Object-Oriented Programming II

객체지향프로그래밍 II

Lecture 6

제13장 클래스 다형성 (Part 1)

1. Introduction
2. Polymorphism 사례
3. 핸들의 자료형에 의한 멤버 함수 호출 (without Polymorphism)

박인규 교수

1. Introduction

상속 계층 구조에서의 다형성 (polymorphism)

- ✓ "program in the general" vs. "program in the specific"
- ✓ 클래스 계층 구조에서의 다양한 객체들에 대해서, 마치 객체들이 기반 클래스의 객체인 것처럼 처리
- ✓ 이 때, 각 객체는 자신에게 적합한 작업을 수행
  - 즉, 서로 다른 형의 객체는 서로 다른 작업을 수행
- ✓ 기존의 코드를 수정하지 않고 새로운 클래스를 추가할 수 있는 효율적인 방법임

사례: Animal 클래스 계층에서 다형성 동작

- ✓ Animal 기본 클래스와 모든 파생 클래스(Fish, Frog, Bird)는 move 함수를 갖는다.
- ✓ 서로 다른 파생 클래스 객체 포인터들은 Animal* 형의 핸들에 저장
- ✓ 프로그램은 같은 메시지를 (예: move) 각 Animal 객체에 전달한다. 즉, 핸들로 move 함수 호출
- ✓ 이 때, 각 객체는 자신에게 적당한 함수를 호출한다.
  - Fish 는 수영하며 move를 수행
  - Frog 는 점프하며 move를 수행
  - Bird 는 날아가며 move를 수행

2. Polymorphism 사례

Polymorphism 동작 메커니즘

1. Derived class 객체를 base class의 포인터/참조형 핸들로 받음

2. Base class에서 가상(virtual) 함수로 선언된 함수를 실행시킬 때 동작

   - C++는 객체를 실제로 생성한 클래스(즉, derived class)의 함수를 실행시키게 됨
   - 핸들(handle)의 자료형(즉, base class)의 함수가 아님

virtual function을 이용하여 동작 메커니즘을 구현 가능

예제: SpaceObjects

- ✓ 어떤 비디오 게임 프로그램은 멤버 함수 draw를 포함하는 SpaceObject로부터 상속받은 파생 클래스의 객체들을 이용한다.
- ✓ draw 함수는 파생 클래스 마다 다르게 구현된다.
- ✓ 스크린 매니저 프로그램은 SpaceObject 객체의 포인터들을 저장하고 있다.
- ✓ 각 객체의 draw 호출은 저장된 SpaceObject 포인터의 사용으로 구현된다.
  - draw 함수는 실제 객체의 자료형에 따라 적합하게 호출된다.
- ✓ SpaceObject로부터 새로 파생된 클래스는 스크린 매니저 프로그램을 수정하지 않고 추가 할 수 있다.

3. 핸들의 자료형에 의한 멤버 함수 호출

(without Polymorphism)

virtual 함수를 사용하지 않을 때 (without using polymorphism)

- ✓ Base class의 포인터 핸들이 base class의 객체를 가리키면
  - 핸들의 자료형과 실제 객체의 자료형이 같음
  - 핸들 자료형의 함수를 호출 → base class 함수 호출
- ✓ Derived class의 포인터 핸들이 derived class의 객체를 가리키면
  - 핸들의 자료형과 실제 객체의 자료형이 같음
  - 핸들 자료형의 함수를 호출 → derived class 함수 호출
- ✓ Base class의 포인터 핸들이 derived class의 객체를 가리키면
  - 핸들은 base class의 포인터, 실제 객체는 파생 클래스의 객체임
  - 핸들 자료형의 함수를 호출 → base class 함수 호출

가상 함수 (virtual function)의 기능

- 즉, virtual 함수를 사용하지 않을 때 (without using polymorphism), 함수의 호출은 객체의 핸들(handle)의 자료형에 의해서 결정됨
  - ✓ 생성된 객체의 실제 자료형이 아님

- virtual 함수를 이용할 때 (using polymorphism)
  - ✓ (핸들의 자료형이 아닌) 실제 객체의 자료형이 호출할 가상함수를 결정하게 된다.
  - ✓ C++에서 객체지향 설계의 다형성(polymorphism) 구현의 방법임

핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

```
1  // Fig. 13.1: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
24    void setGrossSales( double ); // set gross sales amount
25    double getGrossSales() const; // return gross sales amount
```

핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

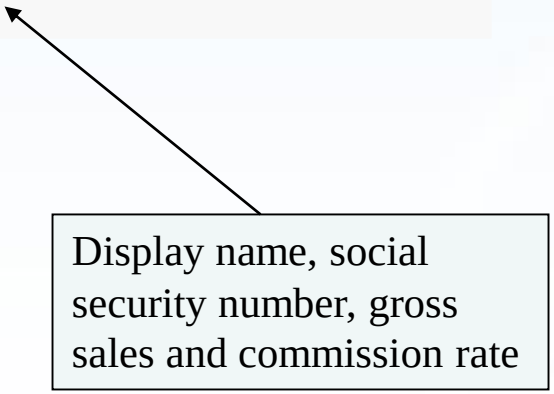
```
26
27    void setCommissionRate( double ); // set commission rate
28    double getCommissionRate() const; // return commission rate
29
30    double earnings() const; // calculate earnings
31    void print() const; // print CommissionEmployee object
32 private:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // gross weekly sales
37    double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

Function **`earnings`** will be redefined in derived classes to calculate the employee's earnings

Function **`print`** will be redefined in derived class to print the employee's information

핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

```
83
84 // print CommissionEmployee object
85 void CommissionEmployee::print() const
86 {
87    cout << "commission employee: "
88       << getFirstName() << ' ' << getLastName()
89       << "\nsocial security number: " << getSocialSecurityNumber()
90       << "\ngross sales: " << getGrossSales()
91       << "\ncommission rate: " << getCommissionRate();
92 } // end function print
```



핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

```
1  // Fig. 13.3: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from class
3  // CommissionEmployee.
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    double earnings() const; // calculate earnings
22    void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

Redefine functions **`earnings`** and **`print`**

핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

```
30
31 // calculate earnings
32 double BasePlusCommissionEmployee::earnings() const
33 {
34    return getBaseSalary() + CommissionEmployee::earnings();
35 } // end function earnings
36
37 // print BasePlusCommissionEmployee object
38 void BasePlusCommissionEmployee::print() const
39 {
40    cout << "base-salaried ";
41
42    // invoke CommissionEmployee's print function
43    CommissionEmployee::print();
44
45    cout << "\nbase salary: " << getBaseSalary();
46 } // end function print
```

Redefined `earnings` function incorporates base salary

Redefined `print` function displays additional **`BasePlusCommissionEmployee`** details

핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

```
1  // Fig. 13.5: fig13_05.cpp
2  // Aiming base-class and derived-class pointers at base-class
3  // and derived-class objects, respectively.
4  #include <iostream>
5  using std::cout;
6  using std::endl;
7  using std::fixed;
8
9  #include <iomanip>
10 using std::setprecision;
11
12 // include class definitions
13 #include "CommissionEmployee.h"
14 #include "BasePlusCommissionEmployee.h"
15
16 int main()
17 {
18    // create base-class object
19    CommissionEmployee commissionEmployee(
20       "Sue", "Jones", "222-22-2222", 10000, .06 );
21
22    // create base-class pointer
23    CommissionEmployee *commissionEmployeePtr = 0;
```

핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

```
24
25    // create derived-class object
26    BasePlusCommissionEmployee basePlusCommissionEmployee(
27       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
28
29    // create derived-class pointer
30    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
31
32    // set floating-point output formatting
33    cout << fixed << setprecision( 2 );
34
35    // output objects commissionEmployee and basePlusCommissionEmployee
36    cout << "Print base-class and derived-class objects:\n\n";
37    commissionEmployee.print(); // invokes base-class print
38    cout << "\n\n";
39    basePlusCommissionEmployee.print(); // invokes derived-class print
40
41    // aim base-class pointer at base-class object and print
42    commissionEmployeePtr = &commissionEmployee; // perfectly natural
43    cout << "\n\n\nCalling print with base-class pointer to "
44       << "\nbase-class object invokes base-class print function:\n\n";
45    commissionEmployeePtr->print(); // invokes base-cla
```

Aiming base-class pointer at base-class object and invoking base-class functionality

핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

```
46
47    // aim derived-class pointer at derived-class object and print
48    basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
49    cout << "\n\n\nCalling print with derived-class pointer to "
50       << "\nderived-class object invokes derived-class "
51       << "print function:\n\n";
52    basePlusCommissionEmployeePtr->print(); // invokes derived-class print
53
54    // aim base-class pointer at derived-class object and print
55    commissionEmployeePtr = &basePlusCommissionEmployee;
56    cout << "\n\n\nCalling print with base-class pointer to "
57       << "derived-class object\ninvokes base-class print "
58       << "function on that derived-class object:\n\n";
59    commissionEmployeePtr->print(); // invokes base-class print
60    cout << endl;
61    return 0;
62 } // end main
```

Aiming derived-class pointer at derived-class object and invoking derived-class functionality

Aiming base-class pointer at derived-class object and invoking base-class functionality

핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

```
Print base-class and derived-class objects:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Calling print with base-class pointer to
base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06
```

(Continued at top of next slide…)

핸들의 자료형에 의한 함수 호출 예제 (without polymorphism)

(…Continued from bottom of previous slide)

```
Calling print with derived-class pointer to
derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Calling print with base-class pointer to derived-class object
invokes base-class print function on that derived-class object:

commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
```

*

print() 함수를 virtual로 선언했을 때와 아닐 때의 차이점(위에서 * 부분)을 확인할 것

파생 클래스의 포인터가 기본 클래스의 객체를 가리킬 때

- ✓ C++ 컴파일러에서 에러 생성
  - CommissionEmployee (기본 클래스 객체) is not a BasePlusCommissionEmployee (파생 클래스 객체)
- ✓ 만약 이것이 허용된다면, 프로그래머는 존재하지 않는 파생 클래스의 멤버에 접근할 것이다.
  - 전혀 엉뚱한 데이터에 사용되는 메모리 내용을 변경하는 오류가 발생 가능

예제 (파생 클래스의 포인터가 기본 클래스의 객체를 가리킬 때)

```
1  // Fig. 13.6: fig13_06.cpp
2  // Aiming a derived-class pointer at a base-class object.
3  #include "CommissionEmployee.h"
4  #include "BasePlusCommissionEmployee.h"
5
6  int main()
7  {
8     CommissionEmployee commissionEmployee(
9        "Sue", "Jones", "222-22-2222", 10000, .06 );
10    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
11
12    // aim derived-class pointer at base-class object
13    // Error: a CommissionEmployee is not a BasePlusCommissionEmployee
14    basePlusCommissionEmployeePtr = &commissionEmployee;
15    return 0;
16 } // end main
```

Cannot assign base-class object to derived-class pointer because *is-a* relationship does not apply

Microsoft Visual C++.NET compiler error messages:

```
C:\cpphtp5_examples\ch13\Fig13_06\fig13_06.cpp(14) : error C2440:
   '=' : cannot convert from 'CommissionEmployee *__w64 ' to
   'BasePlusCommissionEmployee *'
         Cast from base to derived requires dynamic_cast or static_cast
```

13주차 1교시 수업을 마칩니다.

Object-Oriented Programming II

객체지향프로그래밍 II

Lecture 6

제13장 클래스 다형성 (Part 2)

1. (계속) 핸들의 자료형에 의한 멤버 함수 호출 (without Polymorphism)
2. Polymorphism 구현 (가상 함수에 의한 동적 바인딩)

박인규 교수

1. 핸들의 자료형에 의한 멤버함수 호출

(without Polymorphism)

기본클래스의 포인터가 파생클래스 객체를 가리킬 때

- ✓ 기본 클래스에 존재하는 함수를 호출하면 기본 클래스의 기능이 수행
- ✓ 파생 클래스에만 존재하는 함수를 호출하면 컴파일 에러 발생

  - 기본적으로 파생 클래스의 멤버는 기본 클래스의 객체 포인터로부터 접근 할 수 없다.
  - 억지로 downcasting을 이용하면 가능 (Section 13.8)

기본 클래스의 핸들로 파생 클래스에만 존재하는 함수 호출 사례 (오류)

```
7  int main()
8  {
9     CommissionEmployee *commissionEmployeePtr = 0; // base class
10    BasePlusCommissionEmployee basePlusCommissionEmployee(
11       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 ); // derived class
12
13    // aim base-class pointer at derived-class object
14    commissionEmployeePtr = &basePlusCommissionEmployee;
15
16    // invoke base-class member functions on derived-class
17    // object through base-class pointer
18    string firstName = commissionEmployeePtr->getFirstName();
19    string lastName = commissionEmployeePtr->getLastName();
20    string ssn = commissionEmployeePtr->getSocialSecurityNumber();
21    double grossSales = commissionEmployeePtr->getGrossSales();
22    double commissionRate = commissionEmployeePtr->getCommissionRate();
23
24    // attempt to invoke derived-class-only member functions
25    // on derived-class object through base-class pointer
26    double baseSalary = commissionEmployeePtr->getBaseSalary();
27    commissionEmployeePtr->setBaseSalary( 500 );
28    return 0;
29 } // end main
```

Cannot invoke derived-class-only members from base-class pointer

기본 클래스의 핸들로 파생 클래스에만 존재하는 함수 호출 사례 (오류)

Microsoft Visual C++.NET compiler error messages:

```
C:\cpphtp5_examples\ch13\Fig13_07\fig13_07.cpp(26) : error C2039:
   'getBaseSalary' : is not a member of 'CommissionEmployee'
      C:\cpphtp5_examples\ch13\Fig13_07\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
C:\cpphtp5_examples\ch13\Fig13_07\fig13_07.cpp(27) : error C2039:
   'setBaseSalary' : is not a member of 'CommissionEmployee'
      C:\cpphtp5_examples\ch13\Fig13_07\CommissionEmployee.h(10) :
         see declaration of 'CommissionEmployee'
```

2. Polymorphism 구현

(가상 함수에 의한 동적 바인딩)

가상 함수 (Virtual Function) 사용에 의한 동적 바인딩

- 일반적인 경우 (without virtual function)
  - ✓ 핸들(handle)이 어떠한 클래스 함수를 호출할지 결정한다
- 가상 함수 사용의 경우
  - ✓ 핸들의 자료형이 아닌, 가리키고 있는 객체의 실제 자료형이 가상 함수의 어떤 구현을 사용할 지 결정하게 된다.
  - ✓ 프로그램이 동적으로 (컴파일 시간이 아닌 실행시간에) 어떤 클래스의 함수를 사용할 지 결정한다.
    - 동적 바인딩 (dynamic binding) 이라고 함

가상 함수 (Virtual Function) 선언

- ✓ 기본 클래스 선언에서 virtual 키워드를 함수 원형 앞에 붙임으로써 가상함수를 선언
  - 예) virtual void print()
- ✓ 파생 클래스는 기본 클래스의 가상 함수를 override 한다.
- ✓ 기본 클래스의 가상 함수는, 모든 계층의 파생 클래스에서 역시 virtual 함수이다.

정적 바인딩과 동적 바인딩

- 정적 바인딩 (static binding)
  - ✓ 특정 객체가 dot operator를 사용하여 멤버 함수를 호출하면, virtual 여부에 관계없이 호출된 함수는 (핸들에 의해) 컴파일 시간에 결정된다.

- 동적 바인딩 (dynamic binding)
  - ✓ 가상함수의 동적 바인딩은 포인터 또는 참조형의 핸들에서만 이루어진다.

동적 바인딩 예제 (CommissionEmployee.h)

```
1  // Fig. 13.8: CommissionEmployee.h
2  // CommissionEmployee class definition represents a commission employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include <string> // C++ standard string class
7  using std::string;
8
9  class CommissionEmployee
10 {
11 public:
12    CommissionEmployee( const string &, const string &, const string &,
13       double = 0.0, double = 0.0 );
14
15    void setFirstName( const string & ); // set first name
16    string getFirstName() const; // return first name
17
18    void setLastName( const string & ); // set last name
19    string getLastName() const; // return last name
20
21    void setSocialSecurityNumber( const string & ); // set SSN
22    string getSocialSecurityNumber() const; // return SSN
23
24    void setGrossSales( double ); // set gross sales amount
25    double getGrossSales() const; // return gross sales amount
```

동적 바인딩 예제 (CommissionEmployee.h)

```
26
27    void setCommissionRate( double ); // set commission rate
28    double getCommissionRate() const; // return commission rate
29
30    virtual double earnings() const; // calculate earnings
31    virtual void print() const; // print CommissionEmployee object
32 private:
33    string firstName;
34    string lastName;
35    string socialSecurityNumber;
36    double grossSales; // gross weekly sales
37    double commissionRate; // commission percentage
38 }; // end class CommissionEmployee
39
40 #endif
```

Declaring **earnings** and **print** as **virtual** allows them to be overridden, not redefined

동적 바인딩 예제 (BasePlusCommissionEmployee.h)

```
4  #ifndef BASEPLUS_H
5  #define BASEPLUS_H
6
7  #include <string> // C++ standard string class
8  using std::string;
9
10 #include "CommissionEmployee.h" // CommissionEmployee class declaration
11
12 class BasePlusCommissionEmployee : public CommissionEmployee
13 {
14 public:
15    BasePlusCommissionEmployee( const string &, const string &,
16       const string &, double = 0.0, double = 0.0, double = 0.0 );
17
18    void setBaseSalary( double ); // set base salary
19    double getBaseSalary() const; // return base salary
20
21    virtual double earnings() const; // calculate earnings
22    virtual void print() const; // print BasePlusCommissionEmployee object
23 private:
24    double baseSalary; // base salary
25 }; // end class BasePlusCommissionEmployee
26
27 #endif
```

Functions **`earnings`** and **`print`** are already **`virtual`** – good practice to declare **`virtual`** even when overriding function

동적 바인딩 예제 (driver)

```
3  #include <iostream>
4  using std::cout;
5  using std::endl;
6  using std::fixed;
7
8  #include <iomanip>
9  using std::setprecision;
10
11 // include class definitions
12 #include "CommissionEmployee.h"
13 #include "BasePlusCommissionEmployee.h"
14
15 int main()
16 {
17    // create base-class object
18    CommissionEmployee commissionEmployee(
19       "Sue", "Jones", "222-22-2222", 10000, .06 );
20
21    // create base-class pointer
22    CommissionEmployee *commissionEmployeePtr = 0;
23
24    // create derived-class object
25    BasePlusCommissionEmployee basePlusCommissionEmployee(
26       "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
27
28    // create derived-class pointer
29    BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
```

동적 바인딩 예제 (driver)

```
30
31    // set floating-point output formatting
32    cout << fixed << setprecision( 2 );
33
34    // output objects using static binding
35    cout << "Invoking print function on base-class and derived-class "
36       << "\nobjects with static binding\n\n";
37    commissionEmployee.print(); // static binding
38    cout << "\n\n";
39    basePlusCommissionEmployee.print(); // static binding
40
41    // output objects using dynamic binding
42    cout << "\n\n\nInvoking print function on base-class and "
43       << "derived-class \nobjects with dynamic binding";
44
45    // aim base-class pointer at base-class object and print
46    commissionEmployeePtr = &commissionEmployee;
47    cout << "\n\nCalling virtual function print with base-class pointer"
48       << "\nto base-class object invokes base-class "
49       << "print function:\n\n";
50    commissionEmployeePtr->print(); // invokes base-class print
```

Aiming base-class pointer at base-class object and invoking base-class functionality

동적 바인딩 예제 (driver)

```
51
52    // aim derived-class pointer at derived-class object and print
53    basePlusCommissionEmployeePtr = &basePlusCommissionEmployee;
54    cout << "\n\nCalling virtual function print with derived-class "
55       << "pointer\nto derived-class object invokes derived-class "
56       << "print function:\n\n";
57    basePlusCommissionEmployeePtr->print(); // invokes derived-class print
58
59    // aim base-class pointer at derived-class object and print
60    commissionEmployeePtr = &basePlusCommissionEmployee;
61    cout << "\n\nCalling virtual function print with base-class p
62       << "\nto derived-class object invokes derived-class "
63       << "print function:\n\n";
64
65    // polymorphism; invokes BasePlusCommissionEmployee's print;
66    // base-class pointer to derived-class object
67    commissionEmployeePtr->print();
68    cout << endl;
69    return 0;
70 } // end main
```

Aiming derived-class pointer at derived-class object and invoking derived-class functionality

Aiming base-class pointer at derived-class object and invoking derived-class functionality via polymorphism and **`virtual`** functions

동적 바인딩 예제 (실행 결과)

```
Invoking print function on base-class and derived-class
objects with static binding

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00


Invoking print function on base-class and derived-class
objects with dynamic binding

Calling virtual function print with base-class pointer
to base-class object invokes base-class print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

Calling virtual function print with derived-class pointer
to derived-class object invokes derived-class print function:
```

(Coninued at the top of next slide …)

동적 바인딩 예제 (실행 결과)

(…Continued from the bottom of previous slide)

```
base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00

Calling virtual function print with base-class pointer
to derived-class object invokes derived-class print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00
```

13주차 2교시 수업을 마칩니다.

동적 바인딩 예제 (참고자료)

(전역 범위)

```cpp
#include <iostream>

using namespace std;

enum AnimalName { DOG = 1, CAT = 2, PIG = 3, DUCK = 4 };

#define EXIT 5

class Animal
{
public:
    virtual void Speak()
    {
        cout << "동물의 울음소리를 출력하세요. ₩n";
    }

    virtual void Walk()
    {
        cout << "네 발로 걷는다. ₩n";
    }
};

class Dog : public Animal
{
public:
    virtual void Speak()
    {
        cout << "멍멍~~~! ₩n";
    }
};
```

```cpp
class Cat :public Animal
{
public:
    virtual void Speak()
    {
        cout << "야옹~~ ₩n";
    }
};

class Pig : public Animal
{
public:
    virtual void Speak()
    {
        cout << "꿀꿀~~~ ₩n";
    }
};

class Duck : public Animal
{
public:
    virtual void Speak()
    {
        cout << "꽥꽥~~! ₩n";
    }

    virtual void Walk()
    {
        cout << "두 발로 걷는다. ₩n";
    }
};
```

동적 바인딩 예제 (참고자료)

```
int main( )
{
    Animal* pAni = 0;
    int choice;
    while (1)
    {
        cout << "₩n₩n 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit ₩n";
        cout << "Choice : ";
        cin >> choice;
        cout << endl;
        switch (choice)
        {
            case DOG:
                pAni = new Dog;
                break;
            case CAT:
                pAni = new Cat;
                break;
            case PIG:
                pAni = new Pig;
                break;
            case DUCK:
                pAni = new Duck;
                break;
            case EXIT:
                cout << "End ₩n";
                exit(0);
        }

        pAni->Speak( );
        pAni->Walk( );
        delete pAni;
    }
    return 0;
}
```

동적 바인딩 예제 (참고자료)