객체지향형 프로그래밍과 자료구조

Ch 5. 상속과 소프트웨어 재사용

정보통신공학과

교수 김 영 탁

(Tel : +82-53-810-2497; Fax : +82-53-810-4742
http://antl.yu.ac.kr/; E-mail : ytkim@yu.ac.kr)

Outline

- ◆ 상속 (inheritance) 개요
  - 상속 (inheritance) 기본 용어:
    - base class, parent class, ancestor classes
    - derived class, child class, descendent classes
- ◆상속된 클래스의 멤버함수 구현
  - Derived classes with constructors
  - access qualifier “protected”
  - Redefining member functions
  - Non-inherited functions
  - Assignment operators (“=”) and copy constructors
  - Destructors in derived classes
- ◆ 동적 메모리 할당 기능을 포함하는 상속클래스의 생성자 및 소멸자 설계
  - class DynArray, class Stack, class Queue

상속 (inheritance) 개요

상속 (inheritance) 개요

◆객체 지향형 프로그래밍 (Object-oriented programming)

- Powerful programming technique
- 추상화 개념과 함께 상속(***inheritance***) 기능을 제공
- 소프트웨어 재사용 (**software-reuse**) 과 다형성 (***polymorphism***) 기능을 제공

◆일반화 (generalized, generic) 클래스와 특화 (specialized)된 클래스

- 먼저 일반화된 클래스를 설계한 후, 이를 기반으로 필요에 따라 특화 (specialized)된 클래스를 추가 생성함으로써 소프트웨어 재사용
- 특화된 파생 클래스는 일반화된 기반 클래스의 속성을 상속 받은 후, 특화된 속성을 추가

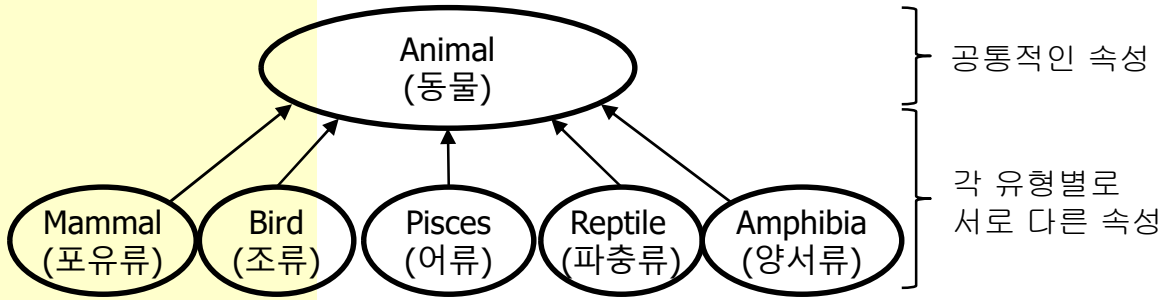
상속이란

◆ "상속(Inheritance)"

- 이미 개발되어 사용되는 클래스에 새로운 멤버 (속성)을 추가하여 새로운 클래스를 생성하게 함
- 새로운 클래스는 기존 클래스의 기능을 다시 구현할 수 있게 하거나, 확장할 수 있게 함
- 상속받은 클래스는 기존 클래스의 특별한 유형으로 볼 수 있으며, 상속받는 자식 클래스는 상속을 해 주는 부모클래스와 "is a"관계가 성립됨

Mammal (e.g., dog, cat) is an animal.
Bird is an animal.
Fish is an animal.

Mammal has 4 **legs**.
Bird has 2 **wings**.
Fish has **fins**.
But, Mammal, Bird and Fish have spine, eye, and blood *in common*.



상속 관련 용어

- **부모 클래스 (parent class), 기반클래스, 수퍼클래스**
  - 상속을 해 주는 기존 기반 클래스 (base class)
  - Super class라고도 함
- **자식 클래스 (child class), 파생클래스, 서브클래스**
  - 부모 클래스의 속성을 상속받고, 추가 속성을 정의하는 새로운 파생 클래스 (derived class)
  - Sub-class라고도 함
- **조상 클래스들 (ancestor classes)**
  - 부모 클래스, 부모의 부모인 조부모 및 상속 관계에 있는 직계 선대 클래스들
- **후손 클래스들 (descendant classes)**
  - 자식 클래스, 손주 클래스 및 상속 관계에 있는 직계 후대(후손) 클래스들

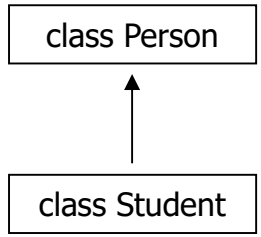
상속 구조의 클래스 예

- **기반 클래스 (base class)**
  - 공통적으로 사용될 수 있는 속성을 가지는 일반화된 클래스 (parent class, superclass)
  - (예) 학생과 직장인 등 모든 사람에게 공통적인 속성을 가지는 class Person
- **파생 클래스 (derived class)**
  - 기반 클래스의 속성을 상속받고, 특화된 속성을 추가한 클래스 (child class, subclass)
  - (예) class Person으로 부터 상속받는 class Student, class Staff

```
class Person // Existing base class
{
public:
   string getName() {return name;}
private:
   string name;
};

class Student : Person  // Derived class
{
public:
    int getSt_ID() {return student_id;}
private:
    int student_id;
};
```

클래스간의 상속을 나타내는 표시



"*Is a*" vs. "*Has a*" 관계

◆상속관계: "is a" 관계

- 자식 클래스는 부모 클래스와 "is a" 관계를 가짐
- 예)
  고양이는 동물이다.
  독수리는 동물이다.

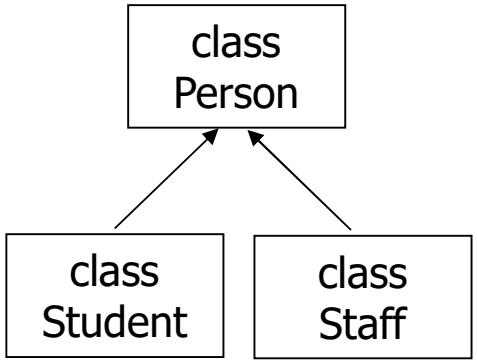
◆포함관계: "has a" 관계

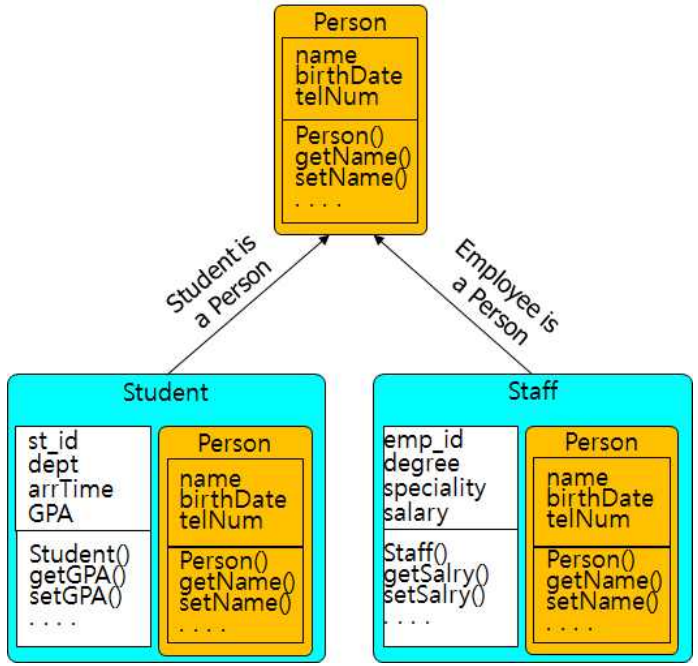
- 하나의 클래스가 다른 클래스를 속성 (데이터 멤버)으로 가지는 경우
- 예)
  동물은 눈(eye)과 척추(spine)을 가진다.
  class Person은 데이터 멤버로 class Date을 가진다.

상속관계에서의 속성들의 포함관계

◆ 상속관계에서 자식클래스와 부모클래스의 속성 포함 관계

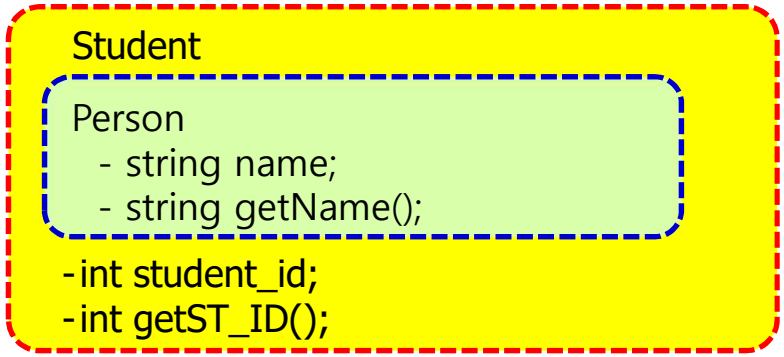
- 자식클래스는 부모클래스의 모든 속성을 모두 포함하게 됨
- 부모클래스는 보다 일반적인 (공통적으로 사용할 수 있는) 속성을 가짐
- 자식 클래스는 보다 일반적인 속성과 함께 특화된 속성을 추가로 가짐





상속관계에서의 멤버들의 포함관계 (2)

```
class Person
{
public:
    string getName() {return name;}
private:
    string name;
};
class Student : Person
{
public:
    int getST_ID(){return student_id;}
private:
    int student_id;
};
```



- **Objects of Person (Parent) have members**

```
string name;
string getName()
  {return name;}
```

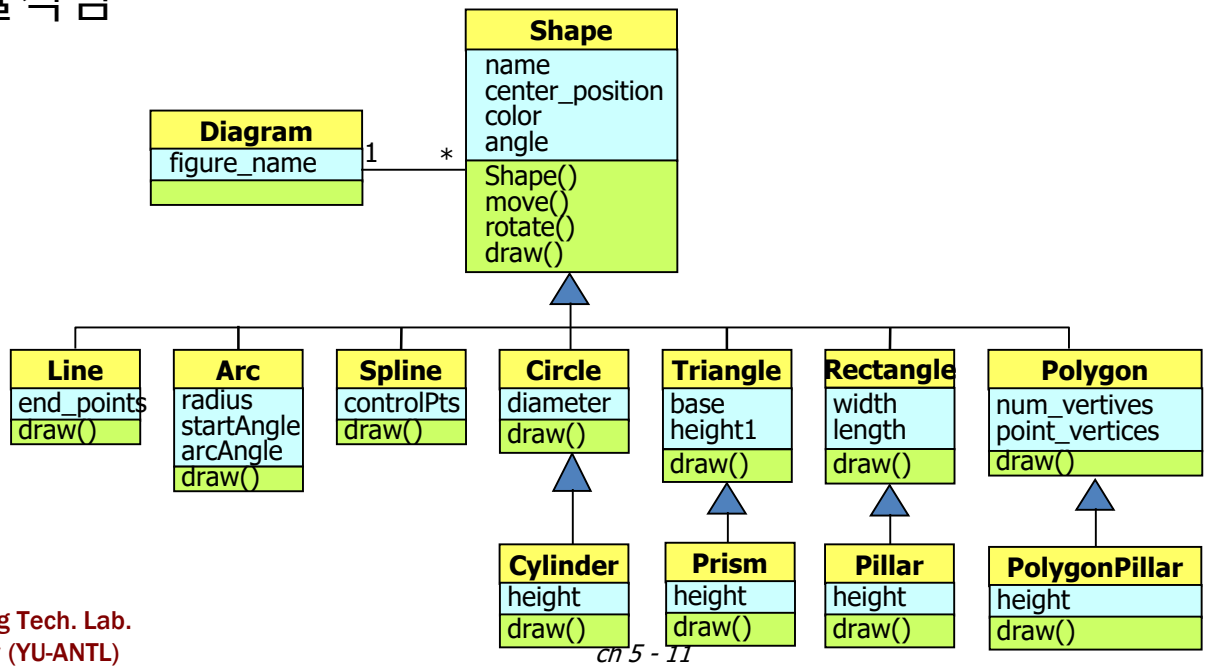
- **Objects of Student (Child) have members**

```
string name;
string getName()
  {return name;}
int student_id;
int getST_ID()
  {return student_id;}
```

상속과 소프트웨어 재사용

◆ 상속은 소프트웨어 재사용 기능 제공

- 상속 기반의 소프트웨어 개발에서는 자식 (파생) 클래스는 이미 기능과 성능이 잘 검증되고 신뢰성이 높은 기존의 부모 클래스를 기반으로 새로운 클래스를 생성하게 하며, 기존 부모 클래스의 소프트웨어를 재사용할 수 있게 함
- 다양하게 파생된 클래스 사용 중에 공통적인 속성의 수정 또는 보완이 필요한 경우, 파생클래스가 상속받은 부모 클래스를 수정 및 보완함으로써 쉽게 정비 (maintenance)할 수 있으며, 각 자식 클래스에서 공통적인 속성을 개별적으로 변경하는 것 보다 효율적임



상속을 기반으로 한 객체 지향형 프로그래밍의 예

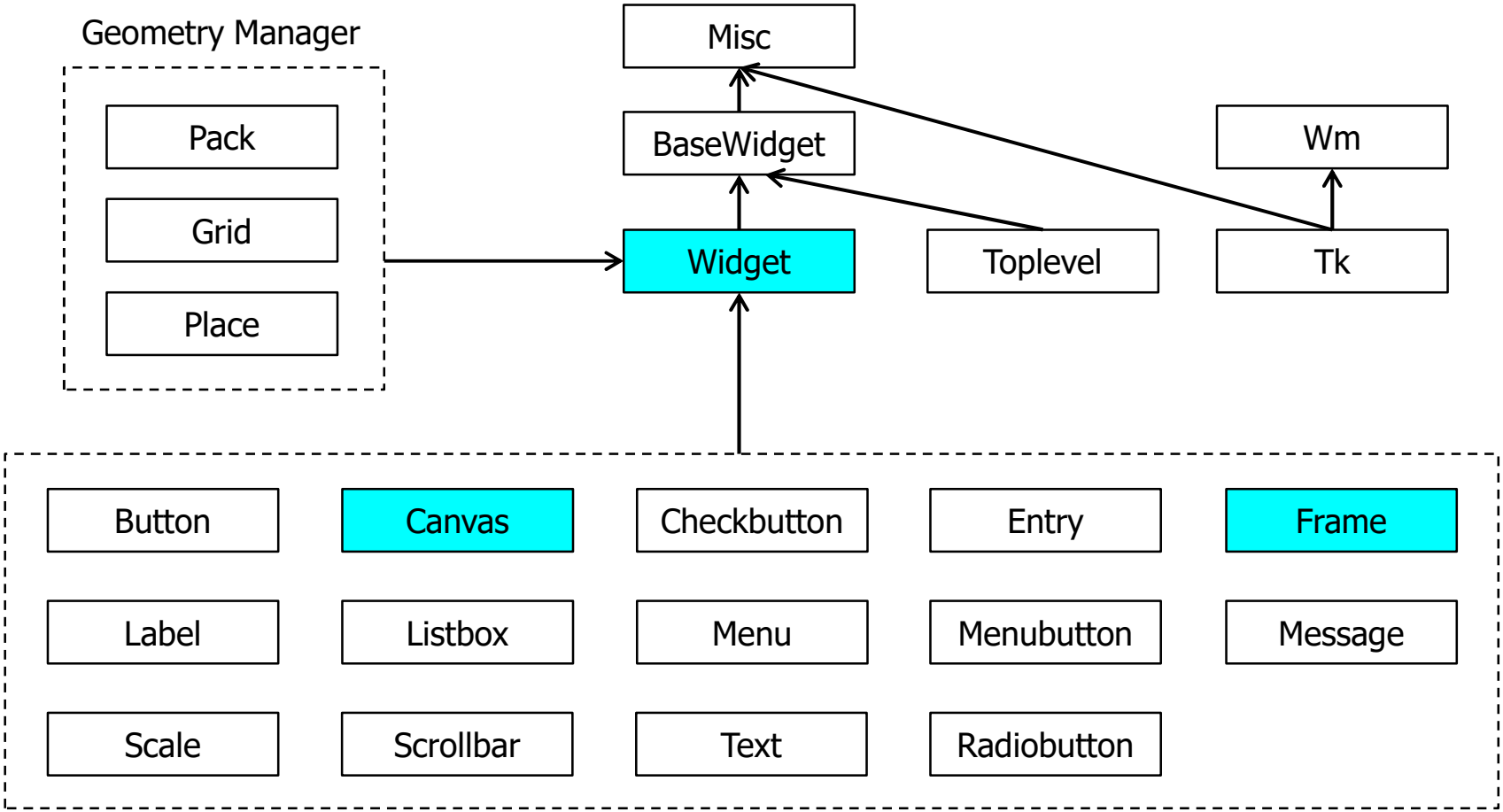
◆ 상속을 고려한 객체 지향형 클래스 설계

- 하나의 부모/조상 클래스는 다양한 자식/자손 클래스에서 재활용
- 부모/조상 클래스를 일반화 (generalized) 소프트웨어로 설계하고, 자식/후손 클래스를 특성화 (specialized) 소프트웨어로 설계

◆ 상속을 기반으로 한 객체 지향형 클래스의 사용 예

- Windows, Java, Python에서의 GUI (graphic user interface)의 클래스들은 모두 상속 개념의 객체 지향형 클래스
- Widget 클래스를 상속받아 Frame, Button, Label, Menu, Text 등의 파생 클래스 생성

class Widget

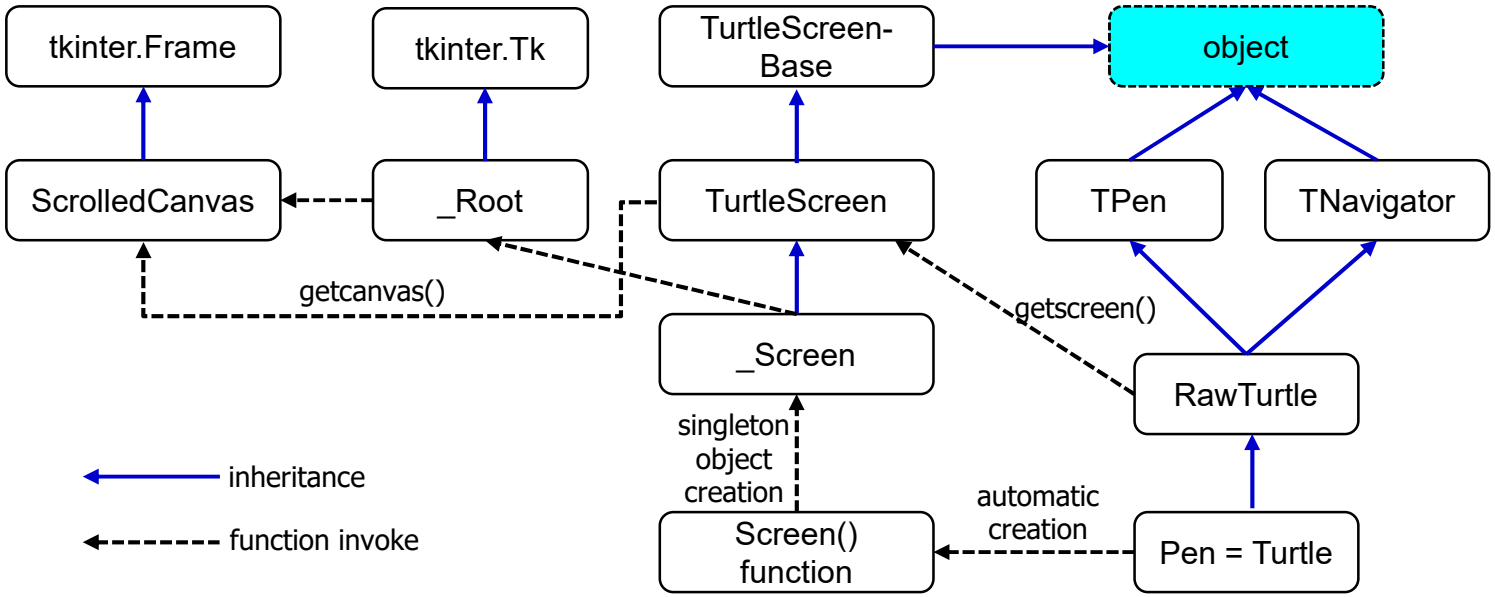


Python Turtle Graphic

◆ turtle graphic in Python

- turtle module supports very simple 2D graphic
- implemented using tkinter package
- document on turtle graphic: https://docs.python.org/3.3/library/turtle.html?highlight=turtle

◆ Hierarchy of classes in turtle module



상속 기반의 C++ 클래스 설계 및 구현

Base Class Person

◆**class Student와 class Staff의 기반 클래스인 class Person은 파생 클래스들이 공통적으로 사용되는 일반화된 속성을 포함**

- 데이터 멤버:
  - string name;
  - Date birthdate;
- 멤버함수:
  - 생성자: Person(string nm, Date dob);
  - 접근자: string getName();
  - 변경자: void setName(string nm);
  - 소멸자: ~Person();

class Person

```
/* Person.h */

#include <iostream>
#include <string>
#include "Date.h"
#include "TelNum.h"
class Person
{
    friend ostream &operator<<(ostream &output, Person &p);
public:
    Person() {}
    Person(string n, Date dob, TelNum tn) { name = n; birthDate = dob; telNum = tn; }
    void setName(string n) { name = n; }
    string getName() const { return name; }
    void setBirthDate(Date bd) { birthDate = bd; }
    Date getBirthDate() const { return birthDate; }
    void setTelNum(TelNum tn) { telNum = tn; }
    TelNum getTelNum() { return telNum; }
protected:
    string name;
    Date birthDate;
    TelNum telNum;
};
```

상속 받는 클래스의 멤버함수 구현

◆ 파생클래스와 기반클래스의 멤버함수 구현

- 파생클래스 (class Student)는 기반 클래스 (class Person)로 속성들(데이터 멤버, 멤버 함수)을 상속 (*inherit)을 받음*
- 기반 클래스의 생성자는 상속되지 않음
- 기반 클래스의 생성자가 기반 클래스의 데이터 멤버들을 초기화 시키도록 구성하여야 함
- 파생 클래스는 기반 클래스로부터 상속 받은 멤버 함수를 재지정 (redefine) 할 수 있음

Class Student

```
/* Student.h */

#include <iostream>
#include "Person.h"
#include "Date.h"
using namespace std;

class Student : public Person
{
friend ostream & operator<< (ostream &, const Student &);
public:
    Student() {}  // default constructor
    Student(int id, string n, Date dob, TelNum telNum, double gpa);
    int getST_id() const { return st_id; }
    void setST_id(int id) { st_id = id; }
    void setDept_name(string dp_n) { dept_name = dp_n; };
    string getDept_name() { return dept_name; }
    double getGPA() const { return gpa; }
    void setGPA(double g) { gpa = g; }
    const Student& operator=(const Student& right);
    bool operator>(const Student& right);
    bool operator==(const Student& right);
private:
    int st_id;
    string dept_name;
    double gpa;
};
```

Class Staff

```
/* Staff.h */

#include "Person.h"
enum DEGREE {HighSchool, Bachelor, Master, PhD};
class Staff : public Person
{
    friend ostream & operator<< (ostream &, const Staff &);
public:
    Staff() {}  // default constructor
    Staff(int id, string nm, Date dob, TelNum telNum, DEGREE dg, string sp, double salary);
    int getID() const { return stf_id; }
    void setID(int id) { stf_id = id; }
    void setDegree(DEGREE dg_n) { degree = dg_n; };
    DEGREE getDegree() { return degree; }
    void setSpeciality(string sp) { speciality = sp; }
    string getSpeciality() const { return speciality; }
    double getSalary() const { return salary; }
    void setSalary(double sl) { salary = sl; }
    const Staff& operator=(const Staff& right);
    bool operator>(const Staff& right);
    bool operator==(const Staff& right);
private:
    int stf_id; // staff ID
    DEGREE degree;
    string speciality;
    double salary;
};
```

파생클래스 (Derived Class)의 생성자

◆ 별도의 인수가 지정되지 않는 Default 생성자의 호출

- Student(); // default constructor
- 생성자가 호출될 때, 인수 (argument)가 설정되어 있지 않으면 default constructor가 호출되며, Base class의 default constructor도 호출됨

◆ 객체가 생성될 때, 인수가 지정되면, 해당 생성자가 호출되며, 해당 base class의 생성자도 호출

- 다양한 종류의 생성자가 사용될 수 있음
- Student(id, nm, dob, tn, gpa) ➔ Person(n, dob, tn) 호출
- Staff(id, nm, dob, tn, dg, sp, slry) ➔ Person(nm, dob, tn) 호출

Base Class의 Private Data 접근

- **파생(자식)클래스는 기반(부모) 클래스의 속성을 모두 상속받음**
  - 기반(부모) 클래스의 private member도 모두 상속받으나, 부모클래스의 private member를 이름으로 직접 접근할 수는 없음
  - 부모클래스의 private member는 부모클래스의 public interface 멤버 함수를 통해서만 접근이 가능함
- **상속관계에 있는 파생(자식) 클래스 구현에서 기본(부모) 클래스의 속성을 간편하게 접근하는 방법?**
  - "protected" 접근 지정자를 사용하여, 상속관계에 있는 파생(자식) 클래스 구현에서 기본(부모) 클래스의 속성을 간편하게 접근할 수 있도록 제한적으로 허용

protected 접근 지정자

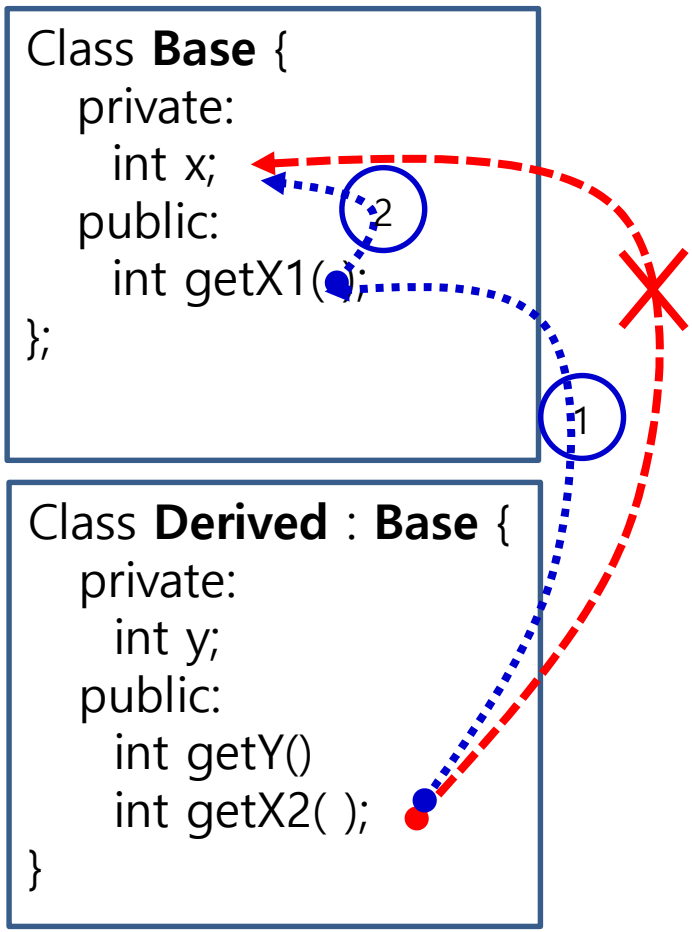
◆protected 접근 지정자

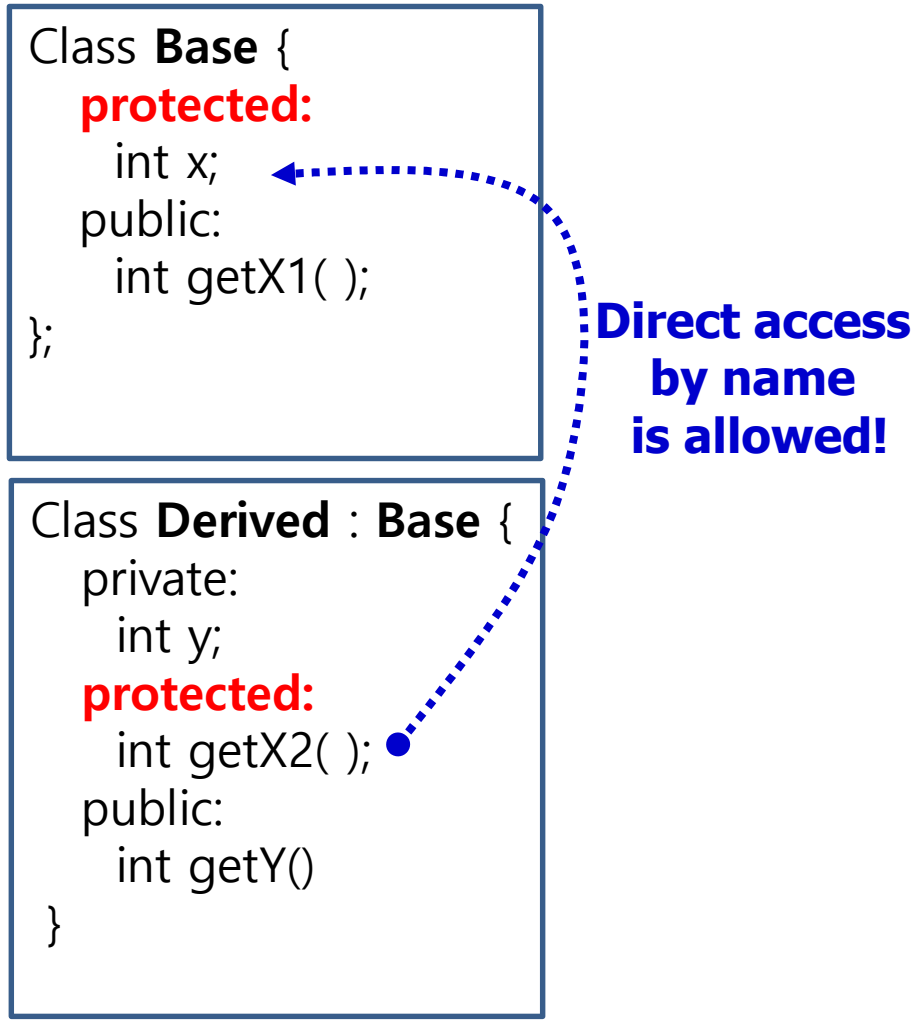
- 상속 관계에 있는 파생(자식) 클래스에서 기반(부모) 클래스의 속성을 이름으로 직접 접근할 수 있도록 제한적으로 허용
- 상속관계가 아닌 다른 클래스에서는 private 처럼 관리되며, 이름으로 직접 접근할 수 없도록 보호됨

◆protected 접근 지정자가 객체지향형 프로그래밍의 정보 보호 원칙을 위배?

- 파생(자식) 클래스에서 기반(부모) 클래스의 속성을 이름으로 직접 접근할 수 있도록 제한적으로 허용하는 것이 객체지향형 프로그래밍의 정보 보호 원칙을 위배한다는 의견도 있음
- 구현에서의 간편성/편리성을 제공하기 위하여 파생(자식) 클래스에게만 제한적으로 허용

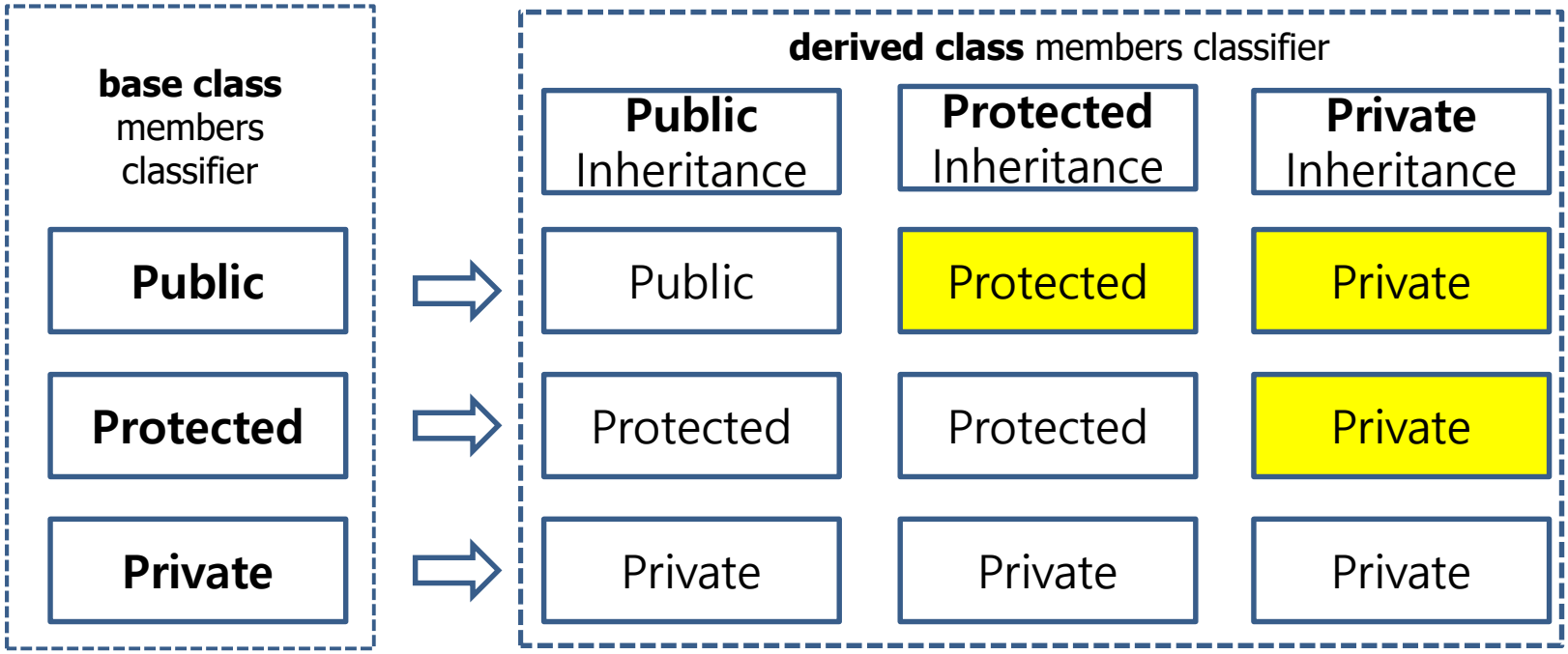
◆ Private vs. Protected





◆ 상속 모드 (inheritance mode)

- 3가지의 상속모드 설정: public, protected, private
- public 상속: base class에서 설정된 접근 지정자 (classifier)에 변경없음
- protected 상속: base class에서 설정된 public 접근지정자가 protected 로 변경됨
- private 상속: base class에서 설정된 public 및 protected 접근지정자가 private 로 변경됨



Protected 상속과 Private 상속

◆ Public 상속에 추가된 protected 상속 및 private 상속

- protected 상속과 private 상속은 거의 사용되지 않음

◆ Protected 상속:

class Student : protected Person
{…}

- 부모(기반)클래스의 public 속성이 상속된 클래스에서는 protected 접근 지정으로 변경되어, 상속관계에서의 자식(파생) 클래스들만 이름으로 직접 접근 가능하게 됨

◆ Private 상속:

class Student : private Person
{…}

- 부모(기반)클래스의 public, protected 속성이 상속된 클래스에서는 private 접근 지정으로 변경되어, 다른 클래스에서는 더 이상 이름으로 직접 접근이 불가능하게 됨

파생 클래스의 생성자 예

◆ class Student의 생성자:

```
Student::Student(int id, string n, Date dob, TelNum tn, double new_gpa)
  : Person(n, dob, tn)
{
   st_id = id;
   gpa = new_gpa;
}
```

◆ 초기화 섹션 (initialization section)

- 클래스 생성자에서 생성자 함수 이름 다음에 ':' 문자로 시작되며, 생성자 함수 구역({, }) 앞에 선언
- 클래스의 데이터 멤버 값을 인수로 전달된 값으로 초기화
- 부모(기반) 클래스의 생성자를 호출하여 부모(기반) 클래스가 담당하는 데이터 멤버의 초기화를 요청

파생클래스에서 기반 클래스 멤버함수의 재 지정 (redefinition)

◆ 파생클래스의 멤버함수

- 기반클래스의 멤버함수는 자동적으로 포함됨
- 특화된 멤버함수 추가
- 기반클래스의 멤버함수를 재지정할 수 있음

재지정 (Redefinition) vs. 오버로딩 (Overloading)

◆ 멤버함수의 재지정과 함수 오버로딩은 서로 다름

◆ 재지정 (Redefinition): 상속관계에서 부모클래스 멤버함수를 자식클래스에서 재지정

- 동일한 함수이름과 동일한 파라메터 (매개변수) 목록 사용
- 부모클래스의 멤버함수를 재 작성하는 것임

◆ 오버로딩 (Overloading):

- 동일한 함수이름이나 파라메터 (매개변수) 목록이 서로 다름
- 함수의 signature가 다르며, 서로 다른 함수로 관리됨

재지정된 멤버함수에서 기본 클래스의 멤버함수 호출

◆재지정된 멤버함수

- 기반클래스의 멤버함수를 파생클래스에서 재지정한 경우에도 기반클래스의 멤버함수 자체가 변경되지는 않음

◆상속관계에서 멤버함수가 재지정된 경우

```
Shape shp;  // base class
Triangle tri;  // class Triangle : class Shape
Prism prsm; // class Prism : class Triangle

shp.draw()  → calls Shape's draw()
tri.draw()  → calls Triangle's draw()
prsm.draw() → calls Prism's draw()
```

상속되지 않는 함수들

◆기반클래스의 멤버함수 중 아래의 특별한 경우를 제외한 멤버함수들을 상속됨

◆상속되지 않는 멤버함수들:

- 생성자 (constructors)
- 소멸자 (destructors)
- 복제 생성자 (copy constructor)
  - 복제 생성자가 구현되지 않으면 default 복제함수가 사용됨
  - 클래스에서 포인터와 동적메모리 생성 기능이 사용되면 복제 생성자를 구현하여야 함
- 대입 연산자 (assignment operator, '=')
  - 대입연산자가 구현되지 않으면 default 대입 연산 기능이 사용됨

대입연산자('=')와
복제 생성자 (copy constructor)

대입연산자 ("=") 와 복제 생성자

◆ 연산자 오버로딩으로 구현된 대입연산자와 복제 생성자 (copy constructor)

- 연산자 오버로딩으로 구현된 대입연산자와 복제 생성자는 상속되지 않음
- 기반 클래스의 대입연산자와 복제 생성자를 파생 클래스에서 사용할 수 있음
  - 파생클래스의 대입연산자 구현에서 기반클래스의 대입연산자를 사용
  - 파생클래스의 복제생성자 구현에서 기반클래스의 복제생성자를 사용

대입연산자 ("=") 구현 예

◆기반 클래스의 대입연산자를 파생 클래스 대입 연산자 오버로딩 구현에서 사용하는 예

```
Derived& Derived::operator=(const Derived & rightSide)
{
   Base::operator=(rightSide);
   ...
}
```

```
Shape& Shape::operator=(const Shape&  right)
{
    pos = right.pos; // position (x, y)
    angle = right.angle;
    name = right.name;
    color = right.color;

    return *this;
}
```

```
Triangle& Triangle::operator=(const Triangle& right)
{
    Shape::operator=(right);
    base = right.base;
    tri_height = right.tri_height;

    return *this;
}
```

복제 생성자 (Copy Constructor) 구현 예

◆복제 생성자 (Copy Constructor)

- 생성자 중에서 전달되는 인수가 그 클래스의 객체인 경우
- 전달된 객체을 복사하여 새로운 객체를 생성

◆상속된 파생 클래스의 복제 생성자에서 부모 클래스의 생성자를 호출

- 파생클래스 객체에는 기반 클래스의 데이터 멤버를 함께 포함하고 있음
- 파생클래스의 초기화 색션에서 복제 대상 객체를 기반 클래스의 생성자에 전달하여 복제를 수행

◆예)

```
Triangle::Triangle(const Triangle &tr)   // copy constructor
:Shape(tr), base(tr.base), tri_height(tr.tri_height)
{
    cout << "  Copy constructor (" << name << ").\n";
}
```

복제 생성자 (Copy Constructor) 실행

- **다음의 경우에서는 복제생성자가 실행:**
  - 클래스의 객체가 생성되어 다른 객체로 복사되는 경우
  - 함수의 실행 결과로 클래스 객체를 반환하는 경우
  - 클래스의 객체가 함수 호출의 call-by-value 인수로 전달되는 경우
- **객체의 임시 복사본 (temporary copy)이 필요할 때**
  - 복제 생성자가 복사본을 생성
- **Default 복제 생성자**
  - 별도의 복제 생성자를 구현하지 않았을 때 실행
  - 기본 대입연산자 (=)와 동일하게 member-wise 복사
  - 동적메모리 할당 기능은 제공되지 않음
- **클래스에서 포인터와 동적메모리할당이 사용되는 경우, 복제 생성자를 구현하여야 함**

```
class Position
{
public:
    Position();
    Position(string nm);
    Position(string nm, int x, int y);
    ~Position();
    void setName(string nm) {name = nm;}
    Position& operator=(Position& p);
private:
    string name;
    int pos_x;
    int pos_y;
};
```

```
/* Position.cpp  (1) */
#include <iostream>
#include "Position.h"

Position::Position()
{
    cout << "Position::default constructor ()"
         << endl;
}

Position::Position(string nm)
{
   name = nm;
   cout << "Position::constructor (“
        << name << ")" << endl;
}

Position::Position(string nm, int x, int y)
{
   name = nm;
   pos_x = x;
   pos_y = y;
   cout << "Position::constructor ("
     << name << ", " << pos_x << ", “
     << pos_y << ")" << endl;
}

Position::Position(const Position& p)
{
   name = p.name;
   pos_x = p.pos_x;
   pos_y = p.pos_y;
   cout << "Position::copy constructor ("
        << name << ", " << pos_x
         << ", " << pos_y << ")" << endl;
}
```

```
/* Position.cpp  (2) */

Position::~Position()
{
   cout << "Position::destructor (" << name
      << ", " << pos_x << ", " << pos_y
      << ")" << endl;
}

Position& Position::operator= (Position& p)
{
    name = p.name;
    pos_x = p.pos_x;
    pos_y = p.pos_y;

    cout << "Position::operator=("
       << name << ", " << pos_x
       << ", " << pos_y << ")" << endl;

    return *this;
    // for chained operation, p3 = p2 = p1;
}
```

```
/* main.cpp */
#include <iostream>
#include "Position.h"

Position getNewPosition()
{
    cout << "Start of getNewPosition()" << endl;

    Position newPos("localPos", 7, 8);

    cout << "End of getNewPosition()" << endl;
    return newPos; // return-by-value
}

Position& getNewPosRef()
{
    cout << "Start of getNewPosRef()" << endl;

    Position refPos("localRefPos", 9, 10);

    cout << "End of getNewPosRef()" << endl;
    return refPos; // return-by-reference
}
```

```
void main()
{
    Position p1("p1", 3, 4), p2("p2"), p3("p3"), p4("p4"), p5("p5");
    cout << "Position p1 : " << p1 << endl << endl;

    p2 = Position("anonymous for p2", 5, 6);
    p2.setName("p2");
    cout << endl << "Position p2 : " << p2 << endl << endl;

    p3 = p2;
    p3.setName("p3");
    cout << endl << "Position p3 : " << p3 << endl << endl;

    p4 = getNewPosition();
    p4.setName("p4");
    cout << endl << "Position p4 : " << p4 << endl << endl;

    //p5 = getNewPosRef();
    /* Error may occur here !!! Guess why ? */
    //p5.setName("p5");
    //cout << endl << "Position p5 : " << p5 << endl << endl;

    cout << "End of main() ..." << endl;
}
```

```
Position::constructor (p1, 3, 4)
Position::constructor (p2)
Position::constructor (p3)
Position::constructor (p4)
Position::constructor (p5)
Position p1 : Position(p1, 3, 4)

Position::constructor (anonymous for p2, 5, 6)
Position::operator= (anonymous for p2, 5, 6)
Position::destructor (anonymous for p2, 5, 6)

Position p2 : Position(p2, 5, 6)

Position::operator= (p2, 5, 6)

Position p3 : Position(p3, 5, 6)

Start of getNewPosition()
Position::constructor (localPos, 7, 8)
End of getNewPosition()
Position::copy constructor (localPos, 7, 8)
Position::destructor (localPos, 7, 8)
Position::operator= (localPos, 7, 8)
Position::destructor (localPos, 7, 8)

Position p4 : Position(p4, 7, 8)

End of main() ...
Position::destructor (p5, -858993460, -858993460)
Position::destructor (p4, 7, 8)
Position::destructor (p3, 5, 6)
Position::destructor (p2, 5, 6)
Position::destructor (p1, 3, 4)
```

결과값 반환에서의 return-object를 위한 복제 생성자 실행

```
Position getNewPosition()
{
    cout << "Start of getNewPosition()" << endl;

    Position localPos("localPos", 0, 0);

    cout << "End of getNewPosition()" << endl;
    return localPos;
}
```

```
void main()
{
    Position p3;

    . . . .

    p3 = getNewPosition();

    . . . .
}
```

1) create a local object in getNewPosition(), localPos

"localPos", 0, 0

2) create a **return-object** as a copy of the localPos

"localPos", 0, 0

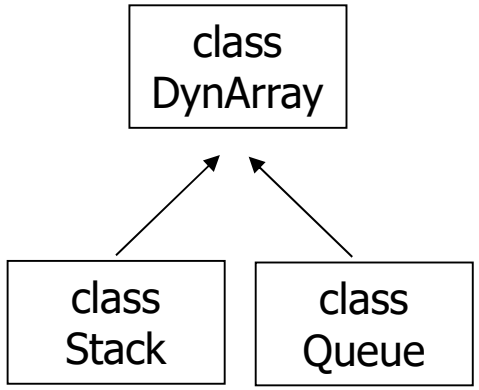
3) delete the local object localPos
4) return the **return-object** as the copy of localPos
5) perform assignment in main()
6) delete the **return-object**

동적 배열 생성이 포함되는
C++ 상속 클래스 예제
- DynArray, Stack, Queue

class DynArray

```
/* DynArray.h */
#ifndef DYN_ARRAY_H
#define DYN_ARRAY_H
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;

class DynArray
{
public:
    DynArray(string nm, int capa);
    ~DynArray();
    string getName() { return name; }
    int getCapacity() { return capacity; }
    int getSize() { return num_entry; }
    int& operator[](int idx);
protected:
    bool isValidIndex(int idx);
    string name;
    int* array;
    int capacity;
    int num_entry;
};
#endif
```



```
/* DynArray.cpp  (1) */
#include "DynArray.h"

DynArray::DynArray(string nm, int capa)
    :name(nm), capacity(capa)
{
    cout << "DynArray::DynArray(" << nm
         << ", " << capa << ")" << endl;
    array = (int*)new int[capacity];
    if (array == NULL)
    {
        cout << "Error in creation of dynamic array
                    of size (";
        cout << capacity << ") in DynArray
                constructor !!" << endl;
        exit;
    }
    num_entry = 0;
}
DynArray::~DynArray()
{
    cout << "DynArray::~DynArray()" << endl;
    delete[] array;
    array = NULL;
    num_entry = 0;
}
```

```
/* DynArray.cpp  (2) */

bool DynArray::isValidIndex(int idx)
{
    if ((idx >= 0) && (idx < capacity))
        return true;
    else
        return false;
}

int& DynArray::operator[](int idx)
{
    if (isValidIndex(idx))
        return array[idx];
    else {
        cout << "Error  - Invalid Index !!" << endl;
        exit;
    }
}
```

main() – testing dyn_array

```
/* main.cpp (1) */
#include <iostream>
#include <fstream>
#include "DynArray.h"
#include "Stack.h"
#include "Queue.h"
using namespace std;

#define ARRAY_SIZE 10
#define STACK_CAPACITY 10
#define QUEUE_CAPACITY 10
void main()
{
    int* pE;

    DynArray dyn_arr("DynArray", ARRAY_SIZE);

    for (int i = 0; i < ARRAY_SIZE; i++)
    {
        dyn_arr[i] = i;
    }

    cout << "DynArray : ";
    for (int i = 0; i < ARRAY_SIZE; i++)
    {
        cout << setw(3) << dyn_arr[i];
    }
    cout << endl;
```

```
DynArray::DynArray(DynArray, 10)
DynArray :   0  1  2  3  4  5  6  7  8  9
DynArray::DynArray(Int_Stack, 10)
Stack::Stack(Int_Stack,10)
int_stack.push() :   0  1  2  3  4  5  6  7  8  9
int_stack.pop()  :   9  8  7  6  5  4  3  2  1  0
DynArray::DynArray(Int_Queue, 10)
Queue::Queue(Int_Queue, 10)
int_queue.enque()  :   0  1  2  3  4  5  6  7
int_queue.dequeue():   0  1  2  3  4  5  6  7
int_queue.enque()  :   8  9 10 11 12 13 14 15
int_queue.dequeue():   8  9 10 11 12 13 14 15
int_queue.enque()  :  16 17 18 19 20 21 22 23
int_queue.dequeue():  16 17 18 19 20 21 22 23
Queue::~Queue(Int_Queue)
DynArray::~DynArray(Int_Queue)
Stack::~Stack(Int_Stack)
DynArray::~DynArray(Int_Stack)
DynArray::~DynArray(DynArray)
```

class Stack

```
/* Stack.h */
#include "DynArray.h"

class Stack : DynArray
{
public:
    Stack(string nm, int capa);
    ~Stack();
    bool isEmpty() { if (stack_top < 0) return true; else return false; }
    bool isFull() { if (stack_top >= capacity) return true; else return false; }
    int* pop();
    int* push(int element);
private:
    int stack_top;
};
```

```
/* Stack.cpp (1) */

#include "Stack.h"

Stack::Stack(string nm, int capa)
    : DynArray(nm, capa), stack_top(-1)
{
    cout << "Stack::Stack(" << nm
         << "," << capa << ")" << endl;
}

Stack::~Stack()
{
    cout << "Stack::~Stack()" << endl;
}

int* Stack::pop()
{
    if (isEmpty())
        return NULL;
    else
    {
        int* pE;
        pE = &array[stack_top];
        stack_top--;
        num_entry--;
        return pE;
    }
}
```

```
/* Stack.cpp (2) */

int* Stack::push(int element)
{
    if (isFull())
        return NULL;
    else
    {
        stack_top++;
        array[stack_top] = element;
        num_entry++;
        return &array[stack_top];
    }
}
```

main() – testing stack

```
/* main.cpp (2) */

    /* Testing Stack */
    Stack int_stack("Int_Stack", STACK_CAPACITY);
    cout << "int_stack.push() : ";
    for (int i = 0; i < STACK_CAPACITY; i++)
    {
        int_stack.push(i);
        cout << setw(3) << i;
    }
    cout << endl;
    cout << "int_stack.pop()  : ";
    for (int i = 0; i < STACK_CAPACITY; i++)
    {
        pE = int_stack.pop();
        cout << setw(3) << *pE;
    }
    cout << endl;
```

```
DynArray::DynArray(DynArray, 10)
DynArray :   0  1  2  3  4  5  6  7  8  9
DynArray::DynArray(Int_Stack, 10)
Stack::Stack(Int_Stack,10)
int_stack.push() :   0  1  2  3  4  5  6  7  8  9
int_stack.pop()  :   9  8  7  6  5  4  3  2  1  0
DynArray::DynArray(Int_Queue, 10)
Queue::Queue(Int_Queue, 10)
int_queue.enque()  :   0  1  2  3  4  5  6  7
int_queue.dequeue():   0  1  2  3  4  5  6  7
int_queue.enque()  :   8  9 10 11 12 13 14 15
int_queue.dequeue():   8  9 10 11 12 13 14 15
int_queue.enque()  :  16 17 18 19 20 21 22 23
int_queue.dequeue():  16 17 18 19 20 21 22 23
Queue::~Queue(Int_Queue)
DynArray::~DynArray(Int_Queue)
Stack::~Stack(Int_Stack)
DynArray::~DynArray(Int_Stack)
DynArray::~DynArray(DynArray)
```

class Queue

```
/* Queue.h */
#ifndef QUEUE_H
#define QUEUE_H
#include "DynArray.h"
class Queue : DynArray
{
public:
    Queue(string nm, int capa);
    ~Queue();
    int* enqueue(const int element);
    int* dequeue();
    bool isEmpty() {
        if (num_entry < 0) return true;
       else return false; }
    bool isFull() {
        if (num_entry >= capacity) return true;
       else return false; }
private:
    int front;
    int back;
};
#endif
```

```
/* Queue.cpp (1) */
#include "Queue.h"

Queue::Queue(string nm, int capa)
    :DynArray(nm, capa)
{
    cout << "Queue::Queue(" << nm
         << ", " << capa << ")" << endl;
    front = back = 0;
}

Queue::~Queue()
{
    cout << "Queue::~Queue()" << endl;
}

int* Queue::enqueue(const int
element)
{
    if (isFull())
        return NULL;
    int* pE;
    array[back] = element;
    pE = &array[back];
    back++;
    if (back >= capacity)
        back = back % capacity;
    num_entry++;
    return pE;
}
```

```
/* Queue.cpp (2) */

int* Queue::dequeue()
{
    if (isEmpty())
        return NULL;
    int* pE;
    pE = &array[front];
    front++;
    num_entry--;
    if (front >= capacity)
        front = front % capacity;
    return pE;
}
```

main() – testing queue

```
/* main.cpp (3) */

    /* Testing Queue */
    Queue int_queue("Int_Queue", QUEUE_CAPACITY);
    int count = 0;
    for (int k = 0; k < 3; k++)
    {
        cout << "int_queue.enque()  : ";
        for (int i = 0; i < 8; i++)
        {
            int_queue.enqueue(count);
            cout << setw(3) << count;
            count++;
        }
        cout << endl;
        cout << "int_queue.dequeue(): ";
        for (int j = 0; j < 8; j++)
        {
            pE = int_queue.dequeue();
            cout << setw(3) << *pE;
        }
        cout << endl;
    }
}
```

```
DynArray::DynArray(DynArray, 10)
DynArray :   0  1  2  3  4  5  6  7  8  9
DynArray::DynArray(Int_Stack, 10)
Stack::Stack(Int_Stack,10)
int_stack.push() :   0  1  2  3  4  5  6  7  8  9
int_stack.pop()  :   9  8  7  6  5  4  3  2  1  0
DynArray::DynArray(Int_Queue, 10)
Queue::Queue(Int_Queue, 10)
int_queue.enque()  :   0  1  2  3  4  5  6  7
int_queue.dequeue():   0  1  2  3  4  5  6  7
int_queue.enque()  :   8  9 10 11 12 13 14 15
int_queue.dequeue():   8  9 10 11 12 13 14 15
int_queue.enque()  :  16 17 18 19 20 21 22 23
int_queue.dequeue():  16 17 18 19 20 21 22 23
Queue::~Queue(Int_Queue)
DynArray::~DynArray(Int_Queue)
Stack::~Stack(Int_Stack)
DynArray::~DynArray(Int_Stack)
DynArray::~DynArray(DynArray)
```

상속관계 클래스들의 생성자 호출/실행 순서

◆클래스 상속 관계가 A ← B ← C 인 경우:

- class B가 class A로 부터 상속
- class C는 class B로 부터 상속

◆class C의 객체가 생성될 때:

- 맨 먼저 class C의 생성자가 실행되며, 내부에서 class B의 생성자를 먼저 호출됨
- class B의 생성자가 실행되며, 내부에서 class A의 생성자가 먼저 호출됨
- 실질적으로 class A의 생성자가 제일 먼저 실행됨
- 즉, 가장 내부에 있는 클래스의 생성자가 제일 먼저 실행됨

◆클래스 상속 관계에서 가장 내부 (상속 단계의 최상위) 클래스의 생성자가 제일 먼저 실행됨

파생클래스의 소멸자

◆파생클래스 소멸자의 구현

- 기반 클래스의 소멸자가 효과적으로 동작하면 파생 클래스의 소멸자를 쉽게 구현할 수 있음

◆파생클래스의 소멸자가 실행되는 경우

- 기반 클래스의 소멸자가 자동적으로 실행됨
- 파생 클래스에서 기반 클래스의 소멸자를 명시적으로 호출할 필요는 없음

◆파생클래스의 소멸자는 파생 클래스의 데이터 멤버와 그 데이터 멤버와 연관된 데이터에 대해서만 고려하면 됨

상속관계 클래스들의 소멸자 호출/실행 순서

◆클래스 상속 관계가 A ← B ← C 인 경우:

- class B가 class A로 부터 상속
- class C는 class B로 부터 상속

◆class C의 객체가 소멸될 때:

- 맨 먼저 class C의 소멸자가 실행
- 다음으로 class B의 소멸자가 실행
- 마지막으로 class A의 소멸자가 실행

◆소멸자는 생성자의 역순으로 호출/실행 됨

Homework 5

Homework 5

5.1 Object-oriented programming with inheritance

(1) class Shape

```
class Shape
{
    friend ostream& operator<<(ostream &, Shape &);
public:
    Shape(); // default constructor
    Shape(string name);
    Shape(int px, int py, double angle, COLORREF color, string name); // constructor
    ~Shape();
    void draw();
    void rotate(double rt_ang) { angle += rt_ang; }
    void move(int dx, int dy) { pos_x +=dx; pos_y += dy; }
    void print(ostream &);
    int get_pos_x() const { return pos_x; }
    int get_pos_y() const { return pos_y; }
    void set_pos_x(int x) { pos_x = x; }
    void set_pos_y(int y) { pos_y = y; }
    void setName(string n) { name = n; }
    string getName() { return name; }
    Shape& operator=(const Shape& s);
protected:
    int pos_x; // position x
    int pos_y; // position y
    double angle; // in radian
    string name;
    COLORREF color; // COLORREF is defined in <Windows.h>
};
```

```
/** Color.h */

#ifndef COLOR_H
#define COLOR_H
#include <Windows.h>
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

// COLORREF is defined in <Windows.h>
// The COLORREF value is used to specify an RGB color,
//     in hexadecimal form of 0x00bbggrr
const COLORREF RGB_BLACK   = 0x00000000;
const COLORREF RGB_RED     = 0x000000FF;
const COLORREF RGB_GREEN   = 0x0000FF00;
const COLORREF RGB_BLUE    = 0x00FF0000;
const COLORREF RGB_ORANGE  = 0x0000A5FF;
const COLORREF RGB_YELLOW  = 0x0000FFFF;
const COLORREF RGB_MAGENTA = 0x00FF00FF;
const COLORREF RGB_WHITE   = 0x00FFFFFF;
ostream& printRGB(ostream& ostr, const COLORREF color);
// RGB color code chart: https://www.rapidtables.com/web/color/RGB_Color.html
/* Note: RGB(red, green, blue) macro also provides COLORREF data
   . RGB(FF, 00, 00) => 0x000000FF (RGB_RED)
   . RGB(00, FF, 00) => 0x0000FF00 (RGB_GREEN)
   . RGB(00, 00, FF) => 0x00FF0000 (RGB_BLUE)
*/
#endif
```

```
class Elps : public Shape  // Ellipse
{
   friend ostream& operator<<(ostream &, const Elps &);
public:
   Elps();
   Elps(string name);
   Elps(int px, int py, double r1, double r2, double ang, COLORREF clr, string name);
   ~Elps();
   double getArea() const;
   void draw();
   void print(ostream &);
   double getRadius_1() const { return radius_1; }
   double getRadius_2() const { return radius_2; }
   void setRadius(double r1, double r2) { radius_1 = r1; radius_2 = r2; }
   Elps& operator=(const Elps& elp);
protected:
   double radius_1;
   double radius_2;
};
```

```
class ElpsCylinder : public Elps
{
    friend ostream& operator<<(ostream &ostr, const ElpsCylinder &elpcyl);
public:
    ElpsCylinder();  // default constructor
    ElpsCylinder(string n);
    ElpsCylinder(int px, int py, double r1, double r2, double h, double ang, COLORREF clr, string n);
    virtual ~ElpsCylinder();
    double getArea() const;
    double getVolume() const;
    void draw();
    void print(ostream &);
    ElpsCylinder& operator=(const ElpsCylinder& right);
protected:
    double height; // Cylinder height
};
```

```
/** main.cpp  */
#include <iostream>
#include <fstream>
#include <string>
#include "Color.h"
#include "Shape.h"
#include "Elps.h"
#include "ElpsCylinder.h"

using namespace std;

int main()
{
    fstream fout;
    Shape shape(1, 1, 0, RGB_BLACK, "Shape");
    Elps red_elps(8, 8, 3.0, 4.0, 0, RGB_RED, "Red_Elps");
    ElpsCylinder blue_elpcyl(9, 9, 5.0, 6.0, 7.0, 0.0, RGB_BLUE, "Blue_Elp_Cyl");

    fout.open("output.txt", 'w');
    if (fout.fail())
    {
        cout << "Failed in opening output.txt file !!" << endl;
        exit;
    }
    fout << "List of shapes using operator<<() friend function " << endl;
    fout << shape << endl;
    fout << red_elps << endl;
    fout << blue_elpcyl << endl;

    fout.close();
    return 0;

}  // end of main()
```

◆ 실행 결과

```
List of shapes using operator<<() friend function
     Shape: pos (  1,   1), angle ( 0.00), color (000000)
  Red_Elps: pos (  8,   8), angle ( 0.00), color (0000FF), radius (3.00, 4.00), ellipse_area (37.70)
Blue_Elp_Cyl: pos (  9,   9), angle ( 0.00), color (FF0000), radius (5.00, 6.00), height (7.00),
         ellipse_area (94.25), elp_cyl area (430.90), elp_cyl volume (659.73)
```