Inheritance with Examples

Computer Programming for Engineers (DSAF003-42)

Instructor:

Youngjoong Ko (nlp.skku.edu)

RECAP WITH EXAMPLES

#0: Class Basics

- Class objects can be declared in various ways.
  - As a variable, an array, a pointer or a reference.
  - In demo, counting the number of objects will be shown.
    - You can do it with a static variable.

Class의 변수, 포인터, 레퍼런스 등 다양한 방법 사용.

```
Person personA;
personA.printInfo();

Person personB("Mike");
personB.printInfo();

Person persons[10];
for(int i=0; i<10; i++)
  persons[i].printInfo();

Person& personC = personB;
```

- Check the usage of static and non-static member variables. (1_basic_static.cpp)


```
class Person {
  public:
    ...
  private:
    string name;
    int myCount;
    static int personCount;
};
int Person::personCount = 0; // static member variable initialization

Person::Person():name("Not set") {
  myCount = personCount++; // setting my count using static variable
}

Person::Person(string name):name(name) {
  myCount = personCount++;
}

Person::~Person() {
  cout << "in ~Person() #" << myCount << endl; // showing my count
}
```

모든 객체에서 접근 가능

INHERITANCE RECAP #1

Inheritance Recap

```
class HourlyEmployee : public Employee
{
    ...
};
```

- Base/Derived class
  - A derived class *inherits* members from the base class.

파생은 기본에서 멤버 상속.

- Redefining member functions

멤버 func 재정의 ㅍ

  - We can change the behavior of inherited member functions.

상속된 멤버 func의 동작 변경 ㅇ.

- Constructors cannot be inherited.

생성자는 상속 X.

  - But, they **can be invoked** within derived class constructor.

DC에서 호출은 ㄱ

- Protected qualifier allows access "by name" in derived class.

는 이름빅 액세스 ㅇㄱ.

#1: Calling and Redefining Parent's Member Functions

- Remind Person/Student Example
  - Functions about name are defined in Person class.
  - Functions about student id are defined in Student class.

- In the demo we will
  - add marks at names of students by calling base class's member function.
    - References can be used.
  - redefine setName function to add marks as default.
    - protected qualifier can be used.

- Fill in the blank during demo


```
// 2_student.cpp
void Student::addMark()
{
    
}
```

```
// 3_student_reference.cpp
// We need to change the definition of getName function for it.
void Student::addMark()
{
    
}
```

getName이 필요 X

Fill in the blank during demo

DEMO

```
// 4_student_redefine_setName.cpp
void Student::setName(string name)
{

}
```

D(en setname에서
B(en " 을 받고
인자에 name+"(student)"로
합쳐버림

```
// 5_student_redefine_setName_protected.cpp
// We need to change the private qualifier in Person class.
void Student::setName(string name)
{

}
```

protected 상속 받아서
private로 사용.

INHERITANCE RECAP #2

#2: Ctor and Dtor Call Sequence

- We will vividly see the call sequences of ctors and dtors.

Remind about the ctors of child classes:

- Should always invoke one of the base class's constructors

BC의 생성자 호출 안 하면 default가 자동 호출.

- If you do not, **default** base class constructor **automatically called**.

- Initialization with parent's 5ctor (6_parent_initialize_ctor.cpp)


```
class Parent : public GrandParent
{
  public:
    Parent(int age) : GrandParent(age){
      cout << "in Parent(int age)" << endl;
    }
    ~Parent() { cout << "in ~Parent()" << endl; }
    // We cannot initialize parents' member variable.
    //Parent(int age) : age(age) { cout << "in Parent(int age)" << endl; }
};
```

↳ 이렇게 하면 초기화 안돼~

- Initialization with protected quailfier

이게 되려면 → 여기 age가 protected로 선언

```
class Parent : public GrandParent
{
  public:
    Parent(int age) {
      this->age = age;
      cout << "in Parent(int age)" << endl;
    }
    ~Parent() { cout << "in ~Parent()" << endl; }
};
```

- Following code has an error
  - Hint: Remind “Should always invoke one of the base class’s constructors”


```
class GrandParent
{
  public:
    GrandParent():age(1000) { cout << "in GrandParent()" << endl; }
    GrandParent(int age) : age(age)
      { cout << "in GrandParent(int age)" << endl; }
    void printAge() { cout << age << endl; }
private:
    int age;
};
class Parent : public GrandParent
{
  public:
    Parent(int age) {
      this->age = age;
      cout << "in Parent(int age)" << endl;
    }
};
```