

객체지향프로그래밍 응용



Lecture 1

Introduction to Classes and Objects (교과서 3장, Part 3)

1. Initializing Objects with Constructors
2. Separating `.h` and `.cpp` Files





1. Initializing Objects with Constructors



클래스 생성자 (constructor)

☞ 객체가 만들어질 때 객체의 데이터를 초기화 하기 위해 사용되는 함수

- ✓ 객체가 생성될 때 암묵적으로 (implicitly) 호출
- ✓ 반드시 클래스와 같은 이름으로 정의 되어야 한다.
- ✓ 값을 반환하지 않음
 - ➡ 반환형 없음 (`void` 형도 아님)

☞ 기본 생성자 (default constructor)는 매개변수를 가지지 않는다.

- ✓ 클래스가 생성자를 프로그래머가 선언하지 않았다면, 컴파일러가 자동으로 기본 생성자를 제공
 - ➡ 오직 객체 클래스의 데이터 멤버의 생성자만 호출하는 역할

클래스 생성자 예제

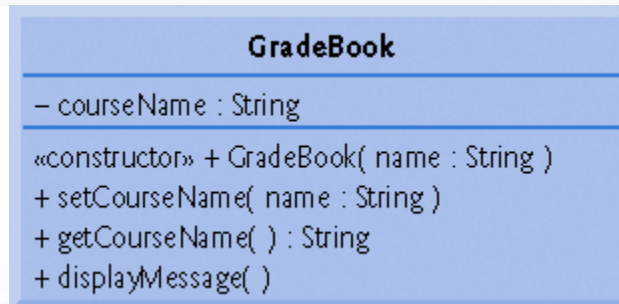
```
1 // Fig. 3.7: fig03_07.cpp
2 // Instantiating multiple objects of the GradeBook class and using
3 // the GradeBook constructor to specify the course name
4 // when each GradeBook object is created.
5 #include <iostream>
6 using std::cout;
7 using std::endl;
8
9 #include <string> // program uses C++ standard string class
10 using std::string;
11
12 // GradeBook class definition
13 class GradeBook
14 {
15 public:
16     // constructor initializes courseName with string supplied as argument
17     GradeBook( string name )
18     {
19         setCourseName( name ); // call set function to initialize courseName
20     } // end GradeBook constructor
21
22     // function to set the course name
23     void setCourseName( string name )
24     {
25         courseName = name; // store the course name in the object
26     } // end function setCourseName
27
```

Constructor has same name as class and no return type

Initialize data member

클래스 생성자 예제

```
28 // function to get the course name
29 string getCourseName()
30 {
31     return courseName; // return object's courseName
32 } // end function getCourseName
33
34 // display a welcome message to the GradeBook user
35 void displayMessage()
36 {
37     // call getCourseName to get the courseName
38     cout << "welcome to the grade book for\n" << getCourseName()
39         << "!" << endl;
40 } // end function displayMessage
41 private:
42     string courseName; // course name for this GradeBook
43 }; // end class GradeBook
44
```



<UML Diagram>

클래스 생성자 예제

```
45 // function main begins program execution
46 int main()
47 {
48     // create two GradeBook objects
49     GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
50     GradeBook gradeBook2( "CS102 Data Structures in C++" );
51
52     // display initial value of courseName for each GradeBook
53     cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
54         << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
55         << endl;
56     return 0; // indicate successful termination
57 } // end main
```

Creating objects implicitly calls the constructor

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```



Error-Prevention Tip 3.2

Unless no initialization of your class's data members is necessary (almost never), provide a constructor to ensure that your class's data members are initialized with meaningful values when each new object of your class is created.

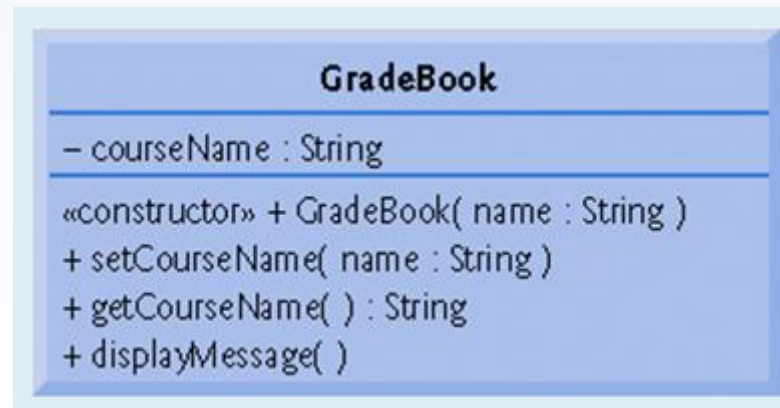


Data members can be initialized in a constructor of the class or their values may be *set* later after the object is created. However, it is a good software engineering practice to ensure that an object is fully initialized before the client code invokes the object's member functions. In general, you should not rely on the client code to ensure that an object gets initialized properly.



UML에서 클래스 생성자 표기

- 세번째 섹션에 등장하며 <<constructor>> 라고 명기하여 일반 멤버함수와 구분
- 일반적으로 멤버함수보다 앞에 위치함





2. Separating `.h` and `.cpp` Files





용도에 따른 파일의 분리

☞ `.cpp` 는 소스 코드 파일로 알려져 있다.

☞ 헤더 파일 (header files)

✓ 클래스 선언부가 위치하며, 소스 파일과 보통 분리됨

- 클래스가 어디에서 사용되더라도 컴파일러가 클래스를 인식할 수 있도록 해준다.

✓ 일반적으로 `.h` 확장자를 가진다.

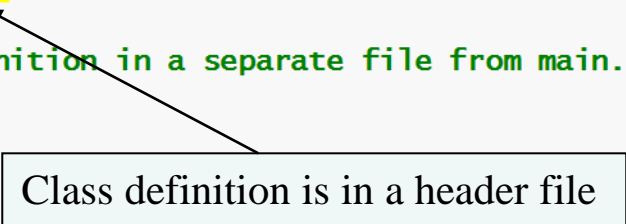
☞ 드라이버 파일 (driver files)

✓ 작성한 클래스나 함수를 운용하기 위해 사용되는 프로그램

✓ 실행 될 수 있게 하기 위해 하나의 `main` 함수를 포함

헤더파일이 소스파일을 포함한 (바람직하지 않은) 예제

```
1 // Fig. 3.9: GradeBook.h
2 // GradeBook class definition in a separate file from main.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include <string> // class GradeBook uses C++ standard string class
8 using std::string;
9
10 // GradeBook class definition
11 class GradeBook
12 {
13 public:
14     // constructor initializes courseName with string supplied as argument
15     GradeBook( string name )
16     {
17         setCourseName( name ); // call set function to initialize courseName
18     } // end GradeBook constructor
19
20     // function to set the course name
21     void setCourseName( string name )
22     {
23         courseName = name; // store the course name in the object
24     } // end function setCourseName
25
```



Class definition is in a header file

헤더파일이 소스파일을 포함한 (바람직하지 않은) 예제

```
26 // function to get the course name
27 string getCourseName()
28 {
29     return courseName; // return object's courseName
30 } // end function getCourseName
31
32 // display a welcome message to the GradeBook user
33 void displayMessage()
34 {
35     // call getCourseName to get the courseName
36     cout << "welcome to the grade book for\n" << getCourseName()
37         << "!" << endl;
38 } // end function displayMessage
39 private:
40     string courseName; // course name for this GradeBook
41 }; // end class GradeBook
```

헤더 파일과 소스 파일을 분리하는 것이 바람직
(다음 시간 강의내용)

드라이버 cpp 파일 예제

```
1 // Fig. 3.10: fig03_10.cpp
2 // Including class GradeBook from file GradeBook.h for use in main.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "GradeBook.h" // include definition of class GradeBook
8
9 // function main begins program execution
10 int main()
11 {
12     // create two GradeBook objects
13     GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
14     GradeBook gradeBook2( "CS102 Data Structures in C++" );
15
16     // display initial value of courseName for each GradeBook
17     cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
18         << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
19         << endl;
20     return 0; // indicate successful termination
21 } // end main
```

Including the header file causes the class definition to be copied into the file

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```

#include 전처리기 지시자

- ☞ 헤더 파일을 포함(include)하기 위해 사용됨
 - ✓ C++ 전처리기는 이 지시자를 헤더파일로 바꿔치기함
- ☞ 사용자가 생성한 헤더파일을 포함할 때는 " "를 이용
 - ✓ 예: `#include "myclass.h"`
- ☞ C++ 표준 라이브러리를 포함할 때는 < >를 이용
 - ✓ 예: `#include <iostream>`



컴파일 과정에서 헤더 파일의 역할

☞ 컴파일러는 객체의 크기를 알아야 한다.

✓ C++ 객체는 일반적으로 데이터 멤버만 포함한다.

✓ 컴파일러는 오직 하나의 클래스 멤버 함수의 복사본을 생성

- 이 복사본은 모든 클래스 객체에서 공유

✓ 예) 4개의 정수형 데이터 멤버와 2개의 멤버 함수를 가진 클래스 객체

- 16바이트 (데이터 멤버)의 크기를 가짐

☞ 컴파일러는 헤더 파일을 통해 객체의 크기를 알아내고 메모리를 할당함.

객체지향프로그래밍 II



Lecture 1

Introduction to Classes and Objects (교과서 3장, Part 4)

1. Separating Interface from Implementation
2. Validating Data with *set* Functions





1. Separating Interface from Implementation





인터페이스 (interface)

- ☞ 사용자가 어떤 (which) 서비스를 사용할 수 있으며, 어떻게 (how) 그 서비스를 요청하는지 정의된 약속

- ✓ 그러나 클래스가 어떻게 (how) 서비스를 수행하는지에 관한 정보는 없음
- ✓ 멤버 함수 이름, 반환형과 매개 변수형 만으로 정의됨
 - ➡ 함수 원형 (function prototype)

- ☞ 클래스의 인터페이스는 클래스의 **public** 멤버 함수로 정의된다.

구현(implementation)과 인터페이스의 분리

☞ 별도의 소스코드 파일에서 클래스 멤버함수를 구현한다.

✓ 클래스의 소스코드 파일

- Binary scope resolution operator (::)를 이용하여 클래스 이름과 멤버 함수의 정의를 연결
- 예: `void GradeBook::displayMessage()`

✓ 구체적인 구현 내용은 숨겨진다.

- 클라이언트 코드는 구현 정보를 필요로 하지 않고, 입출력만 필요로 함

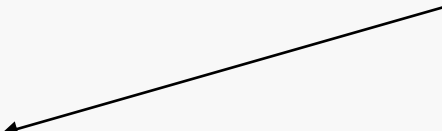
☞ 클래스의 헤더 파일

✓ 멤버 함수의 원형은 클래스의 **public** 인터페이스를 기술함

헤더와 소스 파일의 분리 예제 – 헤더 파일 (클래스 선언)

```
1 // Fig. 3.11: GradeBook.h
2 // GradeBook class definition. This file presents GradeBook's public
3 // interface without revealing the implementations of GradeBook's member
4 // functions, which are defined in GradeBook.cpp.
5 #include <string> // class GradeBook uses C++ standard string class
6 using std::string;
7
8 // GradeBook class definition
9 class GradeBook
10 {
11 public:
12     GradeBook( string ); // constructor that initializes courseName
13     void setCourseName( string ); // function that sets the course name
14     string getCourseName(); // function that gets the course name
15     void displayMessage(); // function that displays a welcome message
16 private:
17     string courseName; // course name for this GradeBook
18 }; // end class GradeBook
```

Interface contains data members
and member function prototypes



헤더와 소스 파일의 분리 예제 - 소스 파일 (멤버함수 구현)

```
1 // Fig. 3.12: GradeBook.cpp
2 // GradeBook member-function definitions. This file contains
3 // implementations of the member functions prototyped in GradeBook.h.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "GradeBook.h" // include definition of class GradeBook
9
10 // constructor initializes courseName with string supplied as arg
11 GradeBook::GradeBook( string name )
12 {
13     setCourseName( name ); // call set function to initialize courseName
14 } // end GradeBook constructor
15
16 // function to set the course name
17 void GradeBook::setCourseName( string name )
18 {
19     courseName = name; // store the course name in the object
20 } // end function setCourseName
21
```

GradeBook implementation is placed in a separate source-code file

Include the header file to access the class name **GradeBook**

Binary scope resolution operator ties a function to its class

헤더와 소스 파일의 분리 예제 - 소스 파일 (멤버함수 구현)

```
22 // function to get the course name
23 string GradeBook::getCourseName()
24 {
25     return courseName; // return object's courseName
26 } // end function getCourseName
27
28 // display a welcome message to the GradeBook user
29 void GradeBook::displayMessage()
30 {
31     // call getCourseName to get the courseName
32     cout << "welcome to the grade book for\n" << getCourseName()
33         << "!" << endl;
34 } // end function displayMessage
```

헤더와 소스 파일의 분리 예제 - 드라이버 파일

```
1 // Fig. 3.13: fig03_13.cpp
2 // GradeBook class demonstration after separating
3 // its interface from its implementation.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "GradeBook.h" // include definition of class GradeBook
9
10 // function main begins program execution
11 int main()
12 {
13     // create two GradeBook objects
14     GradeBook gradeBook1( "CS101 Introduction to C++ Programming" );
15     GradeBook gradeBook2( "CS102 Data Structures in C++" );
16
17     // display initial value of courseName for each GradeBook
18     cout << "gradeBook1 created for course: " << gradeBook1.getCourseName()
19         << "\ngradeBook2 created for course: " << gradeBook2.getCourseName()
20         << endl;
21     return 0; // indicate successful termination
22 } // end main
```

```
gradeBook1 created for course: CS101 Introduction to C++ Programming
gradeBook2 created for course: CS102 Data Structures in C++
```




Common Programming Error 3.8

Forgetting the semicolon at the end of a function prototype is a syntax error.



컴파일과 링크 과정

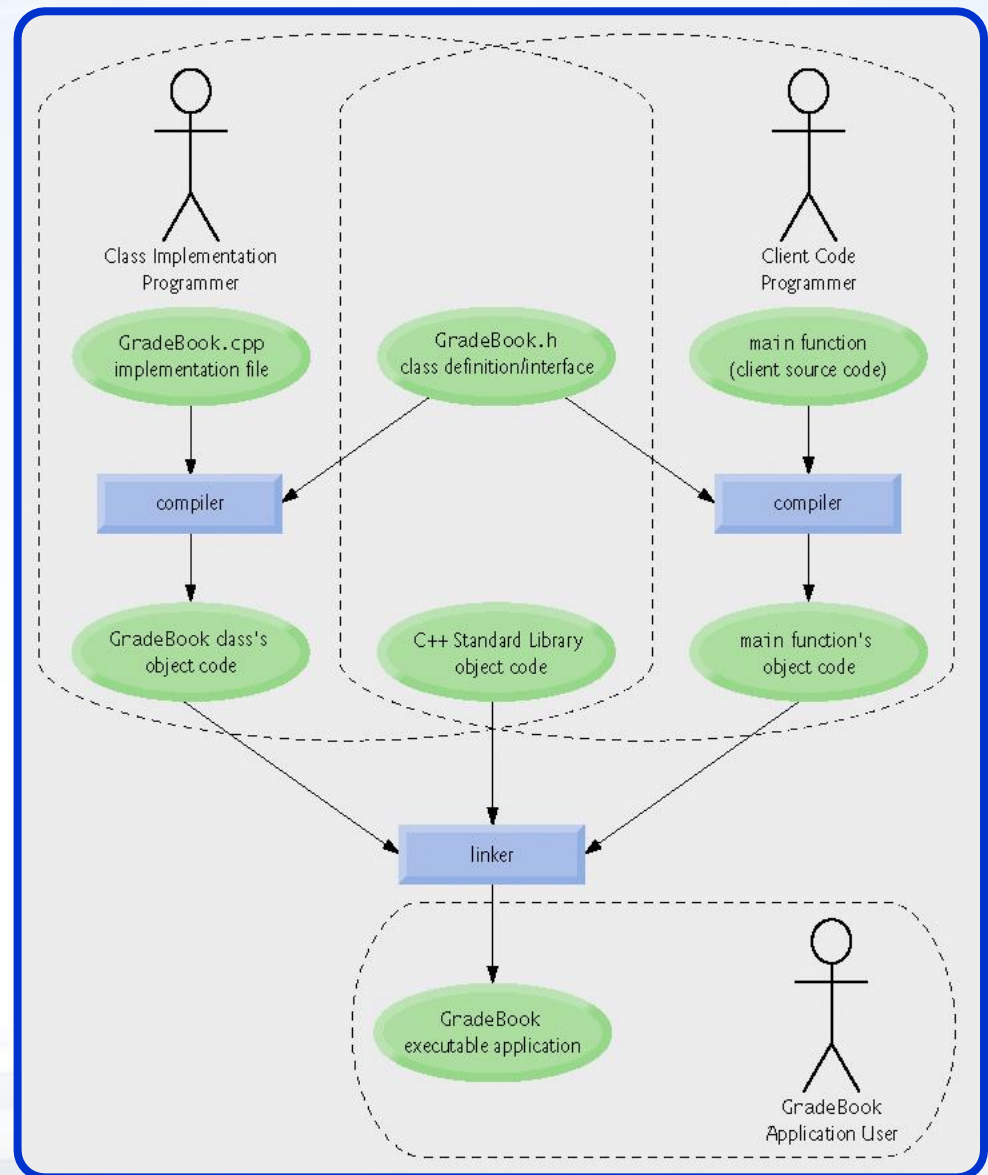
- ☞ 소스코드 파일은 컴파일 되어 클래스 목적 코드(object code)를 생성한다.
 - ✓ 클래스를 구현하는 프로그래머는 헤더파일과 목적 코드를 클라이언트에 제공해 주면 됨 (목적코드는 기계어)
- ☞ 클라이언트는 자신의 코드에 반드시 `#include` "헤더파일" 을 선언했어야 한다.
 - ✓ 그렇지 않으면 컴파일러는 클래스의 존재를 모름
 - ✓ 함수의 원형이 선언된 헤더 파일을 include하는 원리와 같음



컴파일과 링크

☞ 실행 가능한 어플리케이션을 생성하기 위해

- ✓ 클라이언트 목적 코드(.obj)는 반드시 클래스의 목적 코드 및 C++ 표준 라이브러리 목적 코드와 링크되어야 함





2. Validating Data with *set* Functions

private 멤버에 대한 *set* 함수의 용도

☞ *set* 함수는 데이터를 검증(validation) 할 수 있다.

- ✓ 유효성 검사(validity checking)로 알려져 있다.
- ✓ 객체를 안정적인 상태로 유지
 - 데이터 멤버는 유효한 값만을 가짐
- ✓ 유효하지 않은 값으로 설정할 경우, 오류에 해당되는 코드를 리턴할 수 있음

참고- *string* 멤버 함수

- ✓ *length* 는 *string* 내부의 문자 개수를 반환한다.
- ✓ *substr* 는 주어진 *string* 에서 일부를 반환한다.

set 함수를 이용한 private 멤버변수로의 접근 예제

```
1 // Fig. 3.15: GradeBook.h
2 // GradeBook class definition presents the public interface of
3 // the class. Member-function definitions appear in GradeBook.cpp.
4 #include <string> // program uses C++ standard string class
5 using std::string;
6
7 // GradeBook class definition
8 class GradeBook
9 {
10 public:
11     GradeBook( string ); // constructor that initializes a GradeBook object
12     void setCourseName( string ); // function that sets the course name
13     string getCourseName(); // function that gets the course name
14     void displayMessage(); // function that displays a welcome message
15 private:
16     string courseName; // course name for this GradeBook
17 }; // end class GradeBook
```

set 함수를 이용한 private 멤버변수로의 접근 예제

```
1 // Fig. 3.16: GradeBook.cpp
2 // Implementations of the GradeBook member-function definitions.
3 // The setCourseName function performs validation.
4 #include <iostream>
5 using std::cout;
6 using std::endl;
7
8 #include "GradeBook.h" // include definition of class GradeBook
9
10 // constructor initializes courseName with string supplied as argument
11 GradeBook::GradeBook( string name )
12 {
13     setCourseName( name ); // validate and store courseName
14 } // end GradeBook constructor
15
16 // function that sets the course name;
17 // ensures that the course name has at most 25 characters
18 void GradeBook::setCourseName( string name )
19 {
20     if ( name.length() <= 25 ) // if name has 25 or fewer characters
21         courseName = name; // store the course name in the object
22 }
```

Constructor calls *set* function to perform validity checking

set functions perform validity checking to keep *courseName* in a consistent state

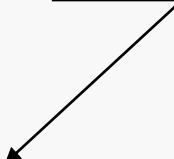
set 함수를 이용한 private 멤버변수로의 접근 예제

```
23  if ( name.length() > 25 ) // if name has more than 25 characters
24  {
25      // set courseName to first 25 characters of parameter name
26      courseName = name.substr( 0, 25 ); // start at 0, length of 25
27
28      cout << "Name \"" << name << "\" exceeds maximum length (25).\n"
29          << "Limiting courseName to first 25 characters.\n" << endl;
30  } // end if
31 } // end function setCourseName
32
33 // function to get the course name
34 string GradeBook::getCourseName()
35 {
36     return courseName; // return object's courseName
37 } // end function getCourseName
38
39 // display a welcome message to the GradeBook user
40 void GradeBook::displayMessage()
41 {
42     // call getCourseName to get the courseName
43     cout << "Welcome to the grade book for\n" << getCourseName()
44         << "!" << endl;
45 } // end function displayMessage
```


set 함수를 이용한 private 멤버변수로의 접근 예제

```
1 // Fig. 3.17: fig03_17.cpp
2 // Create and manipulate a GradeBook object; illustrate validation.
3 #include <iostream>
4 using std::cout;
5 using std::endl;
6
7 #include "GradeBook.h" // include definition of class GradeBook
8
9 // function main begins program execution
10 int main()
11 {
12     // create two GradeBook objects;
13     // initial course name of gradeBook1 is too long
14     GradeBook gradeBook1( "CS101 Introduction to Programming in C++" );
15     GradeBook gradeBook2( "CS102 C++ Data Structures" );
16 }
```

Constructor will call *set* function to perform validity checking



set 함수를 이용한 private 멤버변수로의 접근 예제

```
17 // display each GradeBook's courseName
18 cout << "gradeBook1's initial course name is: "
19     << gradeBook1.getCourseName()
20     << "\ngradeBook2's initial course name is: "
21     << gradeBook2.getCourseName() << endl;
22
23 // modify myGradeBook's courseName (with a valid-length string)
24 gradeBook1.setCourseName( "CS101 C++ Programming" );
25
26 // display each GradeBook's courseName
27 cout << "\ngradeBook1's course name is: "
28     << gradeBook1.getCourseName()
29     << "\ngradeBook2's course name is: "
30     << gradeBook2.getCourseName() << endl;
31 return 0; // indicate successful termination
32 } // end main
```

Call *set* function to perform validity checking

Name "CS101 Introduction to Programming in C++" exceeds maximum length (25).
Limiting courseName to first 25 characters.

gradeBook1's initial course name is: CS101 Introduction to Pro
gradeBook2's initial course name is: CS102 C++ Data Structures

gradeBook1's course name is: CS101 C++ Programming
gradeBook2's course name is: CS102 C++ Data Structures



Making data members **private** and controlling access, especially write access, to those data members through **public** member functions helps ensure data integrity (온전성, 진실성).



Error-Prevention Tip 3.5

The benefits of data integrity are not automatic simply because data members are made **private**—the programmer must provide appropriate validity checking and report the errors.



Member functions that set the values of **private** data should verify that the intended new values are proper; if they are not, the set functions should place the **private** data members into an appropriate state.