

# 객체지향프로그래밍 II



## Lecture 3

### 제10장 Class의 이해 및 활용 (Part 2)

#### 1. Composition: Objects as Members of Classes





# 1. Composition: Objects as Members of Classes





## 복합 (Composition)

- ✓ *has-a* 관계라고 표현하기도 함
- ✓ 클래스는 다른 클래스의 객체를 멤버로 가질 수 있다
- ✓ 예시
  - AlarmClock 객체는 Time의 객체를 멤버로 가진다.



# 복합 관계에서 멤버 객체 초기화

- ✓ 멤버 초기화기(member initializers)에서 객체 생성자의 인자를 통해 멤버 객체의 생성자에게 인수를 전달
- ✓ 멤버 객체는 클래스 정의에 선언된 순서대로 생성됨
- ✓ 만약 멤버 초기화기가 제공되지 않는다면
  - 멤버 객체의 디폴트 생성자가 내부적으로 호출된다.

# Composition 예제 (Date.h)

```
1 // Fig. 10.10: Date.h
2 // Date class definition; Member functions defined in Date.cpp
3 #ifndef DATE_H
4 #define DATE_H
5
6 class Date
7 {
8 public:
9     Date( int = 1, int = 1, int = 1900 ); // default constructor
10    void print() const; // print date in month/day/year format
11    ~Date(); // provided to confirm destruction order
12 private:
13     int month; // 1-12 (January-December)
14     int day; // 1-31 based on month
15     int year; // any year
16
17     // utility function to check if day is proper for month and year
18     int checkDay( int ) const;
19 }; // end class Date
20
21 #endif
```

# Composition 예제 (Date.cpp)

```
1 // Fig. 10.11: Date.cpp
2 // Member-function definitions for class Date.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "Date.h" // include Date class definition
8
9 // constructor confirms proper value for month; calls
10 // utility function checkDay to confirm proper value for day
11 Date::Date( int mn, int dy, int yr )
12 {
13     if ( mn > 0 && mn <= 12 ) // validate the month
14         month = mn;
15     else
16     {
17         month = 1; // invalid month set to 1
18         cout << "Invalid month (" << mn << ") set to 1.\n";
19     } // end else
20
21     year = yr; // could validate yr
22     day = checkDay( dy ); // validate the day
23
24     // output Date object to show when its constructor is called
25     cout << "Date object constructor for date ";
26     print();
27     cout << endl;
28 } // end Date constructor
```

# Composition 예제 (Date.cpp)

```
29
30 // print Date object in form month/day/year
31 void Date::print() const
32 {
33     cout << month << '/' << day << '/' << year;
34 } // end function print
35
36 // output Date object to show when its destructor is called
37 Date::~~Date()
38 {
39     cout << "Date object destructor for date ";
40     print();
41     cout << endl;
42 } // end ~Date destructor
```

# Composition 예제 (Date.cpp)

```
43
44 // utility function to confirm proper day value based on
45 // month and year; handles leap years, too
46 int Date::checkDay( int testDay ) const
47 {
48     static const int daysPerMonth[ 13 ] =
49         { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
50
51     // determine whether testDay is valid for specified month
52     if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
53         return testDay;
54
55     // February 29 check for leap year
56     if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
57         ( year % 4 == 0 && year % 100 != 0 ) ) )
58         return testDay;
59
60     cout << "Invalid day (" << testDay << ") set to 1.\n";
61     return 1; // leave object in consistent state if bad value
62 } // end function checkDay
```



# Composition 예제 (Employee.h)

```
1 // Fig. 10.12: Employee.h
2 // Employee class definition.
3 // Member functions defined in Employee.cpp.
4 #ifndef EMPLOYEE_H
5 #define EMPLOYEE_H
6
7 #include "Date.h" // include Date class definition
8
9 class Employee
10 {
11 public:
12     Employee( const char * const, const char * const,
13             const Date &, const Date & );
14     void print() const;
15     ~Employee(); // provided to confirm destruction order
16 private:
17     char firstName[ 25 ];
18     char lastName[ 25 ];
19     const Date birthDate; // composition: member object
20     const Date hireDate; // composition: member object
21 }; // end class Employee
22
23 #endif
```

Parameters to be passed via member initializers to the constructor for class **Date**

**const** objects of class **Date** as members

# Composition 예제 (Employee.cpp)

```
1 // Fig. 10.13: Employee.cpp
2 // Member-function definitions for class Employee.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <cstring> // strlen and strncpy prototypes
8 using std::strlen;
9 using std::strncpy;
10
11 #include "Employee.h" // Employee class definition
12 #include "Date.h" // Date class definition
13
14 // constructor uses member initializer list to pass initializer
15 // values to constructors of member objects birthDate and hireDate
16 // [Note: This invokes the so-called "default copy constructor" which the
17 // C++ compiler provides implicitly.]
18 Employee::Employee( const char * const first, const char * const last,
19     const Date &dateOfBirth, const Date &dateOfHire )
20 : birthDate( dateOfBirth ), // initialize birthDate
21   hireDate( dateOfHire ) // initialize hireDate
22 {
23     // copy first into firstName and be sure that it fits
24     int length = strlen( first );
25     length = ( length < 25 ? length : 24 );
26     strncpy( firstName, first, length );
27     firstName[ length ] = '\0';
```

**Member initializers** that pass arguments to **Date**'s implicit default copy constructor

# Composition 예제 (Employee.cpp)

```
28
29 // copy last into lastName and be sure that it fits
30 length = strlen( last );
31 length = ( length < 25 ? length : 24 );
32 strncpy( lastName, last, length );
33 lastName[ length ] = '\0';
34
35 // output Employee object to show when constructor is called
36 cout << "Employee object constructor: "
37     << firstName << ' ' << lastName << endl;
38 } // end Employee constructor
39
40 // print Employee object
41 void Employee::print() const
42 {
43     cout << lastName << ", " << firstName << "   Hired: ";
44     hireDate.print();
45     cout << "   Birthday: ";
46     birthDate.print();
47     cout << endl;
48 } // end function print
49
50 // output Employee object to show when its destructor is called
51 Employee::~~Employee()
52 {
53     cout << "Employee object destructor: "
54         << lastName << ", " << firstName << endl;
55 } // end ~Employee destructor
```

# Composition 예제 (driver)

```
1 // Fig. 10.14: fig10_14.cpp
2 // Demonstrating composition--an object with member objects.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "Employee.h" // Employee class definition
8
9 int main()
10 {
11     Date birth( 7, 24, 1949 );
12     Date hire( 3, 12, 1988 );
13     Employee manager( "Bob", "Blue", birth, hire );
14
15     cout << endl;
16     manager.print();
17
18     cout << "\nTest Date constructor with invalid values:\n";
19     Date lastDayOff( 14, 35, 1994 ); // invalid month and day
20     cout << endl;
21     return 0;
22 } // end main
```

Passing objects to a host object constructor

# Composition 예제 (실행 결과)

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Employee object constructor: Bob Blue
```

```
Blue, Bob  Hired: 3/12/1988  Birthday: 7/24/1949
```

```
Test Date constructor with invalid values:
Invalid month (14) set to 1.
Invalid day (35) set to 1.
Date object constructor for date 1/1/1994
```

```
Date object destructor for date 1/1/1994
Employee object destructor: Blue, Bob
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
Date object destructor for date 3/12/1988
Date object destructor for date 7/24/1949
```

# 객체지향프로그래밍 II



## Lecture 3

### 제10장 Class의 이해 및 활용 (Part 3)

1. `friend` Functions and `friend` Classes
2. Using `this` Pointer





# 1. friend Functions and friend Classes





## 클래스의 friend function / friend class

- ✓ Class scope의 외부에 정의됨
  - Class의 멤버 함수 아님
- ✓ 그러나 클래스의 멤버에 접근할 수 있음
  - **public** 멤버가 아닌 **private** 멤버에도 접근 가능
- ✓ 독립적인 함수나 다른 클래스가 어떤 클래스의 **friend** 로 선언될 수 있음
- ✓ Friend 클래스간의 직접적인 데이터 멤버 접근이 가능하므로 수행 속도 향상
- ✓ 멤버 함수만으로는 수행하기 힘든 동작을 구현할 때 사용





# 클래스의 friend function / friend class

## 클래스의 friend 함수 선언

- ✓ 클래스 정의에 **friend** 로 시작되는 함수 원형 선언

## 클래스의 friend 클래스 선언

- ✓ 예) ClassTwo 클래스를 ClassOne 클래스의 friend로 선언

- **friend** class ClassTwo;를 ClassOne 클래스 정의에 추가
- ClassTwo 클래스의 모든 멤버 함수는 ClassOne 클래스의 friend가 됨



## Friendship 관계의 특성

- 친구 관계 (Friendship relation)는 허용되는 것 (취득하는 것이 아님)
  - Class B 가 class A의 friend가 되기 위해서, class A는 class B를 friend로 명시적으로 선언해야 함 (A가 B를 허용)
- Friendship relation은 일방적이고, 또한 전이되지 않음
  - Class A가 class B의 friend여도, class B가 Class A의 friend가 되지 않음
    - 즉, 짝사랑 관계
  - Class A가 class B의 friend이고, class B가 class C의 friend여도, class A가 class C의 friend가 되지 않음
    - 즉, 친구의 친구는 저절로 친구가 아님

# Friend function 예제

```
1 // Fig. 10.15: fig10_15.cpp
2 // Friends can access private members of a class.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 // Count class definition
8 class Count
9 {
10     friend void setX( Count &, int ); // friend declaration
11 public:
12     // constructor
13     Count()
14         : x( 0 ) // initialize x to 0
15     {
16         // empty body
17     } // end constructor Count
18
19     // output x
20     void print() const
21     {
22         cout << x << endl;
23     } // end function print
24 private:
25     int x; // data member
26 }; // end class Count
```

**friend** function declaration (can appear anywhere in the class)

# Friend function 예제

```
27
28 // function setX can modify private data of Count
29 // because setX is declared as a friend of Count (line 10)
30 void setX( Count &c, int val )
31 {
32     c.x = val; // allowed because setX is a friend of Count
33 } // end function setX
34
35 int main()
36 {
37     Count counter; // create Count object
38
39     cout << "counter.x after instantiation: ";
40     counter.print();
41
42     setX( counter, 8 ); // set x using a friend function
43     cout << "counter.x after call to setX friend function: ";
44     counter.print();
45     return 0;
46 } // end main
```

friend function can modify **Count's private** data

Calling a **friend** function; note that we pass the **Count** object to the function

```
counter.x after instantiation: 0
counter.x after call to setX friend function: 8
```

# Friend function 예제 (non-friend 함수와의 비교)

```
1 // Fig. 10.16: fig10_16.cpp
2 // Non-friend/non-member functions cannot access private data of a class.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 // Count class definition (note that there is no friendship declaration)
8 class Count
9 {
10 public:
11     // constructor
12     Count()
13         : x( 0 ) // initialize x to 0
14     {
15         // empty body
16     } // end constructor Count
17
18     // output x
19     void print() const
20     {
21         cout << x << endl;
22     } // end function print
23 private:
24     int x; // data member
25 }; // end class Count
```

# Friend function 예제 (non-friend 함수와의 비교)

```
26
27 // function cannotSetX tries to modify private data of Count,
28 // but cannot because the function is not a friend of Count
29 void cannotSetX( Count &c, int val )
30 {
31     c.x = val; // ERROR: cannot access private member in Count
32 } // end function cannotSetX
33
34 int main()
35 {
36     Count counter; // create Count object
37
38     cannotSetX( counter, 3 ); // cannotSetX is not a friend
39     return 0;
40 } // end main
```

Non-**friend** function cannot access the class's **private** data

*Microsoft Visual C++.NET compiler error messages:*

```
C:\cpphttp5_examples\ch10\Fig10_16\fig10_16.cpp(31) : error C2248: 'Count::x'
: cannot access private member declared in class 'Count'
    C:\cpphttp5_examples\ch10\Fig10_16\fig10_16.cpp(24) : see declaration
        of 'Count::x'
    C:\cpphttp5_examples\ch10\Fig10_16\fig10_16.cpp(9) : see declaration
        of 'Count'
```

cannotsetX가 일반 함수인 경우 (friend 함수 아님)



## 2. Using [this](#) Pointer



# this 포인터란 무엇인가?

## this pointer

- ✓ 객체 내부에서 자기 자신을 가리키는 포인터 (self-reference)

## 멤버 함수는 this pointer를 통해 자신이 속한 객체를 안다.

- ✓ 모든 객체는 C++의 키워드인 this 포인터를 통해 자신의 주소에 접근할 수 있다.

- ✓ 객체의 this 포인터는 객체 자신의 일부는 아님

## 객체는 this pointer를 암시적으로(implicitly) 또는 명시적으로(explicitly) 사용

- ✓ 멤버 함수에서 데이터 멤버에 접근할 때 암시적으로 사용

- ✓ this 키워드를 사용하면 명시적으로 사용

- ✓ this 포인터의 타입은 객체 타입에 의해 결정됨



# this Pointer 예제

```
1 // Fig. 10.17: fig10_17.cpp
2 // Using the this pointer to refer to object members.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 class Test
8 {
9 public:
10     Test( int = 0 ); // default constructor
11     void print() const;
12 private:
13     int x;
14 }; // end class Test
15
16 // constructor
17 Test::Test( int value )
18     : x( value ) // initialize x to value
19 {
20     // empty body
21 } // end constructor Test
```

# this Pointer 예제

```
22
23 // print x using implicit and explicit this pointers;
24 // the parentheses around *this are required
25 void Test::print() const
26 {
27     // implicitly use the this pointer to access the member x
28     cout << "          x = " << x;
29
30     // explicitly use the this pointer and the arrow operator
31     // to access the member x
32     cout << "\n this->x = " << this->x;
33
34     // explicitly use the dereferenced this pointer and
35     // the dot operator to access the member x
36     cout << "\n(*this).x = " << ( *this ).x << endl;
37 } // end function print
38
39 int main()
40 {
41     Test testObject( 12 ); // instantiate and initialize testObject
42
43     testObject.print();
44     return 0;
45 } // end main
```

Implicitly using the **this** pointer to access member **x**

Explicitly using the **this** pointer to access member **x**

x = 12  
this->x = 12  
(\*this).x = 12

# HW #5 : Class의 이해 및 활용

❖ 주어진 멤버함수들과 UML 구조를 이용하여 다음 기능을 포함하도록 강의노트 p7(Fig. 10.10)의 클래스 Date를 수정하라.

1. 다음과 같은 다중 형식으로 날짜를 출력한다.

DDD YY

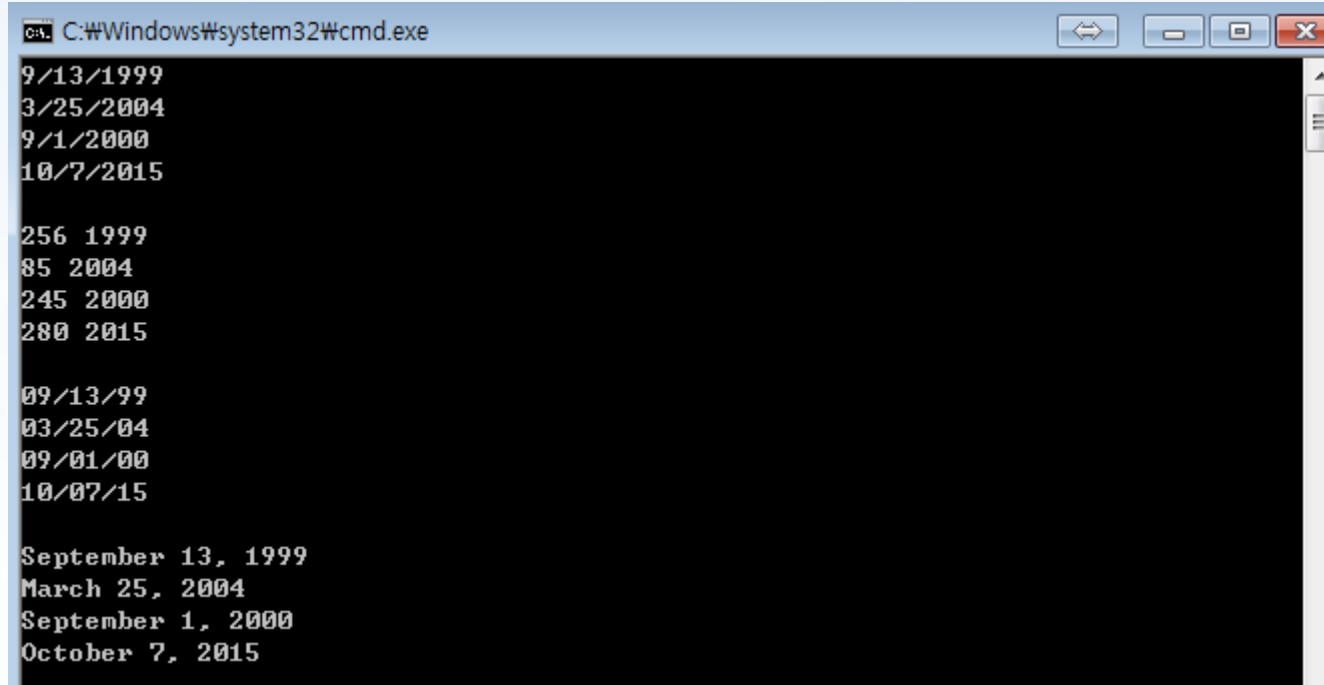
MM/DD/YY

June 14, 1992

2. 위와 같은 형식의 날짜를 이용하여 Date 오브젝트를 생성하고 초기화하기 위해 오버로딩 된 생성자를 사용하라.

3.<ctime> 헤더의 표준 라이브러리 함수를 사용하여 시스템 날짜를 읽고 Data 멤버를 설정하는 Data 생성자를 작성하라. 헤더 <ctime>의 함수들에 대한 정보는 컴파일러의 레퍼런스 문서나 [www.cplusplus.com/ref/ctime/index.html](http://www.cplusplus.com/ref/ctime/index.html) 를 참고하기 바란다.

# HW #5 : Class의 이해 및 활용



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the following text:

```
9/13/1999
3/25/2004
9/1/2000
10/7/2015

256 1999
85 2004
245 2000
280 2015

09/13/99
03/25/04
09/01/00
10/07/15

September 13, 1999
March 25, 2004
September 1, 2000
October 7, 2015
```

❖ **Deadline** : 다음 실습 수업 1초 전까지