



Chapter 10. Classes and Objects

목차

1. Features of Object-Oriented Programming
2. Understanding of the class
3. Constructors and destructors

학습목표

- The understanding of object-oriented programming.
- Defining classes and Learning how to use to create an object.
- Watch out for the members of the class member variables and member functions.
- Study for the access specifier for encapsulation.

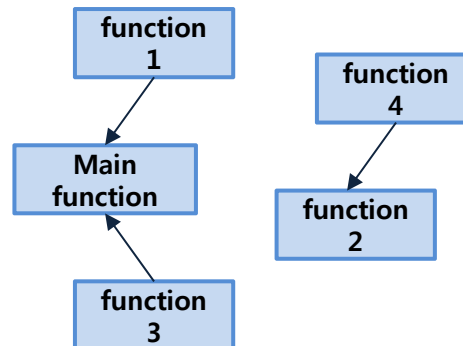
00. Object-oriented programming features.

■ Object-oriented language C ++.

- ① Procedural programming: Since the program is composed of a set of functions used to define the function and the data required for the function declaration. Representative language has C.
- ② Object-Oriented Programming: after designing the class to produce objects by defining a function (user interface), it should cover to cover the object with the function. Representative language may include C ++, Java.

■ Procedural Programming

- A program designed around the function and then define the data required for the function.

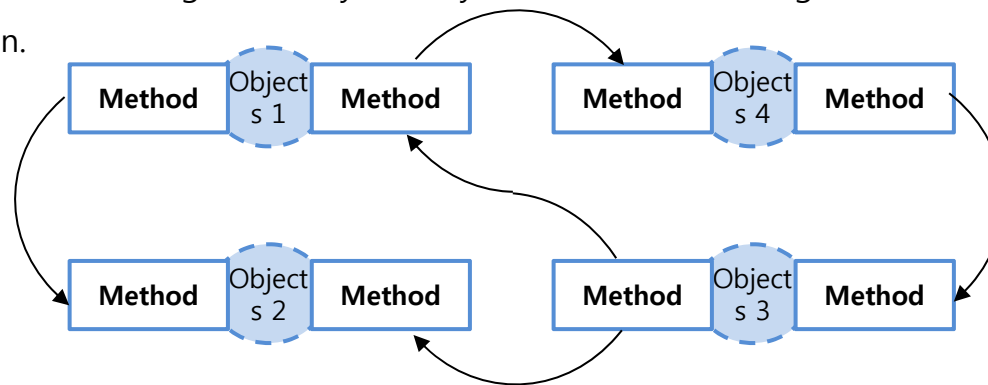


[Picture 1-2] Procedural Programming

00. Object-oriented programming features.

■ Object Oriented Programming

- Program technique for solving various problems by developing a model of the real world as objects.
The object here includes both the operation (process, method, function), the actual (data) associated with that entity.
- Example) For sale to the ticket at the train station for example, the reality of 'guest' and the procedure 'Buy Tickets' is a single object, and one object is also a reality' attendant 'procedure and the' ticket sale
For ticket sale between objects send and receive messages ("KTX Busan, please go single table").
Object receiving the message and executes the operations (steps).
- Object-oriented programming is a technique that promotes modularity, encapsulation of the complicated system rebuilding, with easy-to-easy-to-use software that geodaehwa, filling, easy to maintain direction.



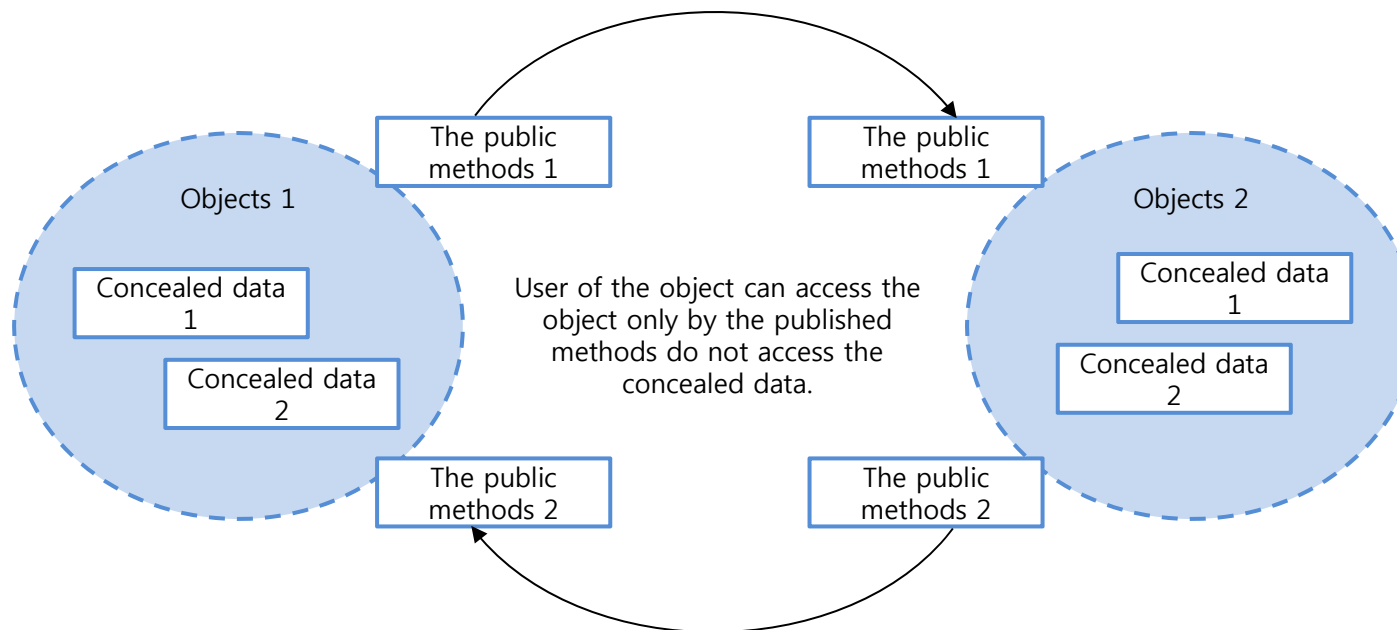
[Picture 1-3] Object Oriented Programming

00. Object-oriented programming features.

■ The main features of object-oriented programming

① Encapsulation

- Poor handling of the data directly, so data may be damaged it is encapsulated and data hiding is provided to prevent this.
- That is, thoroughly hide the details of the object to an object external to simply make the interaction with the object, only the message.



[Picture 1-4] Public functions and concealment of data objects

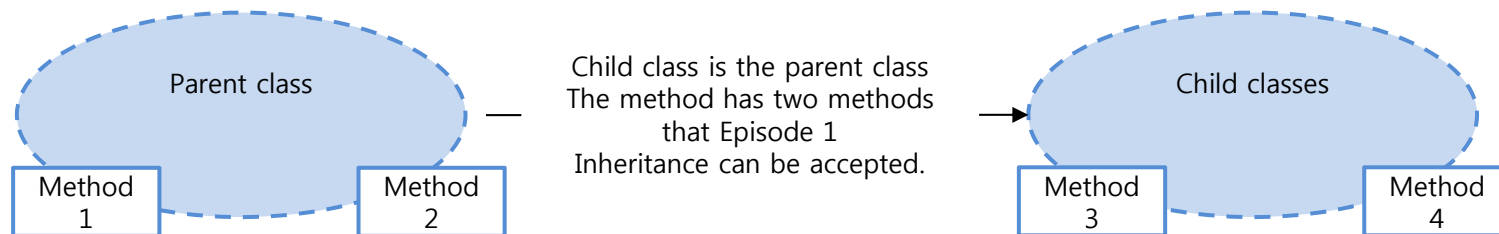
00. Object-oriented programming features.

② Polymorphism

- Polymorphism is to provide a different implementation using a single interface.
- Through overloading or overriding also to operate differently depending on if the function or operator of the same name.

③ Inheritance

- Inheritance is to make available to other objects the nature of certain objects (data and methods) inherit.



01 Understanding of the class

■ The declaration of the class

- The reserved word to declare a class in C ++ is class. class is to abstract the data of the tools that can be implemented in C ++ user-defined data types. Classes are composed into a class declaration and class member function definition.

The class declaration	Class member function definition
<pre>class Class people { Access specifier: Member variable types: Access specifier: Member function types (); };</pre>	<pre>Datatype class member function Name:() { }</pre>

[Table 10-1] Access specifier

division	My current class	Currently out of class
private	o	x
public	o	o

- private : Available only member functions of the class the member belongs to, and is encapsulated (data hiding).
- public : If members of the public to the extent possible approaches that can be used anywhere an object is used to define a member function to be used mainly for private members from outside the class.
- protected: Available by inheritance.

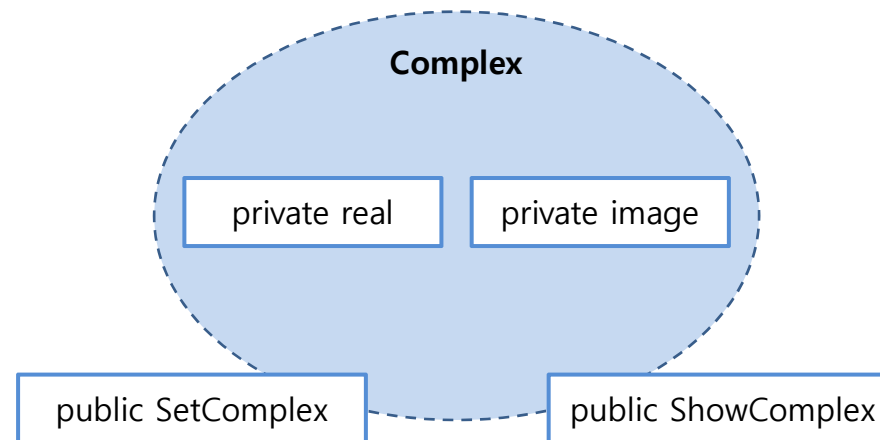
01 Understanding of the class

example)

- Let's design a complex class named Complex.
- This new class data type, and the data is a member variable, its functions to process the data are the member functions.

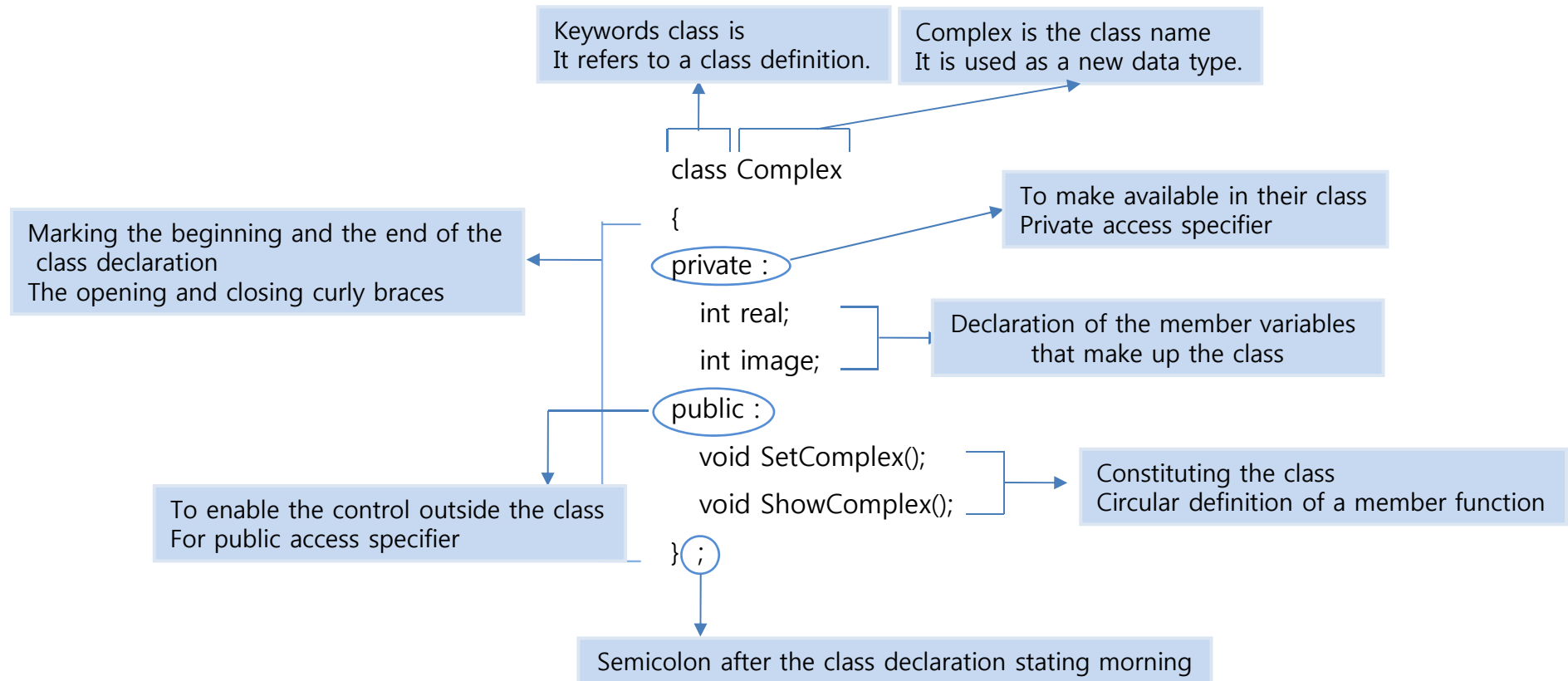
The new data type (class) = stored data (member variables) + function to process the data (member functions)

- A member variable to store the real part, the member variables for storing the imaginary image. It is generally based on data hiding to declare a private member variable of access specifiers.
- Two members of the public should set the value of the variable (Set Complex), or use the public access specifier to allow access from outside the class member functions for viewing (Show Complex).



[Picture 10-1] Complex Class Design

01 Understanding of the class

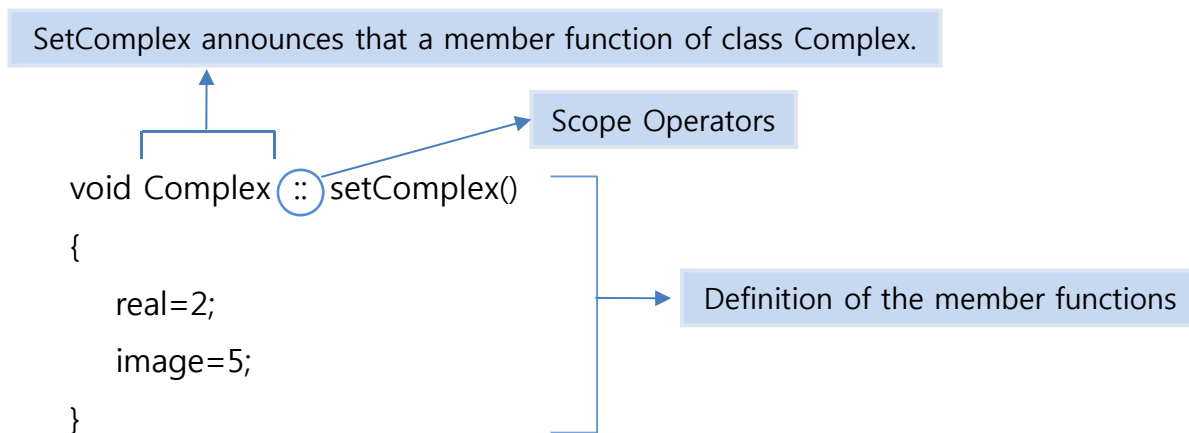


[Picture 10-2] Complex Class implementation

01 Understanding of the class

■ Member function definition (implementation) of the class

- Features of the member function
 - ❶ When you define a member function must specify the class name before the function name to indicate that the function belongs to which class. The scope operator (the class name, and then: Use :).
 - ❷ The purpose of defining a member function of class with a scope operator put it to access the private members defined in the class. So the member function known as 'the method'.



[Picture 10-3] Definition of SetComplex

01 Understanding of the class

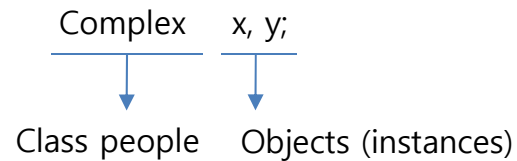
■ Object reference declaration and members.

- Wreaths object is a class instance (instance). So the object is also called 'instances'.

[class] Name one object class name, object name 2, ..., object name n;

객체 선언 기본 형식

- Complex With the example in which class generating the two objects, x and y.



01 Understanding of the class

■ Approach to class members.

- To use the structure as a member. And -> is used as a reference operator member.

The object type member variables;
The object member function name ();

. 연산자를 이용한 클래스 멤버 접근 방법

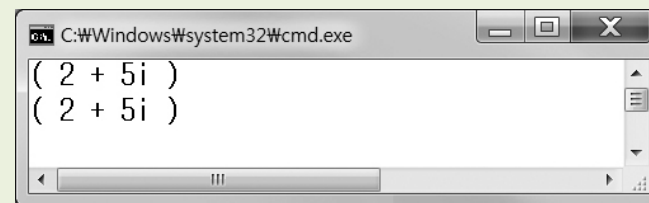
- After you create an object that implements a complex number class Complex An example of calling a member function.
Complex x, y;
x.SetComplex();
y.SetComplex();
- SetComplex should describe the object name with a member reference operator in front of the function name, so the member function.

SetComplex(); // Incorrect error ----- (X)

x.SetComplex(); // The object member function name (); For ---- correct (O)

Example 10-1. Designing a complex number as a class (10_01.cpp)

```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private :
06     int real;
07     int image;
08 public :
09     void SetComplex();
10     void ShowComplex();
11 };
12
13 void Complex::SetComplex()
14 {
15     real=2;
16     image=5;
17 }
18 void Complex::ShowComplex()
19 {
20     cout<<"( " <<real <<" + " <<image << "i)" <<endl ;
21 }
22 void main()
23 {
24     Complex x, y;
25
26     x.SetComplex();
27     x.ShowComplex();
28     y.SetComplex();
29     y.ShowComplex();
30 }
```



01 Understanding of the class

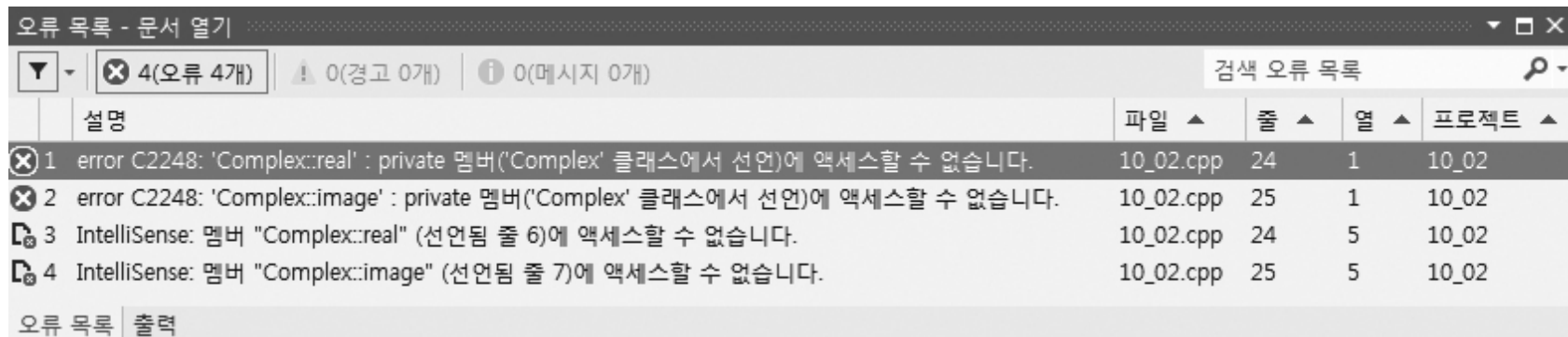
■ Access specifier of class, private/public

- private access specifier
 - ① If the access specifier is omitted, the default applies to private (default). It may be described explicitly: private.
 - ② Range is limited to members of the private member functions within the class affiliation.
 - ③ Generally, setting the member variables as private.
- public access specifier
 - ① If you want to specify as public members of public: it should describe explicitly.
 - ② Member function to create for the purpose of processing a private member variable is typically set to the public member.

Example 10-2. Members to identify private nature (10_02.cpp)

```
#include <iostream>
using namespace std;
class Complex {
private:
    int real;
    int image;
public:
    void SetComplex();
    void ShowComplex();
};
void Complex::SetComplex() {
    real = 2;
    image = 5;
}
```

```
void Complex::ShowComplex()
{
    cout << "( " << real << " + " << image << "i )" << endl;
}
void main()
{
    Complex x;
    x.real = 5;    // private 멤버데이터에 접근 ==> 컴파일 에러
    x.image = 10; // private 멤버데이터에 접근 ==> 컴파일 에러
    x.ShowComplex();
}
```



01 Understanding of the class

- Examples of how to resolve the error [10-2] is?
 - ① Everyone changes as public access specifier of these two member variables to be used to access the real and image.
 - ② By defining a public member function to access the private members variable calling their member functions.

Example 10-3. Add a member function to deal with private members (10_03.cpp)

```
#include <iostream>
using namespace std;
```

```
class Complex {
private:
    int real;
    int image;
public:
    void SetComplex();
    void ShowComplex();
    void SetReal(int r);
    void SetImage(int i);
};
```

```
void Complex::SetComplex() {
    real = 2;
    image = 5;
}
```

```
void Complex::ShowComplex() {
    cout << "( " << real << " + " << image << "i )" << endl;
}
void Complex::SetReal(int r) {
    real = r;
}
void Complex::SetImage(int i) {
    image = i;
}
```

```
void main() {
    Complex x;
    x.SetReal(5);
    x.SetImage(10);
    x.ShowComplex();
}
```

01 Understanding of the class

■ Defining the class member functions inside.

- Inline functions.
 - Inline functions are mainly compiled between use when the function definition is shorter and, when you define an inline function gives written a reserved word that inline before the function declaration (not the function is not actually called in order to reduce the time delay function call program code has the function code is inserted).

인라인 함수 기본 형식

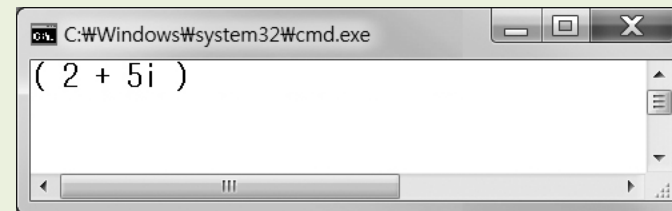
```
inline Data type function name (parameter list)
{
    Declaration of variables;
    sentence;
    return (result);
}
```

- Automatic Inline Functions
 - The definition of a member function very short, the class declaration may directly define a function inside. And the class is defined inside a function without the inline before the function declaration is automatically inline functions.

Example 10-4. Using an inline function (10_04.cpp)

```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private :
06     int real;
07     int image;
08 public :
09 void SetComplex()    // 자동 인라인 함수
10 {
11     real=2;
12     image=5;
13 }
14 void ShowComplex();
15 };
16
17
18 inline void Complex::ShowComplex() // 인라인 함수
19 {
20     cout<<"( " <<real <<" + " <<image << "i )" <<endl ;
21 }
```

```
22 void main()
23 {
24     Complex x;
25
26     x.SetComplex();
27     x.ShowComplex();
28 }
```



01 Understanding of the class

■ const constants and const member functions

- Use the reserved word const as a way to define constants (const constant).

General variable declarations similar but must give an initial value.

```
const data type variable name = initial value;
```

const 상수 기본 형식

- const constants are used for the same purpose with constant macro (#define statement to define a constant).
- However, the macro does not specify a type const constant constants is different in that it specifies the data type.
- The function in the internal use of the reserved word const when you do not change the member variable values (const member functions).

```
void Complex::ShowComplex() const
{
    cout<<"( " << real << " + " << image << "i )" <<endl;
}
```

Example 10-5. Using const member functions (10_05.cpp)

```
#include <iostream>
using namespace std;

class Complex
{
private:
    int real;
    int image;
public:
    void SetComplex();
    void ShowComplex() const; // const 멤버함수 원형
};

void Complex::SetComplex()
{
    real = 2;
    image = 5;
}
```

```
void Complex::ShowComplex() const // const 멤버함수 정의
{
    cout << "( " << real << " + " << image << "i )" << endl;
}

void main()
{
    Complex x;
    x.SetComplex();
    x.ShowComplex();
}
```

01 Understanding of the class

■ Overloading of functions

■ The signature of a function.

- An element for identifying a function is called signature (signature).
 - ❶ Function name
 - ❷ Number of parameters
 - ❸ Data type of the parameter
- Overloading of functions
 - Polymorphism is the name of the function that defines the function of a number of different types of parameters or parameters to avoid ambiguity when calling the function, but using the same.
- Why it is overloading the functions required.
 - If a function that defines a different name in the same sense both would be built to memorize give the function name each time you write a program separately. For example, if you define a function to obtain the absolute values as follows?

```
int abs(int x);  
double fabs(double x);  
long int labs(long int x);
```

Example 10-8. How to determine the absolute value of the parameter types using the overloading of other functions (10_08.cpp)

```
#include <iostream>
using namespace std;
```

```
int myabs(int num) {
    if (num<0)
        num = -num;
    return num;
}
```

```
double myabs(double num) {
    if (num<0)
        num = -num;
    return num;
}
```

```
long myabs(long num) {
    if (num<0)
        num = -num;
    return num;
}
```

```
void main()
{
    int a = -10;
    cout << a << "의 절댓값은-> " << myabs(a) << endl;

    double b = -3.4;
    cout << b << "의 절댓값은-> " << myabs(b) << endl;

    long c = -20L;
    cout << c << "의 절댓값은-> " << myabs(c) << endl;
}
```


Example 10-9. View the number of parameters at different function overloading (10_09.cpp)

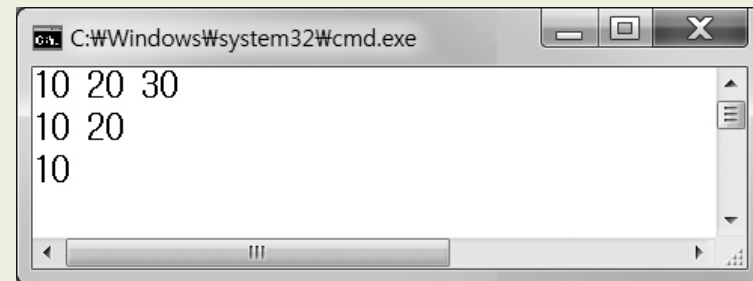
```
#include <iostream>
using namespace std;

void print(int x, int y, int z) {
    cout<<x<<" "<<y<<" "<<z<<endl;
}

void print(int x, int y) {
    cout<<x<<" "<<y<<endl;
}

void print(int x) {
    cout<<x<<endl;
}

void main() {
    print(10, 20, 30);
    print(10, 20);
    print(10);
}
```



01 Understanding of the class

■ Basic parameters of the function.

- You can set the value to the type parameter of the function, so that you have set the value of the type parameter and that is basic parameters.

```
void print(int x=10, int y=20, int z=30);
```

- The default parameters are parameters that can be applied to the corresponding actual parameters when calling the function without both automatic technology. The default parameter lets you specify in the declaration part of the function (circle).

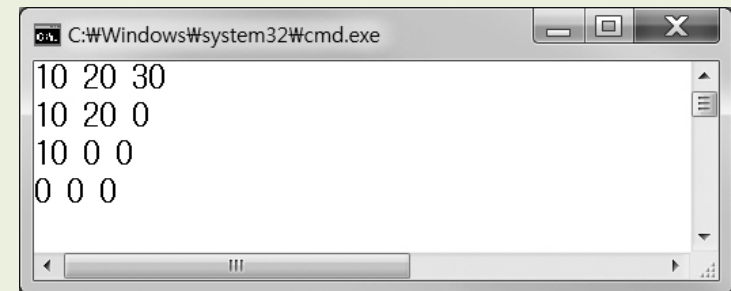
Example 10-10. To specify the default function parameters (10_10.cpp).

```
#include <iostream>
using namespace std;

void print(int x=0, int y=0, int z=0);

void main()
{
    print(10, 20, 30);
    print(10, 20);
    print(10);
    print();
}

void print(int x, int y, int z)
{
    cout<<x<<" "<<y<<" "<<z<<endl;
}
```



02 Constructors and destructors

■ The meaning and characteristics of the constructor

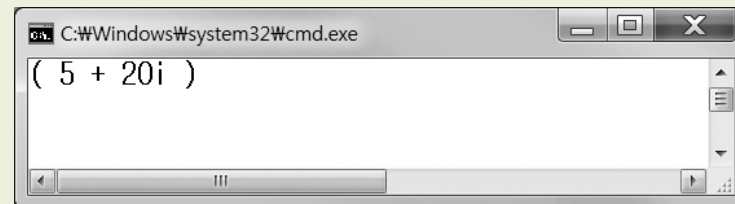
- Classes can give an initial value as the default data type when you create an object, the constructor that would make this possible.
- Features of the constructor
 - ① Constructor is a special member function.
 - ② Constructor name is the same class name.
 - ③ The constructor does not specify the data type of the return value.
 - ④ Call the constructor is not explicit.
 - ⑤ Constructor is invoked automatically by the compiler when declaring (generate) an object.
 - ⑥ Initialization means to initialize the parameters of the object is a member.
- If the programmer to explicitly create a constructor C ++ compiler creates a constructor with no parameters automatically. These constructors are called "default constructor".

Complex x; // The default constructor calls
- When you create an object that will initialize the member variables can be overridden through the constructor to a specific value.

Example 10-11. To create a constructor parameter (10_11.cpp)

```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private :
06     int real;
07     int image;
08 public :
09     Complex();
10     void ShowComplex() const;
11 };
12
13 Complex::Complex()
14 {
15     real=5;
16     image=20;
17 }
18
19 void Complex::ShowComplex() const
20 {
```

```
21     cout<<"( " <<real <<" + " <<image << "i )" <<endl ;
22 }
23
24 void main()
25 {
26     Complex x;
27     x.ShowComplex();
28 }
```



Example 10-12. Constructor to create using the parameters of the various initial values (10_12.cpp)

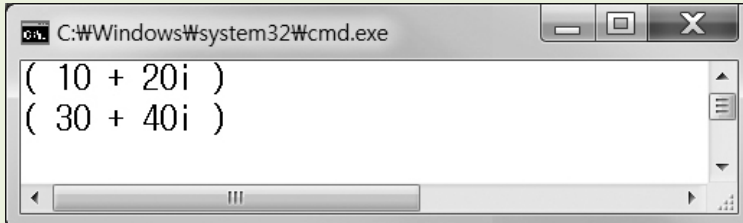
```
#include <iostream>
using namespace std;

class Complex
{
private :
    int real;
    int image;
public :
    Complex(int r, int i);      // 매개변수를 가지는 생성자
    void ShowComplex() const;
};

Complex::Complex(int r, int i) // 매개변수를 가지는 생성자
{
    real=r;
    image=i;
}
```

```
void Complex::ShowComplex() const
{
    cout<<" ( " <<real <<" + " <<image <<"i )" <<endl ;
}

void main()
{
    Complex x(10, 20);
    Complex y(30, 40);
    // Complex z;
    x.ShowComplex();
    y.ShowComplex();
}
```



A screenshot of a Windows command prompt window. The title bar reads "C:\Windows\system32\cmd.exe". The window contains two lines of output: "(10 + 20i)" and "(30 + 40i)".

02 Constructors and destructors

■ Overloading Constructors

- The default constructor with no parameters gives provides by C ++ compiler.

So, give a programmer to create a default constructor with no parameters, the compiler provides a default constructor anymore. For example, if you remove the comment of the line 28 in Example 10-12] it generates the following error:

오류 목록

오류 2개

경고 0개

메시지 0개

검색 오류 목록

설명	파일	줄	열	프로젝트
1 error C2512: 'Complex' : 사용할 수 있는 적절한 기본 생성자가 없습니다.	10_12.cpp	28	1	10_12
2 IntelliSense: "Complex" 클래스의 기본 생성자가 없습니다.	10_12.cpp	28	10	10_12

- To eliminate the compile error, the programmer must define a default constructor to add one more.
- Function overloading to define the function of the same name multiple times is applied to the constructor.
The number of different parameter types can be defined by multiple times, this is called constructor overloading.

Example 10-13. To overloaded constructor (10_13.cpp).

```
#include <iostream>
using namespace std;

class Complex
{
private:
    int real;
    int image;
public:
    Complex();           // 사용자 정의 기본 생성자 원형
    Complex(int r, int i); // 생성자 오버로딩
    void ShowComplex() const;
};

Complex::Complex()      // 사용자 정의 기본 생성자
{
    real = 0;
    image = 0;
}
```

```
Complex::Complex(int r, int i) // 매개변수를 가지는 생성자 정의
{
    real = r;
    image = i;
}

void Complex::ShowComplex() const
{
    cout << "(" << real << " + " << image << "i )" << endl;
}

void main()
{
    Complex x(10, 20);
    Complex y(30, 40);
    Complex z;           // 사용자 정의 기본 생성자 호출

    x.ShowComplex();
    y.ShowComplex();
    z.ShowComplex();
}
```


02 Constructors and destructors

■ The default constructor to specify parameter values.

- You can set the following parameters in the constructor, the default constructor values Statement (Circular).

```
Complex(int r=0 , int i=0);
```

- When describing the default parameter values it can be called to define a constructor once the parameter values in a variety of forms.

```
Complex x(10, 20); // Setting the two actual parameters by calling
```

```
Complex y(30);      // Actual parameters to invoke only one set.
```

```
Complex z;          // Called with no parameters.
```

■ Constructor initializes the colon.

- You can set the initial value of the member variables in the header section of the generator, wherein the initialization colon is used.

```
Complex::Complex(int r, int i) : real(r), image(i)
{
}
```

Example 10-14. Constructor to set default parameter values (10_14.cpp)

```
#include <iostream>
using namespace std;

class Complex
{
private:
    int real;
    int image;
public:
    Complex(int r = 0, int i = 0);    // 기본 매개변수
    void ShowComplex() const;
};

Complex::Complex(int r, int i)
{
    real = r;
    image = i;
}
```

```
void Complex::ShowComplex() const
{
    cout << "( " << real << " + " << image << "i )" << endl;
}

void main()
{
    // 기본 매개변수를 이용해 다양한 형태로 호출
    Complex x(10, 20);
    Complex y(30);
    Complex z;

    x.ShowComplex();
    y.ShowComplex();
    z.ShowComplex();
}
```

Example 10-15. Constructor to initialize the colon (10_15.cpp)

```
#include <iostream>
using namespace std;
class Complex
{
private:
    int real;
    int image;
public:
    Complex(int r = 0, int i = 0);
    void ShowComplex() const;
};

// 콜론을 이용한 멤버변수 초기화
Complex::Complex(int r, int i) : real(r), image(i)
{
}
```

```
void Complex::ShowComplex() const
{
    cout << "( " << real << " + " << image << "i )" << endl;
}

void main()
{
    Complex x(10, 20);
    Complex y(30);
    Complex z;

    x.ShowComplex();
    y.ShowComplex();
    z.ShowComplex();
}
```

02 Constructors and destructors

■ The meaning and features of destructors.

- Destructors are the opposite of the constructor.
 - ① Constructor is invoked automatically when an object is created, the destructor is automatically invoked when the object is destroyed.
 - ② If the constructor is a member function to initialize the object, the destructor (tasks, such as releasing resources) member function to arrange the objects.
- Features of the destructor
 - ① The destructor function is a member function.
 - ② The destructor function name is also used as the name of the class constructor.
 - ③ Destructor functions are denoted by the sign in front of the function name - try to separate from the constructor function.
 - ④ Destructor also does not specify a return type.
 - ⑤ Call the destructor does not explicit.
 - ⑥ Destructor is automatically called when an object is destroyed.
 - ⑦ Destructor can not specify the transmission parameter.
 - ⑧ Destructor can not specify the delivery parameters can not be overloaded.

Example 10-16. To define a destructor (10_16.cpp)

```
#include <iostream>
using namespace std;

class Complex
{
private:
    int real;
    int image;
public:
    Complex(int r = 0, int i = 0);
    ~Complex();           // 소멸자 원형
    void ShowComplex() const;
};

Complex::Complex(int r, int i) : real(r), image(i)
{
}
```

```
Complex::~~Complex()    // 소멸자 정의
{
    cout << "소멸자가 호출된다. \n";
}

void Complex::ShowComplex() const
{
    cout << "(" << real << " + " << image << "i )" << endl;
}

void main() {
    Complex x(10, 20);
    Complex y(30);
    Complex z;

    x.ShowComplex();
    y.ShowComplex();
    z.ShowComplex();
}
```

Homework

- Chapter 10 Exercise: 3, 4, 5, 6