

윈도우 프로그래밍

6. 클래스(3)

2018. 4.13.
심미나 교수



목 차

- I. 정보은닉과 캡슐화
- II. 생성자와 소멸자
- III. 실습

I. 정보은닉과 캡슐화



객체지향프로그래밍의 특징

- 정보은닉

- 객체는 자신의 데이터와 함수를 외부에 공개하거나 숨길 수 있음
 - 예를 들어, “A클래스의 정보(멤버변수)는 A내 함수외에 B나 C클래스에서 접근제한”
- 만약 외부에 객체의 데이터는 숨기고 데이터를 처리하는 멤버함수만 공개한다면 데이터의 잘못된 수정을 막을 수 있어서 프로그램의 안정성을 높일 수 있음

- 캡슐화

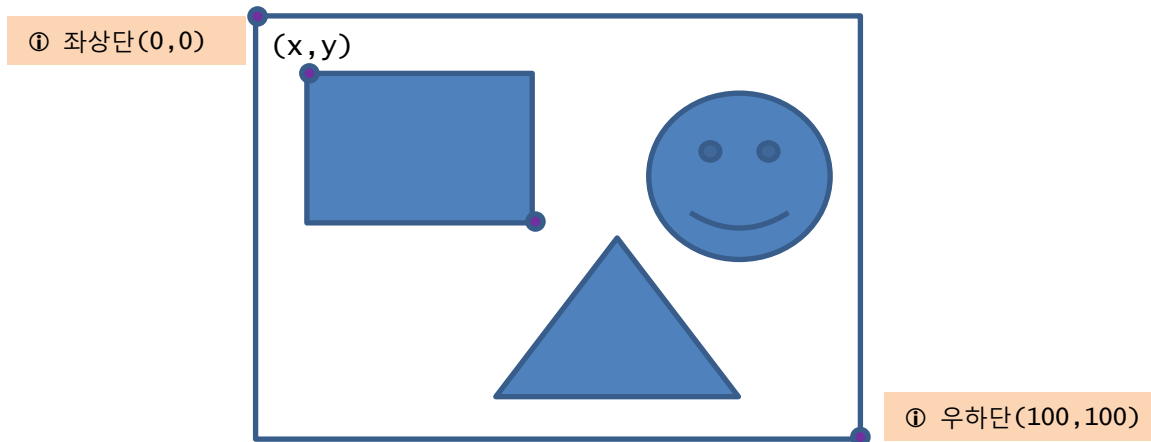
- 데이터와 해당 데이터를 처리할 수 있는 함수들을 결합하여 하나의 단위로 묶는 것
- 캡슐화를 통해서 비로소 클래스가 프로그램의 부품처럼 사용될 수 있음
 - 캡슐화가 잘 안될 경우, 독립성을 명확하게 유지하기 어려움



정보은닉의 이해

• 정보은닉

- 멤버변수의 외부접근을 허용할 경우, 잘못된 값이 저장되는 문제 발생 가능
- 따라서, (private 선언을 통해) 멤버변수의 외부접근을 막는 것
- (예시) 그림판
 - 그림판의 좌 상단(0,0)과 우 하단(100,100)의 범위 내에서 그림을 그려야 함
 - 하나의 그림에서 좌표의 좌우정보는 서로 바뀌어 저장되면 안됨



정보은닉과 캡슐화



정보은닉의 이해

• 그림판의 구현 - 좌표 클래스, 직사각형 클래스

- Point의 멤버변수에는 0~100 이외의 값이 들어가지 못하도록 막을 수 없음
- Rectangle의 멤버변수에는 좌우정보가 서로 바뀌어 저장되지 못하도록 막을 수 없음

① 정보은닉
실패

```
class Point
{
public:
    int x;    // x좌표의 범위는 0이상 100이하
    int y;    // y좌표의 범위는 0이상 100이하
};
```

① 정보은닉
실패

```
class Rectangle
{
public:
    Point upLeft;
    Point lowRight;
public:
```

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

pos1 pos2

-2, 4 5, 9

복사

pos2 pos1

5, 9 -2, 4

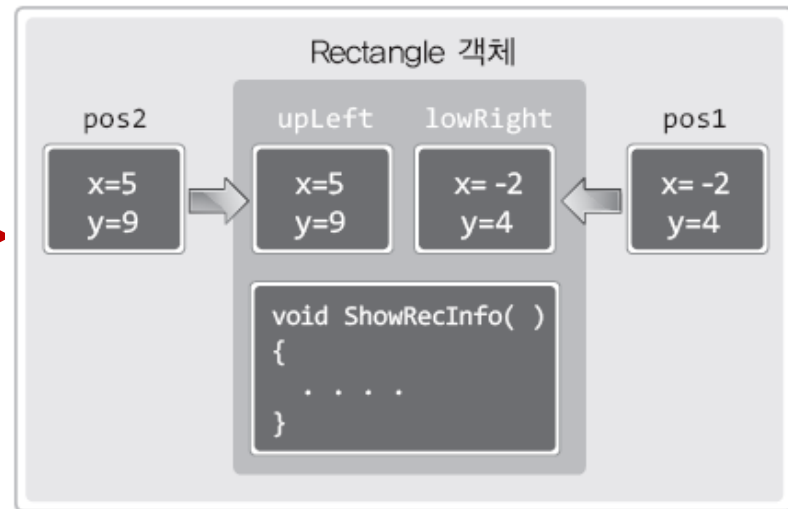
```
int main(void)
{
    Point pos1={-2, 4};
    Point pos2={5, 9};
    Rectangle rec={pos2, pos1};
    rec.ShowRecInfo();
    return 0;
}
```



정보은닉의 이해

- 직사각형(Rectangle) 객체의 이해
 - 클래스의 객체도 다른 객체의 멤버가 될 수 있음

```
int main(void)
{
    Point pos1={-2, 4};
    Point pos2={5, 9};
    Rectangle rec={pos2, pos1};
    rec.ShowRecInfo();
    return 0;
}
```



정보은닉과 캡슐화



정보은닉의 구현

• 좌표(Point) 클래스의 정보은닉

- 클래스의 멤버변수를 **private**으로 선언하고, 해당 변수에 접근하는 함수를 별도정의
- 이로써 안전한 형태의 멤버변수 접근을 유도함 즉, '정보은닉' 구현

```
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);
};
```

① 정보은닉

(1)

(2)

① 액세스함수

- ① GetX() 함수는 x값을 반환함. GetY() 함수는 y값을 반환함
- ① 이들은 객체안에서 멤버변수의 값이 변경되지 않도록 함
- ① 액세스함수는 정보은닉으로 인하여 추가됨

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

① 제어함수
SetX()

① x값 유입의 유일한 경로를 SetX함수가 되도록 함

① 액세스 함수 정의 규칙

- ① AAA 변수 선언 시,
 - GetAAA() 함수 만듦; 값 반환용
 - SetAAA() 함수 만듦; 값 저장용

정보은닉과 캡슐화



정보은닉의 구현

직사각형(Rectangle) 클래스의 정보은닉

- 클래스의 멤버변수를 **private**으로 선언하고, 해당 변수에 접근하는 함수를 별도정의
- 이로써 안전한 형태의 멤버변수 접근을 유도함 즉, '정보은닉' 구현

```
class Rectangle
{
(1) private:
    Point upLeft;
    Point lowRight;
    ① 정보은닉

public:
(2) bool InitMembers(const Point &ul, const Point &lr);
    void ShowRecInfo() const;
    ① 액세스함수
};
```

```
(2) bool Rectangle::InitMembers(const Point &ul, const Point &lr)
{
    if(ul.GetX()>lr.GetX() || ul.GetY()>lr.GetY())
    {
        cout<<"잘못된 위치정보 전달"<<endl;
        return false;
        ① 오류정보 알림
    }
    upLeft=ul;
    lowRight=lr;
    ① 값 저장
    return true;
}
```

① 좌상단, 우하단이 바뀌는 것을 차단

정보은닉과 캡슐화



정보은닉의 구현

- const 함수

- 정보은닉 구현 시, 프로그램의 안정성 위해 멤버함수에 const 선언
- ① const함수 내에서는 동일 클래스에 선언된 멤버변수 값을 변경하지 못함
- ② const함수는 const가 아닌 함수를 호출할 수 없음
- ③ const로 상수화된 객체를 대상으로는 const 멤버함수만 호출이 가능함

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

① 값변경불가

```
int GetNum()
{
    return num;
}

void ShowNum() const
{
    cout<<GetNum()<<endl; //컴파일에러 발생
}
```

① 호출불가

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

정보은닉과 캡슐화



캡슐화의 이해

- 캡슐화

- 데이터와 해당 데이터를 처리할 수 있는 함수들을 결합하여 하나의 단위로 묶는 것
- 즉, 관련 있는 기능을 수행하는 것들(클래스)을 하나의 클래스로 묶어줌



정보은닉과 캡슐화



캡슐화의 구현

• 캡슐화

- 관련 있는 기능을 수행하는 것들(클래스)을 하나의 클래스로 묶어줌
 - 사용 여부에 따라 캡슐화의 적절성은 달라질 수 있음
 - 조합의 제한이 많다면 캡슐화 실패할 수 있음

```
class SinivelCap    // 콧물 처치용 캡슐
{
public:
    void Take() const {cout<<"콧물이 짹~ 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;}
};
```

```
class CONTACT600    ① 코감기용 종합캡슐
{
private:             ① 작은 클래스 모두 포함
    SinivelCap sin;
    SneezeCap sne;
    SnuffleCap snu;

public:
    void Take() const
    {                ① 복용순서 고려
        sin.Take();
        sne.Take();
        snu.Take();
    }
};
```



캡슐화의 구현

• 캡슐화된 CONTAC600

- CONTAC600 클래스에 변경(복용순서 등)되더라도 이와 관련된 ColdPatient는 변경되지 않거나 변경되더라도 그 범위가 최소화됨

```
class CONTAC600 ① 코감기용 종합캡슐
{
private:
    SinivelCap sin;
    SneezeCap sne;
    SnuffleCap snu;

public:
    void Take() const
    {
        sin.Take();
        sne.Take();
        snu.Take();
    }
};
```

Cf. 캡슐화된 CONTAC600의 형태로 구현하지 않을 경우,
아래 함수를 모두 별도로 호출해야 하며 하나의 변경에도
모든 함수호출이나 내용이 함께 변경되어야 함

```
takeCONTAC600 (SinivelCap, SneezeCap, SnuffleCap);
sin.Take();
sne.Take();
snu.Take();
```

```
class ColdPatient ① 감기환자
{
public:
    void TakeCONTAC600(const CONTAC600 &cap) const { cap.Take(); }
};
```

① 종합감기약 복용 함수

① 복용순서는 환자가
직접 몰라도 됨

II. 생성자와 소멸자

생성자와 소멸자



생성자와 소멸자의 이해

- 생성자

- 함수의 일종으로, 객체가 생성될 때 자동으로 호출되는 함수
- 객체를 생성하고 InitMember함수로 초기화하는 역할을 대신함
- 즉, 생성자를 통해 객체를 생성과 동시에 초기화!

- 소멸자

- 객체가 소멸될 때 자동으로 호출되는 함수
- 정리가 필요한 연산자의 수행에 동반되므로 객체를 자동 소멸시킴

생성자와 소멸자



생성자의 이해

• 생성자의 조건

- ① 클래스의 이름과 동일한 이름의 함수
- ② 반환형을 선언하지 않음
- ③ 실제로 반환하지 않는 함수
- ④ 생성자는 객체 생성시 반드시 딱 한번만 호출됨 → **멤버변수 초기화에 사용**

```
class SimpleClass
{
private:
    int num;
public:
    SimpleClass(int n) // 생성자(constructor)
    {
        num=n;
    }
    int GetNum() const
    {
        return num;
    }
};
```

④

① sc객체생성
① 멤버변수 num을 20으로 초기화

① 스택할당

```
SimpleClass sc(20)
```

// 생성자에 20을 전달

① (힙)동적할당

```
SimpleClass * ptr = new SimpleClass(30);
```

// 생성자에 30을 전달

생성자와 소멸자



생성자의 이해

• 생성자의 조건

⑤ 생성자도 오버로딩 및 함수의 디폴트값 설정이 가능

- 선언된 생성자 없다면, 디폴트생성자 (void형 생성자)가 컴파일러에 의해 자동 호출됨

```
SimpleClass() { }
```

① 함수 원형 선언의 형태는 불가!

① 함수 오버로딩

⑤

① 디폴트값 설정

```
SimpleClass(int n1=0, int n2=0)
{
    num1=n1;
    num2=n2;
}
```

SimpleClass()

```
{
    num1=0;
    num2=0;
}
```

SimpleClass(int n)

```
{
    num1=n;
    num2=0;
}
```

SimpleClass(int n1, int n2)

```
{
    num1=n1;
    num2=n2;
}
```

```
SimpleClass sc1(); (x)
SimpleClass sc1; (o)
SimpleClass *ptr1=new SimpleClass; (o)
SimpleClass *ptr1=new SimpleClass(); (o)
```

```
SimpleClass sc2(100);
SimpleClass *ptr2=new SimpleClass(100);
```

```
SimpleClass sc3(100,200);
SimpleClass *ptr3=new SimpleClass(100,200);
```

생성자와 소멸자



생성자의 이해

• 디폴트 생성자

- 생성자를 정의하지 않을 경우, 컴파일러에 의해 자동 생성되는 생성자
 - 인자도 받지 않고, 하는 일도 없는 형태
- 따라서, 모든 객체는 무조건 생성자의 호출 과정을 거쳐 완성됨

```
class AAA
{
private:
    int num;
public:
    int GetNum { return num; }
};
```

자동생성



```
class AAA
{
private:
    int num;
public:
    AAA(){ } // 디폴트 생성자
    int GetNum { return num; }
};
```

생성자와 소멸자



생성자의 이해

• 디폴트 생성자

- 이미 생성자를 정의한 경우, 디폴트 생성자는 자동 할당되지 않음
- 따라서, 인자 없는 void형 생성자의 호출은 불가능

```
class SoSimple
{
private:
    int num;
public:
    SoSimple(int n) : num(n) { }
};
```

```
SoSimple simObj1(10);           (o)
SoSimple *simPtr1=new SoSimple(2); (o)
```

```
SoSimple simObj2;               (x)
SoSimple *simPtr2=new SoSimple; (x)
```

⇒ 이런 형태의 객체생성을 위한 생성자 추가 필요

```
SoSimple() : num(0) { }
```

생성자와 소멸자



생성자의 이해

- Private 생성자

- 생성자가 public이 아닌 private에 존재하는 경우, 클래스 내에서만 호출 가능
- 즉, 객체 안의 멤버함수를 통해서만 객체 생성 가능

```
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) {}
};
```

① 클래스 내부에서 private 생성자 호출이 가능

② 클래스 외부에서 이 생성자 호출을 통해 객체생성 불가능

생성자와 소멸자



생성자의 구현

- Point, Rectangle 클래스의 생성자 구현

- Point 클래스의 생성자

```
Point::Point(const int &xpos, const int &ypos)
{
    x = xpos;
    y = ypos;
}
```

```
class Point
{
private:
    int x;
    int y;
public:
    Point(const int &xpos, const int &ypos); // 인자 있는 생성자
    int GetX() const;
    int GetY() const;
    bool SetX(int xpos);
    bool SetY(int ypos);
};
```

생성자와 소멸자



생성자의 구현

- Point, Rectangle 클래스의 생성자 구현

- Rectangle 클래스의 생성자 → **이니셜라이저 필요**

- Point 생성자는 인자 있는 기본생성자로 지정됨 -> void형 Point 생성자가 자동 생성 안됨
- 따라서 Rectangle 객체 생성시 필요한 void형 Point 형태로 UpLeft, LowRight 객체생성 불가

```
class Rectangle
{
private:
    Point upLeft;
    Point lowRight;
public:
    Rectangle(const int &x1, const int &y1, const int &x2, const int &y2);
    void ShowRecInfo() const;
};
```

① 인자없는 void형 Point 생성자 필요 -> 호출할 생성자를 명시할 수 없음

upLeft

lowRight

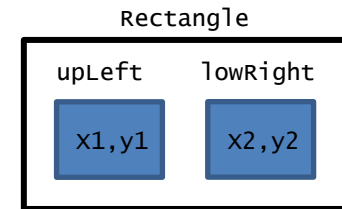
생성자와 소멸자



생성자의 구현

- **멤버 이니셜라이저 이용한 Rectangle 클래스 멤버 초기화(생성자 호출)**
 - **멤버 이니셜라이저는 함수의 정의부에 명시**
 - Void형 생성자가 없어 객체 생성이 실패하는 경우, 이니셜라이저로 인자를 전달하여 객체생성을 완료함
 - 앞의 예시에서, Point 클래스의 인자 있는 생성자를 호출하여 객체생성이 완료됨
 - **(객체생성 과정)**

- 1단계: (객체를 생성할) 메모리 공간을 할당
- 2단계: 이니셜라이저를 이용하여 멤버변수(객체)를 초기화
- 3단계: 생성자의 몸체부분을 실행



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

① 객체 upLeft의 생성과정에서, x1과 y1을 인자로 전달받는 (Point클래스의) 생성자를 호출하라!

① 객체 lowRight의 생성과정에서, x2와 y2를 인자로 전달받는 (Point클래스의) 생성자를 호출하라!

생성자와 소멸자



생성자의 구현

- 이니셜라이저 이용한 변수 및 상수 초기화
 - 이니셜라이저를 통해 멤버변수의 초기화 수행
 - 선언과 동시에 초기화되는 형태로 바이너리 구성
 - 따라서, const 상수와 선언된 멤버변수 초기화도 가능!

```
class SoSimple
{
private:
    int num1;
    int num2;
public:
    SoSimple(int n1, int n2) : num1(n1)
    {
        num2=n2;
    }
    . . . . .
};
```

① int num1 = n1;

① int num2;
num2 = n2;

```
class FruitSeller
{
private:
    const int APPLE_PRICE;
    int numOfApples;
    int myMoney;
public:
    FruitSeller(int price, int num, int money)
        : APPLE_PRICE(price), numOfApples(num), myMoney(money)
    {
        const int APPLE_PRICE = price;
    }
};
```


생성자와 소멸자



생성자의 구현

- 이니셜라이저 이용한 참조자 초기화

- 이니셜라이저 초기화는 선언과 동시에 초기화되는 형태 → 참조자 초기화도 가능
- 따라서 멤버변수로 참조자 선언이 가능함!

```
class BBB
{
private:
    AAA &ref;
    const int &num;
public:
    BBB(AAA &r, const int &n)
        : ref(r), num(n)
    { // empty constructor body
    }
```

Cf. 참조자 특성

```
int &ref;           (x)
int &ref = 10;      (x)
int &ref = Null;    (x)

int &ref = Num;     (o)
```

생성자와 소멸자



소멸자의 이해

• 소멸자

- 객체가 소멸될 때 자동으로 호출 → ①~클래스명(), ②인자 없음, ③반환값 없음

```
class AAA
{
    // empty class
};
```

```
~AAA() { }
```

- (생성자와 동일하게) 소멸자가 정의되지 않은 경우, 디폴트 소멸자가 삽입됨

```
class AAA
{
public:
    AAA() { }
    ~AAA() { }
};
```

① 디폴트 소멸자

생성자와 소멸자



소멸자의 활용

• 소멸자

- 생성자에서 할당한 메모리 공간을 소멸
- 오버로딩 불가
 - 무조건 인자값을 받지 않도록 정의함

① Person::Person

① Person::~~Person

```
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;
```

```
}
```

```
    ~Person()
```

```
{
```

```
        delete []name;
```

```
        cout<<"called destructor!"<<endl;
```

```
}
```

① 생성된 메모리공간 소멸
① 이름 name삭제

III. 실습

복소수를 클래스로 설계하기

```
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 }
```

실행 결과

```
( 2 + 5i )
( 2 + 5i )
```

private 멤버 성격 파악하기

```
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.real = 5; //컴파일 에러
27     x.image = 10; //컴파일 에러
28     y.SetComplex();
29     y.ShowComplex();
30 }
```

```
error C2248: 'Complex::real' : private 멤버('Complex' 클래스에서 선언)에 액세스할 수 없습니다.
error C2248: 'Complex::image' : private 멤버('Complex' 클래스에서 선언)에 액세스할 수 없습니다.
IntelliSense: 멤버 "Complex::real" (선언됨 줄 6)에 액세스할 수 없습니다.
IntelliSense: 멤버 "Complex::image" (선언됨 줄 7)에 액세스할 수 없습니다.
```

private 멤버를 다루기 위한 멤버함수 추가하기

```
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     void SetReal(int r);
12     void SetImage(int i);
13 };
14
15 void Complex::SetComplex()
16 {
17     real=2;
18     image=5;
19 }
```

실행 결과

(5 + 10i)

```
20 void Complex::ShowComplex()
21 {
22     cout<<"( " <<real <<" + " <<image
        <<"i )" <<endl ;
23 }
24 void Complex::SetReal(int r)
25 {
26     real = r;
27 }
28 void Complex::SetImage(int i)
29 {
30     image = i;
31 }
32
33 void main()
34 {
35     Complex x;
36     x.SetReal(5);
37     x.SetImage(10);
38     x.ShowComplex();
39 }
```

private 멤버를 다루기 위한 멤버함수 추가하기

```

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     void SetReal(int r);
12     void SetImage(int i);
13 };
14
15 void Complex::SetComplex()
16 {
17     real=2;
18     image=5;
19 }
    
```

실행 결과

(5 + 10i)

```

20 void Complex::ShowComplex()
21 {
22     cout<<"( " <<real <<" + " <<image
    << "i )" <<endl ;
23 }
24
25 {
26
27 }
28
29 {
30
31 }
32
33 void main()
34 {
35     Complex x;
36     x.SetReal(5);
37     x.SetImage(10);
38     x.ShowComplex();
39 }
    
```


인라인 함수 사용하기

```
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     {
11         real=2;
12         image=5;
13     }
14     void ShowComplex();
15 };
16
17
```

```
18 inline void Complex::ShowComplex()
19 {
20     cout<<" ( " <<real <<" + " <<image
        << "i )" <<endl ;
21 }
22 void main()
23 {
24     Complex x;
25
26     x.SetComplex();
27     x.ShowComplex();
28 }
```

실행 결과

(2 + 5i)

const 멤버함수 사용하기

```
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() const;
11 };
12
13 void Complex::SetComplex()
14 {
15     real=2;
16     image=5;
17 }
18
19 void Complex::ShowComplex() const
20 {
21     cout<<"( " <<real <<" + " <<image
        <<"i )" <<endl ;
22 }
```

```
23
24 void main()
25 {
26     Complex x;
27
28     x.SetComplex();
29     x.ShowComplex();
30 }
```

실행 결과

(2 + 5i)

매개변수가 없는 생성자 작성하기

```
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 }
```

실행 결과

(5 + 20i)

다양한 초기값의 매개변수를 사용하는 생성자 작성하기

생성자 오버로딩하기

```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private:
06     int real;
07     int image;
08 public:
09     Complex();
10     Complex(int r, int i);
11     void ShowComplex() const;
12 };
13 Complex::Complex()
14 {
15     real=0;
16     image=0;
17 }
18
19 Complex::Complex(int r, int i)
20 {
```

```
21     real=r;
22     image=i;
23 }
24
25 void Complex::ShowComplex() const
26 {
27     cout<<"( " <<real <<" + " <<image
28         << "i )" <<endl ;
29 }
30 void main()
31 {
32     Complex x(10, 20);
33     Complex y(30, 40);
34     Complex z;
35     x.ShowComplex();
36     y.ShowComplex();
37     z.ShowComplex();
38 }
```

실행 결과

```
( 10 + 20i )
( 30 + 40i )
( 0 + 0i )
```

생성자의 기본 매개변수값 설정하기

```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private:
06     int real;
07     int image;
08 public:
09     Complex(int r=0, int i=0);
10     void ShowComplex() const;
11 };
12
13 Complex::Complex(int r, int i)
14 {
15     real=r;
16     image=i;
17 }
18
```

```
19 void Complex::ShowComplex() const
20 {
21     cout<<"( " <<real <<" + " <<image
22         << "i )" <<endl ;
23 }
24
25 void main()
26 {
27     Complex x(10, 20);
28     Complex y(30);
29     Complex z;
30     x.ShowComplex();
31     y.ShowComplex();
32     z.ShowComplex();
33 }
```

실행 결과

```
( 10 + 20i )
( 30 + 0i )
( 0 + 0i )
```

생성자의 기본 매개변수값 설정하기

```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private:
06     int real;
07     int image;
08 public:
09     Complex(int r=0, int i=0);
10     void ShowComplex() const;
11 };
12
13 Complex::Complex(
14 {
15
16
17 }
18
```

```
19 void Complex::ShowComplex() const
20 {
21     cout<<"( " <<real <<" + " <<image
22         << "i )" <<endl ;
23 }
24
25 void main()
26 {
27     Complex x(10, 20);
28     Complex y(30);
29     Complex z;
30     x.ShowComplex();
31     y.ShowComplex();
32     z.ShowComplex();
33 }
```

실행 결과

```
( 10 + 20i )
( 30 + 0i )
( 0 + 0i )
```

생성자 콜론 초기화하기(이니셜라이저 활용)

```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private:
06     int real;
07     int image;
08 public:
09     Complex(int r=0, int i=0);
10     void ShowComplex() const;
11 };
```

```
12
13 Complex::Complex(int r, int i) : real(r),
    image(i)
14 {
15 }
```

```
16
```

```
17 void Complex::ShowComplex() const
18 {
19     cout<<"( " <<real <<" + " <<image
        << "i )" <<endl ;
20 }
21
22 void main()
23 {
24     Complex x(10, 20);
25     Complex y(30);
26     Complex z;
27     x.ShowComplex();
28     y.ShowComplex();
29     z.ShowComplex();
30 }
```

실행 결과

```
( 10 + 20i )
( 30 + 0i )
( 0 + 0i )
```


생성자 콜론 초기화하기(이니셜라이저 활용)

```
01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private:
06     int real;
07     int image;
08 public:
09     Complex(int r=0, int i=0);
10     void ShowComplex() const;
11 };
```

```
12
13 Complex::Complex
14 {
15 }
16
```

```
17 void Complex::ShowComplex() const
18 {
19     cout<<"( " <<real <<" + " <<image
20         <<"i )" <<endl ;
21 }
22 void main()
23 {
24     Complex x(10, 20);
25     Complex y(30);
26     Complex z;
27     x.ShowComplex();
28     y.ShowComplex();
29     z.ShowComplex();
30 }
```

실행 결과

```
( 10 + 20i )
( 30 + 0i )
( 0 + 0i )
```

소멸자 정의하기

```

01 #include <iostream>
02 using namespace std;
03 class Complex
04 {
05 private:
06     int real;
07     int image;
08 public:
09     Complex(int r=0, int i=0);
10     ~Complex();
11     void ShowComplex() const;
12 };
13
14 Complex::Complex(int r, int i) : real(r),
    image(i)
15 {
16 }
17

```

실행 결과

```

( 10 + 20i )
( 30 + 0i )
( 0 + 0i )
소멸자가 호출된다.
소멸자가 호출된다.
소멸자가 호출된다.

```

```

18 Complex::~~Complex()
19 {
20     cout<<"소멸자가 호출된다. \n";
21 }
22
23 void Complex::ShowComplex() const
24 {
25     cout<<"( " <<real <<" + " <<image
        <<"i )" <<endl ;
26 }
27
28 void main()
29 {
30     Complex x(10, 20);
31     Complex y(30);
32     Complex z;
33     x.ShowComplex();
34     y.ShowComplex();
35     z.ShowComplex();
36 }

```

헤더파일 만들기 - Car.h, Car.cpp, Racing Main.cpp

- 클래스의 선언 - .h
 - 컴파일 정보로 사용
 - 클래스를 구성하는 외형적 구조

```
#ifndef __CAR_H__
#define __CAR_H__
...
#endif
```

- 클래스의 정의(멤버함수정의) - .cpp

- 컴파일 정보로 미사용
- 컴파일후, 링커에 의해 하나의 실행파일로 구성

```
#include "Car.h"
```

```
class Car
{
private:
    char gamerID[CAR_CONST::ID_LEN];
    int fuelGauge;
    int curSpeed;
public:
    void InitMembers(char * ID, int fuel);
    void ShowCarState();
    void Accel();
    void Break();
};
```

car.h

① 멤버함수 원형선언

```
void Car::InitMembers(char * ID, int fuel) { . . . . }
void Car::ShowCarState() { . . . . }
void Car::Accel() { . . . . }
void Car::Break() { . . . . }
```

① 멤버함수의 몸체 정의

car.cpp

에로가 발생하지 않도록 수정하기

```
01 #include <iostream>
02 using namespace std;
03 class CRect{
04     int left;
05     int top;
06     int right;
07     int bottom;
08 public :
09     void print();
10 };
11 void CRect::print()
12 {
13     cout << "( " << left << ", " << top
14     << ", " << right << ", " <<
15     bottom << " )" << endl ;
16 }
```

```
16 void main()
17 {
18     CRect rc;
19     rc.left=0;
20     rc.top=0;
21     rc.right=20;
22     rc.bottom='20';
23     rc.print();
24 }
```

멤버함수 SetRect()를 완성하기

```
01 #include <iostream>
02 using namespace std;
03 class CRect{
04     int left;
05     int top;
06     int right;
07     int bottom;
08 public :
09     void print();
10     void SetRect(int l, int t, int r, int b);
11 };
12 void CRect::print()
13 {
14     cout << "( " << left << ", " << top
15     << ", " << right << ", " <<
16     bottom << " )" << endl ;
17 }
```

```
17 void main()
18 {
19     CRect rc;
20     rc.SetRect(0, 0, 20, 20); //error
21     rc.print();
22 }
```

객체를 초기화할 수 있도록 생성자 오버로딩하기

```
01 #include <iostream>
02 using namespace std;
03 class CRect{
04     int left;
05     int top;
06     int right;
07     int bottom;
08 public :
09     void prn();
10 };
11 void CRect::prn()
12 {
13     cout << "( " << left << ", " << top
14     << ", " << right << ", " <<
15     bottom << " )" << endl ;
16 }
```

```
17 void main()
18 {
19     CRect rc(0, 0, 20, 20);
20     CRect rc2(20, 20);
21     CRect rc3;
22     rc.prn();
23     rc2.prn();
24     rc3.prn();
25 }
```

생성자에 기본 매개변수 값으로 할당받도록 정의하기

```
01 #include <iostream>
02 using namespace std;
03 class CRect{
04     int left;
05     int top;
06     int right;
07     int bottom;
08 public :
09     void prn();
10 };
11
12 void CRect::prn()
13 {
14     cout << "( " << left << ", " << top
15         << ", " << right << ", " <<
16     bottom << " )" << endl ;
17 }
```

```
17
18 void main()
19 {
20     CRect rc( 20, 20, 20, 20 );
21     CRect rc2( 20 ,20, 20 );
22     CRect rc3( 20, 20 );
23     CRect rc4( 20 );
24     CRect rc5;
25
26     rc.prn();
27     rc2.prn();
28     rc3.prn();
29     rc4.prn();
30     rc5.prn();
31 }
```

과제6



파일명 “과제6_분반_학번_이름”으로 제출

- 과제6

- 실습6-10 ~ 6-14 : 소스코드(.cpp or .h), 실행결과화면(.jpg) 각각 생성

- 제출 시 주의사항

- 실행결과 마지막에는 “학과, 학년, 분반, 학번, 이름” 출력할 것
 - (형식: 컴퓨터공학부 2학년 1반, 2017000번, 홍길동입니다.)
 - 각 예제의 소스코드(*.cpp)와 실행결과화면(*.jpg 등)를 한 개의 zip파일로 만들어 제출할 것



감사합니다

mnshim@sungkyul.ac.kr

