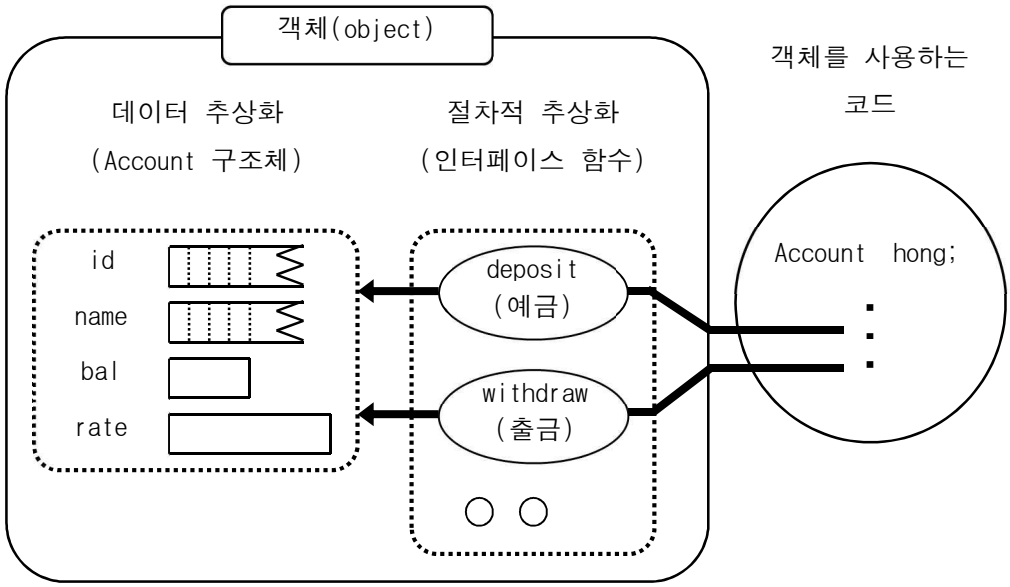
제2장 객체지향 문법

1. 객체지향 문법의 특징 및 생성자 소멸자

• 객체지향 프로그래밍(Object Oriented Programming)은 프로그래밍 방식의 하나로써. 객체를 주로 사용하여 프로그래밍하는 것이며, 객체는 데이터에 함수를 결합하여 추상화한 것이다.
C++은 객체지향 프로그래밍을 문법적으로 지원하는 언어이다.

• 특정 대상과 관련된 데이터를 묶어서 추상화하고 인터페이스함수를 통해 그 데이터를 처리 하는 프로그래밍 방식을 객체지향 프로그래밍이라고 한다.



◈ 객체 지향 프로그래밍의 다른 특징들

- 추상화를 지원하는 캡슐화
  인터페이스 함수를 통한 일관된 데이터의 사용,
  클래스를 이용하여 데이터와 함수를 명시적으로 묶어주는 방법

- 코드 재활용을 위한 계층성
  OOP가 지향하는 목표인 코드의 재활용
  이미 설계된 클래스를 새로운 클래스의 구성요소로 포함 또는 파생(상속).

- 다형성을 통한 추상화의 확장
  동일한 코드로 다양한 형태의 데이터를 처리할 수 있는 프로그래밍 방식
  예) 함수오버로딩, 함수오버라이딩, 연산자 오버로딩

* 객체를 생성해서 사용하기 위해서는 클래스라는 응용자료형이 사용된다.

◈ 클래스의 이해

- 클래스는 쉽게 생각해서 구조체 안에 인터페이스함수를 포함시킨 것이다.

- 클래스의 실제 메모리 블록을 객체(Object)라 한다.

- class의 접근지정자
  ① private : 클래스 외부에서 멤버에 직접 접근할 수 없다. 해당 클래스의 멤버함수에서만 접근 가능. 주로 멤버 데이터를 private로 선언한다.

  ② protected : private과 같으나 파생된 클래스의 멤버 함수에서 접근 가능

  ③ public : 클래스 외부, 내부에서 언제든지 접근 가능
  public 의 멤버함수는 private 변수와 연결이 가능하다. (인터페이스기능).
  주로 멤버 함수를 public 으로 선언한다.

◈ this 포인터란?

- class의 객체를 이용하여 멤버함수를 호출할 때 그 멤버함수를 호출한 객체의 주소를 암시적으로 전달 받아 저장하는 특수한 포인터
- 즉, this 포인터는 class의 멤버함수들 마다 갖고 있는 암시적인 전달인자이다.

[예제 2-1] this 포인터의 암시적, 명시적 사용

```
#include<iostream>
using namespace std;
#include <windows.h>
class   Point
{
   private :
      int x;
      int y;
   protected :
      void gotoxy();
   public :
      void init(int, int);
      void put_char(char);
};
void Point::gotoxy()
{
      COORD Pos = {x - 1, y - 1};
      SetConsoleCursorPosition(
      GetStdHandle(STD_OUTPUT_HANDLE),
      Pos);
}
void Point::init(int a, int b)
{
      x = a;
      y = b;
}
void Point::put_char(char ch)
{
      gotoxy();
      cout << ch;
}
int main()
{
      Point ob1, ob2;
      ob1.init(20,10);
      ob1.put_char('*');
      ob2.init(60,10);
      ob2.put_char('#');
      cout << "₩n₩n종료..." << endl;
      return 0;
}
```

```
#include<iostream>
using namespace std;
#include <windows.h>
class   Point
{
   private :
      int x;
      int y;
   protected :
      void gotoxy();
   public :
      void init(int, int);
      void put_char(char);
};
void Point::gotoxy()
{
      COORD Pos = {this->x-1, this->y-1};
      SetConsoleCursorPosition(
      GetStdHandle(STD_OUTPUT_HANDLE),
      Pos);
}
void Point::init(int x, int y)
{
      this->x = x;
      this->y = y;
}
void Point::put_char(char ch)
{
      this->gotoxy();
      cout << ch;
}
int main()
{
      Point ob1, ob2;
      ob1.init(20,10);
      ob1.put_char('*');
      ob2.init(60,10);
      ob2.put_char('#');
      cout << "₩n₩n종료..." << endl;
      return 0;
}
```

◈ 생성자(constructor)와 소멸자(destructor)

• 생성자
- 클래스의 객체가 생성될 때 자동으로 호출되어 데이터 멤버를 초기화하고 객체 생성을 완료해주는 역할을 하는 클래스의 멤버함수가 바로 생성자 (constructor) 함수이다.
- 생성자 함수는 클래스와 같은 이름의 멤버함수로서 객체가 생성될 때 자동으로 호출된다.

• 생성자의 특징
- 함수의 형은 정의하지 않음
- 클래스내에 생성자가 하나도 지정되어있지 않을 경우 컴파일러에 의해 default생성자가 추가됨

• 생성자의 종류

① default 생성자 : 전달인자 없이 호출될 수 있는 생성자

생성자를 전혀 지정하지 않을시 컴파일러가 디폴트 생성자를 자동생성 한다.
컴파일러가 자동생성하는 default constructor는 다음과 같은 형태로 정의된다.

```
class SomeClass
{
        :        ← class 선언시 생성자가 하나도 지정되어 있지 않으면,
        :          SomeClass(void) { };  와 같이 초기화 기능이 없는
        :          default constructor 가 컴파일러에 의해 자동생됨.
};
```

② overloaded 생성자 : 적어도 하나 이상의 전달인자를 받아 처리 하는 생성자

③ 복사 생성자 : 클래스 자신의 참조객체를 전달인자로 받는 생성자

객체 초기화시 이미 생성된 개체로 초기화하는 경우 사용한다.
프로그래머가 class내에 constructor를 직접 정의하더라도 그중에 copy constructor가 없으면 컴파일러가 default copy constructor를 자동생성한다.
Default copy constructor는 멤버데이터 대 멤버데이터를 그대로 Bit copy 하므로 Bit copy constructor라 부르며 사용자 정의 복사생성자를 Deep copy constructor라 부른다.

• 소멸자 : 객체 소멸 시 자동호출 되어 객체에서 사용했던 부가 메모리 공간을 반납 하고 객체 소멸을 완료하는 멤버함수(함수명 앞에 ~기호를 붙여 생성자와 구분함)

[예제2-2] 다양한 생성자의 사용

```
1:  #include <iostream>
2:  #include <string.h>
3:  using namespace std;
4:  // 클래스 선언
5:  class MoneyBox{
6:  private:
7:          char name[20];
8:          int sum;
9:  public:
10:         MoneyBox();   // 디폴트 생성자
11:         MoneyBox(const char *np, int m=0);   // 오버로디드 생성자
12:         MoneyBox(const MoneyBox &br);   // 복사 생성자
13:         ~MoneyBox() {}   // 소멸자
14:         void save(int m);
15:         void use(int m);
16:         void prn();
17: };
18: // 멤버함수 정의
19: MoneyBox::MoneyBox()
20: {
21:         strcpy(this->name, "아무개");
22:         this->sum=0;
23:         cout << "디폴트 생성자 호출..." << this->name << endl;
24: }
25:
26: MoneyBox::MoneyBox(const char *np, int m)
27: {
28:         strcpy(this->name, np);
29:         this->sum=m;
30:         cout << "오버로디드 생성자 호출..." << this->name << endl;
31: }
32:
33: MoneyBox::MoneyBox(const MoneyBox &br)
34: {
35:         strcpy(this->name, br.name);
36:         this->sum=br.sum;
37:         cout << "복사 생성자 호출..." << this->name << endl;
38: }
39:
40: void MoneyBox::save(int m)
41: {
42:         if(m>=0) { this->sum+=m; }
43: }
```

```
44:
45: void MoneyBox::use(int m)
46: {
47:         if(m>0 && m<=this->sum) { this->sum-=m; }
48: }
49:
50: void MoneyBox::prn()
51: {
52:         cout << "이름 : " << this->name << endl;
53:         cout << "잔액 : " << this->sum << endl;
54: }
55:
56: MoneyBox func(MoneyBox m);   // 객체를 리턴하는 일반함수
58: int main()
59: {
60:         MoneyBox a;
61:         MoneyBox b("홍길동");
62:         MoneyBox c("이순신", 500);
63:         MoneyBox d=b;
64:         MoneyBox *mp1=new MoneyBox("홍길동");
65:         MoneyBox *mp2=new MoneyBox(c);
66:         func(b);
67:
68:        delete mp1;
69:        delete mp2;
70:         return 0;
71: }
72:
73: MoneyBox func(MoneyBox m)
74: {
75:         return m;
76: }
```

** MoneyBox클래스의 name 멤버를 char *형으로 바꾸어 프로그램을 수정해보자

2. 객체의 형태

객체를 생성해서 활용하는 방법은 크게 4가지가 존재한다.
(1) 일반 객체 (2) 객체 포인터 (3) 참조 객체 (4) 객체 배열

◈ 일반 객체

선언과 동시에 메모리 할당, 멤버참조연산자(.)를 통하여 멤버 접근

```
class ClassName
{
public:
    void memberFunction1();
    int memberFunction2();
    int var1;
private:
    void memberFunction3() { ... }
    int var2;
};
```

```
int main()
{
    ClassName object;
    object.memberFunction1();
    object.memberFunction2();
    object.memberFunction3(); // error
    object.var1 = 10;
    object.var2 = 20; // error
}
```

◈ 객체 포인터

선언 후 new 연산자를 통하여 실제 객체 메모리 할당하여 사용
간접멤버참조연산자(->)를 통하여 멤버 접근

```
class ClassName
{
public:
    void memberFunction1();
    int memberFunction2();
    int var1;
private:
    void memberFunction3() { ... }
    int var2;
};
```

```
int main()
{
    ClassName *pObject;
    pObject = new ClassName;
    pObject->memberFunction1();
    pObject->memberFunction2();
    pObject->var1 = 10;
    delete pObject;
}
```

◈ 참조 객체

이미 선언된 객체의 별명으로 사용됨
멤버참조연산자(.)를 통하여 멤버 접근

```
class ClassName
{
public:
    void memberFunction1();
    int memberFunction2();
    int var1;
private:
    void memberFunction3() { ... }
    int var2;
};
```

```
int main()
{
    ClassName Object;
    ClassName &rObject1 = Object;
    rObject1.memberFunction1();
    rObject1.memberFunction2();
    rObject1.var1 = 10;
    ClassName *pObject;
    pObject = new ClassName;
    ClassName &rObject2 = *pObject;
    rObject2.memberFunction1();
    rObject2.memberFunction2();
    rObject2.var1 = 20;
    delete pObject;
}
```

◈ 객체 배열

객체들로 이루어진 배열을 선언하여 사용
방이름으로 접근할때에는 멤버참조연산자(.)를 통하여 멤버 접근하고
배열명(주소)로 접근할때에는 간접멤버참조연산자(->)를 통하여 멤버 접근

```
class ClassName
{
public:
    void memberFunction1();
    int memberFunction2();
    int var1;
private:
    void memberFunction3() { ... }
    int var2;
} ;
```

```
int main()
{
    ClassName obAry[2];
    obAry[0].memberFunction1();
    obAry[0].memberFunction2();
    obAry[0].var1 = 10;
    (obAry+1)->memberFunction1();
    (obAry+1)->memberFunction2();
    (obAry+1)->var1 = 10;
}
```

◈ 객체 배열의 초기화

객체 배열 선언 시 배열의 각 원소를 초기화 할때에는 생성자를 명시적으로 호출한다.

ClassName obAry[5] = { ClassName (1,2), ClassName (10,20) };

◈ 객체 배열의 동적할당 및 해제

객체 배열을 new[] 연산자로 할당할 경우 반드시 delete[]로 해제해야한다.

```
#include<iostream>
using namespace std;
class Robot {
public :
  Robot() {
      cout << "Robot constructor" << endl;
  }
  ~Robot() {
      cout << "Robot destructor" << endl;
  }
};
int main()
{
        Robot *rp;
        rp = new Robot[3];
        delete  rp;  ← 문제 발생!!
        return 0;
}
```

```
#include<iostream>
using namespace std;
class Robot {
public :
  Robot() {
      cout << "Robot constructor" << endl;
  }
  ~Robot() {
      cout << "Robot destructor" << endl;
  }
};
int main()
{
        Robot *rp;
        rp = new Robot[3];
        delete[] rp;
        return 0;
}
```

3. 콜론 초기화 기법

• class의 형틀 선언시에는 일반적으로 멤버데이터를 초기화 할 수 없으나 콜론초기화기법을 사용하면 class의 형틀 선언시에도 초기화 데이터를 표기할 수 있다.
단, 실제로 초기화 되는 시점은 객체가 생성될 때이다.

```
class Point{
      int x = 7;          <= 불가능
      int y =10;          <= 불가능
   public :
         :
         :
};
```

일반적으로 객체의 초기화는 constructor 의 정의부 내에서 이루어진다.

```
 class Point{
      int x;
      int y;
   public :
     Point(void){ x=7; y=10 };
         :
};
```

• 콜론초기화는 constructor의 정의부의 머리부에서 이루어진다.

(클래스 선언부내에서의 초기화)

```
class Point
{
   private :
      int x;
      int y;
   public :
      Point(void) :x(7),y(10) { };
               :
};
```

(constructor 정의부에서의 초기화)

```
class Point
{
private :
        int x;
        int y;
   public :
        Point(void);
};
Point::Point(void) : x(7), y(10)
{   }
```

• 배열멤버는 콜론초기화 기법으로 초기화가 불가능하다.

[예제 2-3] char 배열 멤버의 초기화

```
#include<iostream>
using namespace std;
#include<string.h>
class Wow
{
   private :
            char name[20];
            int age;
   public :
            Wow(char * ="무명씨", int=0);
            ~Wow(void){ };
            void update(char *, int);
            void output();
};
Wow::Wow(char *p, int n) : age(n)   // 콜론초기화
{
            strcpy(name,p); // name 멤버는 콜론초기화 기법을 사용할 수 없다
}
void Wow::update(char *p, int n)
 {
            strcpy(name, p);
            age = n;
 }
 void Wow::output()
 {
            cout << "* 이름 : " << name << endl;
            cout << "* 나이 : " << age << endl;
 }
  int main()
  {
             Wow person1;
             Wow person2("King", 20);
             Wow person3=person2;
             person1.output();
             person3.output();
             person3.update("King Kong", 77);
             person3.output();
             return 0;
  }
```

- 콜론초기화 기법을 사용하는 경우
  - 일반적인 클래스 데이터멤버의 초기화 (배열멤버 제외)
  - 포함된 객체의 초기화 (사용 권장)
  - 상수멤버(const member)의 초기화 (필수 사항)

4. const 멤버와 static 멤버

◈ const멤버 변수와 const 멤버함수 그리고 const 객체

- const 멤버변수
  - class의 멤버변수 선언 시 const를 붙이면 이 멤버변수는 상수적 의미를 갖게 된다.
  - const 멤버변수는 생성자에서 초기화 되고 나면 프로그램 수행 도중 변경이 불가능하다.
  - const 멤버변수는 colon 초기화 기법에 의해서만 초기화가 가능하다.

- const 멤버 함수
  - class의 멤버함수 중 const를 붙여서 선언 및 정의된 함수
  - const 멤버함수 내에서는 멤버데이터를 변경 할 수 없다.
  - const 멤버함수는 const 멤버함수만 호출 할 수 있다.

- const 객체
  - 객체의 정적 할당 시 const로 선언된 객체를 const 객체라 한다.
  - const 객체는 선언 시 반드시 초기값을 갖도록 해야 하며, 이후 절대로 데이터멤버의 값을 변경 할 수 없다.
  - const 객체는 const 멤버함수만 호출 할 수 있다.

[예제2-4]

```
1: #include <iostream>
2: using namespace std;
3: class Letter{
4:   private:
5:         const char ch;   // const 데이터멤버
6:         int num;
7:   public:
8:         Letter(char c='*', int n=0);
9:         void n_prn();
10:        void c_prn() const;   // const 멤버함수
11:        void show();
12: };
13:
14: Letter::Letter(char c, int n) : ch(c)
15: {
16:        num=n;
17: }
```

```
18:
19: void Letter::n_prn()
20: {
21:         // ch='@';   -> 에러, const 데이터멤버는 수정할 수 없다.
22:         num+=10;
23:         cout << "const가 아닌 멤버함수 호출..." << endl;
24:         show();
25: }
26:
27: void Letter::c_prn() const
28: {
29:         // num+=10;   -> 에러, const멤버함수이므로 수정할 수 없다.
30:         cout << "const 멤버함수 호출..." << endl;
31:         // show();   -> 에러, const멤버함수는 일반 멤버함수 호출불가
32: }
33:
34: void Letter::show()
35: {
36:         int i;
37:         for(i=0; i<num; i++){
38:               cout << ch;
39:         }
40:         cout << endl;
41: }
42:
43: int main()
44: {
45:         Letter a('@', 5);
46:         const Letter b('#', 3);
47:
48:         a.n_prn();
49:         a.c_prn();
50:         // b.n_prn();   -> 에러, const객체는 const가 아닌 멤버 호출불가
51:         b.c_prn();
52:
53:         return 0;
54: }
```

◈ static 멤버변수와 static 멤버함수

• static 멤버변수
- class의 멤버변수중 static 키워드를 이용해서 선언된 멤버변수
- static 멤버변수는 해당 class의 모든 객체 사이에서 공동으로 사용되는 멤버로 해당 클래스의 객체가 여러가 선언되더라도 static 멤버변수는 딱 1개만 할당된다.
- 객체생성 이전에 미리 할당되어있는 멤버로 static멤버변수는 선언부와 정의부를 모두 지정해야 한다. (정의부에서 초기화 하므로 생성자함수에서는 초기화 시키지 않는 것이 좋다)

• static 멤버함수
- static 멤버함수는 static 멤버변수를 제어하기 위하여 선언하여 사용한다.
- 객체 생성 이전에도 static 멤버함수를 호출하여 사용할 수 있다. (:: 연산자 이용하여 호출)
- static 멤버함수는 static 멤버변수에만 접근 할 수 있다.

[예제2-5]

```
1:  #include<iostream>
2:  using namespace std;
3:
4:   class Sample{
5:    private:
6:       int num;
7:       static int cnt;        // 정적멤버변수 선언부
8:    public:
9:       Sample(int n=0);
10:      void prn() const;
11:      static void setNum(int n);  // 정적멤버함수선언
12:  };
13:
14:  int Sample::cnt=0;  // 정적 멤버변수 정의부
15:
16:  void Sample::setNum(int n)  // 정적 멤버함수 정의
17:  {
18:      cnt=n;  // 정적 멤버변수의 값을 변경한다.
19:  }
20:  Sample::Sample(int n)  // 오버로디드생성자정의
21:  {
22:      num=n;
23:      cnt++;  // 객체가 생성될 때 마다 1씩 증가한다.
24:  }
25:
26:  void Sample::prn() const  // 출력멤버함수정의
27:  {
```

```
28:         cout << "num : " << num << ", ";
29:         cout << "cnt : " << cnt << endl;
30:   }
31:
32:   int main()
33:   {
34:         int start;
35:
36:         cout << "시작값을입력하세요: ";
37:         cin >> start;
38:         Sample::setNum(start);   // 객체 없이 독립적으로 호출 가능
39:
40:         Sample a(10), b(20), c(30);   // 시작값3을입력하면...
41:         a.prn();
42:         b.prn();
43:         c.prn();
44:         return 0;
45:   }
```

◈ static 멤버만으로 이루어진 모노톤 패턴 클래스

[예제2-6]

```
#include <iostream>
using namespace std;
class Math     //MonoTone
{
public:
            static double CircleArea( int r)
            {
                        return   r*r*3.14;
            }
            static int RectArea( int h, int w){
                        return   h*w;
            }
};
int main()
{
            cout << Math::CircleArea(5) << endl;
            cout << Math::RectArea(3,6 ) << endl;
            return 0;
}
```