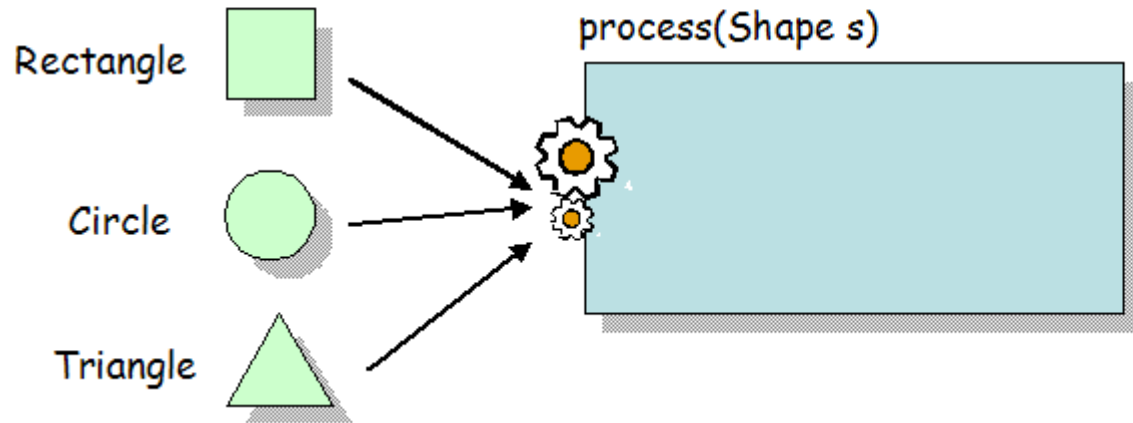
국민대학교 소프트웨어학부

다형성이란?

- 다형성(polymorphism)이란 객체들의 타입이 다르면 같은 이름의 함수가 호출되더라도 서로 다른 동작을 하는 것
- 다형성은 객체 지향 기법에서 하나의 코드로 다양한 타입의 객체를 처리하는 중요한 기술이다.



다형성이란?



다형성은 다양한 객체들을 하나의 코드로 처리하는 기술입니다.



객체 간의 형변환

- 먼저 객체 간의 형변환을 살펴보자.

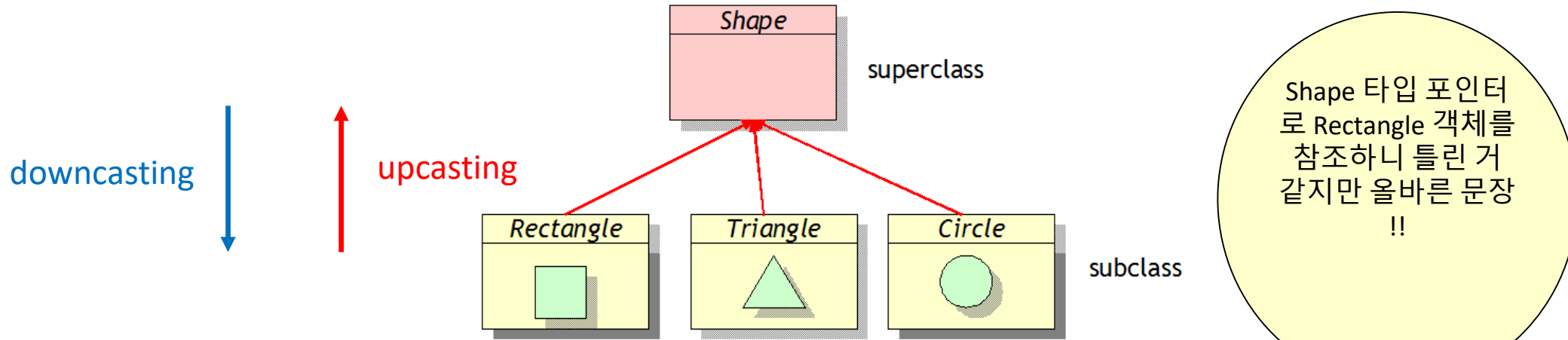
- pointer, reference 자료형에만 적용

객체 간의 형변환

상향 형변환(upcasting):
서브클래스 타입을 슈퍼클래스타입으로 변환
implicit 하게 일어남

하향 형변환(downcasting):
슈퍼클래스 타입을 서브클래스타입으로 변환
implicit 하게 일어나지 않음 (명시적 형변환 연산자를 사용해야 함)

상속과 객체 참조



Shape 타입 포인터
로 Rectangle 객체를
참조하니 틀린 거
같지만 올바른 문장
!!

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

자동적 upcasting
but 자동적 downcasting 은 없다.

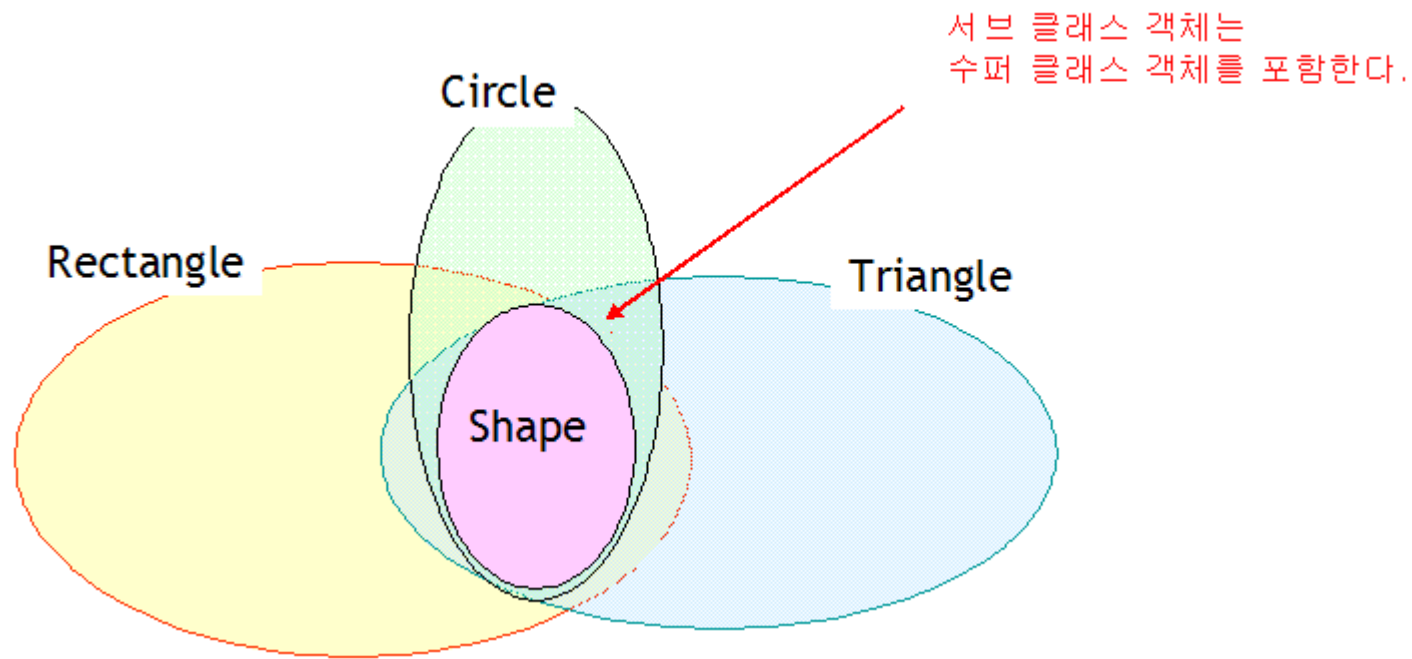
Is-a relationship

- Rectangle is a shape (yes)
- Shape is a rectangle (no)



왜 그럴까?

- 서브 클래스 객체는 슈퍼 클래스 객체를 포함하고 있기 때문이다.



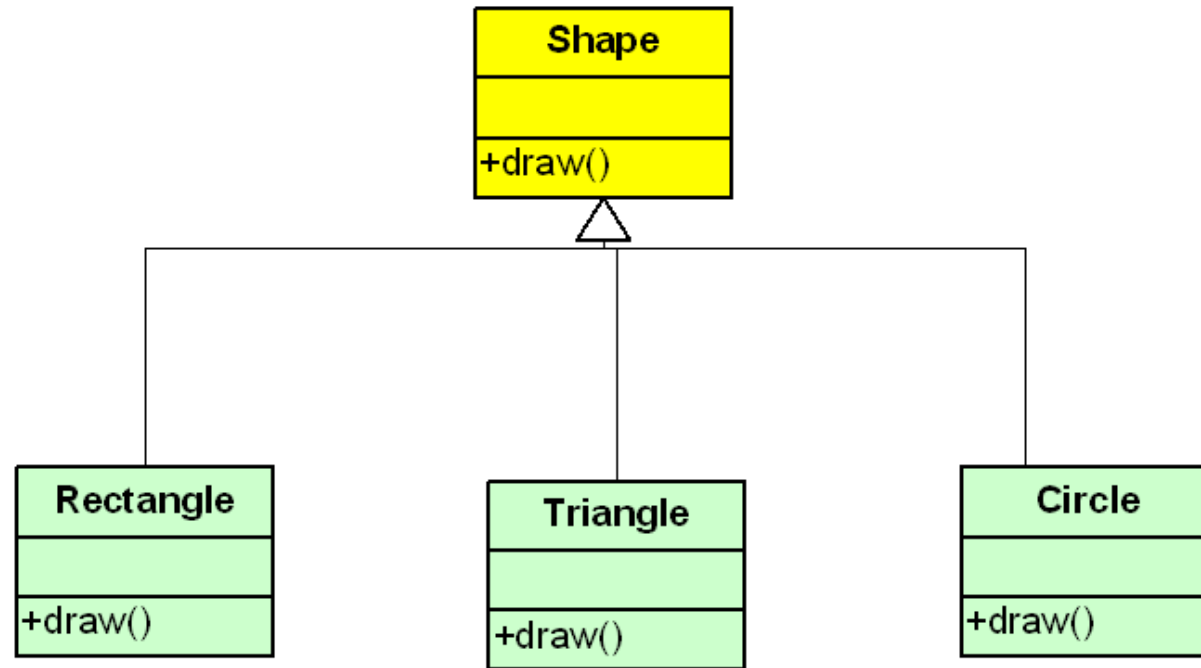


그림 14.6 도형의 UML

Unified Modelling Language

```

4  class Shape{
5  protected:
6      int x,y;
7  public:
8      void setOrigin(int x, int y){
9          this->x = x; this->y = y;
10     }
11     void draw(){
12         cout << "Shape ( " << x << ", " << y << " ) ";
13     }
14 };
15 class Rectangle : public Shape{
16     int width, height;
17 public:
18     void setWidth(int w){ width = w; }
19     void setHeight(int h){ height = h; }
20     void draw(){
21         Shape::draw();
22         cout << "Rectangle : " << width << " x " << height;
23     }
24 };

```

Rectangle, Circle 에서 접근 가능

```

25 class Circle : public Shape{
26     int radius;
27 public:
28     void setRadius(int r){ radius = r; }
29     void draw(){
30         Shape::draw();
31         cout << "Circle : " << radius;
32     }
33 };
34 void move(Shape& s, int dx, int dy){
35     s.setOrigin(dx, dy);
36 }

```


객체 포인터의 상향 형변환

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

객체 포인터의 상향 형변환
Rectangle * → Shape *

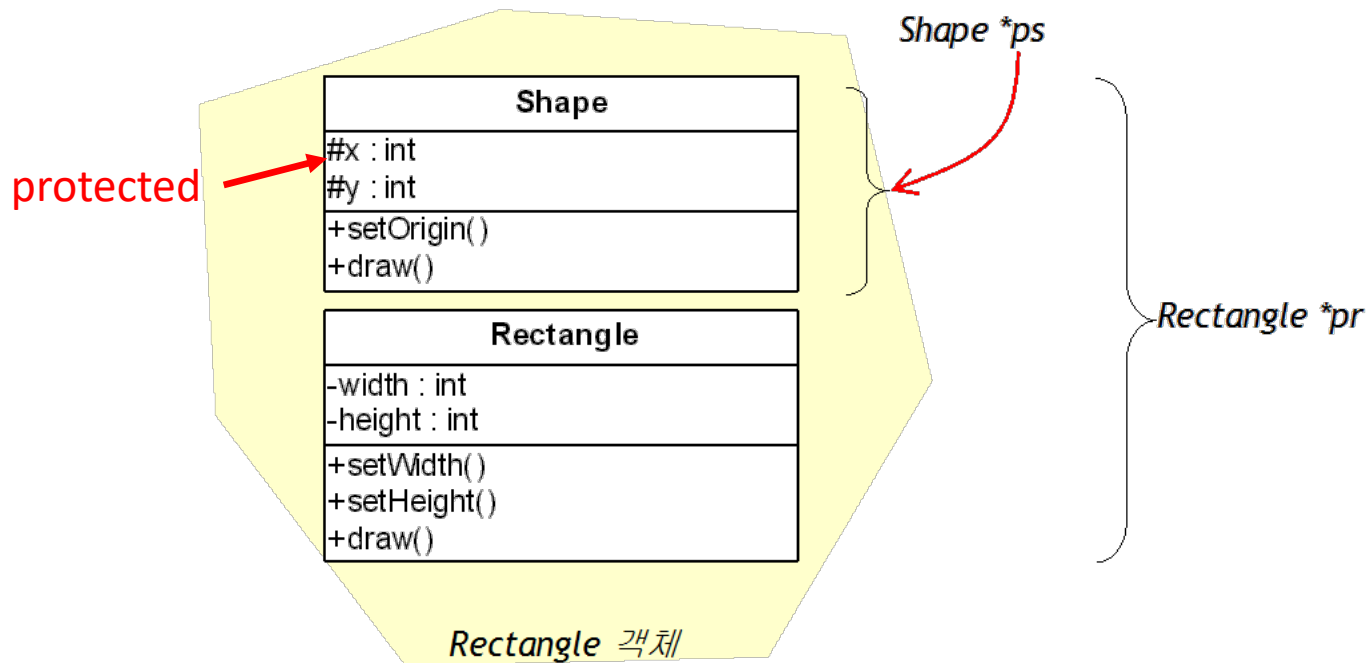


그림 14.4 Rectangle 객체를 Shape 포인터로 가리키면 Shape에 정의된 부분밖에 가리키지 못한다.

객체 포인터의 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);

객체 포인터의 하향 형변환
Shape * → Rectangle *

```

4  class Shape{
5  protected:
6      int x,y;
7  public:
8  >   void setOrigin(int x, int y){=}
11 >   void draw(){=}
14 };
15 class Rectangle : public Shape{
16     int width, height;
17 public:
18     void setWidth(int w){ width = w; }
19     void setHeight(int h){ height = h;}
20 >   void draw(){=}
24 };
25 class Circle : public Shape{
26     int radius;
27 public:
28     void setRadius(int r){ radius = r;}
29 >   void draw(){=}
33 };

```

```

Shape ( 1, 1 ) Rectangle : 3 x 21966
Shape ( 10, 10 )
Shape ( 8, 8 ) Rectangle : 16 x 21966
Shape ( 8, 8 ) Circle : 11
Shape ( 8, 8 ) Rectangle : 11 x 21966
Shape ( 5, 5 )

```

```

37 int main(){
38     Rectangle r;
39     r.setOrigin(1,1);
40     r.setWidth(3);
41     r.draw(); cout << endl;
42
43     Shape *p_s;
44     p_s = &r; // Rectangle * --> Shape *
45     p_s->setOrigin(10, 10);
46     p_s->setWidth(100); // compile error
47     p_s->draw(); cout << endl;
48
49     Rectangle *p_r;
50     p_r = (Rectangle *)p_s; // Shape * --> Rectangle *
51     p_r->setOrigin(8, 8);
52     p_r->setWidth(16);
53     p_r->draw(); cout << endl;
54
55     ((Circle *)p_s)->setRadius(11);
56     ((Circle *)p_s)->draw(); cout << endl;
57     p_r->draw(); cout << endl;
58
59     Shape& s = r;
60     s.setOrigin(5,5);
61     s.setWidth(7); // compile error
62     s.draw(); cout << endl;
63
64     Rectangle& r_r = (Rectangle)s; // compile error

```

(Shape& s)r

Shape *p_s
Rectangle *p_r
(Circle *)p_s

x,y
width
height

Shape *→Rectangle * 변환 생성자는 필요없음

Shape → Rectangle 변환 생성자가 필요함

```

4 class Shape{
5 protected:
6     int x,y;
7 public:
8 > void setOrigin(int x, int y){=}
11 > void draw(){=}
14 };
15 class Rectangle : public Shape{
16     int width, height;
17 public:
18     void setWidth(int w){ width = w; }
19     void setHeight(int h){ height = h;}
20 > void draw(){=}
24 };
25 class Circle : public Shape{
26     int radius;
27 public:
28     void draw();
29 > void draw();
33 };

```

```

Shape ( 1,
Shape ( 10,
Shape ( 8,
Shape ( 8,
Shape ( 8,
Shape ( 5, 5 )

```

Microsoft Visual C++ Runtime Library



Debug Error!

Program: C:\WC++\HelloWorld\Debug\HelloWorld.exe
Module: C:\WC++\HelloWorld\Debug\HelloWorld.exe
File:

Run-Time Check Failure #2 - Stack around the variable 'ss' was corrupted.

(Press Retry to debug the application)

종단(A)

다시 시도(R)

무시(I)

```

37 int main(){
38     Rectangle r;
39     r.setOrigin(1,1);
40     r.setWidth(3);
41     r.draw(); cout << endl;
42
43     Shape *p_s;
44     p_s = &r; // Rectangle * --> Shape *
45     p_s->setOrigin(10, 10);
46     p_s->setWidth(100); // compile error
47     p_s->draw(); cout << endl;
48
49     Rectangle *p_r;
50     p_r = (Rectangle *)p_s; // Shape * --> Rectangle *
51     p_r->setOrigin(8, 8);
52     p_r->setWidth(16);
53     p_r->draw(); cout << endl;

```

Shape *p_s
Rectangle *p_r
(Circle *)p_s

(Shape& s)r

x,y
width
height

Shape * → Rectangle * 변환 생성자는 필요없음

```

((Circle *)p_s)->setRadius(11);
((Circle *)p_s)->draw(); cout << endl;
p_r->draw(); cout << endl;

```

```

Shape& s = r;
s.setOrigin(5,5);
s.setWidth(7); // compile error
s.draw(); cout << endl;

```

```

Shape ss;
ss.setOrigin(3, 3);
p_r = (Rectangle*)&ss;
p_r->setWidth(99);
p_r->draw(); cout << endl;

```

Shape → Rectangle 변환 생성자가 필요함

```

Rectangle& r_r = (Rectangle)s; // compile error

```

함수의 매개 변수

함수 호출에서 인자를 전달할 때에도 implicit upcasting 이 일어난다.
따라서 함수의 매개 변수는 서브클래스보다는 수퍼클래스 타입으로 선언하는 것이 좋다.

```
34 void move(Shape& s, int dx, int dy){  
35     s.setOrigin(dx, dy);  
36 }
```

```
65 Rectangle r;  
66 Circle c;  
67 move(r, 1,1);  
68 move(c, 2,2);
```

모든 도형을 받을 수 있다.

가상 함수

- 단순히 서브클래스 객체를 수퍼클래스 객체로 취급하는 것이 어디에 쓸모가 있을까?
- 다음과 같은 상속 계층도를 가정하여 보자.

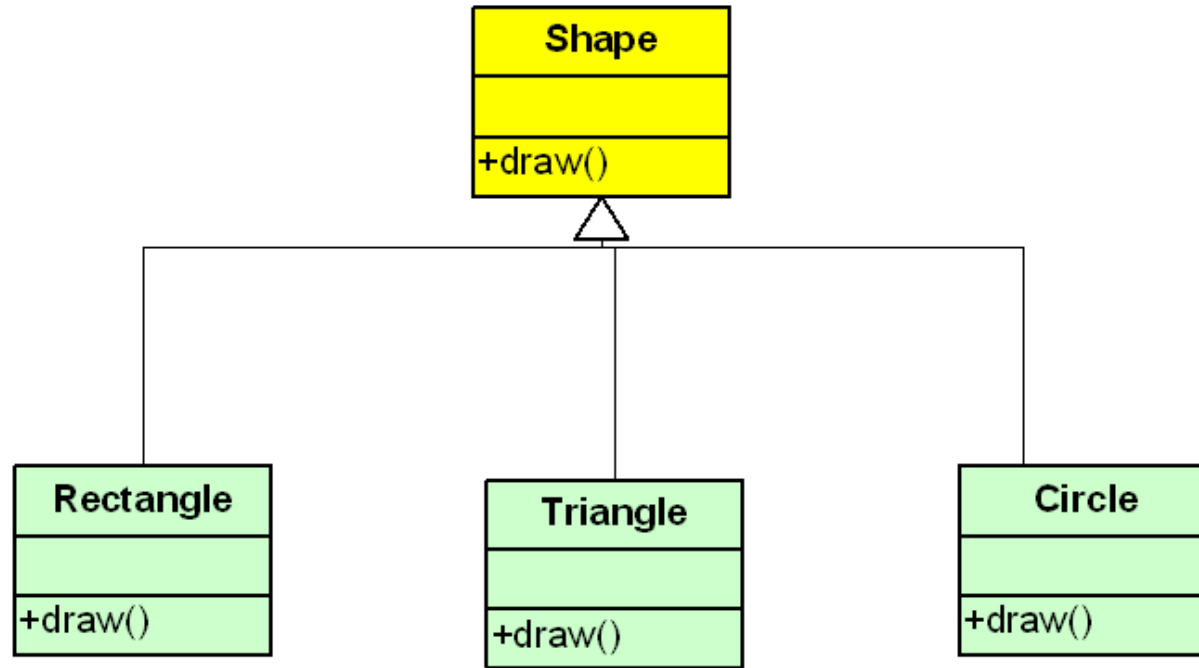


그림 14.6 도형의 UML


```

4 class Shape{
5 protected:
6     int x,y;
7 public:
8 > void setOrigin(int x, int y){=}
11 void draw(){
12     cout << "Shape ( " << x << ", " << y << " )
13 }
14 };
15 class Rectangle : public Shape{
16     int width, height;
17 public:
18     void setWidth(int w){ width = w; }
19     void setHeight(int h){ height = h;}
20     void draw(){
21         Shape::draw();
22         cout << "Rectangle : " << width << " x " << height << endl;
23     }
24 };
25 class Circle : public Shape{
26     int radius;
27 public:
28     void setRadius(int r){ radius = r;}
29     void draw(){
30         Shape::draw();
31         cout << "Circle : " << radius;
32     }
33 };

```

```

37 int main(){
38     Rectangle r;
39     r.setOrigin(1,1);
40     r.setWidth(3);
41     r.draw(); cout << endl;
42
43     Circle c;
44     c.setOrigin(4,4);
45     c.setRadius(5);
46     c.draw(); cout << endl;
47
48     Shape *p_s;
49     p_s = &r; // Rectangle * --> Shape *
50     p_s->draw(); cout << endl;
51     p_s = &c; // Circle * --> Shape *
52     p_s->draw(); cout << endl;
53
54     Shape& s1 = r;
55     s1.draw(); cout << endl;
56     Shape& s2 = c;
57     s2.draw(); cout << endl;

```

(Shape& s1)r

x,y
width
height

(Shape& s2)c

x,y
radius

```

Shape ( 1, 1 ) Rectangle : 3 x 21873
Shape ( 4, 4 ) Circle : 5
Shape ( 1, 1 )
Shape ( 4, 4 )
Shape ( 1, 1 )
Shape ( 4, 4 )

```

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


```

4 class Shape{
5     protected:
6         int x,y;
7     public:
8 >     void setOrigin(int x, int y){=}
11     virtual void draw(){
12         cout << "Shape ( " << x << ", " << y << " ) ";
13     }
14 };
15 class Rectangle : public Shape{
16     int width, height;
17     public:
18     void setWidth(int w){ width = w; }
19     void setHeight(int h){ height = h;}
20     void draw(){
21         Shape::draw();
22         cout << "Rectangle : " << width << " x " << height;
23     }
24 };
25 class Circle : public Shape{
26     int radius;
27     public:
28     void setRadius(int r){ radius = r;}
29     void draw(){
30         Shape::draw();
31         cout << "Circle : " << radius;
32     }
33 };

```

```

37 int main(){
38     Rectangle r;
39     r.setOrigin(1,1);
40     r.setWidth(3);
41     r.draw(); cout << endl;
42
43     Circle c;
44     c.setOrigin(4,4);
45     c.setRadius(5);
46     c.draw(); cout << endl;
47
48     Shape *p_s;
49     p_s = &r; // Rectangle * --> Shape *
50     p_s->draw(); cout << endl;
51     p_s = &c; // Circle * --> Shape *
52     p_s->draw(); cout << endl;
53
54     Shape& s1 = r;
55     s1.draw(); cout << endl;
56     Shape& s2 = c;
57     s2.draw(); cout << endl;

```

```

Shape ( 1, 1 ) Rectangle : 3 x 32536
Shape ( 4, 4 ) Circle : 5
Shape ( 1, 1 ) Rectangle : 3 x 32536
Shape ( 4, 4 ) Circle : 5
Shape ( 1, 1 ) Rectangle : 3 x 32536
Shape ( 4, 4 ) Circle : 5

```

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.

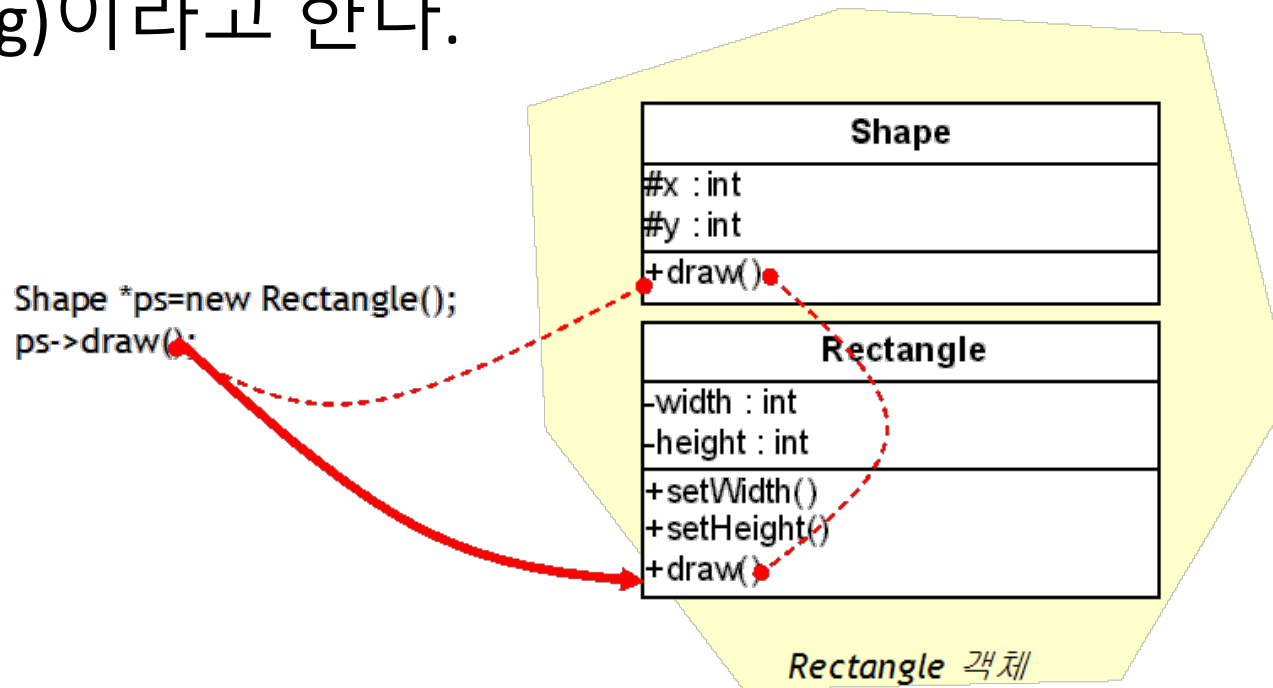


그림 14.8 동적 바인딩

정적 바인딩과 동적 바인딩

바인딩의 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.	느다	가상 함수

가상 함수의 구현

- V-table을 사용한다.

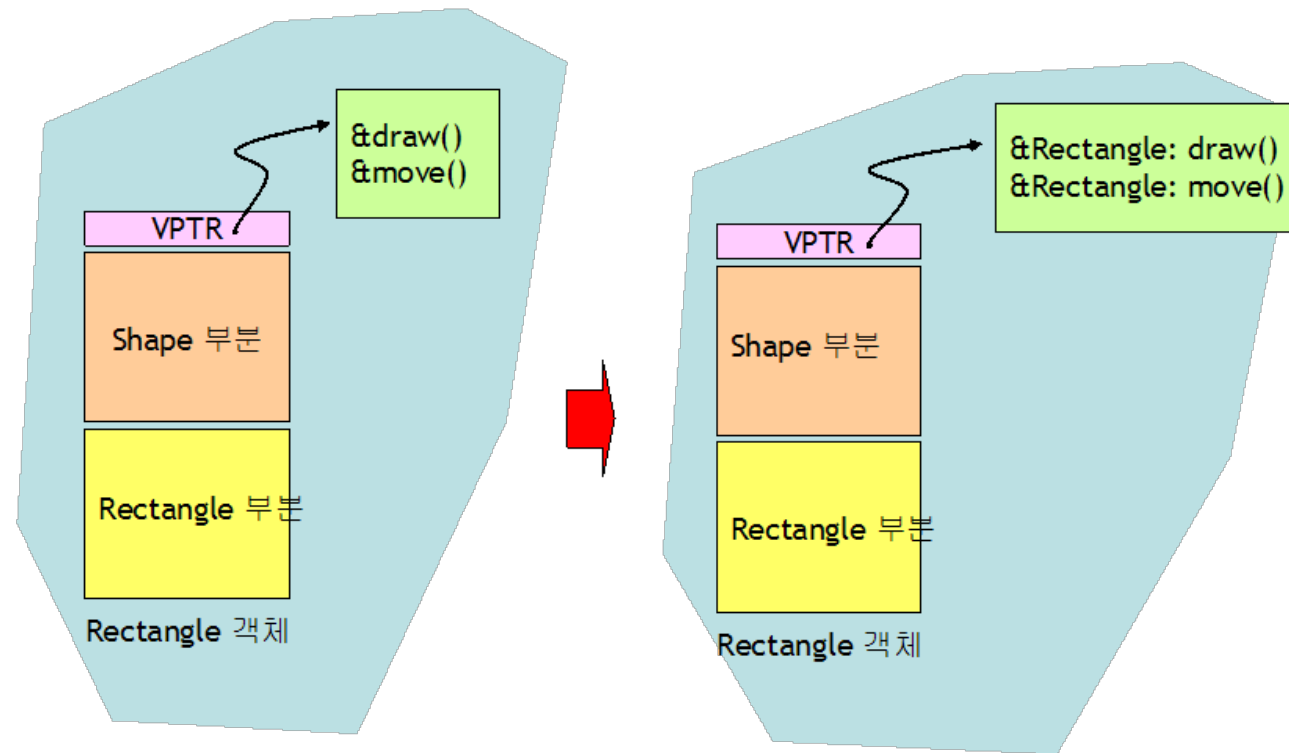


그림 14.9 가상 함수의 구현

```

4 class Shape{
5     protected:
6         int x,y;
7     public:
8 >     void setOrigin(int x, int y){=}
11     virtual void draw(){
12         cout << "Shape ( " << x << ", " << y << " ) ";
13     }
14 };
15 class Rectangle : public Shape{
16     int width, height;
17     public:
18     void setWidth(int w){ width = w; }
19     void setHeight(int h){ height = h;}
20     void draw(){
21         Shape::draw();
22         cout << "Rectangle : " << width << " x " << height;
23     }
24 };
25 class Circle : public Shape{
26     int radius;
27     public:
28     void setRadius(int r){ radius = r;}
29     void draw(){
30         Shape::draw();
31         cout << "Circle : " << radius;
32     }
33 };

```

```

Shape ( 1, 1 )
Shape ( 4, 4 )
Shape ( 1, 1 ) Rectangle : 3 x 3
Shape ( 4, 4 ) Circle : 5

```

```

37 int main(){
38     Rectangle r;
39     r.setOrigin(1,1);
40     r.setWidth(3); r.setHeight(3)
41
42     Circle c;
43     c.setOrigin(4,4);
44     c.setRadius(5);
45
46     Shape shapeArray[2];
47
48     shapeArray[0] = r;
49     shapeArray[1] = c;
50
51     for (int i=0; i<2; i++){
52         shapeArray[i].draw();
53         cout<< endl;
54     }
55
56     Shape *shapePtrs[2];
57
58     shapePtrs[0] = &r;
59     shapePtrs[1] = &c;
60
61     for (int i=0; i<2; i++){
62         shapePtrs[i]->draw();
63         cout<< endl;
64     }

```

Upcasting/Downcasting 으로 부르지 않는 경우

(아래와 같은 생성자가 있다고 가정)

```
Rectangle r(3, 5, 10, 20);
```

r

x : 3
y : 5
width : 10
height : 20

```
Shape s(2, 4);
```

s

x : 2
y : 4

```
s = r; // OK, Rectangle is a Shape
```

s

x : 3
y : 5

r

x : 3
y : 5
width : 10
height : 20

object slicing

s 는 Rectangle 의 Shape 멤버 데이터만 취함
(object slicing 이라고 부름)

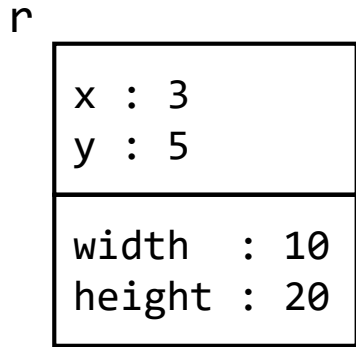
즉 s 는 Rectangle 이 아니고 Shape 임 (당연)

```
r = (Rectangle) s;  
    // compiler error  
    // Shape is not a Rectangle
```

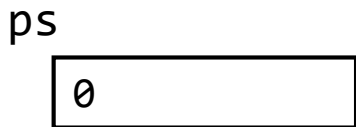
Upcasting/Downcasting (pointer)

(아래와 같은 생성자가 있다고 가정)

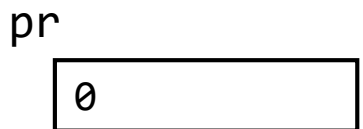
```
Rectangle r(3, 5, 10, 20);
```



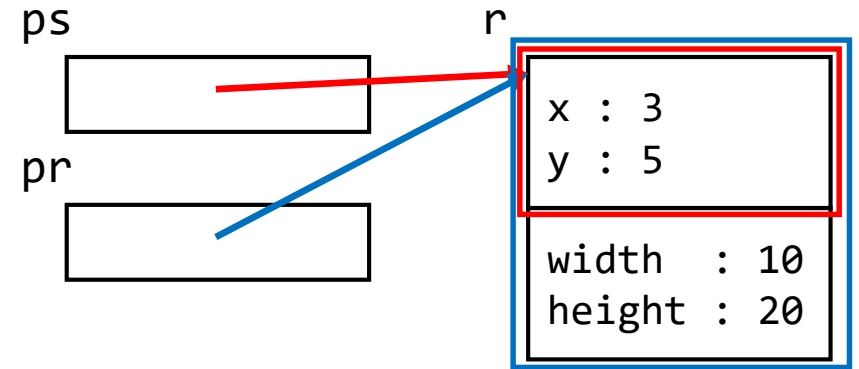
```
Shape *ps = NULL;
```



```
Rectangle *pr = NULL;
```



```
ps = &r; // upcasting (implicit)
```



ps 는 Rectangle 객체를 가리키고 있음
그러나, ps 를 통해서는 r 의 x, y 만 보임
(즉 Rectangle의 Shape 멤버데이터만 접근)

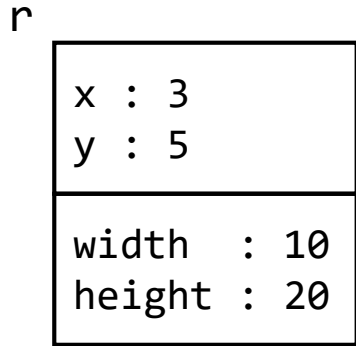
```
pr = (Rectangle *)ps; // downcasting
```

pr 는 Rectangle 객체를 가리키게 됨
pr 를 통해서는 r 의 모든
멤버 데이터를 볼 수 있음

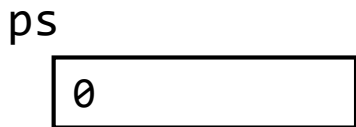
Upcasting/**Unsafe** Downcasting (pointer)

(아래와 같은 생성자가 있다고 가정)

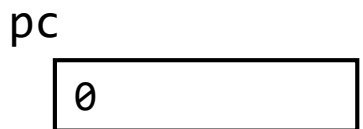
```
Rectangle r(3, 5, 10, 20);
```



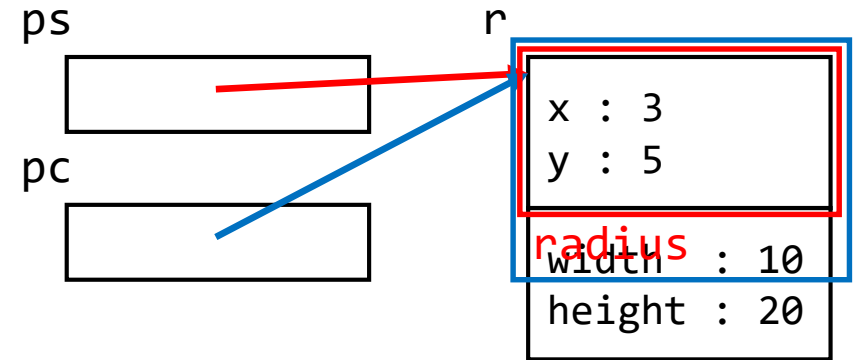
```
Shape *ps = NULL;
```



```
Circle *pc = NULL;
```



```
ps = &r; // upcasting (implicit)
```



ps 는 Rectangle 객체를 가리키고 있음
그러나, ps 를 통해서는 r 의 x, y 만 보임
(즉 Rectangle의 Shape 멤버데이터만 접근)

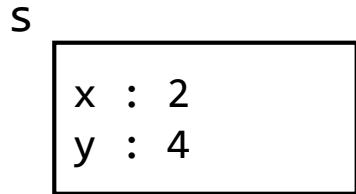
```
pc = (Circle *)ps; //unsafe downcasting
```

pc 는 Rectangle 객체를 가리키게 됨
pc 를 통해서는 r의 멤버데이터가 Circle의
멤버 데이터로 보임 (즉 width가 radius로 보임)

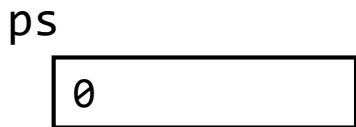
Upcasting/**Unsafe** Downcasting (pointer)

(아래와 같은 생성자가 있다고 가정)

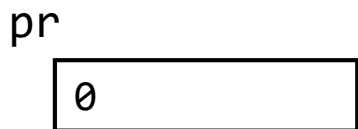
```
Shape s(2, 4);
```



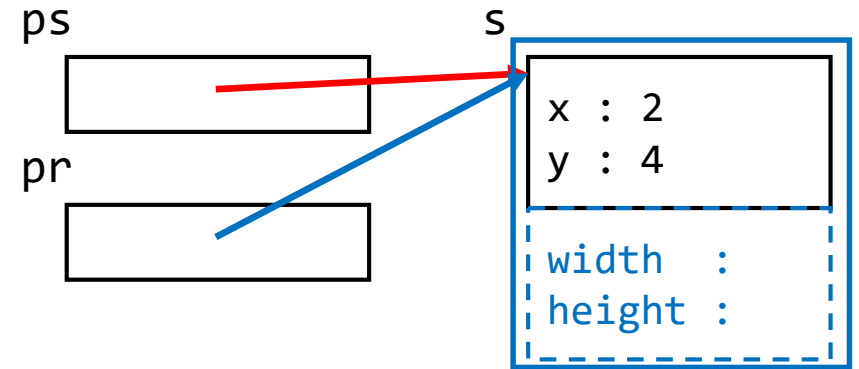
```
Shape *ps = NULL;
```



```
Rectangle *pr = NULL;
```



```
ps = &s; //
```



ps 는 Shape 객체를 가리키고 있음

```
pr = (Rectangle *)ps; //unsafe downcasting
```

pr 는 Shape 객체를 가리키게 됨

pr 을 통해서는 Rectangle 멤버를 접근하려 함
width, height는 메모리 할당안된 멤버변수임

- 읽을 경우는 쓰레기값을 읽어옴
- 쓸 경우는 프로그램 강제종료 혹은 논리에러 발생

Upcasting/Downcasting (pointer)

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

class Shape
{
protected:
    int x, y;
public:
    Shape(int x, int y): x(x), y(y) {}
    void draw() {
        cout << "Shape ( " << x << ", " << y << " ) ";
    }
};

class Rectangle : public Shape
{
    int width, height;
public:
    Rectangle(int x, int y, int w, int h) : Shape(x, y), width(w), height(h) {}
    void draw() {
        Shape::draw();
        cout << "Rectangle : " << width << " x " << height;
    }
};

class Circle : public Shape
{
    int radius;
public:
    Circle(int x, int y, int r) : Shape(x, y), radius(r) {}
    void draw() {
        Shape::draw();
        cout << "Circle : " << radius;
    }
};
```

```
int main()
{
    Shape s(2, 4);
    Rectangle r(3, 5, 10, 20);
    Circle c(0, 0, 100);
    Shape* ps = NULL;
    Rectangle* pr = NULL;
    Circle* pc = NULL;

    s.draw(); cout << endl;
    r.draw(); cout << endl;
    c.draw(); cout << endl;

    s = r;
    s.draw(); cout << endl;

    ps = &r;
    ps->draw(); cout << endl;

    pc = (Circle*)ps;
    pc->draw(); cout << endl;

    ps = &s;
    pr = (Rectangle*)ps;
    pr->draw(); cout << endl;

    return 0;
}
```

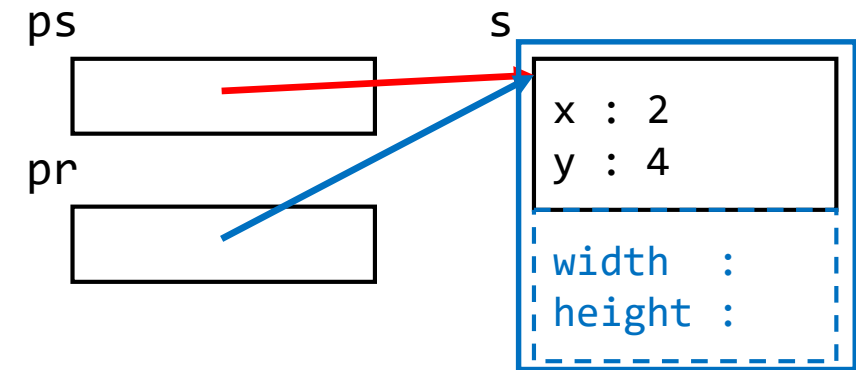
```
Shape (2, 4 )
Shape (3, 5 ) Rectangle :10 x 20
Shape (0, 0 ) Circle :100
Shape (3, 5 )
Shape (3, 5 )
Shape (3, 5 ) Circle :10
Shape (3, 5 ) Rectangle :-858993460 x -858993460
```

dynamic_cast

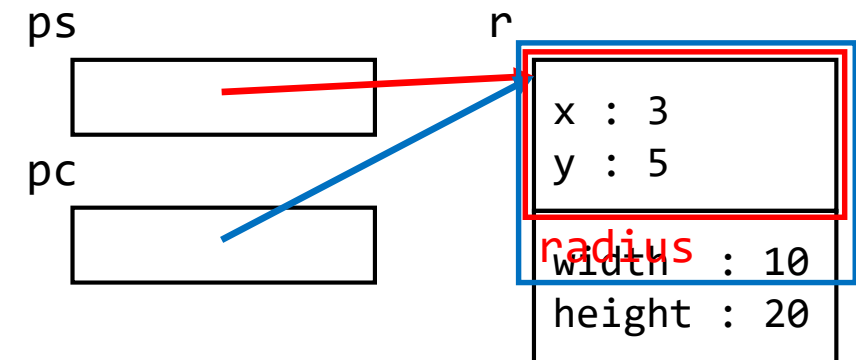
- dynamic_cast
 - unsafe 한 downcasting 방지

```
ChildClass *p = dynamic_cast<ChildClass *>(ParentClass 포인터 변수);
```

```
Shape s(2, 4);  
Shape *ps = &s;  
Rectangle *pr = dynamic_cast<Rectangle *>(ps);
```



```
Rectangle r(3, 5, 10, 20);  
Shape *ps = &r;  
Circle *pc = dynamic_cast<Circle *>(ps);
```



dynamic_cast

- dynamic_cast

- unsafe 한 downcasting 방지

```
ChildClass *p = dynamic_cast<ChildClass *>(ParentClass 포인터 변수);
```

- 결과

- safe downcasting 인 경우
: 포인터 p에 정상적인 주소값(ParentClass 포인터 변수값)을 저장함
- unsafe downcasting 인 경우
: 포인터 p 에 NULL 값이 저장됨

- 사용의 예

```
Rectangle *pr = dynamic_cast<Rectangle *>(ps);  
if(pr != NULL)  
    // safe 한 작업 수행  
else  
    // unsafe한 downcasting 인 경우를 처리하는 작업 수행
```

멤버함수는 어디에 저장이 되나요?

- 객체의 멤버 변수 저장 장소
 - 일반 변수와 동일하게 stack, heap 등의 메모리에 저장됨
- 그러면 객체의 멤버 함수는?
 - 멤버 변수와 별도로 저장됨
 - 멤버 변수와 달리 각 객체마다 저장되는 것이 아님.
 - 멤버함수의 머신 코드는 메모리 영역의 텍스트 영역에 저장됨

멤버함수는 어디에 저장이 되나요?

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

class sample
{
    int value;
public:
    int getValue() { return value; }
    void setValue(int v) { value = v; }
};

int main()
{
    sample data;

    cout << sizeof(data) << endl;
    return 0;
}
```

4

다형성의 장점

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

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

```

#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; }
};

```

```

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();
    delete a1;
    Animal *a2 = new Cat();
    a2->speak();
    delete a2;
    return 0;
}

```

```

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

```


소멸자의 virtual 선언

- 소멸자를 virtual로 선언하지 않으면 문제가 발생할 수 있다.
- 소멸자를 정의해 주어야 하는 class 는 ?
 - 생성자에서 동적 할당을 하는 class
- 객체의 포인터가 슈퍼클래스 타입 포인터인 경우 소멸자의 virtual 선언을 하지 않으면 슈퍼클래스의 소멸자만 실행된다.

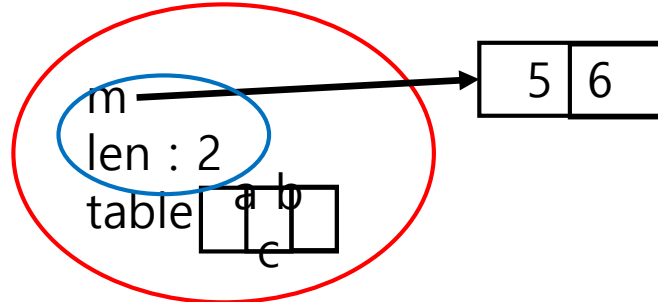
Kvector

```
5  class Kvector{
6  protected:
7      int *m;
8      int len;
9  public:
10     Kvector(int sz = 0, int value = 0): len(sz){
11         cout << this << " : Kvector(" << sz << ", " << value << ") \n";
12         if (!sz){ m = NULL; return; }
13         m = new int[sz];
14         for (int i=0; i<sz; i++) m[i] = value;
15     }
16     Kvector(const Kvector& v){
17         cout << this << " : Kvector(*" << &v << ")\n";
18         len = v.len;
19         if (!len){ m = NULL; return; }
20         m = new int[len];
21         for (int i=0; i<len; i++) m[i] = v.m[i];
22     }
23     ~Kvector(){
24         cout << this << " : ~Kvector() \n";
25         delete[] m;
26     }
```

Avector 에 출력문이 있는 copy constructor 와 destructor 를 추가


```
59  #define N 3
60  class Avector : public Kvector{
61      char table[N];
62  public:
63 >   Avector(int sz=0, int v=0, const char *t="abc"): Kvector(sz, v){=}
67 >   void setTable(const char *t){=}
70  friend ostream& operator<<(ostream& os, const Avector& v);
71  };
72 > ostream& operator<<(ostream& os, const Avector& v){=}
```

Avector a



a.print(); → 5 6
cout << a; → c a

```
97 int main(int argc, char *argv[]){
98     Avector v(3,1,"abc"); v.print();
99     Avector *p = new Avector(v); p->print();
100     Kvector *kp = new Avector(2,5,"xyz"); kp->print();
101     delete kp;
102     delete p;
103     return 0;
104 }
```



```
0x7fffa668ad10 : Kvector(3,1)
0x7fffa668ad10 : Avector(3,1,"abc")
1 1 1
0x55ffd5b0b2c0 : Kvector(*0x7fffa668ad10)
0x55ffd5b0b2c0 : Avector(*0x7fffa668ad10)
1 1 1
0x55ffd5b0b330 : Kvector(2,5)
0x55ffd5b0b330 : Avector(2,5,"xyz")
5 5
0x55ffd5b0b330 : ~Kvector()
0x55ffd5b0b2c0 : ~Avector()
0x55ffd5b0b2c0 : ~Kvector()
0x7fffa668ad10 : ~Avector()
0x7fffa668ad10 : ~Kvector()
```

```


61 ~ class Avector : public Kvector{
62     char *table;
63     int table_len;
64 public:
65 ~ Avector(int sz=0, int v=0, const char *t="abc"): Kvector(sz, v){
66     cout << this << " : Avector(" <<sz <<',' << v << ",\"\" << t << "\")\n";
67     table_len = strlen(t);
68     table = new char[table_len+1];
69     strcpy(table, t);
70 }
71 ~ Avector(const Avector& v) : Kvector(v) {
72     cout << this << " : Avector(*\" << &v << ")\"n";
73     setTable(v.table);
74 }
75 ~ ~Avector(){
76     cout << this << " : ~Avector() \"n";
77     delete[] table;
78 }
79 ~ void setTable(const char *t){
80     delete[] table;
81     table_len = strlen(t);
82     table = new char[table_len+1];
83     strcpy(table, t);
84 }
85 ~ Avector& operator=(const Avector& v){
86     setTable(v.table);
87     this->Kvector::operator=(v);
88     return *this;
89 }

```

- Avector class 가 table 배열을 생성자에서 동적으로 할당받도록 변경하면

- 깊은 복사를 하는 복사 생성자
- 깊은 복사를 하는 =연산자
- 소멸자도 정의해야 함
- setTable() 함수도 변경

```
97 int main(int argc, char *argv[]){
98     Avector v(3,1,"abc"); v.print();
99     Avector *p = new Avector(v); p->print();
100     Kvector *kp = new Avector(2,5,"xyz"); kp->print();
101     delete kp;
102     delete p;
103     return 0;
104 }
```



```
0x7fffa668ad10 : Kvector(3,1)
0x7fffa668ad10 : Avector(3,1,"abc")
1 1 1
0x55ffd5b0b2c0 : Kvector(*0x7fffa668ad10)
0x55ffd5b0b2c0 : Avector(*0x7fffa668ad10)
1 1 1
0x55ffd5b0b330 : Kvector(2,5)
0x55ffd5b0b330 : Avector(2,5,"xyz")
5 5
0x55ffd5b0b330 : ~Kvector()
0x55ffd5b0b2c0 : ~Avector()
0x55ffd5b0b2c0 : ~Kvector()
0x7fffa668ad10 : ~Avector()
0x7fffa668ad10 : ~Kvector()
```

```
5  class Kvector{
6  protected:
7      int *m;
8      int len;
9  public:
10     Kvector(int sz = 0, int value = 0): len(sz){
11         cout << this << " : Kvector(" << sz << ", " << value << ") \n";
12         if (!sz){ m = NULL; return; }
13         m = new int[sz];
14         for (int i=0; i<sz; i++) m[i] = value;
15     }
16     Kvector(const Kvector& v){
17         cout << this << " : Kvector(*" << &v << ")\n";
18         len = v.len;
19         if (!len){ m = NULL; return; }
20         m = new int[len];
21         for (int i=0; i<len; i++) m[i] = v.m[i];
22     }
23     virtual ~Kvector(){
24         cout << this << " : ~Kvector() \n";
25         delete[] m;
26     }
```

```
97 int main(int argc, char *argv[]){
98     Avector v(3,1,"abc"); v.print();
99     Avector *p = new Avector(v); p->print();
100    Kvector *kp = new Avector(2,5,"xyz"); kp->print();
101    delete kp;
102    delete p;
103    return 0;
104 }
```

```
0x7ffc27099560 : Kvector(3,1)
0x7ffc27099560 : Avector(3,1,"abc")
1 1 1
0x55aaf73052c0 : Kvector(*0x7ffc27099560)
0x55aaf73052c0 : Avector(*0x7ffc27099560)
1 1 1
0x55aaf7305330 : Kvector(2,5)
0x55aaf7305330 : Avector(2,5,"xyz")
5 5
0x55aaf7305330 : ~Avector() delete[] table;
0x55aaf7305330 : ~Kvector() delete[] m;
0x55aaf73052c0 : ~Avector() delete[] table;
0x55aaf73052c0 : ~Kvector() delete[] m;
0x7ffc27099560 : ~Avector() delete[] table;
0x7ffc27099560 : ~Kvector() delete[] m;
```

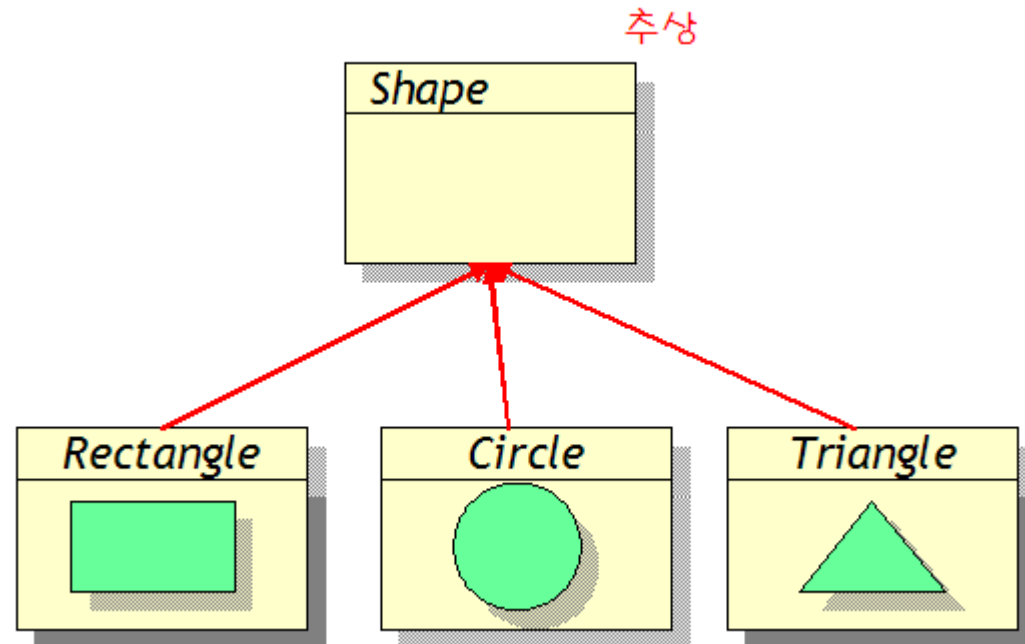

순수 가상 함수

- ^{순수}_{함수} 가상 함수(pure virtual function): 함수 헤더만 존재하고 함수의 몸체는 없는

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

- (예) virtual void draw() = 0;
- 추상 클래스(abstract class): 순수 가상 함수를 하나라도 가지고 있는 클래스
→ 객체를 생성할 수 없는 클래스

추상 클래스의 예



추상 클래스의 예

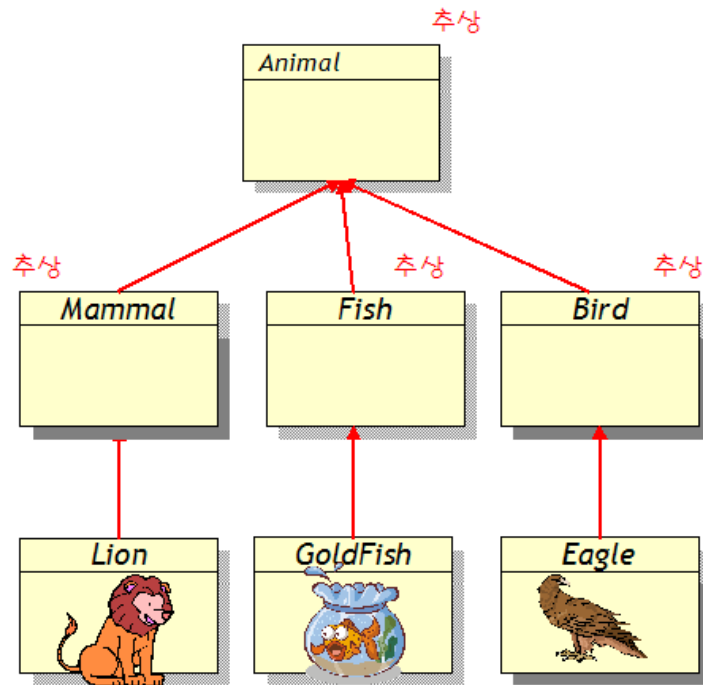


```
class Shape {  
protected:  
    int x, y;  
  
public:  
    ...  
    virtual void draw() = 0;  
};  
  
class Rectangle : public Shape {  
private:  
    int width, height;  
  
public:  
    void draw() {  
        cout << "Rectangle Draw" << endl;  
    }  
};
```

추상 클래스

- 추상 클래스(abstract class): 순수 가상 함수를 가지고 있는 클래스
- 추상 클래스는 추상적인 개념을 표현하는데 적당하다.
- 추상 클래스로는 객체를 생성할 수 없다.

Animal a; // compile error

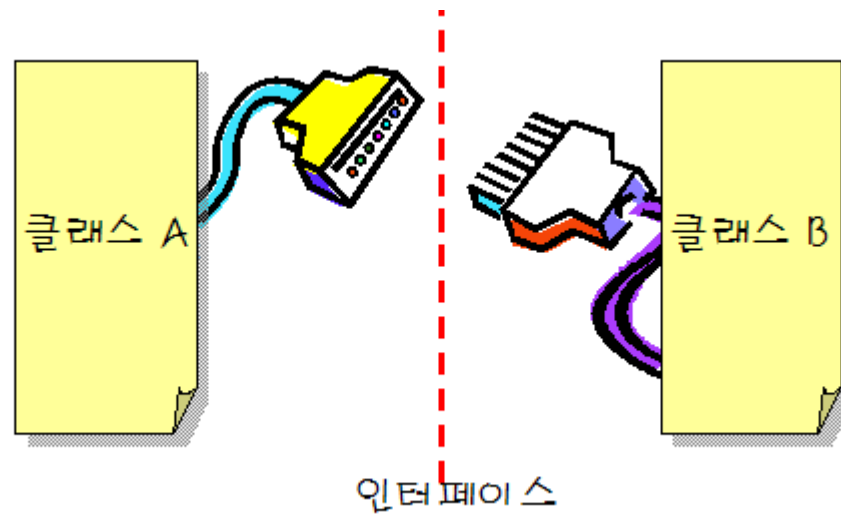


```
class Animal {
    virtual void move() = 0;
    virtual void eat() = 0;
    virtual void speak() = 0;
};

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

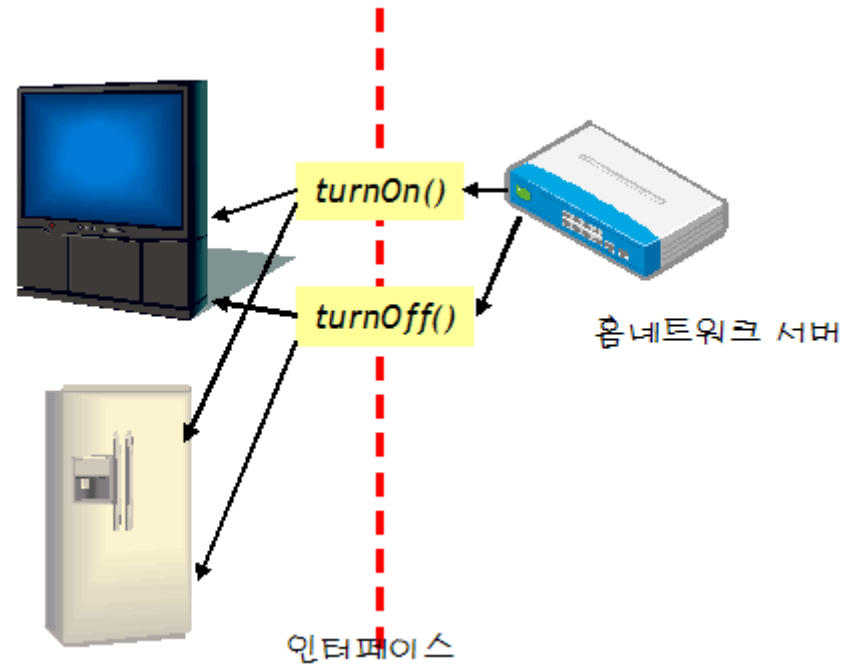
추상 클래스를 인터페이스로

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



인터페이스의 예

- 홈 네트워킹 예제



```

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

```

```

int main()
{

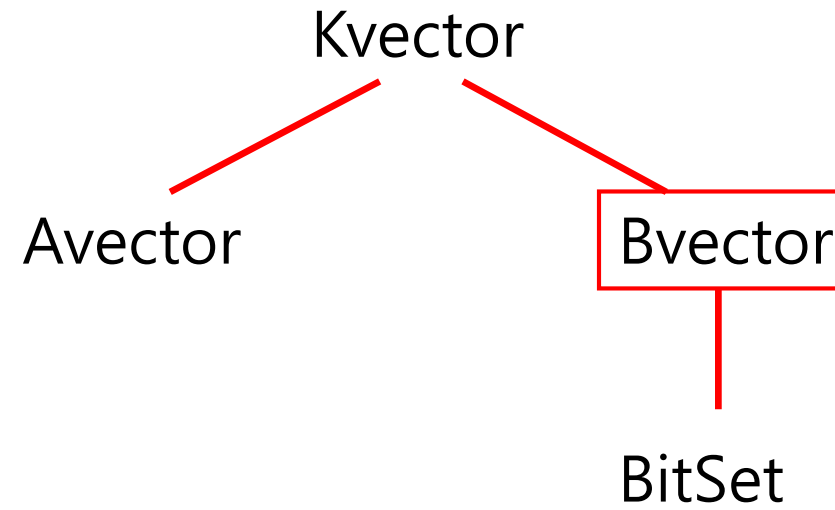
    Television *pt = new Television();
    pt->turnOn();
    pt->turnOff();

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

    delete pt;
    delete pr;
    return 0;
}

```

실습 class hierarchy

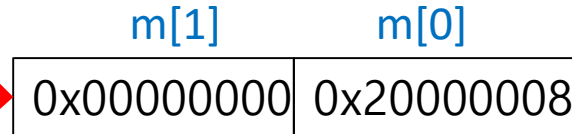
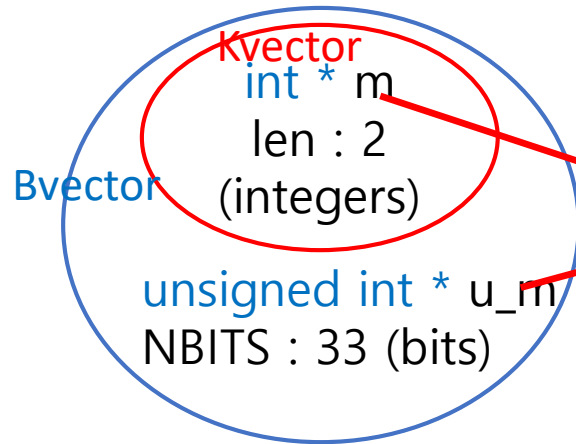


Kvector 를 상속하여 bit vector 를 표현하는 Bvector class 를 만들어 본다.

33bit bit vector

0000 0000 0000 0000 0000 0000 0000 0000
zero-padding

0 0010 0000 0000 0000 0000 0000 0000 1000



$\text{len} = \text{ceil}(33/32) = (33+32-1)/32 = 2$ integers needed to represent

```
1 // Kvector.h by ejim@kookmin.ac.kr
```

```
2 #include <iostream>
```

```
3 #ifndef __KVECTOR__
```

```
4 #define __KVECTOR__
```

같은 화일이 여러번 include 되었을 때 class 가 한 번만 선언되도록 하는 장치

```
5
```

```
6 class Kvector{
```

```
7 protected:
```

```
8     int *m;
```

```
9     int len;
```

```
10 public:
```

```
11     Kvector(int sz = 0, int value = 0);
```

```
12     Kvector(const Kvector& v);
```

```
13     virtual ~Kvector();
```

```
14     virtual void print() const{ std::cout << "Kvector\n"; }
```

```
15     void clear(){ delete[] m;    m = NULL;    len = 0;}
```

size() 와 연산자들은 virtual 함수가 아님

```
16     int size() const { return len; }
```

```
17     Kvector& operator=(const Kvector& v);
```

```
18     friend bool operator==(const Kvector& v, const Kvector& w);
```

```
19     friend bool operator!=(const Kvector& v, const Kvector& w);
```

```
20     int& operator[](int idx){ return m[idx]; }
```

```
21     const int& operator[](int idx) const { return m[idx]; }
```

```
22     friend std::ostream& operator<<(std::ostream& os, const Kvector& v);
```

```
23 };
```

```
24 #endif
```

```
// main.cpp by ejim
```

```
#include <iostream>
```

```
#include "Avector.h"
```

```
#include "Bvector.h"
```

```
using namespace std;
```

```

1 // Bvector.h by ejim@kookmin.ac.kr
2 #include <iostream>
3 #include "Kvector.h"
4
5 class Bvector : public Kvector{
6 protected:
7     unsigned int *u_m;          동적 할당을 받지 않으므로 소멸자를 정의할 필요가 없음
8     const int NBITS;
9 public:
10    Bvector(int nbits=32);
11    Bvector(const Bvector& e);
12    bool bit(int pos) const;      pos 번째 bit 의 값을 0 (false) or 1(true) 로 return
13    void set(int pos);           pos 번째 bit 의 값을 1 로 씌 (다른 bit 값은 바뀌면 안 됨!)
14    void reset(int pos);        pos 번째 bit 의 값을 0 으로 씌 (다른 bit 값은 바뀌면 안 됨!)
15    int size() const { return NBITS; }    len 이 아니라 NBITS 를 return
16    void clear(){ for(int i=0; i<len; i++) m[i]=0; }    모든 bits 를 false 로 clear
17    void print() const ;
18    bool operator[](int idx) const { return bit(idx); } reference 를 반환하지 않고 객체를 반환
19    Bvector& operator=(const Bvector& v);           v[0] = 1; // compile error → v.set(0);
20    friend std::ostream& operator<<(std::ostream& os, const Bvector& v);
21 };

```

overridden/
overloaded
functions

reference 를 반환하지 않고 객체를 반환
v[0] = 1; // compile error → v.set(0);

```

1  // Bvector.cpp by ejim@kookmin.ac.kr
2  #include <iostream>
3  #include "Bvector.h"
4  using namespace std;
5  Bvector::Bvector(int nbits): Kvector((nbits+31)/32, 0), NBITS(nbits){
6      cout << this << " : Bvector(" << nbits << ")\n";
7      u_m = (unsigned int *)m; ← bitwise 연산자 사용을 하려면 unsigned type 으로
8  }
9  Bvector::Bvector(const Bvector& e): Kvector(e), NBITS(e.NBITS){
10     cout << this << " : Bvector(" << &e << ")\n";
11     u_m = (unsigned int *)m;
12 }

```

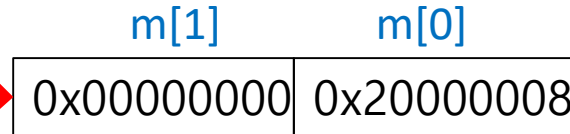
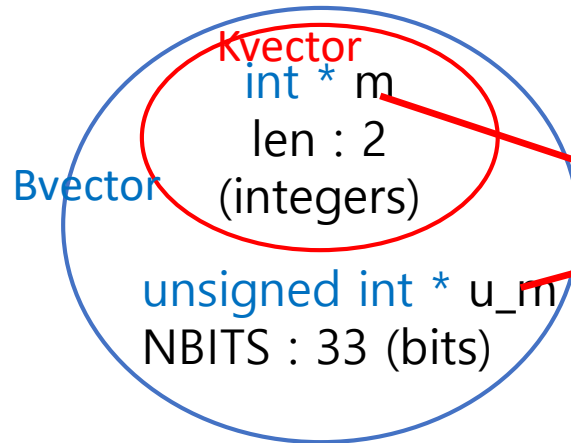
ceil(nbits/32)
↓

Kvector 를 상속하여 bit vector 를 표현하는 Bvector class

33bit bit vector

0000 0000 0000 0000 0000 0000 0000 0000
zero-padding

0 0010 0000 0000 0000 0000 0000 0000 1000



$\text{len} = \text{ceil}(33/32) = (33+32-1)/32 = 2$ integers needed to represent

```
13 Bvector& Bvector::operator=(const Bvector& v){
14 // default operator= is not created if there is a const member NBITS
15     cout << "Bvector::operator= " << &v << endl;
16     if (NBITS == v.NBITS){ 크기가 다르면 치환이 일어나지 않음
17         for (int i=0; i<len; i++) m[i] = v.m[i]; bit 단위로 복사하는 것보다 훨씬 빠르다.
18     }
19     return *this;
20 }
21 void Bvector::print() const { Kvector 의 virtual function
22     cout << "Bvector with " << NBITS << " bits\n";
23     for (int i=0; i<NBITS; i++) cout << bit(i) << " ";
24     cout << endl;
25 }
26 ostream& operator<<(ostream& os, const Bvector& v){
27     for (int i=0; i<v.NBITS; i++) os << v.bit(i) << " ";
28     return os;
29 }
```

bit 단위로 출력 (function overriding)

value == 0 이면 false(0) 아니면 true(1) 을 return

NBITS = 33, pos = 3

u_m[0]

$$\text{pos}/32 = 0, \text{pos_bit} = \text{pos} \% 32 = 3$$
 $0 \ \& \ x = 0$

&

value

→ true(=1)


```

45 void Bvector::reset(int pos){
46     if (pos >= NBITS) return;
47     unsigned int &element = u_m[pos/32];
48     int pos_bit = pos % 32;
49     unsigned int mask = 1 << pos_bit;
50     mask = ~mask;
51     element &= mask;
52 }

```

0010 0000 0000 0000 0000 0000 0000 0000
(← 1 << 29)

reset(29) 를 실행한다면

NBITS = 33, pos = 29

u_m[1]

u_m[0]

0000 0000 0000 0000 0000 0000 0000 000 0 0010 0000 0000 0000 0000 0000 0000 1000

pos/32 = 0, pos_bit = pos % 32 = 29

element 0010 0000 0000 0000 0000 0000 0000 1000

0 & x = 0

1 & x = x

&

mask

1101 1111 1111 1111 1111 1111 1111 1111

0000 0000 0000 0000 0000 0000 0000 1000

element 의 나머지 bits 들은 그대로 두고 pos_bit 번째 bit 만 0으로


```

38 void Bvector::set(int pos){
39     if (pos >= NBITS) return;
40     unsigned int &element = u_m[pos/32];
41     int pos_bit = pos % 32;
42     unsigned int mask = 1 << pos_bit;
43     element |= mask;
44 }

```

set(5) 를 실행한다면

NBITS = 33, pos = 5

u_m[1]

u_m[0]

0000 0000 0000 0000 0000 0000 0000 0000 0 0010 0000 0000 0000 0000 0000 0000 1000

pos/32 = 0, pos_bit = pos % 32 = 5

element 0010 0000 0000 0000 0000 0000 0000 1000

0 | x = x

1 | x = 1

| mask 0000 0000 0000 0000 0000 0000 0010 0000 (← 1 << 5)

0010 0000 0000 0000 0000 0000 0010 1000

element 의 나머지 bits 들은 그대로 두고 pos_bit 번째 bit 만 1로

```
1  // Kvector.cpp by ejim@kookmin.ac.kr
2  #include <iostream>
3  #include "Kvector.h"
4  using namespace std;
5  Kvector::Kvector(int sz, int value): len(sz){
6      cout << this << " : Kvector(" << sz << ", " << value << ")\n";
7      if (!sz){ m = NULL; return; }
8      m = new int[sz];
9      for (int i=0; i<sz; i++) m[i] = value;
10 }
11 Kvector::Kvector(const Kvector& v){
12     cout << this << " : Kvector(*" << &v << ")\n";
13     len = v.len;
14     if (!len){ m = NULL; return; }
15     m = new int[len];
16     for (int i=0; i<len; i++) m[i] = v.m[i];
17 }
18 Kvector::~Kvector(){
19     cout << this << " : ~Kvector() \n";
20     delete[] m;
21 }
```

```
22 Kvector& Kvector::operator=(const Kvector& v){
23     cout << "Kvector::operator= " << &v << endl;
24     delete[] m;
25     len = v.len;
26     m = new int[len];
27     for (int i=0; i<len; i++) m[i] = v.m[i];
28     return *this;
29 }
30 bool operator==(const Kvector& v, const Kvector& w){
31     if (v.len != w.len) return false;
32     for (int i=0; i<v.len; i++)
33         if (v.m[i] != w.m[i]) return false;
34     return true;
35 }
36 bool operator!=(const Kvector& v, const Kvector& w){
37     return !(v==w);
38 }
39 ostream& operator<<(ostream& os, const Kvector& v){
40     for (int i=0; i<v.len; i++) os << v.m[i] << " ";
41     return os;
42 }
```

```
1  // Avector.h  by ejim@kookmin.ac.kr
2  #include <iostream>
3  #include "Kvector.h"
4
5  #define N 3
6  class Avector : public Kvector{
7      char table[N];
8  public:
9      Avector(int sz=0, int v=0, const char *t="abc");
10     void setTable(const char *t);
11     void print() const ; ← Kvector 의 virtual function
12     friend std::ostream& operator<<(std::ostream& os, const Avector& v);
13 };

```

← Kvector 의 member function 도
virtual function 도 아님

```
1  // Avector.cpp by ejim@kookmin.ac.kr
2  #include <iostream>
3  #include "Avector.h"
4  using namespace std;
5
6  Avector::Avector(int sz, int v, const char *t): Kvector(sz, v){
7      cout << this << " : Avector(" << sz << ", " << v << ", " << t << ")\n";
8      for (int i=0; i<N; i++) table[i] = t[i];
9  }
10 void Avector::setTable(const char *t){
11     for (int i=0; i<N; i++) table[i] = t[i];
12 }
13 void Avector::print() const {
14     cout << "Avector with table " << table << endl;
15     for (int i=0; i<len; i++) cout << m[i] << " ";
16     cout << endl;
17 }
18 ostream& operator<<(ostream& os, const Avector& v){
19     for (int i=0; i<v.len; i++) os << v.table[v.m[i]%N] << " ";
20     return os;
21 }
```

Avector 의 사용

```

1 // main.cpp by ejim@kookmin.ac.kr
2 #include <iostream>
3 #include "Avector.h"
4 #include "Bvector.h"
5 using namespace std;
6 int main(int argc, char *argv[]){
7     Avector a1(7, 4, "stu");
8     Avector a2(a1);
9     cout << "a1 == a2 " << (a1==a2) << endl;
10    a1[0] = 0;
11    a2[1] = 5;
12    a2.setTable("ijk");
13    a1.print();
14    a2.print();
15    cout << "a1.size() : " << a1.size() << endl;
16    cout << "a2.size() : " << a2.size() << endl;
17    cout << "a1 : " << a1 << endl;
18    cout << "a2 : " << a2 << endl;
19    a1.clear();
20    a1.print();
21    cout << "a1.size() : " << a1.size() << endl;
22    cout << "a1 : " << a1 << endl;
23    a1 = a2;
24    a1.print();
25    cout << "a1.size() : " << a1.size() << endl;
26    cout << "a1 : " << a1 << endl;

```

m-> 4 4 4 4 4 4 4 table = stu
 a1 : m-> 4 5 4 4 4 4 4 table = ijk
 default 생성된 Avector::op= 에서 Kvector::op= 이 호출된다

```

a1 0x7fff7a88e280 : Kvector(7,4)
0x7fff7a88e280 : Avector(7,4,stu)
a2 0x7fff7a88e2a0 : Kvector(*0x7fff7a88e280)
a1 == a2 1
Avector with table stu
0 4 4 4 4 4 4
Avector with table ijk
4 5 4 4 4 4 4
a1.size() : 7
a2.size() : 7
a1 : s t t t t t t
a2 : j k j j j j j
Avector with table stu

a1.size() : 0
a1 :
Kvector::operator= 0x7fff7a88e2a0
Avector with table ijk
4 5 4 4 4 4 4
a1.size() : 7
a1 : j k j j j j j

```

```

1  // Kvector.h by ejim@kookmin.ac.kr
2  #include <iostream>
3  #ifndef __KVECTOR__
4  #define __KVECTOR__
5
6  class Kvector{
7  protected:
8      int *m;
9      int len;
10 public:
11     Kvector(int sz = 0, int value = 0);
12     Kvector(const Kvector& v);
13     virtual ~Kvector();
14     virtual void print() const{ std::cout << "Kvector\n"; }
15     void clear(){ delete[] m;    m = NULL;    len = 0;}
16     int size() const { return len; }
17     Kvector& operator=(const Kvector& v);
18     friend bool operator==(const Kvector& v, const Kvector& w);
19     friend bool operator!=(const Kvector& v, const Kvector& w);
20     int& operator[](int idx){ return m[idx]; }
21     const int& operator[](int idx) const { return m[idx]; }
22     friend std::ostream& operator<<(std::ostream& os, const Kvector& v);
23 };
24 #endif

```

size() 와 연산자들은
virtual 함수가 아님

할당받은 메모리를 반환

```

1 // Bvector.h by ejim@kookmin.ac.kr
2 #include <iostream>
3 #include "Kvector.h"
4
5 class Bvector : public Kvector{
6 protected:
7     unsigned int *u_m;          동적 할당을 받지 않으므로 소멸자를 정의할 필요가 없음
8     const int NBITS;
9 public:
10    Bvector(int nbits=32);
11    Bvector(const Bvector& e);
12    bool bit(int pos) const;      pos 번째 bit 의 값을 0 (false) or 1(true) 로 return
13    void set(int pos);            pos 번째 bit 의 값을 1 로 씌 (다른 bit 값은 바뀌면 안 됨!)
14    void reset(int pos);          pos 번째 bit 의 값을 0 으로 씌 (다른 bit 값은 바뀌면 안 됨!)
15    int size() const { return NBITS; }   len 이 아니라 NBITS 를 return
16    void clear(){ for(int i=0; i<len; i++) m[i]=0; }   모든 bits 를 false 로 clear
17    void print() const ;
18    bool operator[](int idx) const { return bit(idx); } reference 를 반환하지 않고 객체를 반환
19    Bvector& operator=(const Bvector& v);              v[0] = 1; // compile error → v.set(0);
20    friend std::ostream& operator<<(std::ostream& os, const Bvector& v);
21 };

```

overridden/
overloaded
functions

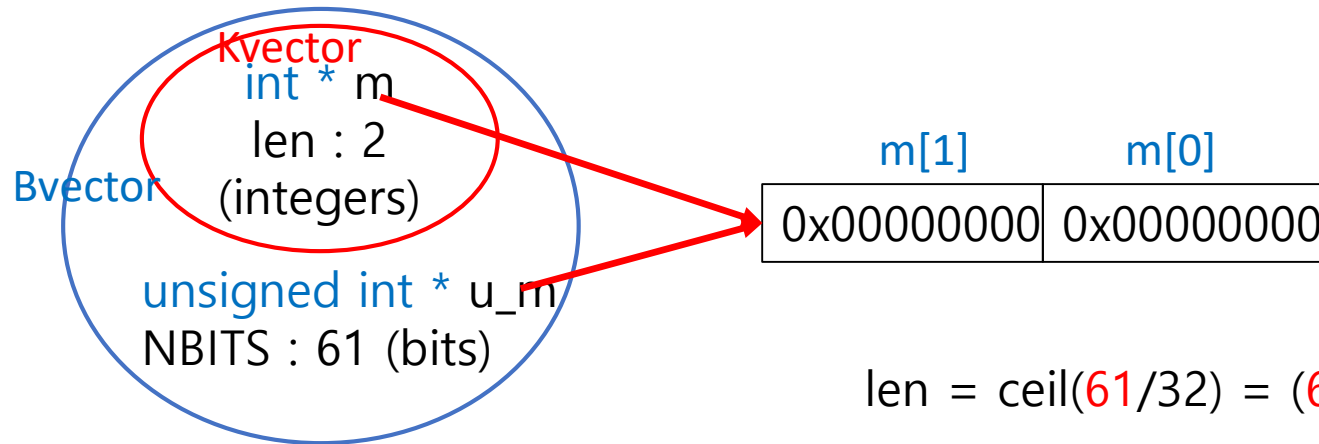
reference 를 반환하지 않고 객체를 반환
v[0] = 1; // compile error → v.set(0);

Bvector b1(61);

61bit bit vector

000 zero-padding

0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
--



$\text{len} = \text{ceil}(61/32) = (61+32-1)/32 = 2$ integers needed to represent

```
13 Bvector& Bvector::operator=(const Bvector& v){
14 // default operator= is not created if there is a const member
15     cout << "Bvector::operator= " << &v << endl;
16     if (NBITS == v.NBITS){ 크기가 다르면 치환이 일어나지 않음
17         for (int i=0; i<len; i++) m[i] = v.m[i]; bit 단위로 복사하는 것보다 훨씬 빠르다.
18     }
19     return *this;
20 }
```

```
21 void Bvector::print() const { Kvector 의 virtual function
22     cout << "Bvector with " << NBITS << " bits\n";
23     for (int i=0; i<NBITS; i++) cout << bit(i) << " ";
24     cout << endl;
25 }
```

```
26 ostream& operator<<(ostream& os, const Bvector& v){
27     for (int i=0; i<v.NBITS; i++) os << v.bit(i) << " ";
28     return os;
29 }
```

bit 단위로 출력 (function overriding)

객체를 pointer 를 통해 접근할 때 virtual function 의 사용

```

47  Avector *ap = new Avector(5, 2, "xyz");
48  Bvector *bp = new Bvector(50);
49  Kvector *karray[2] = {ap, bp};
50
51  ap->print();
52  cout << *ap << endl;
53  cout << "ap->size() : " << ap->size() << endl;
54  bp->print();
55  cout << *bp << endl;
56  cout << "bp->size() : " << bp->size() << endl;
57
58  for(int i=0; i<2; i++){
59      cout << i << endl;
60      karray[i]->print();
61      cout << *(karray[i])<< endl;
62      cout << "size() : " << karray[i]->size() << endl;
63  }

```

```

0x564a93077320 : Kvector(5,2)
0x564a93077320 : Avector(5,2,xyz)
0x564a93077360 : Kvector(2,0)
0x564a93077360 : Bvector(50)
Avector with table xyz
2 2 2 2 2
z z z z z      operator<<(ostream&, Avector&)
ap->size() : 5   Avector::size()
Bvector with 50 bits
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0      operator<<(ostream&, Bvector&)
bp->size() : 50      Bvector::size()
0
Avector with table xyz      operator<< 과 size() 는
2 2 2 2 2      Avector::print()      virtual function 이 아님
2 2 2 2 2      operator<<(ostream&, Kvector&)
size() : 5      Kvector::size()
1
Bvector with 50 bits
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0      Bvector::print()
0 0      operator<<(ostream&, Kvector&)
size() : 2      Kvector::size()

```

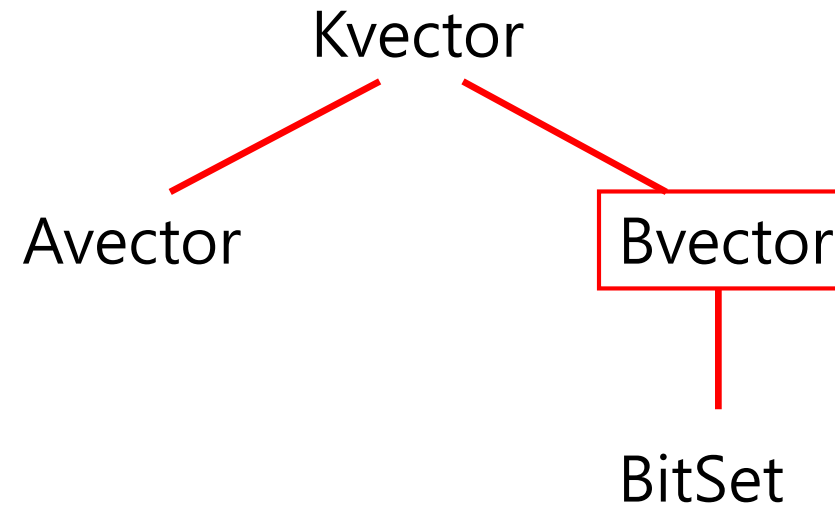
```

1  // Kvector.h by ejim@kookmin.ac.kr
2  #include <iostream>
3  #ifndef __KVECTOR__
4  #define __KVECTOR__
5
6  class Kvector{
7  protected:
8      int *m;
9      int len;
10 public:
11     Kvector(int sz = 0, int value = 0);
12     Kvector(const Kvector& v);
13     virtual ~Kvector();
14     virtual void print() const{ std::cout << "Kvector\n"; }
15     void clear(){ delete[] m;    m = NULL;    len = 0;}
16     int size() const { return len; }
17     Kvector& operator=(const Kvector& v);
18     friend bool operator==(const Kvector& v, const Kvector& w);
19     friend bool operator!=(const Kvector& v, const Kvector& w);
20     int& operator[](int idx){ return m[idx]; }
21     const int& operator[](int idx) const { return m[idx]; }
22     friend std::ostream& operator<<(std::ostream& os, const Kvector& v);
23 };
24 #endif

```

size() 와 연산자들은 virtual 함수가 아님 할당받은 메모리를 반환

실습 class hierarchy




```
1  # Makefile
2  CC = g++
3  CCFLAGS = -g
4
5  Bvector: main.o Bvector.o Avector.o Kvector.o
6      $(CC) $(CCFLAGS) -o Bvector main.o Bvector.o Avector.o Kvector.o
7
8  BitSet: BitSet.o Bvector.o Kvector.o
9      $(CC) $(CCFLAGS) -o BitSet BitSet.o Bvector.o Kvector.o
10
11 clean:
12     rm -f *.o
13
14 %.o : %.cpp %.h
15     $(CC) $(CCFLAGS) -c $<
16
17 %.o : %.cpp
18     $(CC) $(CCFLAGS) -c $<
19
20 % : %.cpp
21     $(CC) $(CCFLAGS) -o $@ $<
```

```
ejim@ejim-VirtualBox:~/C2020/Vectors$ make
g++ -g -c main.cpp
g++ -g -c Bvector.cpp
g++ -g -c Avector.cpp
g++ -g -c Kvector.cpp
g++ -g -o Bvector main.o Bvector.o Avector.o Kvector.o
```

실습

- Bvector 를 상속하여 0~N-1 의 숫자들 중 일부를 원소로 갖는 BitSet class 를 BitSet.h 와 BitSet.cpp 를 만들어 앞의 Makefile 을 이용하여 컴파일하고 수행하라.
- 1) N을 인자로 갖는 생성자: 생성된 집합 객체는 공집합을 표현한다.
 - 2) i 를 집합의 원소로 추가하는 insert() 멤버 함수
 - 3) 합집합 연산자(operator+) 를 friend 함수로 구현하라. (overloading)
 - 4) 출력 연산자(operator<<)를 friend 함수로 구현하라.
- 다음의 main() 함수에 대한 출력이 제시된 바와 같아야 한다. main() 함수는 BitSet.cpp 에 포함하여 작성하라.

BitSet.h

```
1  // BitSet.h by ejim@kookmin.ac.kr
2  #include <iostream>
3  #include "Bvector.h"
4  class BitSet : public Bvector{
5      public:
6          BitSet(int sz=32);
7      void insert(int v);
8      friend BitSet operator+(const BitSet& v1, const BitSet& v2);
9      friend std::ostream& operator<<(std::ostream& os, const BitSet& s);
10 };
```

BitSet.cpp

```
1 // BitSet.cpp
2 #include <iostream>
3 #include "BitSet.h"
4 using namespace std;
```

```
22 int main(int argc, char *argv[]){
23     BitSet b1(131), b2(131);
24     b1.insert(3); b1.insert(5); b1.insert(8);
25     b2.insert(4); b2.insert(5); b2.insert(8); b2.insert(130);
26     b1.print(); b2.print();
27     cout << "b1= " << b1 << endl;
28     cout << "b2= " << b2 << endl;
29     cout << "b1+b2= " << b1+b2 << endl;
```

[illegible]