4. 클래스의 완성

4주차

150-153, 155p

정보 은닉

- class Point
  - (0,0)~(100,100) 좌표계 내의 점을 표현
  - 멤버변수
    - int x;
    - int y;
- class Rectangle
  - 2개의 Point 객체로써 직사각형을 표현
  - 멤버변수
    - Point upLeft;
    - Point lowRight;
- 클래스의 멤버변수들이 public이면 아래 사항을 지킬 방법이 없다. 컴파일 오류가 아니므로.
  - Point에는 정해진 좌표계 내의 점만 저장
  - Rectangle에는 항상 upLeft가 lowRight보다 왼쪽 위에 위치하도록 저장.
- 정보은닉이란
  - 멤버변수를 private으로 선언하고, 해당 변수에 대한 Get/Set 함수를 제공하여 안전한 형태로 멤버변수에 접근하도록 하는 것.

153-158p

정보 은닉
(point.h)

```
#ifndef __POINT_H_
#define __POINT_H_

class Point
{
private:
        int x;
        int y;

public:
        bool InitMembers(int xpos, int ypos);
        int GetX() const;
        int GetY() const;
        bool SetX(int xpos);
        bool SetY(int ypos);
};

#endif
```

멤버 변수들은 private.
멤버변수 각각에 대한 Get/Set 함수를 준비. (당장 사용하지 않더라도 미리 준비)

153-158p

정보 은닉
(point.cpp 1/2)

```
#include <iostream>
#include "Point.h"
using namespace std;

bool Point::InitMembers(int xpos, int ypos) {
    if(xpos<0 || ypos<0) {
        cout << "Point(" << xpos << "," << ypos << "): 벗어난 범위의 값 전달" << end
        return false;
    }

    x=xpos;
    y=ypos;
    return true;
}

int Point::GetX() const {return x;}
int Point::GetY() const {return y;}
```

153-158p

정보 은닉
(point.cpp 2/2)

```
bool Point::SetX(int xpos) {
    if(0<xpos || xpos>100) {
        cout << "Point xpos=" << xpos << ": 벗어난 범위의 값 전달" << endl;
        return false;
    }

    x=xpos;
    return true;
}

bool Point::SetY(int ypos) {
    if(0<ypos || ypos>100) {
        cout << "Point ypos=" << ypos << ": 벗어난 범위의 값 전달" << endl;
        return false;
    }

    y=ypos;
    return true;
}
```

153-158p

정보 은닉
(rectangle.h)

```
#ifndef __RECTANGLE_H_
#define __RECTANGLE_H_

#include "Point.h"

class Rectangle
{
private:
        Point upLeft;
        Point lowRight;

public:
        bool InitMembers(const Point &ul, const Point &lr);
        void ShowRecInfo() const;
};

#endif
```

멤버 변수들은 private.
Set 하는 함수를 별도로 준비.

153-158p

정보 은닉
(rectangle.cpp)

```
#include <iostream>
#include "Rectangle.h"
using namespace std;

bool Rectangle::InitMembers(const Point &ul, const Point &lr) {
        if(ul.GetX()>lr.GetX() || ul.GetY()>lr.GetY()) {
                cout<<"Rectangle 잘못된 위치정보 전달"<<endl;
                return false;
        }
        upLeft.InitMembers(ul.GetX(), ul.GetY());
        lowRight.InitMembers( lr.GetX(), lr.GetY());
        return true;
}

void Rectangle::ShowRecInfo() const {
        cout<<"좌 상단: "<<'['<<upLeft.GetX()<<", ";
        cout<<upLeft.GetY()<<']'<<endl;
        cout<<"우 하단: "<<'['<<lowRight.GetX()<<", ";
        cout<<lowRight.GetY()<<']'<<endl<<endl;
}
```

153-158p

정보 은닉 (RectangleFaultFind.cpp)

```
#include<iostream>
#include "Point.h"
#include "Rectangle.h"
using namespace std;

int main(void) {
	Point pos1;
	if (!pos1.InitMembers(-2, 4))
		cout << "Point 초기화 실패" << endl;
	if (!pos1.InitMembers(2, 4))
		cout << "Point 초기화 실패" << endl;
	Point pos2;
	if (!pos2.InitMembers(5, 9))
		cout << "Point 초기화 실패" << endl;
	Rectangle rec;
	if (rec.InitMembers(pos2, pos1))
		cout << "Rectangle 초기화 실패" << endl;

	if (rec.InitMembers(pos1, pos2))
		cout << "Rectangle 초기화 실패" << endl;
	rec.ShowRecInfo();
	return 0;
}
```

158-160p

정보은닉과 const

멤버함수의 const 선언

```
int GetX() const;
int GetY() const;
void ShowRecInfo() const;
```

const 함수 내에서는 동일 클래스에 선언된 멤버 변수의 값을 변경 못함!

```
int GetNum()
{
    return num;
}
void ShowNum() const
{
    cout<<GetNum()<<endl;     // 컴파일 에러 발생
}
```

이 둘이 멤버함수일 때...

const 함수는 const 멤버함수만 호출 가능.

```
void InitNum(const EasyClass &easy)
{
    num=easy.GetNum();     // 컴파일 에러 발생
}
```

const 객체는 const 멤버함수만 호출 가능!

GetNum이 const 함수가 아닐 때...

161-162p

캡슐화

- 캡슐화란!
  - 관련 있는 모든 것을 하나의 클래스 안에 묶어 두는 것!
  - 자신의 데이터나 실제 구현 내용의 일부를 외부에 대하여 감춤.
  - 어디까지 어떻게 감싸야 할지를 고민하는 것이 프로그래머의 몫.
- 캡슐화의 이점
  - 외부에 공개되지 않는(private) 멤버함수, 멤버변수의 수정이 발생해도 외부에서 사용하는 코드의 수정은 필요하지 않게 됨.

캡슐화 이전 소스
(Encaps1.cpp 1/2)

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

class SnivelCap {   // 콧물 처치용 캡슐
public:
        void Take() const { cout << "콧물이 싹~ 납니다." << endl; }
};

class SneezeCap {   // 재채기 처치용 캡슐
public:
        void Take() const { cout << "재채기가 멎습니다." << endl; }
};

class SnuffleCap {  // 코막힘 처치용 캡슐
public:
        void Take() const { cout << "코가 뻥 뚫립니다." << endl; }
};
```

캡슐화 이전 소스
(Encaps1.cpp 2/2)

```
class ColdPatient {
public:
        void TakeSnivelCap(const SnivelCap& cap) const { cap.Take(); }
        void TakeSneezeCap(const SneezeCap& cap) const { cap.Take(); }
        void TakeSnuffleCap(const SnuffleCap& cap) const { cap.Take(); }
};

int main(void) {
        SnivelCap scap;
        SneezeCap zcap;
        SnuffleCap ncap;

        ColdPatient sufferer;
        sufferer.TakeSnivelCap(scap);
        sufferer.TakeSneezeCap(zcap);
        sufferer.TakeSnuffleCap(ncap);
        return 0;
}
```

(1) 여러 단계를 거쳐야 해서 복잡.
(2) 약 복용에 순서를 지켜야 한다면 처리가 힘듬.

캡슐화 된 소스
(Encaps2.cpp 1/2)

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

class SnivelCap    // 콧물 처치용 캡슐
{
public:
        void Take() const { cout << "콧물이 싹~ 납니다." << endl; }
};

class SneezeCap    // 재채기 처치용 캡슐
{
public:
        void Take() const { cout << "재채기가 멎습니다." << endl; }
};

class SnuffleCap   // 코막힘 처치용 캡슐
{
public:
        void Take() const { cout << "코가 뻥 뚫립니다." << endl; }
};
```

캡슐화 된 소스
(Encaps2.cpp 2/2)

```
class CONTAC600 {
        SnivelCap sin;
        SneezeCap sne;
        SnuffleCap snu;
public:
        void Take() const {
                sin.Take();     sne.Take();     snu.Take();
        }
};
class ColdPatient {
public:
        void TakeCONTAC600(const CONTAC600& cap) const { cap.Take(); }
};
int main(void) {
        CONTAC600 cap;
        ColdPatient sufferer;
        sufferer.TakeCONTAC600(cap);
        return 0;
}
```

캡슐화에는 class 타입을 포함한 어떤 타입의 데이터든 포함 가능.

168-170p

생성자 (constructor)
(Constructor1.cpp)

- 클래스의 이름과 동일한 이름의 함수
- 반환형이 없음 (실제로 반환하지 않음)
- 함수 overloading 가능 (여러 개 존재 가능)
- 매개변수에 default값 사용 가능
- Default constructor : 인자 없이 호출될 수 있는 constructor

```
class SimpleClass {
        int num1;
        int num2;
public:
        SimpleClass() {
                num1 = 0;       num2 = 0;
        }
        SimpleClass(int n) {
                num1 = n;       num2 = 0;
        }
        SimpleClass(int n1, int n2) {
                num1 = n1;      num2 = n2;
        }
```

Overload된 여러 개의 constructor 존재

생성자 (constructor)
(Constructor1.cpp)

```
        /*
        SimpleClass(int n1=0, int n2=0) {
                num1=n1;
                num2=n2;
        }
        */
        void ShowData() const {
                cout << num1 << ' ' << num2 << endl;
        }
};
int main(void) {
        SimpleClass sc1;
        sc1.ShowData();

        SimpleClass sc2(100);
        sc2.ShowData();

        SimpleClass sc3(100, 200);
        sc3.ShowData();
        return 0;
}
```

매개변수에 default값 사용.
다른 constructor들과 매개변수 사용이 애매해지므로 가려 놓음.

생성자 사용 부분을 보고
3개 또는 1개의 생성자를 작성할 수 있어야 함.

171-172p

Default constructor(인자 없음) 의 호출

- SimpleClass sc1; (O)
- SimpleClass sc1(); (X)
- SimpleClass *p1 = new SimpleClass; (O)
- SimpleClass *p1 = new SimpleClass(); (O)

이 형태만 사용할 수 없음.
함수 원형선언으로 인식하므로.

173-175p

이전 예제에 생성자 활용 (FruitSaleSim2.cpp 1/3)

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

class FruitSeller {
        int APPLE_PRICE;
        int numOfApples;
        int myMoney;
public:
        FruitSeller(int price, int num, int money) {
                APPLE_PRICE = price;
                numOfApples = num;
                myMoney = money;
        }
        int SaleApples(int money) {
                int num = money / APPLE_PRICE;
                numOfApples -= num;
                myMoney += money;
                return num;
        }
        void ShowSalesResult() const {
                cout << "남은 사과: " << numOfApples << endl;
                cout << "판매 수익: " << myMoney << endl << endl;
        }
};
```

이전 예제에 생성자 활용
(FruitSaleSim2.cpp 2/3)

```
class FruitBuyer {
        int myMoney;
        int numOfApples;
public:
        FruitBuyer(int money) {
                myMoney = money;
                numOfApples = 0;
        }
        void BuyApples(FruitSeller& seller, int money) {
                numOfApples += seller.SaleApples(money);
                myMoney -= money;
        }
        void ShowBuyResult() const {
                cout << "현재 잔액: " << myMoney << endl;
                cout << "사과 개수: " << numOfApples << endl << endl;
        }
};
```

이전 예제에 생성자 활용
(FruitSaleSim2.cpp 3/3)

```
int main(void)
{
        FruitSeller seller(1000, 20, 0);
        FruitBuyer buyer(5000);
        buyer.BuyApples(seller, 2000);

        cout << "과일 판매자의 현황" << endl;
        seller.ShowSalesResult();
        cout << "과일 구매자의 현황" << endl;
        buyer.ShowBuyResult();
        return 0;
}
```

Member initializer를 이용한 멤버 초기화 (point.h)

- **Member initializer**
  - 클래스 constructor 함수 정의부에서 멤버 변수 초기화.
    (객체인 경우: constructor 호출, 객체가 아닌 경우: 초기값 사용)

```
#ifndef __POINT_H_
#define __POINT_H_

class Point
{
        int x;
        int y;
public:
        Point(const int &xpos, const int &ypos);
        int GetX() const;
        int GetY() const;
        bool SetX(int xpos);
        bool SetY(int ypos);
};

#endif
```

Member initializer를 이용한 멤버 초기화 (point.cpp)

176-179p

```
#include <iostream>
#include "Point.h"
using namespace std;
Point::Point(const int &xpos, const int &ypos) { x=xpos; y=ypos; }
int Point::GetX() const {return x;}
int Point::GetY() const {return y;}
bool Point::SetX(int xpos) {
        if(0>xpos || xpos>100) {
                cout<<"벗어난 범위의 값 전달"<<endl;
                return false;
        }
        x=xpos;
        return true;
}
bool Point::SetY(int ypos) {
        if(0>ypos || ypos>100) {
                cout<<"벗어난 범위의 값 전달"<<endl;
                return false;
        }
        y=ypos;
        return true;
}
```

Member initializer를 이용한 멤버 초기화 (Rectangle.h)

176-179p

```
#ifndef __RECTANGLE_H_
#define __RECTANGLE_H_

#include "Point.h"

class Rectangle
{
	Point upLeft;
	Point lowRight;

public:
	Rectangle(const int &x1, const int &y1, const int &x2, const int &y2);
	void ShowRecInfo() const;
};

#endif
```

Member initializer를 이용한 멤버 초기화 (Rectangle.cpp)

176-179p

```
#include <iostream>
#include "Rectangle.h"
using namespace std;

Rectangle::Rectangle(const int &x1, const int &y1, const int &x2, const int &y2)
	:upLeft(x1, y1), lowRight(x2, y2)
{
	// empty
}

void Rectangle::ShowRecInfo() const
{
	cout<<"좌 상단: "<<'['<<upLeft.GetX()<<", ";
	cout<<upLeft.GetY()<<']'<<endl;
	cout<<"우 하단: "<<'['<<lowRight.GetX()<<", ";
	cout<<lowRight.GetY()<<']'<<endl<<endl;
}
```

Member initializer를 이용한 멤버 초기화
(RectangleConstructor.cpp)

176-179p

```
#include<iostream>
#include "Point.h"
#include "Rectangle.h"
using namespace std;

int main(void)
{
	Rectangle rec(1, 1, 5, 5);
	rec.ShowRecInfo();
	return 0;
}
```

Member initializer를 이용한 멤버 초기화

◆ 선언과 함께 초기화가 필요한 형태도 멤버로 가질 수 있게 됨.

- const 멤버

```
// 멤버로 선언
const int APPLE_PRICE;

// constructor에서 초기화
FruitSeller(int price, int num, int money)
 : APPLE_PRICE(price)
```

- 참조자 멤버 (흔하지는 않음)

```
// 참조자를 멤버로 선언
AAA &ref;

// constructor에서 초기화
BBB( AAA &r)
 : ref(r) )
```

Default constructor

- 생성자를 정의하지 않으면
  - 인자 없고
  - 하는 일 없는
  - 디폴트 생성자가 컴파일러에 의해서 추가됨.
- 그러므로 모든 객체는 무조건 생성자의 호출 과정을 거쳐서 완성됨.
- new 대신 malloc 사용하면
  - 생성자가 호출되지 않으므로 문제!
  - 클래스 객체 할당 받을 때는 new 사용!
- Default constructor 아닌 constructor 있을 때는 컴파일러가 자동으로 default constructor를 생성하지 않음.
  - 다른 생성자가 이미 있는데 default constructor가 필요하다면 프로그래머가 이를 만들어 넣어야 함.

186-187p

private constructor
(PrivateConstructor.cpp)

- 생성자가 private이라는 것은 외부에서의 객체 생성을 허용하지 않겠다는 뜻
- 객체의 생성 방법을 제한 하고자 할 때 사용.

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

class AAA {
private:
        int num;
public:
        AAA() : num(0) {}
        AAA& CreateInitObj(int n) const {
                AAA* ptr = new AAA(n);
                return *ptr;
        }
        void ShowNum() const { cout << num << endl; }

private:
        AAA(int n) : num(n) {}
};
```

private constructor
사용하는 생성 함수

private constructor
(PrivateConstructor.cpp)

```
int main(void)
{
	AAA base;
	base.ShowNum();

	AAA& obj1 = base.CreateInitObj(3);
	obj1.ShowNum();

	AAA& obj2 = base.CreateInitObj(12);
	obj2.ShowNum();

	delete &obj1;
	delete &obj2;
	return 0;
}
```

소멸자(destructor)

- 객체 소멸 시 자동으로 반드시 호출됨.
- 소멸자도 정의하지 않으면 컴파일러에 의해 디폴트 소멸자가 삽입됨.
- 클래스 이름 앞에 ~ 붙은 이름.
- 반환값도 인자도 없음.
  - 그러므로 함수 overloading 안되어 유일!
- 멤버에 잡아둔 자원을 delete 하는 데 사용.

소멸자(destructor)
(desctructor.cpp 1/2)

188-190p

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;

class Person {
private:
	char* name;
	int age;
public:
	Person(char* myname, int myage) {
		int len = strlen(myname) + 1;
		name = new char[len];
		strcpy(name, myname);
		age = myage;
	}
	void ShowPersonInfo() const {
		cout << "이름: " << name << endl;
		cout << "나이: " << age << endl;
	}
```

소멸자(destructor)
(desctructor.cpp 2/2)

```
        ~Person()
        {
                delete []name;
                cout << "called destructor!" << endl;
        }
};

int main(void)
{
        Person man1((char *)"Lee dong woo", 29);
        Person man2((char *)"Jang dong gun", 41);
        man1.ShowPersonInfo();
        man2.ShowPersonInfo();
        return 0;
}
```

변수 선언으로 생성되었으므로
블록이 닫힐 때 destructor가 호출됨.