객체지향프로그래밍 II

Lecture 5

제12장 클래스 상속 (Part 3)

1. Inheritance Hierarchy Using protected Data
2. Inheritance Hierarchy Using private Data

박인규 교수

1. Inheritance Hierarchy Using protected Data

protected 데이터 사용

- ✓ BasePlusCommissionEmployee 클래스 (파생 클래스) 가 기본 클래스 (CommissionEmployee) 데이터 멤버에 직접 접근하기 위해서 protected 데이터를 사용해야 한다.
- ✓ 기본 클래스의 protected 멤버는 기본 클래스의 모든 파생 클래스에 상속되며 파생 클래스의 멤버 함수에서 접근할 수 있다.

protected 멤버를 이용한 CommissionEmployee 클래스 개선

```
// Fig. 12.12: CommissionEmployee.h
// CommissionEmployee class definition with protected data.
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string> // C++ standard string class
using std::string;

class CommissionEmployee
{
public:
   CommissionEmployee( const string &, const string &, const string &,
      double = 0.0, double = 0.0 );

   void setFirstName( const string & ); // set first name
   string getFirstName() const; // return first name

   void setLastName( const string & ); // set last name
   string getLastName() const; // return last name

   void setSocialSecurityNumber( const string & ); // set SSN
   string getSocialSecurityNumber() const; // return SSN

```

protected 멤버를 이용한 CommissionEmployee 클래스 개선

```
   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate

   double earnings() const; // calculate earnings
   void print() const; // print CommissionEmployee object
protected:
   string firstName;
   string lastName;
   string socialSecurityNumber;
   double grossSales; // gross weekly sales
   double commissionRate; // commission percentage
}; // end class CommissionEmployee

#endif
```

Declare **`protected`** data

protected 멤버를 이용한 CommissionEmployee 클래스 개선

```cpp
// Fig. 12.14: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class derived from class
// CommissionEmployee.
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <string> // C++ standard string class
using std::string;

#include "CommissionEmployee.h" // CommissionEmployee class declaration

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const string &, const string &,
      const string &, double = 0.0, double = 0.0, double = 0.0 );

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   double earnings() const; // calculate earnings
   void print() const; // print BasePlusCommissionEmployee object
private:
   double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee

#endif
```

BasePlusCommissionEmployee still inherits **publicly** from **CommissionEmployee**

protected 멤버를 이용한 CommissionEmployee 클래스 개선

```cpp
// Fig. 12.15: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
using std::cout;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   // explicitly call base-class constructor
   : CommissionEmployee( first, last, ssn, sales, rate )
{
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary
```

Call base-class constructor using base-class initializer syntax

protected 멤버를 이용한 CommissionEmployee 클래스 개선

```

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   // can access protected data of base class
   return baseSalary + ( commissionRate * grossSales );
} // end function earnings

// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   // can access protected data of base class
   cout << "base-salaried commission employee: " << firstName << ' '
      << lastName << "\nsocial security number: " << socialSecurityNumber
      << "\ngross sales: " << grossSales
      << "\ncommission rate: " << commissionRate
      << "\nbase salary: " << baseSalary;
} // end function print
```

Directly access base class’s **`protected`** data

protected 접근 지정자 사용의 장단점

장점

- 파생 클래스가 기본 클래스의 데이터 멤버를 바로 조절 가능
  - *set* / *get* 호출하는데 소모되는 오버헤드를 피할 수 있다.

    -수행 속도가 조금 향상된다.

단점

- 유효체크 불가 - set/get 함수를 사용하지 않으므로 파생 클래스에 비정상적인 값이 할당될 수 있다.
- 실행 의존적
  - 파생 클래스 함수는 기본 클래스 구현에 더 의존적이 됨
  - 기본 클래스를 수정하면 파생 클래스 또한 조절 되어야 한다.

    - 부서지기 쉬운 (fragile) 소프트웨어

2. Inheritance Hierarchy Using private Data

계층 구조의 개선

- ✓ 소프트웨어 공학 기법의 권고를 따름
  - 데이터 멤버는 private 로 선언
  - public *get* 과 *set* 멤버 함수 제공
  - 기본 클래스에서 상속된 *get* 멤버 함수를 사용하여

    파생 클래스에서 기본 클래스의 private 데이터 멤버의 값을 얻도록 함
- ✓ 파생 클래스에서 재정의된 (override된) 멤버 함수의 호출 방식
  - 파생 클래스 함수 내부에서는 그 함수 이름만으로는 재정의된 파생

    클래스의 멤버 함수를 호출함
  - 따라서, 같은 이름의 기본 클래스의 멤버 함수를 호출하기 위해서는

    "기본클래스이름::함수이름"과 같이 호출해야 함

CommissionEmployee 클래스 개선 (최종버전)

```
// Fig. 12.17: CommissionEmployee.h
// CommissionEmployee class definition with good software engineering.
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string> // C++ standard string class
using std::string;

class CommissionEmployee
{
public:
   CommissionEmployee( const string &, const string &, const string &,
      double = 0.0, double = 0.0 );

   void setFirstName( const string & ); // set first name
   string getFirstName() const; // return first name

   void setLastName( const string & ); // set last name
   string getLastName() const; // return last name

   void setSocialSecurityNumber( const string & ); // set SSN
   string getSocialSecurityNumber() const; // return SSN

   void setGrossSales( double ); // set gross sales amount
   double getGrossSales() const; // return gross sales amount

   void setCommissionRate( double ); // set commission rate
   double getCommissionRate() const; // return commission rate
```

CommissionEmployee 클래스 개선 (최종버전)

```

   double earnings() const; // calculate earnings
   void print() const; // print CommissionEmploye
private:
   string firstName;
   string lastName;
   string socialSecurityNumber;
   double grossSales; // gross weekly sales
   double commissionRate; // commission percentage
}; // end class CommissionEmployee

#endif
```

Declare **`private`** data

CommissionEmployee 클래스 개선 (최종버전)

```cpp
// Fig. 12.18: CommissionEmployee.cpp
// Class CommissionEmployee member-function definitions.
#include <iostream>
using std::cout;

#include "CommissionEmployee.h" // CommissionEmployee class

// constructor
CommissionEmployee::CommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate )
   : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
{
   setGrossSales( sales ); // validate and store gross sales
   setCommissionRate( rate ); // validate and store commission rate
} // end CommissionEmployee constructor

// set first name
void CommissionEmployee::setFirstName( const string &first )
{
   firstName = first; // should validate
} // end function setFirstName

// return first name
string CommissionEmployee::getFirstName() const
{
   return firstName;
} // end function getFirstName
```

Use member initializers to set the values of members **`firstName`**, **`lastname`** and **`socialSecurityNumber`**

CommissionEmployee 클래스 개선 (최종버전)

```

// set last name
void CommissionEmployee::setLastName( const string &last )
{
   lastName = last; // should validate
} // end function setLastName

// return last name
string CommissionEmployee::getLastName() const
{
   return lastName;
} // end function getLastName

// set social security number
void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
{
   socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber

// return social security number
string CommissionEmployee::getSocialSecurityNumber() const
{
   return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void CommissionEmployee::setGrossSales( double sales )
{
   grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales
```

CommissionEmployee 클래스 개선 (최종버전)

```

// return gross sales amount
double CommissionEmployee::getGrossSales() const
{
   return grossSales;
} // end function getGrossSales

// set commission rate
void CommissionEmployee::setCommissionRate( double rate )
{
   commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate

// return commission rate
double CommissionEmployee::getCommissionRate() const
{
   return commissionRate;
} // end function getCommissionRate

// calculate earnings
double CommissionEmployee::earnings() const
{
   return getCommissionRate() * getGrossSales();
} // end function earnings

```

Use *get* functions to obtain the values of data members

CommissionEmployee 클래스 개선 (최종버전)

```
// print CommissionEmployee object
void CommissionEmployee::print() const
{
   cout << "commission employee: "
      << getFirstName() << ' ' << getLastName()
      << "\nsocial security number: " << getSocialSecurityNumber()
      << "\ngross sales: " << getGrossSales()
      << "\ncommission rate: " << getCommissionRate();
} // end function print
```

Use *get* functions to obtain the values of data members

BasePlusCommissionEmployee 클래스 개선 (최종버전)

```cpp
// Fig. 12.19: BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class derived from class
// CommissionEmployee.
#ifndef BASEPLUS_H
#define BASEPLUS_H

#include <string> // C++ standard string class
using std::string;

#include "CommissionEmployee.h" // CommissionEmployee class declaration

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
   BasePlusCommissionEmployee( const string &, const string &,
      const string &, double = 0.0, double = 0.0, double = 0.0 );

   void setBaseSalary( double ); // set base salary
   double getBaseSalary() const; // return base salary

   double earnings() const; // calculate earnings
   void print() const; // print BasePlusCommissionEmployee object
private:
   double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee

#endif
```

BasePlusCommissionEmployee 클래스 개선 (최종버전)

```cpp
// Fig. 12.20: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
using std::cout;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   // explicitly call base-class constructor
   : CommissionEmployee( first, last, ssn, sales, rate )
{
   setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
   baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary

// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary
```

BasePlusCommissionEmployee 클래스 개선 (최종버전)

```cpp

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   return getBaseSalary() + CommissionEmployee::earnings();
} // end function earnings

// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   cout << "base-salaried ";

   // invoke CommissionEmployee's print function
   CommissionEmployee::print();

   cout << "\nbase salary: " << getBaseSalary();
} // end function print
```

Invoke base class's **`earnings`** function

Invoke base class's **`print`** function

설계된 클래스 활용 (driver)

```cpp
// Fig. 12.21: fig12_21.cpp
// Testing class BasePlusCommissionEmployee.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

```

설계된 클래스 활용 (driver)

```cpp
int main()
{
   // instantiate BasePlusCommissionEmployee object
   BasePlusCommissionEmployee
      employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

   // get commission employee data
   cout << "Employee information obtained by get functions: \n"
      << "\nFirst name is " << employee.getFirstName()
      << "\nLast name is " << employee.getLastName()
      << "\nSocial security number is "
      << employee.getSocialSecurityNumber()
      << "\nGross sales is " << employee.getGrossSales()
      << "\nCommission rate is " << employee.getCommissionRate()
      << "\nBase salary is " << employee.getBaseSalary() << endl;

   employee.setBaseSalary( 1000 ); // set base salary

   cout << "\nUpdated employee information output by print function: \n"
      << endl;
   employee.print(); // display the new employee information

   // display the employee's earnings
   cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;

   return 0;
} // end main
```

Create **`BasePlusCommissionEmployee`** object

Use inherited *get* methods to access base class **`private`** members

Use **`BasePlusCommissionEmployee`** *get* method to access **`private`** member

Use **`BasePlusCommissionEmployee`** *set* method to modify **`private`** data member **`baseSalary`**

설계된 클래스 활용 (실행 결과)

```
Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

객체지향프로그래밍 II

Lecture 5

제12장 클래스 상속 (Part 4)

1. 상속 구조에서 생성자, 소멸자의 실행 순서
2. public, protected and private Inheritance

박인규 교수

1. 상속 구조에서 생성자, 소멸자의 실행 순서

파생 클래스 객체의 생성 과정

- 기본클래스의 생성자와 소멸자는 파생클래스에 상속되지 않는다.
- 파생 클래스 객체의 생성 과정
  - ✓ 연쇄적인 생성자 호출/실행/리턴 메커니즘
    - 파생클래스 생성자는 기본클래스 생성자를 호출한다.
      - 명시적(explicitly) 혹은 암시적(implicitly)
    - 최상위 단계의 기본 클래스
      - 마지막으로 생성자가 호출됨
      - 처음으로 생성자 실행을 마치고 리턴
    - CommissionEmployee/BasePlueCommissionEmployee 계층
      - CommissionEmployee : 마지막에 생성자 호출하고 처음으로 생성자 실행을 마침
    - 데이터 멤버 초기화
      - 각 기본 클래스의 생성자는 파생 클래스의 member initializer에서 호출되어 자신의 데이터 멤버를 초기화 한다.

파생 클래스 객체의 소멸 과정

- 연쇄적 소멸자 호출
  - ✓ 연쇄적 생성자 구조의 반대 (함수 body의 실행순서)
  - ✓ 파생 클래스의 소멸자가 처음 호출 및 실행
  - ✓ 그 다음 상위 기본 클래스 소멸자 호출 및 실행
    - 최종 기본 클래스에 도착할 때 까지 계속
    - 리턴 순서는 기본 클래스 → 파생 클래스의 순서
      - 가장 하위의 파생 클래스 소멸자가 최종적으로 리턴되면 메모리에서 객체가 제거된다.

생성자 및 소멸자 호출 순서 예제 (CommissionEmployee.cpp)

```cpp
// Fig. 12.23: CommissionEmployee.cpp
// Class CommissionEmployee member-function definitions.
#include <iostream>
using std::cout;
using std::endl;

#include "CommissionEmployee.h" // CommissionEmployee class definition

// constructor
CommissionEmployee::CommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate )
   : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
{
   setGrossSales( sales ); // validate and store gross sales
   setCommissionRate( rate ); // validate and store commission rate

   cout << "CommissionEmployee constructor: " << endl;
   print();
   cout << "\n\n";
} // end CommissionEmployee constructor

// destructor
CommissionEmployee::~CommissionEmployee()
{
   cout << "CommissionEmployee destructor: " << endl;
   print();
   cout << "\n\n";
} // end CommissionEmployee destructor
```

Constructor and destructor output messages to demonstrate function call order

생성자 및 소멸자 호출 순서 예제 (BasePlusCommissionEmployee.cpp)

```cpp
// Fig. 12.25: BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
using std::cout;
using std::endl;

// BasePlusCommissionEmployee class definition
#include "BasePlusCommissionEmployee.h"

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
   const string &first, const string &last, const string &ssn,
   double sales, double rate, double salary )
   // explicitly call base-class constructor
   : CommissionEmployee( first, last, ssn, sales, rate )
{
   setBaseSalary( salary ); // validate and store base salary

   cout << "BasePlusCommissionEmployee constructor: " << endl;
   print();
   cout << "\n\n";
} // end BasePlusCommissionEmployee constructor

// destructor
BasePlusCommissionEmployee::~BasePlusCommissionEmployee()
{
   cout << "BasePlusCommissionEmployee destructor: " << endl;
   print();
   cout << "\n\n";
} // end BasePlusCommissionEmployee destructor
```

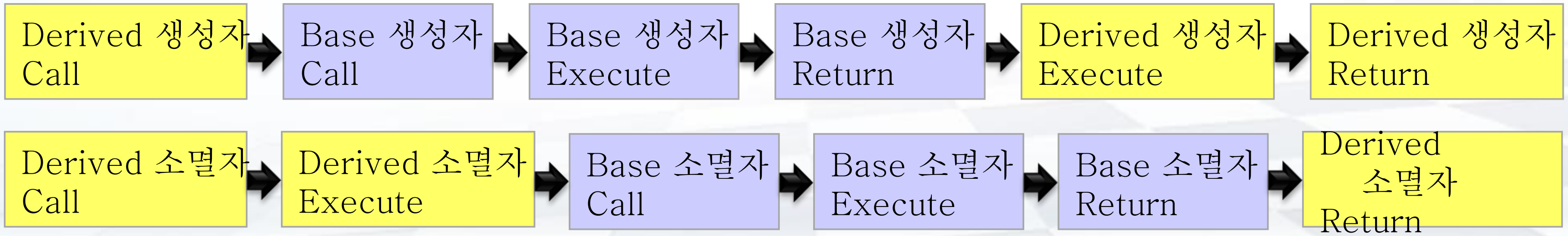
Constructor and destructor output messages to demonstrate function call order

생성자 및 소멸자 호출 순서 예제 (driver)

```cpp
int main()
{
   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

   { // begin new scope
     CommissionEmployee employee1("Bob", 5000 );
   } // end scope

   cout << endl;
   BasePlusCommissionEmployee employee2( "Lisa", 2000, 800 );

   cout << endl;
   BasePlusCommissionEmployee employee3( "Mark", 8000, 2000 );

   cout << endl;
   return 0;
} // end main
```

`CommissionEmployee` object goes in and out of scope immediately

Instantiate two **`BasePlusCommissionEmployee`** objects to demonstrate order of derived-class and base-class constructor/destructor function calls



생성자 및 소멸자 호출 순서 예제 (실행 결과)

```
CommissionEmployee constructor:
commission employee: Bob
gross sales: 5000.00

CommissionEmployee destructor:
commission employee: Bob
gross sales: 5000.00


CommissionEmployee constructor:
commission employee: Lisa
gross sales: 2000.00

BasePlusCommissionEmployee constructor:
base-salaried commission employee: Lisa
gross sales: 2000.00
base salary: 800.00


CommissionEmployee constructor:
commission employee: Mark
gross sales: 8000.00
```

`CommissionEmployee` constructor called for object in block; destructor called immediately as execution leaves scope

Base-class **`CommissionEmployee`** constructor executes first when instantiating derived-class **`BasePlusCommissionEmployee`** object

Derived-class **`BasePlusCommissionEmployee`** constructor body executes after base-class **`Employee`**’s constructor finishes execution

Base-class **`CommissionEmployee`** constructor executes first when instantiating derived-class **`BasePlusCommissionEmployee`** object

생성자 및 소멸자 호출 순서 예제 (실행 결과)

```
BasePlusCommissionEmployee constructor:
base-salaried commission employee: Mark
gross sales: 8000.00
base salary: 2000.00

BasePlusCommissionEmployee destructor:
base-salaried commission employee: Mark
gross sales: 8000.00
base salary: 2000.00

CommissionEmployee destructor:
commission employee: Mark
gross sales: 8000.00

BasePlusCommissionEmployee destructor:
base-salaried commission employee: Lisa
gross sales: 2000.00
base salary: 800.00

CommissionEmployee destructor:
commission employee: Lisa
gross sales: 2000.00
```

Derived-class **`BasePlusCommissionEmployee`** constructor body executes after base-class **`CommissionEmployee`**'s constructor finishes execution

Destructors for **`BasePlusCommissionEmployee`** object called in reverse order of constructors

Destructors for **`BasePlusCommissionEmployee`** object called in reverse order of constructors

2. public, protected and private Inheritance

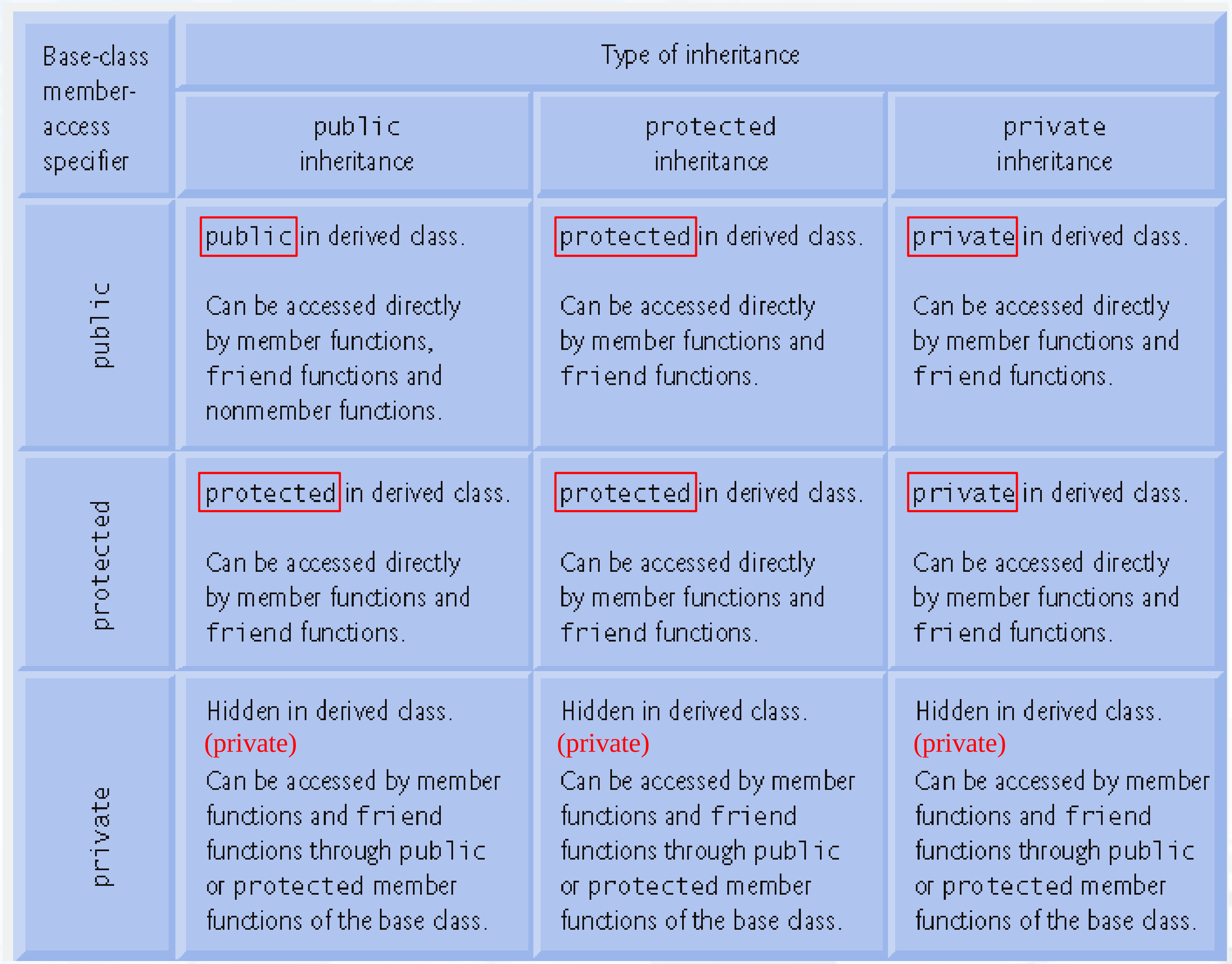
public 상속

- 기본 클래스 public 멤버
  - ➡ 파생 클래스 public 멤버
- 기본 클래스 protected 멤버
  - ➡ 파생 클래스 protected 멤버
- 기본 클래스 private 멤버는 직접 접근 불가능
  - ➡ 기본 클래스의 public 멤버 함수를 통해 접근 가능

protected 상속 및 private 상속

- protected 상속
  - ✓ 기본 클래스 public와 protected 멤버
    - ➡ 파생 클래스 protected 멤버
- private 상속
  - ✓ 기본 클래스 public와 protected 멤버
    - ➡ 파생 클래스 private 멤버

파생 클래스에서의 기본 클래스 데이터 멤버 접근성



Base-class member-access specifier	Type of inheritance		
	public inheritance	protected inheritance	private inheritance
public	public in derived class. Can be accessed directly by member functions, friend functions and nonmember functions.	protected in derived class. Can be accessed directly by member functions and friend functions.	private in derived class. Can be accessed directly by member functions and friend functions.
protected	protected in derived class. Can be accessed directly by member functions and friend functions.	protected in derived class. Can be accessed directly by member functions and friend functions.	private in derived class. Can be accessed directly by member functions and friend functions.
private	Hidden in derived class. (private) Can be accessed by member functions and friend functions through public or protected member functions of the base class.	Hidden in derived class. (private) Can be accessed by member functions and friend functions through public or protected member functions of the base class.	Hidden in derived class. (private) Can be accessed by member functions and friend functions through public or protected member functions of the base class.

12주차 2교시 수업을 마칩니다.