C++ 프로그래밍

한국방송통신대학교
컴퓨터과학과 이병래 교수

제4장 클래스와 객체

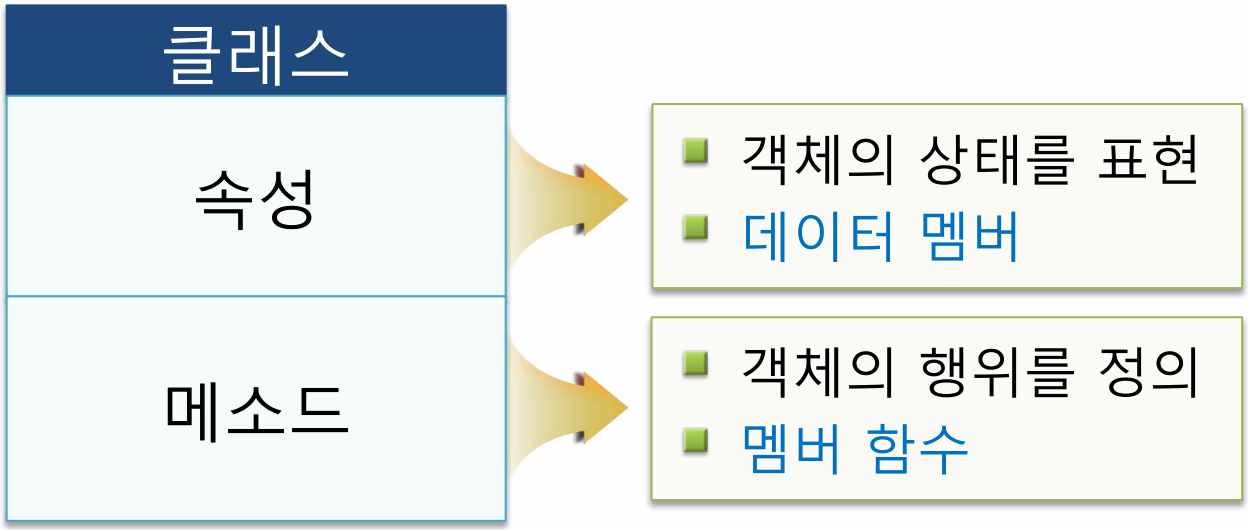
학습활동 4 : 클래스의 선언

교재 121쪽의 [소스코드 4-6]에 하향계수(계수기 값을 1씩 감소시킴) 멤버함수 countDown()을 추가하고, 이를 사용하는 프로그램을 작성하라.

클래스 선언

클래스

- 표현하고자 하는 대상의 메소드와 속성을 선언한 것



클래스 선언

- 클래스 선언문의 형식

```
class ClassName {
가시성_지시어_1:
    데이터 멤버 또는 멤버함수 리스트;
가시성_지시어_2:
    데이터 멤버 또는 멤버함수 리스트;
    ......
};
```

클래스 선언

가시성 지시어

- 클래스의 멤버가 공개되는 범위를 나타냄
- 종류 : private, public, protected

가시성 지시어	공개 범위
private	■ 소속 클래스의 멤버함수 ■ 친구 클래스의 멤버함수 및 친구함수
public	■ 전 범위

☞디폴트는 private임

생성자

- 생성자(constructor)란?
  - 객체가 생성될 때 수행할 작업을 정의하는 특수한 멤버함수

- 생성자의 특성
  - 생성자의 이름은 클래스의 이름과 같다.
  - 생성자는 return 명령으로 값을 반환할 수 없으며, 함수 머리에 반환 자료형을 표시하지 않는다.
  - public으로 선언해야 클래스 외부에서 객체를 생성할 수 있다.
  - 생성자를 다중정의할 수 있다.

생성자

생성자 선언 형식

```
class ClassName {
      ......
public:
    ClassName(fParameterList) {
          ......
    }
      ......
};
```

소멸자

소멸자(destructor)란?

- 객체가 소멸될 때 수행할 작업을 정의하는 특수한 멤버함수

소멸자의 특성

① 소멸자의 이름은 클래스의 이름과 같으나, 앞에 '~'가 붙는다.

② 소멸자는 return 명령으로 값을 반환할 수 없으며, 함수 머리에 반환 자료형을 표시하지 않는다.

③ 인수를 전달받을 매개변수를 포함할 수 없다.

④ 소멸자를 다중정의 할 수 없으며, 클래스에 하나만 정의한다.

⑤ public으로 선언하는 것이 일반적이다.

⑥ 상속을 통해 파생 클래스를 정의하는 경우 virtual을 지정하여 가상함수가 되도록 하는 것이 좋다.

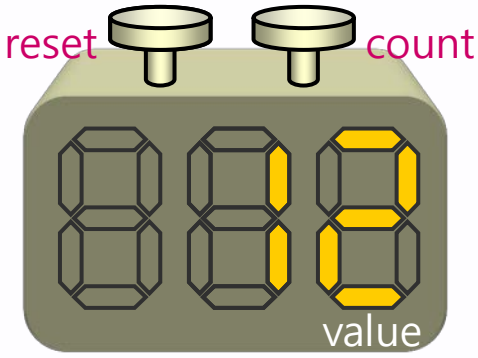
소멸자

소멸자 선언 형식

```
class ClassName {
    ......
public:
    ClassName(fParameterList) { // 생성자
        ......    // 객체 생성을 위한 준비 작업
    }
    ~ClassName() {                    // 소멸자
        ......    // 객체 제거를 위한 정리 작업
    }
    ......
};
```

CounterM 클래스의 명세

CounterM 클래스
계수기를 나타내는 클래스를 선언하라. 계수기 객체는 값을 0으로 지울 수 있고, 값을 1씩 증가시킬 수 있으며, 현재의 계수기 값을 알려줄 수 있다. 계수기는 지정된 최대값까지 계수할 수 있으며, 최대값 다음에는 0으로 되돌아간다.



행위

메소드	비 고
`CounterM(int mVal)`	생성자
`void reset()`	계수기의 값을 0으로 지움
`void count()`	계수기의 값을 +1 증가시킴
`int getValue()`	계수기의 현재 값을 알려 줌

속성

속성	비 고
`const int maxValue`	최대 계수 값
`int value`	계수기의 현재 값을 저장

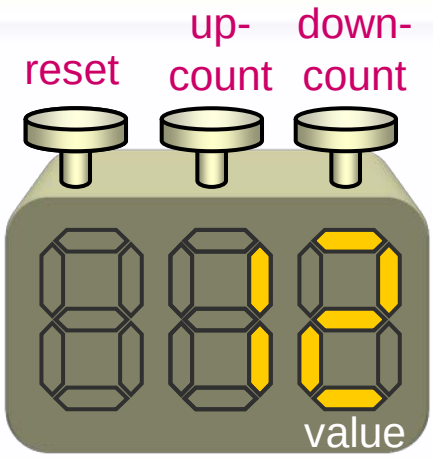
CounterM 클래스의 선언 - CounterM.h

```
1  class  CounterM {                  // 클래스 CounterM의 선언 시작
2      const int maxValue;            // 계수기의 최댓값
3      int  value;                    // private 데이터멤버
4  public:                            // public 멤버함수
5      CounterM(int mVal) : maxValue(mVal+1), value(0)
         { }                          // 생성자
6      void  reset()                  // 계수기의 값을 0으로 지움
         {  value = 0;  }
8      void  count()                  // 계수기의 값을 1 증가시킴
         {  value = (value + 1) % maxValue;  }
10     int  getValue() const          // 계수기의 현재 값을 반환함
11       {  return value;  }
12 };
```

상하향 계수기 클래스의 명세

UDCounterM 클래스

상하향 계수기를 나타내는 클래스를 선언하라. 상하향 계수기 객체는 값을 0으로 지울 수 있고, 값을 1씩 증가 또는 감소시킬 수 있으며, 현재의 계수기 값을 알려줄 수 있다. 계수기는 지정된 최댓값까지 계수할 수 있다.



행위

메소드	비 고
`CounterM(int mVal)`	생성자
`void reset()`	계수기의 값을 0으로 지움
`void countUp()`	계수기의 값을 1 증가시킴
`Void countDown()`	계수기의 값을 1 감소시킴
`int getValue()`	계수기의 현재 값을 알려 줌

속성

속성	비 고
`const int maxValue`	최대 계수 값
`int value`	계수기의 현재 값을 저장

디폴트 생성자 (default constructor)

- 매개변수가 없는 생성자, 또는 모든 매개변수에 디폴트 인수가 지정된 생성자
- 클래스를 선언할 때 생성자를 선언하지 않으면 컴파일러는 아무런 일도 하지 않는 디폴트 생성자를 만듦
- 생성자를 하나라도 선언하면 컴파일러는 디폴트 생성자를 자동으로 만들지 않음

Counter.h

```
class Counter {
  int Value;
public:

  void reset() { value = 0; }
  void count() { ++value; }
  int getValue() const
     { return value; }
};
```

```
int main()
{
  ......
  Counter cnt;
  ......
}
```

디폴트 생성자 (default constructor)

- 매개변수가 없는 생성자, 또는 모든 매개변수에 디폴트 인수가 지정된 생성자
- 클래스를 선언할 때 생성자를 선언하지 않으면 컴파일러는 아무런 일도 하지 않는 디폴트 생성자를 만듦
- 생성자를 하나라도 선언하면 컴파일러는 디폴트 생성자를 자동으로 만들지 않음

CounterM.h

```
class CounterM {
  const int maxValue;
  int Value;
public:
  CounterM(int mVal)
   : maxValue(mVal+1), value(0) {}
  void reset() { value = 0; }
  ......
};
```

```
int main()
{
  ......
  CounterM cnt1(999);
  CounterM cnt2; // 에러
  ......
}
```

복사 생성자 (copy constructor)

- 같은 클래스의 객체를 복사하여 객체를 만드는 생성자
- 복사 생성자를 명시적으로 선언하지 않으면 원본 객체의 데이터 멤버들을 그대로 복사하여 객체를 정의하는 복사 생성자가 자동적으로 선언됨

CounterM.h

```
class CounterM {
  const int maxValue;
  int value;
public:
  CounterM(int mVal)
   : maxValue(mVal+1), value(0) {}



  void reset() { value = 0; }
  ......
};
```

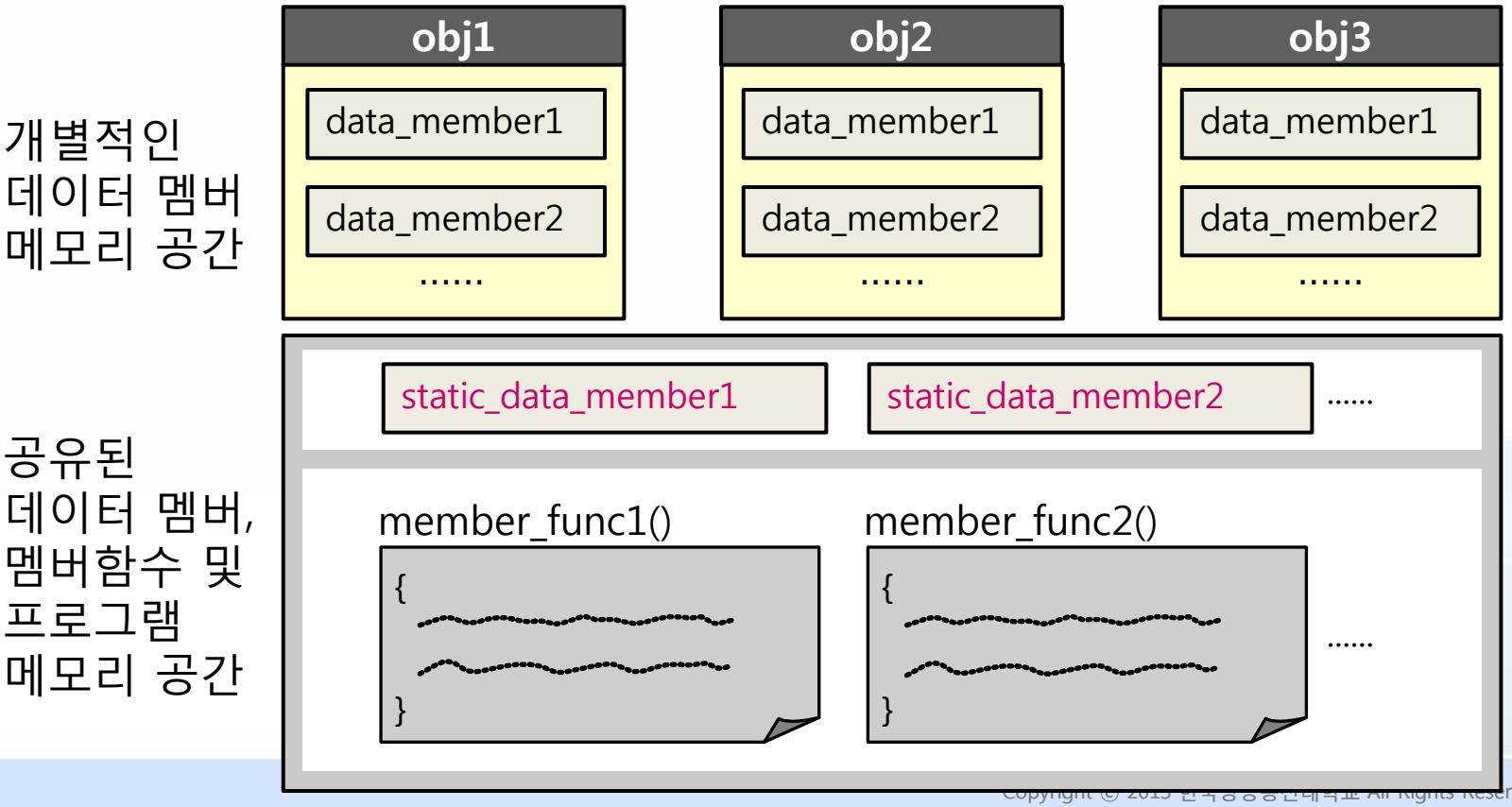
```
int main()
{
  ......
  CounterM cnt4(99);
  CounterM cnt5(cnt4);
  CounterM cnt6=cnt4;
  ......
}
```

학습활동 5 : static 멤버의 활용

이름(string으로 표현)과 일련번호를 저장하는 객체를 정의할 수 있는 클래스 S를 선언하라. 일련번호는 프로그램 시작 후 현재 시점까지 생성된 S 클래스의 객체 수이다.

static 멤버

- static 데이터 멤버
  - 클래스에 속하는 모든 객체들이 공유하는 데이터 멤버
  - 객체 생성과 관계 없이 프로그램이 시작되면 static 데이터 멤버를 위한 메모리 공간이 할당됨



static 멤버

static 데이터 멤버

- 클래스에 속하는 모든 객체들이 공유하는 데이터 멤버
- 객체 생성과 관계 없이 프로그램이 시작되면 static 데이터 멤버를 위한 메모리 공간이 할당됨
- 일반 데이터 멤버와는 달리, static 데이터 멤버는 클래스 외부에서 별도로 정의해야 한다.

static 멤버함수

- 특정 객체에 대한 처리를 하는 것이 아니라, 소속 클래스 전체를 대상으로 하는 작업을 수행하는 함수
- static 멤버함수는 객체가 정의되지 않아도 사용할 수 있음
- static 멤버함수 안에서는 일반 데이터 멤버를 액세스 할 수 없으며, static 데이터 멤버만 액세스 할 수 있음

제5장 연산자 다중정의

학습활동 6 : 연산자 다중정의

1개의 int형 데이터 멤버로 구성되며, 전위표기 및 후위표기 ++ 연산자와 += 연산자를 포함하는 클래스를 선언하라.

연산자 다중정의란?

- 연산자를 일종의 함수로 본다면, 피연산자는 함수에 전달되는 인수로 볼 수 있음
  - 함수 다중정의와 같이, 동일한 연산자라도 피연산자의 자료형에 따라 구체적인 처리는 다르게 정의됨

자료형과 연산자

10 + 20 → int형 + 연산자

10.0 + 20.0 → double형 + 연산자

- 연산자 다중정의
  - 함수 다중정의와 같이, 동일한 연산자라도 피연산자의 자료형에 따라 구체적인 처리는 다르게 정의됨

연산자 다중정의시 주의 사항

- 연산자의 의미를 임의로 바꾸지 않는다.
- 연산자의 고유한 특성이 유지되도록 한다.
  - 연산자의 우선순위를 바꿀 수 없다.
  - 피연산자의 수를 바꿀 수 없다.
  - 전위 표기와 후위 표기 연산자의 의미가 유지되도록 한다.
- 연산자 다중정의 위치
  - 클래스의 멤버로 정의
    - 연산자의 구현 과정에서 객체의 멤버를 액세스 할 수 있음
  - 클래스 외부에서 별도의 일반 함수와 같이 정의

단항 연산자

- 피연산자가 1개인 연산자
- 연산자가 피연산자의 앞에 위치하는가, 뒤에 위치하는가에 따라 전위 표기법과 후위 표기법으로 구분할 수 있음

수식 (a가 10일 때)	실행 결과	
	a의 값	b의 값
b = ++a;	11	11
b = a++;	11	10
b = --a;	9	9
b = a--;	9	10

전위 표기법

- 연산자가 피연산자의 앞에 위치하는 단항 연산자 표기법

전위 표기 연산자 다중정의 형식

```
ReturnClass ClassName::operator opSymbol()
{
    ......
}
```

- opSymbol : ++, -- 등의 단항 연산자 기호
- 형식 매개변수가 없음

후위 표기법

- 연산자가 피연산자의 뒤에 위치하는 단항 연산자 표기법

후위 표기 연산자 다중정의 형식

```
ReturnClass ClassName::operator opSymbol(int)
{
    ......
}
```

- opSymbol : ++, -- 등의 단항 연산자 기호
- 형식 매개변수 표기 위치의 'int'는 인수 전달의 의미가 아니라 단지 후위 표기법을 사용하는 단항 연산자임을 나타냄

이항 연산자 다중정의

- 이항 연산자 : 피연산자가 2개인 연산자

이항 연산자 다중정의 형식

```
ReturnClass ClassName::operator opSymbol(ArgClass arg)
{
    ......
}
```

- opSymbol : +, -, *, /, &&, || 등의 이항 연산자 기호
- 객체 자신이 좌측 피연산자, arg가 우측 피연산자에 해당됨

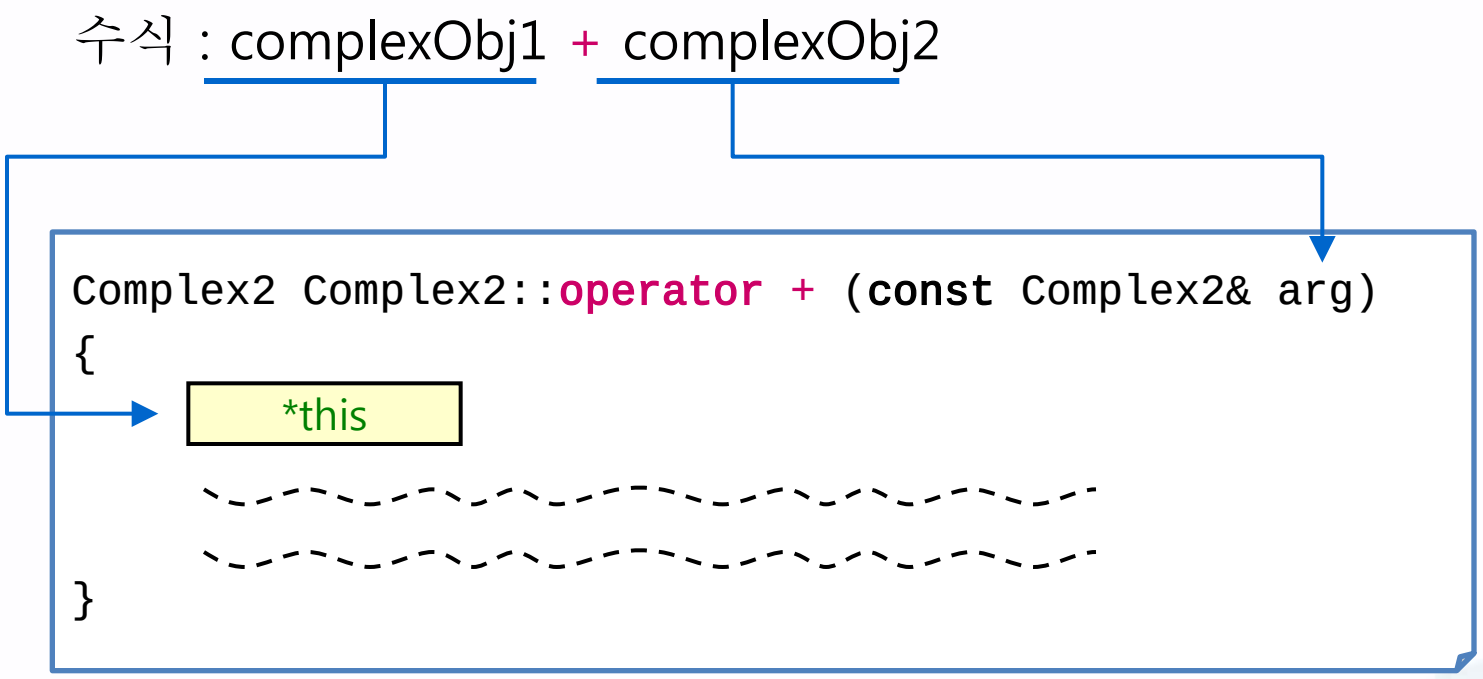
이항 연산자 다중정의

- 예 : 복소수를 표현하는 클래스 Complex2

```
1   #ifndef  COMPLEX2_H_INCLUDED
2   #define  COMPLEX2_H_INCLUDED
3   #include  <iostream>
4
5   class Complex2 {
6       double  rPart, iPart;          // 실수부 및 허수부
7   public:
8       // 생성자
9       Complex2(double r=0, double i=0) : rPart(r), iPart(i) {}
10      Complex2  conj()  const        // 공액 복소수
11          {  return Complex2(rPart, -iPart);  }
-       ......
    };
    #endif  // COMPLEX2_H_INCLUDED
```

이항 연산자 다중정의

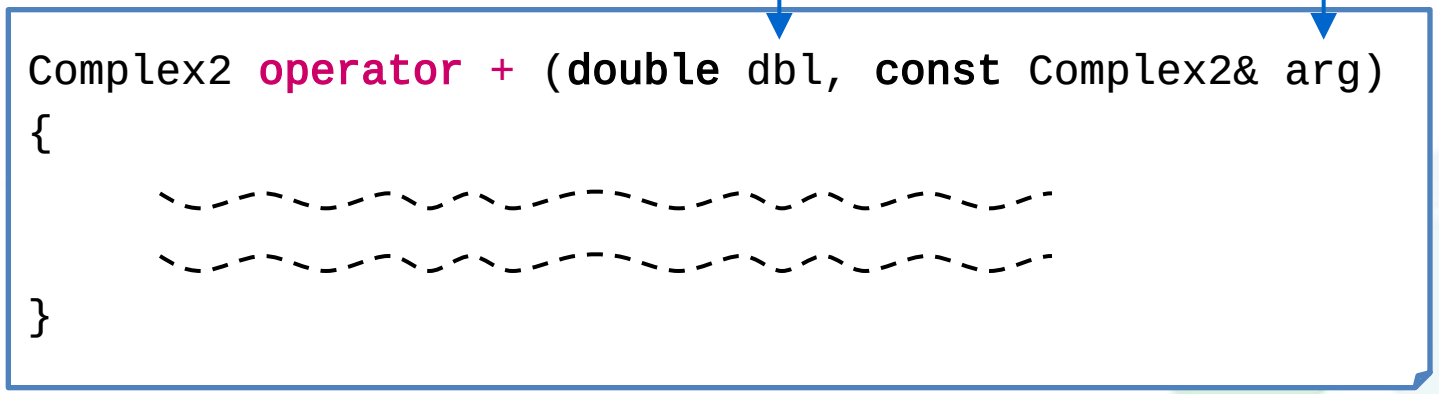
- 예 : 복소수를 표현하는 클래스 Complex2



이항 연산자 다중정의

- 실수와 복소수 객체의 덧셈 연산자
  - 좌측 피연산자가 실수이므로 Complex2 클래스의 멤버로 연산자를 정의할 수 없음
  - 클래스 외부의 별도 연산자로 정의함

수식 : dblVal + complexObj



```
Complex2 operator + (double dbl, const Complex2& arg)
{
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
}
```

이항 연산자 다중정의

- 실수와 복소수 객체의 덧셈 연산자
  - 좌측 피연산자가 실수이므로 Complex2 클래스의 멤버로 연산자를 정의할 수 없음
  - 클래스 외부의 별도 연산자로 정의함

```
1    Complex2  operator+(double r, const Complex2& c)
2    {
3        return Complex2(r+c.rPart, c.iPart);  // 오류! private 멤버 사용
4    }
```

이항 연산자 다중정의

- 실수와 복소수 객체의 덧셈 연산자
  - 좌측 피연산자가 실수이므로 Complex2 클래스의 멤버로 연산자를 정의할 수 없음
  - 클래스 외부의 별도 연산자로 정의함

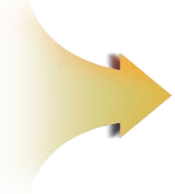
```
class Complex2 {
    ......
public:
    ......
    double getRPart() const { return rPart; }  // 실수부의 값 반환
    double getIPart() const { return iPart; }  // 허수부의 값 반환
};
```

```
1    Complex2  operator+(double r, const Complex2& c)
2    {
3        return Complex2(r+c.getRPart(), c.getIPart());
4    }
```

이항 연산자 다중정의

- 실수와 복소수 객체의 덧셈 연산자
  - 좌측 피연산자가 실수이므로 Complex2 클래스의 멤버로 연산자를 정의할 수 없음
  - 클래스 외부의 별도 연산자로 정의함

```
class Complex2 {
    ......
public:
    ......
    friend Complex2 operator+(double r, const Complex2& c);
};
```



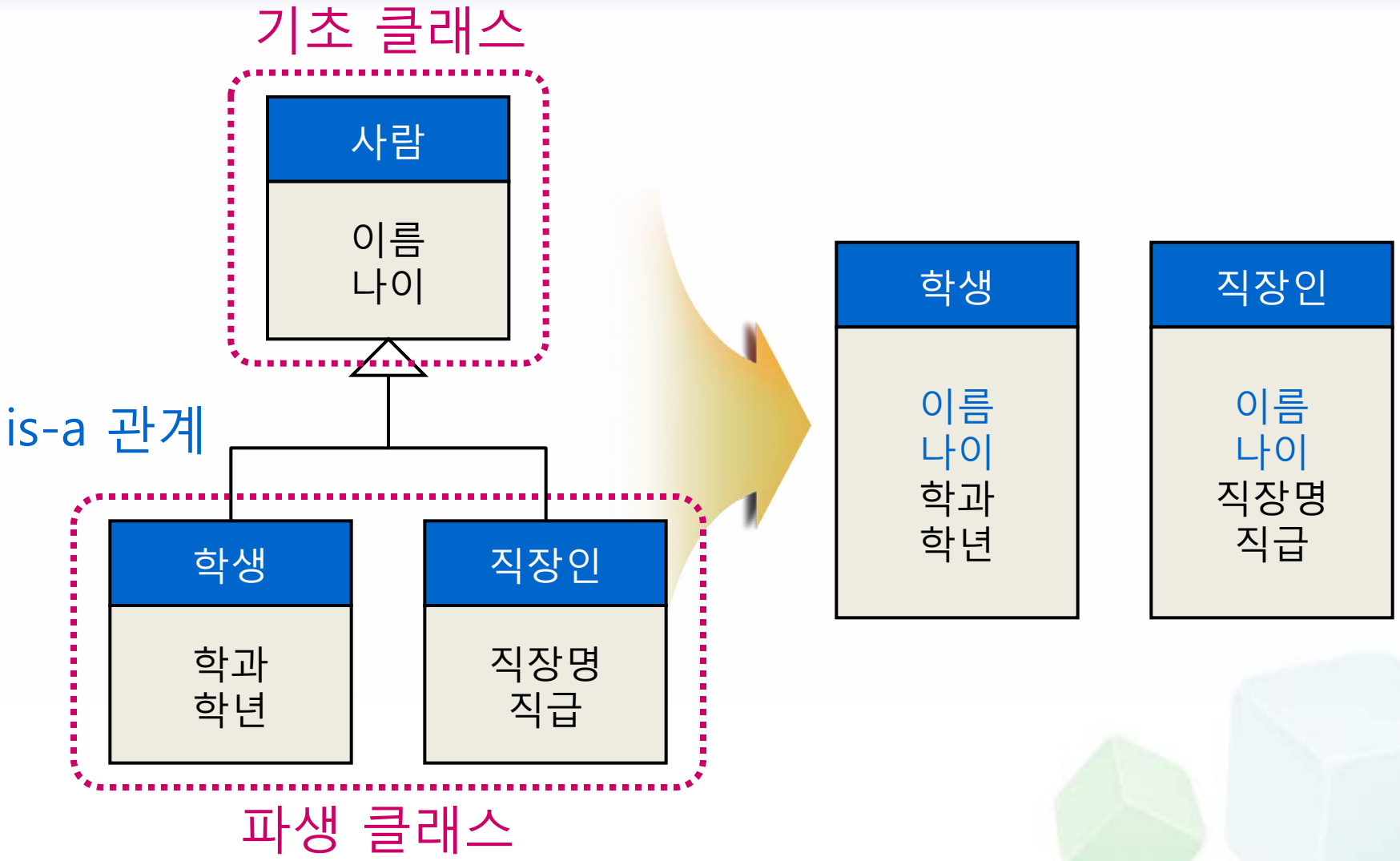
```
1    Complex2  operator+(double r, const Complex2& c)
2    {
3        return Complex2(r+c.rPart, c.iPart);
4    }
```

제6장 상속

학습활동 7 : 파생 클래스의 선언

교재 231쪽 [소스코드 6-9] Person 클래스의 파생 클래스인 Athlete을 선언하라. 운동선수를 나타내는 클래스인 Athlete에는 소속팀을 나타내는 데이터멤버 team을 포함하며, 팀 이름을 지정하거나 저장된 팀 이름을 알리는 기능, 콘솔에 "…팀 소속 선수 …입니다"라는 형식으로 출력하는 기능(print)이 포함된다.

클래스의 상속



파생 클래스의 선언

파생 클래스 선언 형식

```
class DClassName : visibilitySpec BClassName {
visibilitySpec_1:
    데이터 멤버 또는 멤버함수 리스트;
visibiliySpec_2:
    데이터 멤버 또는 멤버함수 리스트;
    ......
};
```

- *DClassName* : 파생 클래스 이름
- *BClassName* : 기초 클래스 이름
- *visibilitySpec* : 가시성 지시어
  - ☞**public, protected, private**

기초 클래스 Person - Person.h

```
1  #ifndef  PERSON_H_INCLUDED
2  #define  PERSON_H_INCLUDED
3  #include  <iostream>
4  #include  <string>
5  using  namespace  std;
6
7  class Person {
8      string  name;
9  public:
10     Person (const string& n) : name(n) {}
11     const string getName() const { return name; }
12     void print() const { cout << name; }
13 };
14
15 #endif  // PERSON_H_INCLUDED
```

학습활동 8 : 가상함수

학습활동7에서 선언한 Person 클래스의 포인터로 Person 객체와 Athlete 객체를 가리키게 한 후 print를 호출하여 본다. 그리고 Person 클래스를 수정하여 동일한 처리가 해당 클래스의 print를 호출하도록 한다.

클래스와 포인터

- 기초 클래스의 포인터는 파생 클래스 객체를 가리킬 수 있다.
- 파생 클래스의 포인터는 기초 클래스 객체를 가리킬 수 없다.

```
int main()
{
    Person   *pPrsn1, *pPrsn2;
    Person   who;
    Student  harry;
    Student  *pStdnt;
    pPrsn1 = &who;          // OK
    pPrsn2 = &harry;        // OK
    pStdnt = &who;          // Error!
    ......
    return 0;
}
```

정적 연결 (static binding)

- 객체에 대한 포인터(또는 참조)에 의해 멤버함수를 호출할 때 포인터(또는 참조)의 유형에 따라 호출되는 멤버함수가 결정됨

SBinding.cpp

```
1    #include  <iostream>
2    #include  "Person.h"
3    #include  "Student.h"
4    using  namespace  std;
5
6    int  main()
7    {
8      Person* p1 = new Person("Dudley");
9      p1->print(); cout << endl;       // Person::print() 호출
10     Person* p2 = new Student("Harry", "Hogwarts");
11     p2->print(); cout << endl;       // Person::print() 호출
12     ((Student *)p2)->print(); cout << endl; // Student::print() 호출
13     return 0;
14   }
```

출력 화면

```
Dudley
Harry
Harry goes to Hogwarts
```

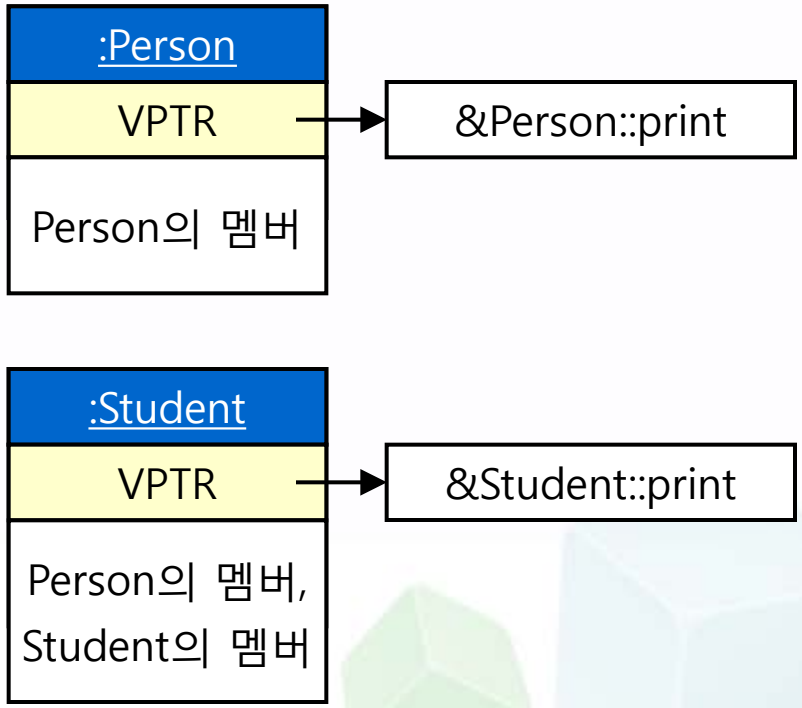
위험! – p2가 Student 객체를 가리키고 있으리라는 보장이 없음

동적 연결 (dynamic binding)

- 프로그램이 실행되는 과정에서 실제 객체에 따라 멤버함수를 결정하는 방법
- C++에서는 가상함수(virtual function)로 동적 연결을 구현함

```
class Person {
  ......
  virtual void print() const
    { cout << name; }
};
```

```
class Student : public Person {
  ......
  void print() const {
    Person::print();
    cout << " goes to "
         << school;
  }
};
```



동적 연결 (dynamic binding)

- 프로그램이 실행되는 과정에서 실제 객체에 따라 멤버함수를 결정하는 방법
- C++에서는 가상함수(virtual function)로 동적 연결을 구현함

```
1    #include  <iostream>
2    #include  "Person.h"
3    #include  "Student.h"
4    using  namespace  std;
5
6    int  main()
7    {
8      Person* p1 = new Person("Dudley");
9      p1->print(); cout << endl;       // Person::print() 호출
10     Person* p2 = new Student("Harry", "Hogwarts");
11     p2->print(); cout << endl;       // Student::print() 호출
12     return 0;
13   }
```

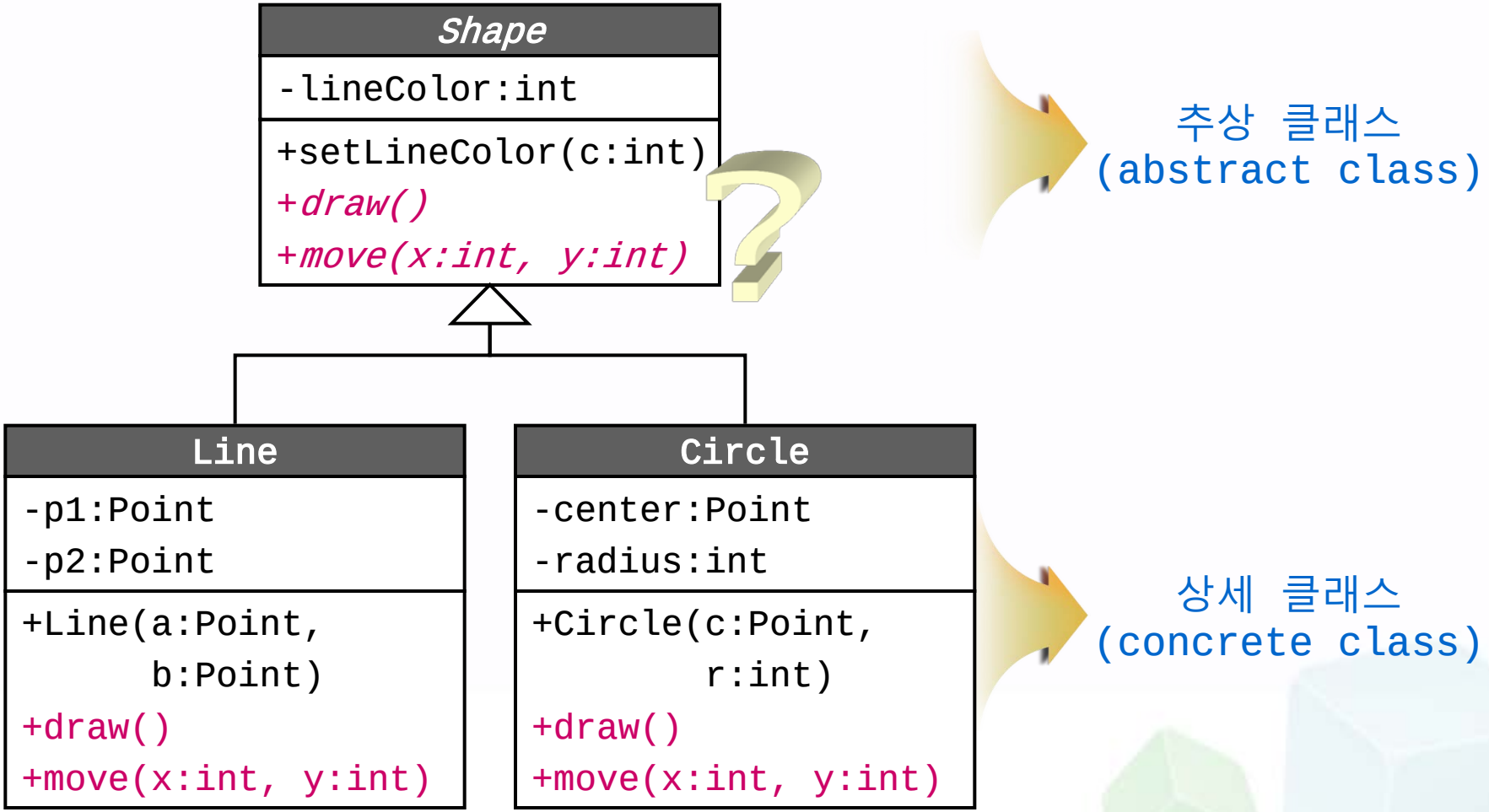
출력 화면

```
Dudley
Harry goes to Hogwarts
```

학습활동 9 : 추상 클래스

도형을 나타내는 추상 클래스 Shape을 선언하라. Shape의 멤버함수에는 면적을 구하는 area()와 도형의 내용을 콘솔에 출력하는 print()가 포함되며, 이들은 순수 가상함수이다. 그리고 이의 파생 클래스인 Circle과 Rectangle을 선언하라. 이때 Circle과 Rectangle이 상세 클래스가 되도록 하라.

추상 클래스와 상세 클래스



추상 클래스 (abstract class)

- 유사한 성격을 갖는 클래스들의 공통적 요소들을 뽑아서 만든 클래스로, 일부 메소드의 구체적 구현이 없어 직접적인 사례가 존재하지 않는 클래스
- 추상 클래스로 객체를 직접 정의할 수 없음
  - 추상 클래스는 그 자체로 사용되는 것이 아니라 파생 클래스를 통해 구현되어 사용됨
- 예) Shape 클래스 : draw, move 등의 메소드가 선언은 되어 있으나, 구체적으로 정의되지 않음
  - Shape 클래스의 객체를 정의할 수 없음
- 사용 목적 : 특정 그룹의 클래스들(추상 클래스의 파생 클래스들)이 반드시 가지고 있어야 할 행위를 지정함으로써, 필요한 행위를 정의하는 것을 누락하지 않도록 함

상세 클래스 (concrete class)

- 클래스의 모든 요소가 구체적으로 구현되어 직접적인 사례가 존재하는 클래스
- 상세 클래스는 직접 객체를 정의할 수 있음
- 예) Line, Circle 클래스 : 기초 클래스인 Shape에서 상속받은 메소드 중 Shape에서 완전히 구현되지 않은 draw, move 등이 구체적으로 정의되어 있음
  - Line이나 Circle 클래스의 객체를 정의할 수 있음

추상 클래스의 선언

- 멤버 함수 중 순수 가상함수를 포함하여 클래스를 선언함
  - 순수 가상함수 : 구현 부분이 없는 가상함수
  - 순수 가상함수의 선언

```
virtual RetType functionName(fParameterList) = 0;
```

  - 예

```
class Shape {
    int lineColor;
public:
    void setLineColor(int c) { lineColor = c; }
    virtual void draw() = 0;
    virtual void move(int x, int y) = 0;
};
```

상세 클래스의 선언

- 모든 멤버함수를 정의하여 순수 가상함수가 없도록 함
  - 기초 클래스로부터 상속된 순수 가상함수가 있다면 이들을 모두 재정의하여 구체적으로 구현해야 함
  - 예

```
class Circle : public Shape {
    Point center;
    int radius;
public:
    Circle(Point c, int r)
        : center(c), radius(r) {}
    void draw()
      { ..... } // 원을 그리는 문장들을 작성함
    void move(int x, int y) {
      center.x += x;  center.y += y;
    }
};
```

제7장 템플릿

학습활동 10 : 클래스 템플릿의 활용

지정된 자료형의 데이터를 지정된 개수만큼 저장할 수 있는 배열 형태의 컨테이너 클래스를 템플릿으로 선언하라. 이 클래스 템플릿에는 데이터를 액세스하기 위한 [] 연산자가 포함된다. 교재 160쪽의 Pencils 클래스의 객체를 5개 저장할 수 있는 컨테이너를 정의하여 사용하는 예를 프로그램으로 작성하라.

템플릿 (template)

- 클래스, 함수 등을 선언하기 위한 형판
- 자료형, 상수 등을 인수를 통해 전달받음

  ⇢ 학습에 사용되는 예의 집합은 다양하게 구성될 수 있음

- 여러 가지 대상을 위한 클래스나 함수를 템플릿으로 선언함으로써, 동일한 코드를 반복적으로 작성하는 것을 방지함

클래스 템플릿의 선언

클래스 템플릿의 선언 형식

```
template <templateArgs>
class ClassTemplateName {
        ......
};
```

- *templateArgs* : 템플릿 인수
- *ClassTemplateName* : 클래스 템플릿 이름

컨테이너 클래스(container class)

- int형 데이터를 저장하는 스택을 표현하는 컨테이너 클래스

```
1  #ifndef  STACK1_H_INCLUDED
2  #define  STACK1_H_INCLUDED
3  typedef int STACK_ITEM;
4  class Stack {
5      enum { MAXSTACK=20 };
6      int  top;
7      STACK_ITEM item[MAXSTACK];
8  public:
9      Stack();     // constructor
10     bool empty();
11     void initialize();
12     void push(STACK_ITEM s);
13     STACK_ITEM pop();
14 };
15 #endif  // STACK1_H_INCLUDED
```

예 : 클래스 템플릿 Stack의 선언 - Stack2.h

```
1  #ifndef  STACK2_H_INCLUDED
2  #define  STACK2_H_INCLUDED
3  #include  <iostream>
4  using  namespace  std;
5
6  template  <class T>
7  class  Stack {
8      T* buf;      // buffer pointer
9      int top;     // stack top
10     int size;    // 스택의 크기
11 public:
12     Stack(int s) : size(s), top(s)    // 생성자
13         {  buf = new T[s];  }
14     virtual ~Stack()                  // 소멸자
15         {  delete [] buf;  }
16     void push(T a);
17     T pop();
18 };
   ......
```

클래스 템플릿의 멤버함수 선언

클래스 템플릿의 멤버함수 선언 형식

```
template <templateArgs>
ReturnType ClassTemplateName<args>::
                        funcName(fParameterList)
{
        ......
}
```

- *ReturnType* : 멤버함수의 반환 자료형
- *funcName* : 멤버함수의 이름
- *args* : *templateArgs*로 전달된 인수
- *fParameterList* : 멤버함수의 형식 매개변수 목록

예 : 클래스 템플릿 Stack의 선언 - Stack2.h

```
      ......
20    template <class T> void Stack<T>::push(T a)
21    {
22        if (top)
23            buf[--top] = a;
24        else
25            cout << "Erorr -- stack full\n";
26    }
27
28    template <class T> T Stack<T>::pop()
29    {
30        if (top < size)
31            return buf[top++];
32        else {
33            cout << "Error -- stack empty\n";
34            return NULL;
35        }
36    }
37
38    #endif  // STACK2_H_INCLUDED
```

클래스 템플릿의 객체 정의

클래스 템플릿의 객체 정의 형식

```
ClassTemplateName<ClassNames>
                        objName(constrArgs);
```

- *ClassNames* : 템플릿 인수에 전달할 클래스 이름
- *objName* : 정의할 객체의 이름
- *constrArgs* : 생성자에 전달할 인수

예 : 스택 클래스 템플릿의 객체 사용 - StackTMain.cpp

```
 1  #include  <iostream
 2  #include  "Stack2.h
 3  using  namespace  s
 4
 5  int main()
 6  {
 7      Stack<char> sc(
 8      Stack<int> si(5
 9
10      sc.push('a');
11      sc.push('b');
12      cout << "CHAR S
13      cout << sc.pop(
14      si.push(5);
15      si.push(10);
16      cout << "INT STACK: " << si.pop() << " ";
17      cout << si.pop() << endl;
18      return 0;
19  }
```

```
template  <class T>
class  Stack {
    T* buf;      // buffer pointer
    int top;     // stack top
    int size;    // 스택의 크기
public:
    Stack(int s) : size(s), top(s)
        { buf = new T[s]; }
    virtual ~Stack()
        {  delete [] buf;  }
    void push(T a);
    T pop();
};
```

C++ 프로그래밍

수고 많이 하셨습니다.

한국방송통신대학교
컴퓨터과학과 이병래 교수