Object-Oriented Programming II

객체지향프로그래밍 II

Lecture 6

제13장 클래스 다형성 (Part 3)

1. Abstract Classes and Pure Virtual Functions
2. Case Study: Payroll System Using Polymorphism

박인규 교수

복습: 핸들 포인터와 그것이 가리키는 실제 객체와의 관계 요약

- 기본 클래스 포인터가 기본 클래스 객체를 가리킬 때 및 파생 클래스 포인터가 파생 클래스 객체를 가리킬 때
  - ✓ 핸들과 실제 자료가 같은 자료형이므로 특별한 내용 없음
- 기본 클래스 포인터가 파생 클래스 객체를 가리킬 때
  - ✓ (Virtual 함수를 이용하여 Polymorphism을 구현하지 않으면) 정적 바인딩에 의해 기본 클래스의 멤버 함수를 호출함. 즉, 핸들 자료형에 의한 멤버 함수 호출
  - ✓ (Virtual 함수를 이용하여 polymorphism을 구현한다면) 동적 바인딩에 의해 파생 클래스의 멤버 함수를 호출함. 즉, 실제 객체 자료형의 멤버 함수 호출.
- 파생 클래스 포인터가 기본 클래스 객체를 가리킬 때
  - ✓ 컴파일 오류

1. Abstract Classes and Pure virtual Functions

추상 클래스 (Abstract Class)

- ✓ Client가 실제로 객체를 생성하지 않는 (불완전한) 클래스
  - 파생 클래스들의 공통적이고 추상적인 특징만 정의함
    → 파생 클래스가 "빠진 부분"을 정의해야 한다.
  - 실제 객체를 정의 하기에 너무 포괄적이다.
- ✓ 일반적으로 기본 클래스로 사용되며, 추상 기본 클래스(abstract base class) 라고 불린다.
  - 다른 클래스, 즉 구체 클래스 (concrete class)에 상속될 수 있는 적합한 기본 클래스를 제공하기 위해 존재

순수 virtual 함수 (Pure Virtual Function)

- ✓ Pure virtual function을 사용하면 그 클래스는 추상 클래스가 됨

```
Virtual  void  draw() const  = 0;
```

- ✓ 함수 구현을 제공하지 않는다 (위 예제와 같은 원형만 존재)

  - 모든 구체 파생 클래스는 기본 클래스의 모든 pure virtual function을 재정의(override)하고 구체적 구현을 제공해야 함
    - ➢ 그렇지 않으면 파생 클래스 또한 추상 클래스가 된다.

- ✓ 기본 클래스에서 멤버 함수를 구현하는 것이 무의미할 때 사용됨

➡ 실제 구현은 구체화된 파생 클래스에서 이루어 짐

추상 기본 클래스의 다형성에의 이용

- 포인터 또는 참조형 객체를 선언하여 (즉, 객체 생성 없이 핸들만 생성) 추상 기본 클래스를 사용할 수 있다.
  - ✓ 파생된 어떠한 구체 클래스의 객체라도 가리킬 수 있음
  - ✓ 프로그램은 일반적으로 이러한 포인터나 참조형을 이용하여 파생 클래스 객체의 다형성을 이용할 수 있다.

- 다형성은 계층화된 소프트웨어 시스템의 구현에 특히 효율적
  - ✓ 예) 여러 다른 장치(device)에서 데이터를 읽거나 쓸 때

    - 추상 클래스는 공통적인 인터페이스를 pure virtual function으로 제공하고, 실제 입출력은 파생 클래스에서 재정의한 함수가 담당

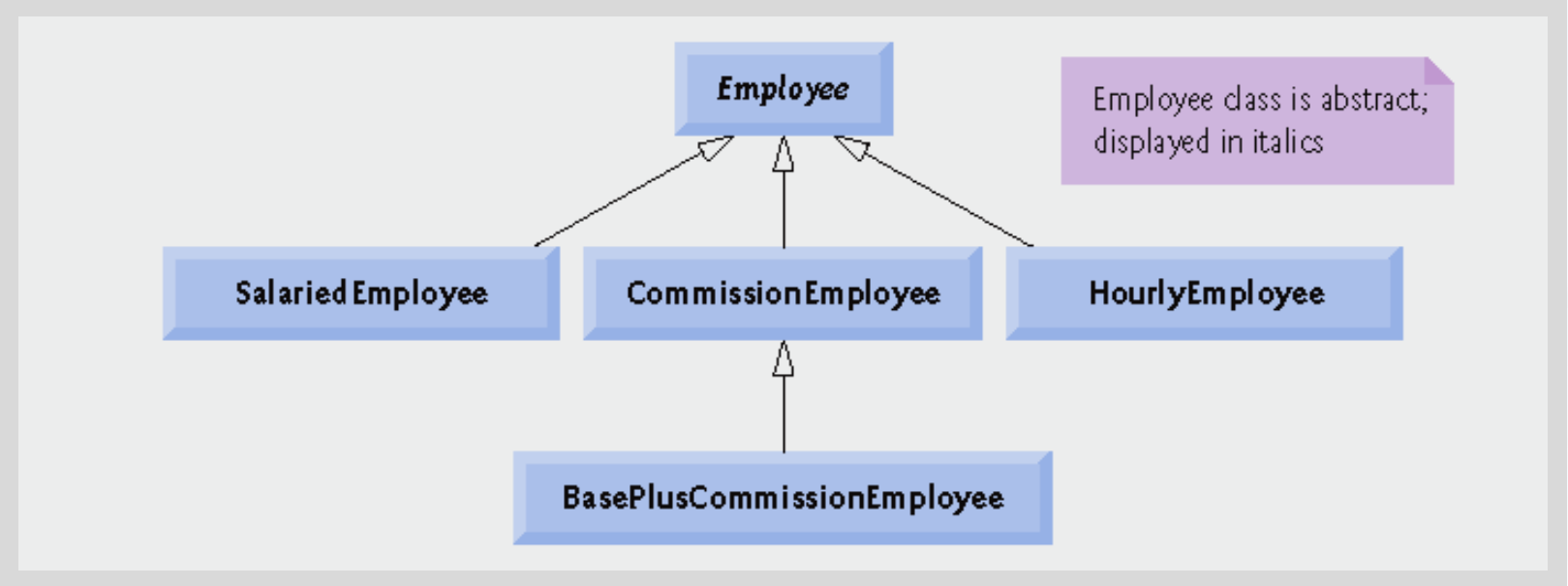
2. Case Study: Payroll System Using Polymorphism

기존 Employee 클래스 계층 구조를 추상 클래스를 이용하여 개선

- ✓ 추상 클래스 Employee 는 일반적인 ‘종업원’의 특징을 표현
  - 계층 구조의 “interface”를 선언
  - 각 종업원의 공통 속성 선언
    - first name, last name, social security number
- ✓ 수입(earnings)의 계산 방법 및 객체 정보의 출력 방법은 구체 파생 클래스마다 모두 다름

Employee 계층구조

- ✓ 실제 구현은 다음 시간으로…



14주차 1교시 수업을 마칩니다.

객체지향프로그래밍 II

Lecture 6

제13장 클래스 다형성 (Part 4)

Case Study: Payroll System Using Polymorphism

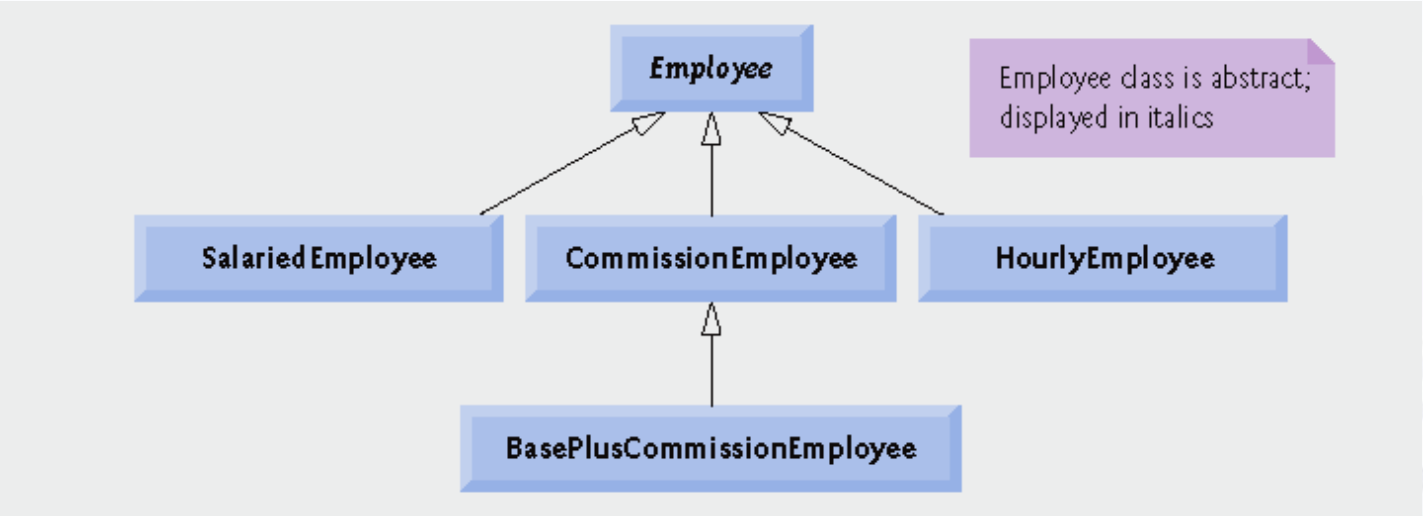
1. Creating Abstract Base Class Employee
2. Creating Concrete Derived Class SalariedEmployee
3. Creating Concrete Derived Class HourlyEmployee
4. Creating Concrete Derived Class CommissionEmployee
5. Indirect Concrete Derived Class BasePlusCommissionEmployee
6. Demonstrating Polymorphic Processing

박인규 교수

1. Creating Abstract Base Class Employee

추상 기본 클래스 Employee

- ✓ 여러 가지 get/set 함수들 제공
- ✓ 함수 earnings() 와 print() 를 선언
  - earnings() 의 구현 방식은 파생 클래스에 따라 각각 다르므로 pure virtual로 선언
    - ➢ Employee 클래스에서 구현하기에는 정보가 불충분함
  - print() 는 가상함수이지만 pure virtual로 선언하지는 않음
    - ➢ Employee 클래스에 default 구현을 가짐 (기본 정보 출력)



Employee 계층 구조에서의 다형적인 (polymorphic) 인터페이스

	earnings	print
Employee	= 0	*firstName lastName* social security number: *SSN*
Salaried-Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *SSN* weekly salary: *weeklysalary*
Hourly-Employee	*If hours <= 40* wage * hours *If hours > 40* (40 * wage) + ((hours - 40) * wage * 1.5)	hourly employee: *firstName lastName* social security number: *SSN* hourly wage: *wage*; hours worked: *hours*
Commission-Employee	commissionRate * grossSales	commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*
BasePlus-Commission-Employee	baseSalary + (commissionRate * grossSales)	base salaried commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales*; commission rate: *commissionRate*; base salary: *baseSalary*

추상 기본 클래스 Employee 구현 (Employee.h)

```
// Fig. 13.13: Employee.h
// Employee abstract base class.
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

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

class Employee
{
public:
   Employee( const string &, const string &, const string & );

   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
```

추상 기본 클래스 Employee 구현 (Employee.h)

```
22
23    // pure virtual function makes Employee abstract base class
24    virtual double earnings() const = 0; // pure virtual
25    virtual void print() const; // virtual
26 private:
27    string firstName;
28    string lastName;
29    string socialSecurityNumber;
30 }; // end class Employee
31
32 #endif // EMPLOYEE_H
```

Function **`earnings`** is pure **`virtual`**, not enough data to provide a default, concrete implementation

Function **`print`** is **`virtual`**, default implementation provided but derived-classes may override

추상 기본 클래스 Employee 구현 (Employee.cpp)

```
// Fig. 13.14: Employee.cpp
// Abstract-base-class Employee member-function definitions.
// Note: No definitions are given for pure virtual functions.
#include <iostream>
using std::cout;

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

// constructor
Employee::Employee( const string &first, const string &last,
   const string &ssn )
   : firstName( first ), lastName( last ), socialSecurityNumber( ssn )
{
   // empty body
} // end Employee constructor

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

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

```

추상 기본 클래스 Employee 구현 (Employee.cpp)

```
29 // set last name
30 void Employee::setLastName( const string &last )
31 {
32    lastName = last;
33 } // end function setLastName
34
35 // return last name
36 string Employee::getLastName() const
37 {
38    return lastName;
39 } // end function getLastName
40
41 // set social security number
42 void Employee::setSocialSecurityNumber( const string &ssn )
43 {
44    socialSecurityNumber = ssn; // should validate
45 } // end function setSocialSecurityNumber
46
47 // return social security number
48 string Employee::getSocialSecurityNumber() const
49 {
50    return socialSecurityNumber;
51 } // end function getSocialSecurityNumber
52
53 // print Employee's information (virtual, but not pure virtual)
54 void Employee::print() const
55 {
56    cout << getFirstName() << ' ' << getLastName()
57       << "\nsocial security number: " << getSocialSecurityNumber();
58 } // end function print
```

2. Creating Concrete Derived Class

SalariedEmployee

Employee에서 파생된 SalariedEmployee 클래스

- ✓ 주급 (weekly salary)을 포함
  - 주급을 반영하기 위해 earnings() 를 재정의
  - 역시 주급을 출력하기 위해 print() 를 재정의
- ✓ SalariedEmployee 는 구체 클래스임
  - 추상 기본 클래스(Employee)의 모든 pure virtual function을 구현함

SalariedEmployee 클래스 구현 (SalariedEmployee.h)

```
// Fig. 13.15: SalariedEmployee.h
// SalariedEmployee class derived from Employee.
#ifndef SALARIED_H
#define SALARIED_H

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

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

   void setWeeklySalary( double ); // set weekly salary
   double getWeeklySalary() const; // return weekly salary

   // keyword virtual signals intent to override
   virtual double earnings() const; // calculate earnings
   virtual void print() const; // print SalariedEmployee object
private:
   double weeklySalary; // salary per week
}; // end class SalariedEmployee

#endif // SALARIED_H
```

SalariedEmployee inherits from **Employee**, must override **earnings** to be concrete

Functions will be overridden (or defined for the first time)

SalariedEmployee 클래스 구현 (SalariedEmployee.cpp)

```
// Fig. 13.16: SalariedEmployee.cpp
// SalariedEmployee class member-function definitions.
#include <iostream>
using std::cout;

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

// constructor
SalariedEmployee::SalariedEmployee( const string &first,
   const string &last, const string &ssn, double salary )
   : Employee( first, last, ssn )
{
   setWeeklySalary( salary );
} // end SalariedEmployee constructor

// set salary
void SalariedEmployee::setWeeklySalary( double salary )
{
   weeklySalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setWeeklySalary

// return salary
double SalariedEmployee::getWeeklySalary() const
{
   return weeklySalary;
} // end function getWeeklySalary
```

Maintain new data member **weeklySalary**

SalariedEmployee 클래스 구현 (SalariedEmployee.cpp)

```
// calculate earnings;
// override pure virtual function earnings in Employee
double SalariedEmployee::earnings() const
{
   return getWeeklySalary();
} // end function earnings

// print SalariedEmployee's information
void SalariedEmployee::print() const
{
   cout << "salaried employee: ";
   Employee::print(); // reuse abstract base-class print function
   cout << "\nweekly salary: " << getWeeklySalary();
} // end function print
```

Overridden `earnings` and `print` functions incorporate weekly salary

3. Creating Concrete Derived Class

HourlyEmployee

Employee에서 파생된 HourlyEmployee 클래스

- 시급(hourly salary)과 노동 시간을 포함
  - 이에 의해 수입을 계산하기 위해 earnings() 를 재정의
  - 역시 이 정보들을 출력하기 위해 print() 를 재정의
- SalariedEmployee 는 구체 클래스임
  - 추상 기본 클래스(Employee)의 모든 pure virtual function을 구현함

HourlyEmployee 클래스 구현 (HourlyEmployee.h)

```
// Fig. 13.17: HourlyEmployee.h
// HourlyEmployee class definition.
#ifndef HOURLY_H
#define HOURLY_H

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

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

   void setWage( double ); // set hourly wage
   double getWage() const; // return hourly wage

   void setHours( double ); // set hours worked
   double getHours() const; // return hours worked

   // keyword virtual signals intent to override
   virtual double earnings() const; // calculate earnings
   virtual void print() const; // print HourlyEmployee object
private:
   double wage; // wage per hour
   double hours; // hours worked for week
}; // end class HourlyEmployee

#endif // HOURLY_H
```

HourlyEmployee inherits from **Employee**, must override **earnings** to be concrete

Functions will be overridden (or defined for first time)

HourlyEmployee 클래스 구현 (HourlyEmployee.cpp)

```
// Fig. 13.18: HourlyEmployee.cpp
// HourlyEmployee class member-function definitions.
#include <iostream>
using std::cout;

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

// constructor
HourlyEmployee::HourlyEmployee( const string &first, const string &last,
   const string &ssn, double hourlyWage, double hoursWorked )
   : Employee( first, last, ssn )
{
   setWage( hourlyWage ); // validate hourly wage
   setHours( hoursWorked ); // validate hours worked
} // end HourlyEmployee constructor

// set wage
void HourlyEmployee::setWage( double hourlyWage )
{
   wage = ( hourlyWage < 0.0 ? 0.0 : hourlyWage );
} // end function setWage

// return wage
double HourlyEmployee::getWage() const
{
   return wage;
} // end function getWage
```

Maintain new data member, **`hourlyWage`**

HourlyEmployee 클래스 구현 (HourlyEmployee.cpp)

```
28
29 // set hours worked
30 void HourlyEmployee::setHours( double hoursWorked )
31 {
32    hours = ( ( ( hoursWorked >= 0.0 ) && ( hoursWorked <= 168.0 ) ) ?
33       hoursWorked : 0.0 );
34 } // end function setHours
35
36 // return hours worked
37 double HourlyEmployee::getHours() const
38 {
39    return hours;
40 } // end function getHours
41
42 // calculate earnings;
43 // override pure virtual function earnings in Employee
44 double HourlyEmployee::earnings() const
45 {
46    if ( getHours() <= 40 ) // no overtime
47       return getWage() * getHours();
48    else
49       return 40 * getWage() + ( ( getHours() - 40 ) * getWage() * 1.5 );
50 } // end function earnings
51
52 // print HourlyEmployee's information
53 void HourlyEmployee::print() const
54 {
55    cout << "hourly employee: ";
56    Employee::print(); // code reuse
57    cout << "\nhourly wage: " << getWage() <<
58       "; hours worked: " << getHours();
59 } // end function print
```

Maintain new data member, **hoursWorked**

Overridden **earnings** and **print** functions incorporate wage and hours

4. Creating Concrete Derived Class

CommissionEmployee

Employee에서 파생된 CommissionEmployee 클래스

- 총매출(gross sales)과 commission rate를 포함
  - 이에 의해 수입을 계산하기 위해 earnings() 를 재정의
  - 역시 이 정보들을 출력하기 위해 print() 를 재정의
- CommissionEmployee 는 구체 클래스임
  - 추상 기본 클래스(Employee)의 모든 pure virtual function을 구현함

CommissionEmployee 클래스 구현 (CommissionEmployee.h)

```
1  // Fig. 13.19: CommissionEmployee.h
2  // CommissionEmployee class derived from Employee.
3  #ifndef COMMISSION_H
4  #define COMMISSION_H
5
6  #include "Employee.h" // Employee class definition
7
8  class CommissionEmployee : public Employee
9  {
10 public:
11    CommissionEmployee( const string &, const string &,
12       const string &, double = 0.0, double = 0.0 );
13
14    void setCommissionRate( double ); // set commission rate
15    double getCommissionRate() const; // return commission rate
16
17    void setGrossSales( double ); // set gross sales amount
18    double getGrossSales() const; // return gross sales amount
19
20    // keyword virtual signals intent to override
21    virtual double earnings() const; // calculate earnings
22    virtual void print() const; // print CommissionEmployee object
23 private:
24    double grossSales; // gross weekly sales
25    double commissionRate; // commission percentage
26 }; // end class CommissionEmployee
27
28 #endif // COMMISSION_H
```

CommissionEmployee inherits from **Employee**, must override **earnings** to be concrete

Functions will be overridden (or defined for first time)

CommissionEmployee 클래스 구현 (CommissionEmployee.cpp)

```
// Fig. 13.20: CommissionEmployee.cpp
// CommissionEmployee class member-function definitions.
#include <iostream>
using std::cout;

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

// constructor
CommissionEmployee::CommissionEmployee( const string &first,
   const string &last, const string &ssn, double sales, double rate )
   : Employee( first, last, ssn )
{
   setGrossSales( sales );
   setCommissionRate( rate );
} // end CommissionEmployee constructor

// 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
```

Maintain new data member, **commissionRate**

CommissionEmployee 클래스 구현 (CommissionEmployee.cpp)

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

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

// calculate earnings;
// override pure virtual function earnings in Employee
double CommissionEmployee::earnings() const
{
   return getCommissionRate() * getGrossSales();
} // end function earnings

// print CommissionEmployee's information
void CommissionEmployee::print() const
{
   cout << "commission employee: ";
   Employee::print(); // code reuse
   cout << "\ngross sales: " << getGrossSales()
      << "; commission rate: " << getCommissionRate();
} // end function print
```

Maintain new data member, **grossSales**

Overridden **earnings** and **print** functions incorporate commission rate and gross sales

5. Indirect Concrete Derived Class

BasePlusCommissionEmployee

BasePlusCommissionEmployee는 CommissionEmployee에서 파생

- 기본급 (base salary)을 포함
  - 이에 의해 수입을 계산하기 위해 earnings() 를 재정의
  - 역시 이 정보를 출력하기 위해 print() 를 재정의
- BasePlusCommissionEmployee 는 구체 클래스임
  - 구체 클래스가 되기 위해 earnings() 를 재정의 한 것이 아님
    - ← CommissionEmployee로부터 상속받을 수 있음
    - 그러나 예제에서는 기본급을 반영하기 위해 재정의했음

BasePlusCommissionEmployee 클래스 구현 (.h)

```
1  // Fig. 13.21: BasePlusCommissionEmployee.h
2  // BasePlusCommissionEmployee class derived from Employee.
3  #ifndef BASEPLUS_H
4  #define BASEPLUS_H
5
6  #include "CommissionEmployee.h" // CommissionEmployee class definition
7
8  class BasePlusCommissionEmployee : public CommissionEmployee
9  {
10 public:
11    BasePlusCommissionEmployee( const string &, const string &,
12       const string &, double = 0.0, double = 0.0, double = 0.0 );
13
14    void setBaseSalary( double ); // set base salary
15    double getBaseSalary() const; // return base salary
16
17    // keyword virtual signals intent to override
18    virtual double earnings() const; // calculate earnings
19    virtual void print() const; // print BasePlusCommissionEmployee object
20 private:
21    double baseSalary; // base salary per week
22 }; // end class BasePlusCommissionEmployee
23
24 #endif // BASEPLUS_H
```

BasePlusCommissionEmployee inherits from **CommissionEmployee**, already concrete

Functions will be overridden

BasePlusCommissionEmployee 클래스 구현 (.cpp)

```
// Fig. 13.22: BasePlusCommissionEmployee.cpp
// 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 )
   : 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
```

Maintain new data member, **baseSalary**

BasePlusCommissionEmployee 클래스 구현 (.cpp)

```
29
30 // calculate earnings;
31 // override pure virtual function earnings in Employee
32 double BasePlusCommissionEmployee::earnings() const
33 {
34     return getBaseSalary() + CommissionEmployee::earnings();
35 } // end function earnings
36
37 // print BasePlusCommissionEmployee's information
38 void BasePlusCommissionEmployee::print() const
39 {
40    cout << "base-salaried ";
41    CommissionEmployee::print(); // code reuse
42    cout << "; base salary: " << getBaseSalary();
43 } // end function print
```

Overridden **earnings** and **print** functions incorporate base salary

6. Demonstrating Polymorphic Processing

Employee 클래스 계층구조 테스트

- ✓ 정적 바인딩 (static binding)을 테스트
  - 포인터 또는 참조형 핸들 대신 이름 핸들을 이용
  - 컴파일러는 어떤 함수가 이용되는지 결정할 수 있다.
    - ➢ 즉, 핸들의 자료형의 멤버함수
- ✓ 동적 바인딩 (dynamic binding)을 이용하여 다형성을 테스트
  - Employee 의 포인터 형 객체의 배열을 이용
  - 포인터를 핸들로 이용하여 가상 함수를 호출

Employee 클래스 계층구조 테스트 예제

```
// Fig. 13.23: fig13_23.cpp
// Processing Employee derived-class objects individually
// and polymorphically using dynamic binding.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

#include <vector>
using std::vector;

// include definitions of classes in Employee hierarchy
#include "Employee.h"
#include "SalariedEmployee.h"
#include "HourlyEmployee.h"
#include "CommissionEmployee.h"
#include "BasePlusCommissionEmployee.h"

void virtualViaPointer( const Employee * const ); // prototype
void virtualViaReference( const Employee & ); // prototype
```

Employee 클래스 계층구조 테스트 예제

```
24
25 int main()
26 {
27    // set floating-point output formatting
28    cout << fixed << setprecision( 2 );
29
30    // create derived-class objects
31    SalariedEmployee salariedEmployee(
32       "John", "Smith", "111-11-1111", 800 );
33    HourlyEmployee hourlyEmployee(
34       "Karen", "Price", "222-22-2222", 16.75, 40 );
35    CommissionEmployee commissionEmployee(
36       "Sue", "Jones", "333-33-3333", 10000, .06 );
37    BasePlusCommissionEmployee basePlusCommissionEmployee(
38       "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );
39
40    cout << "Employees processed individually using static binding:\n\n";
41
42    // output each Employee's information and earnings using static binding
43    salariedEmployee.print();
44    cout << "\nearned $" << salariedEmployee.earnings() << "\n\n";
45    hourlyEmployee.print();
46    cout << "\nearned $" << hourlyEmployee.earnings() << "\n\n";
47    commissionEmployee.print();
48    cout << "\nearned $" << commissionEmployee.earnings() << "\n\n";
49    basePlusCommissionEmployee.print();
50    cout << "\nearned $" << basePlusCommissionEmployee.earnings()
51       << "\n\n";
```

Using objects (rather than pointers or references) to demonstrate static binding

Employee 클래스 계층구조 테스트 예제

```
52
53    // create vector of four base-class pointers
54    vector < Employee * > employees( 4 );
55
56    // initialize vector with Employees
57    employees[ 0 ] = &salariedEmployee;
58    employees[ 1 ] = &hourlyEmployee;
59    employees[ 2 ] = &commissionEmployee;
60    employees[ 3 ] = &basePlusCommissionEmployee;
61
62    cout << "Employees processed polymorphically via dynamic binding:\n\n";
63
64    // call virtualViaPointer to print each Employee's information
65    // and earnings using dynamic binding
66    cout << "Virtual function calls made off base-class pointers:\n\n";
67
68    for ( size_t i = 0; i < employees.size(); i++ )
69       virtualViaPointer( employees[ i ] );
70
71    // call virtualViaReference to print each Employee's information
72    // and earnings using dynamic binding
73    cout << "Virtual function calls made off base-class references:\n\n";
74
75    for ( size_t i = 0; i < employees.size(); i++ )
76       virtualViaReference( *employees[ i ] ); // note dereferencing
77
78    return 0;
79 } // end main
```

vector of **Employee** pointers, will be used to demonstrate dynamic binding

Demonstrate dynamic binding using first pointers, then references

Employee 클래스 계층구조 테스트 예제

```
// call Employee virtual functions print and earnings off a
// base-class pointer using dynamic binding
void virtualViaPointer( const Employee * const baseClassPtr )
{
   baseClassPtr->print();
   cout << "\nearned $" << baseClassPtr->earnings() << "\n\n";
} // end function virtualViaPointer

// call Employee virtual functions print and earnings off a
// base-class reference using dynamic binding
void virtualViaReference( const Employee &baseClassRef )
{
   baseClassRef.print();
   cout << "\nearned $" << baseClassRef.earnings() << "\n\n";
} // end function virtualViaReference
```

Using references and pointers cause **`virtual`** functions to be invoked polymorphically

Employee 클래스 계층구조 테스트 실행 결과

```
Employees processed individually using static binding:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
earned $500.00
```

(Continued at top of next slide...)

Employee 클래스 계층구조 테스트 실행 결과

(...continued from bottom of previous slide)

```
Employees processed polymorphically using dynamic binding:

Virtual function calls made off base-class pointers:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
earned $500.00
```

(Continued at the top of next slide...)

Employee 클래스 계층구조 테스트 실행 결과

(...Continued from bottom of previous page)

```
Virtual function calls made off base-class references:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: 800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: 16.75; hours worked: 40.00
earned $670.00


commission employee: Sue Jones
social security number: 333-33-3333
gross sales: 10000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: 5000.00; commission rate: 0.04; base salary: 300.00
earned $500.00
```

14주차 2교시 수업을 마칩니다.

동적 바인딩 예제 (참고자료)

```
(전역 범위)
#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";
    }
};
```

```
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)

```
C:₩WINDOWS₩system32₩cmd.exe

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 1

멍멍~~~!
네 발로 걷는다.

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 2

야옹~~
네 발로 걷는다.

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 3

꿀꿀~~~
네 발로 걷는다.

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 4

꽥꽥~~!
두 발로 걷는다.

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 5

End
계속하려면 아무 키나 누르십시오 . . .
```

동적 바인딩 예제 (참고자료)

animal.cpp

Cat

```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() = 0;

    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)

```
C:₩WINDOWS₩system32₩cmd.exe

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 1

멍멍~~~!
네 발로 걷는다.

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 2

야옹~~
네 발로 걷는다.

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 3

꿀꿀~~~
네 발로 걷는다.

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 4

꽥꽥~~!
두 발로 걷는다.

 1.Dog   2.Cat   3.Pig   4. Duck   5.Exit
Choice : 5

End
계속하려면 아무 키나 누르십시오 . . .
```