Ch8. 클래스와 객체 숙달하기

Jong-Hyeok Park
akindo19@gmail.com

클래스 정의

- 선언
  - 변수의 타입 또는 함수의 프로토타입을 컴파일러에 미리 알려주는 것
  - 클래스의 메소드를 "**선언**"
- 정의
  - 실행파일에 실체를 작성하는 것
  - 클래스를 "**정의**" 하는 헤더파일
- 구현
  - 추상적으로 정의된 대상을 구체화하는 것
  - 메소드를 정의하는 코드는 소스파일에 "**구현**"한다.

클래스 정의

- 클래스 멤버
  - 멤버 함수: 메서드, 생성자, 소멸자
  - 멤버 변수: 데이터 멤버
  - 객체 단위로 적용
    - 예외: 정적 멤버는 **클래스 단위**로 적용

```
class XXX
{
public:
    void setValue(double inValue);
    double getValue() const;
private:
    double mValue;
}
```

클래스 정의

- 접근 제어
  - 따로 명시하지 않으면, private로 적용
  - struct == default 접근 제한자가 public 인 class
  - 접근 제한자 순서/반복 상관없이 사용 가능

- 접근 제한자
  - public: 메소드, 게터, 세터
  - protected: 외부 클라이언트가 사용하면 안되는 헬퍼 메서드
  - private: default, 데이터 멤버

메서드 정의

- 스코프 지정 연산자 ::
- 클래스 내 멤버, 다른 메소드 호출 가능
- this 포인터
  - 메소드 호출시, 메소드가 속한 객체의 포인터 **this** 가 숨겨진 매개변수 형태로 전달됨.

객체 사용법

- 스택 vs. 힙
  - 스마트 포인터 !!!

```
XXX mmm;
mmm.setValue(7);
```

스택에 생성한 객체

```
auto zzz = make_unique<XXX>();
XXX* nnn = new XXX();
nnn->setValue(5);
zzz->setValue(5);
```

힙에 생성한 객체

객체의 라이프 사이클

- 생성
- 대입 (할당)
- 소멸 (제거)

생성자

- 생성자 (ctor)
  - 주의 : 생성자는 선언과 동시에 또는 선언한 뒤에 호출하면 안됨!s

```
class XXX
{
    public:
    XXX(double inValue);
}

XXX::XXX(double inValue)
{
    setValue(inValue);
}
```

```
XXX test(3), test2(5);

// wrong example #1
XXX test;
test.XXX(3);

// wrong example #2
XXX test.XXX(3);
```

생성자

- 오버로딩
  - 컴파일러가 호출 시점에 매개변수 타입 일치하는 함수 선택
  - 생성자 안에서 다른 생성자 호출 가능 (*위임 생성자*)

```
class XXX
{
public:
    XXX(double inValue);
    XXX(string stringValue);
}
```

생성자

- 디폴트 생성자
  - 아무런 인수 받지 않는 생성자
  - 0인수 (0-argument) 생성자
  - 컴파일러가 자동으로 생성
    - 클래스 객체 멤버에 대해서도 디폴트 생성자 호출해줌.
    - 단, 사용자가 생성자를 1개라도 선언하면 컴파일러는 생성하지 않음.

```
class XXX
{
public:
    XXX();
}

XXX::XXX()
{
    mValue = 0;
}
```

```
// problem without default ctor
XXX cells[3];
XXX *myCellp = new XXX[3];

// solution
XXX cells[3] = {XXX(0), XXX(13),
XXX(23)};
```

생성자

- 디폴트 생성자
  - Most vexing parse

```
XXX myCell;
myCell.setValue(7);
```

```
XXX myCell();
myCell.setValue(7); // compile error
```

- 명시적 디폴트 생성자

```
class XXX
{
public:
    XXX() default;
    XXX(double inValue);
    XXX(string stringValue);
}
```

- 명시적 삭제된 생성자
  - static 메소드로만 구성된 클래스

```
class XXX
{
public:
    XXX() delete;
}
```

생성자

- 생성자 이니셜라이저
  - 객체 생성 시점에 데이터 멤버를 바로 초기화할 수 있음.
  - 나중에 값을 따로 대입하는 것보다 효율적!
    - const 데이터 멤버
    - 레퍼런스 데이터 멤버
    - 디폴트 생성자가 정의되지 않은 객체 데이터 멤버
    - 디폴트 생성자가 없는 베이스 클래스

```
XXX::XXX(double
inValue):mValue(inValue)
{
}
```

생성자

- 복제 생성자
  - 컴파일러가 자동으로 생성해 줌.
  - 생성자 이니셜라이져 사용 (기존 객체 데이터 멤버)

```
class XXX
{
public:
    XXX(const XXX& src);
}
```

생성자

- 복제 생성자
  - 명시적 호출

```
XXX myCell(3);
XXX myCell2(myCell);
```

  - 레퍼런스 객체 전달
    - const 레퍼런스로 전달하는 것이 성능 좋음.

  - 명시적 디폴트/삭제된 복제 생성자
    - 삭제: 객체를 값으로 전달하지 않게 할 때 사용

```
class XXX
{
public:
    XXX(const XXX& src) = default;
    XXX(const XXX& src) = delete;
}
```

생성자

- 이니셜라이저 리스트 생성자
  - std::initializer_list<T>를 첫번째 매개변수로 받고, 다른 매개변수는 없거나, 디폴트값을 가진 매개변수를 추가로 받는 생성자

```
class XXX {

XXX(initializer_lists<double> args)
{
    if (args.size()%2 != 0) {
        throw
        invalid_argument("initializer
        _list should"
        "contain even number of
        elements");
    }
}
    /* ... */
};
```

```
XXX sample = {1.0, 2.0, 3.0};
```

생성자

- 위임 생성자
  - 같은 클래스의 다른 생성자를 생성자 안에서 호출
  - 반드시 생성자 이니셜라이저에서만 호출해야함.

```
XXX::XXX(string_view initValue)
: XXX(stringToDouble(initValue))
{
}
```

```
class MyClass
{
    // wrong example
    MyClass(char c) : MyClass(1.2) {}
    MyClass(double d) : MyClass('m')
    {}
}
```

생성자

- 컴파일러가 생성하는 생성자

직접 정의한 생성자	컴파일러가 만들어주는 생성자	객체 생성 방법

없음	디폴트 생성자
	복제 생성자
		인수가 없는 경우:
		`SpreadsheetCell cell;`
		다른 객체를 복제하는 경우:
		`SpreadsheetCell myCell(cell);`

디폴트 생성자만 정의한 경우	복제 생성자	인수가 없는 경우:
		`SpreadsheetCell cell;`
		다른 객체를 복제하는 경우:
		`SpreadsheetCell myCell(cell);`

복제 생성자만 정의한 경우	없음	이론적으로는 다른 객체를 복제할 수 있지만 실제로는 어떠한 객체도 생성할 수 없다. 복제 방식을 사용하지 않는 생성자가 없기 때문이다.

한 개의 인수 또는 여러 개의 인수를 받는, 복제 생성자가 아닌 생성자만 정의한 경우	복제 생성자	인수가 있는 경우:
		`SpreadsheetCell cell(6);`
		다른 객체를 복제하는 경우:
		`SpreadsheetCell myCell(cell);`

한 개의 인수 또는 여러 개의 인수를 받는, 복제 생성자가 아닌 생성자 또는 디폴트 생성자 하나만 정의한 경우	복제 생성자	인수가 없는 경우:
		`SpreadsheetCell cell;`
		인수가 있는 경우:
		`SpreadsheetCell cell(5);`
		다른 객체를 복제하는 경우:
		`SpreadsheetCell anotherCell(cell);`

객체의 소멸

- 객체의 소멸
  - 스택 객체 : 스코프 (유효범위)를 벗어날 때 자동으로 삭제
  - 힙 객체 : delete 명시 또는 스마트 포인터
  - 객체 삭제 순서 == 생성 순서 **반대**

```
int main() {
    XXX myCell(5);
    if (myCell.getValue() == 5) {
        XXX myCell2 anotherCell(7);
    }
    // anotherCell 삭제
    cout << myCell.getValue();
    return 0;
}
```

```
int main() {
    XXX* myCellPtr1 = new XXX(5);
    XXX* myCellPtr2 = new XXX(7);
    cout << myCellPtr1->getValue();
    delete myCellPtr1;
    myCellPtr1=nullptr;
    // myCellPtr2는 안지워짐!
    return 0;
}
```

객체의 대입

- 대입 연산자
  - 클래스마다 대입을 수행하는 메서드
  - 복제 대입 연산자: 좌변과 우변에 있는 객체가 대입 후에도 남음.
  - 이동 대입 연산자: 이동 후 우변에 있는 객체 삭제

```
class XXX
{
public:
    XXX& operator=(const XXX& rhs);
}

myCell = anotherCell = aThirdCell;
```

객체의 대입

- 대입 연산자
  - 명시적 디폴트/삭제된 대입 연산자

```
XXX& XXX::operator=(const XXX& rhs)
{
    if (this == &rhs) {
    return *this;
    }
    mValue = rhs.mValue;
    return *this;
}
```

```
class XXX
{
public:
    XXX& operator=(const XXX& rhs)
    = default;
    XXX& operator=(const XXX& rhs)
    = delete;
}
```

객체의 대입

- 복제 생성자 vs. 대입 연산자
  - 복제 생성자는 초기화할 때 단 한번만 호출됨.
  - C++ 객체에 자기 자신을 대입할 수 있음. (this로 비교)
  - 이미 생성된 객체는 대입 연산자 사용

```
// 복제 생성자 사용
XXX myCell(5);
XXX anotherCell(myCell);
XXX aThirdCell = myCell;

// 대입 연산자 사용
anotherCell = myCell;
```

References

[1] Marc Gregoire, 2018, Professional C++, 4th edition, WILEY