[temp] 항목 7. 다형성을 가진 기본 클래스에서는 소멸자를 반드시 가상 소멸자로 선언하자

가상소멸자가 필요한 이유에 대한 추상화 된 설명

책에서는 factory 함수를 예를 들어 설명했으나, 먼저 좀 더 추상화 된 예시를 통해 이해해보자.

아래 코드에서 만약 classA 선언부에 가상소멸자를 사용하지 않았다면, A를 delete 했으나 classA에 대한 소멸자만 호출되기 때문에 classB 부분에 있는 자원이 소멸되지 않아 문제가 발생한다.

https://silisian.tistory.com/53

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

class classA
{
    public:
      classA(){  cout << "A 생성" << endl;}
       virtual ~classA(){  cout << "~A 소멸" << endl;}
};

class classB : public classA
{
    public:
      classB(){cout << "B 생성"<< endl;}
      ~classB(){cout << "~B 소멸" << endl;}
};


int main()
{
  cout << "== 소멸자 테스트 시작 ==" << endl;
  classB *B = new classB;
  classA *A = B;
  delete A;
  return 0;
}
```

Colored by Color Scripter

가상소멸자가 필요한 이유 팩토리 함수에서 설명

https://0xd00d00.github.io/2021/08/05/effective_12.html

책에 설명된 코드를 이해해보자. 책에는 많은 부분이 생략되어있기 때문에 위 블로그의 코드를 참고해보겠습니다.

```
class TimeKeeper {
public:
```

```
        TimeKeeper();
        void currentTime() { ... } // 현재 시간을 얻는 함수.
        ...

        virtual void design() = 0;

        ~TimeKeeper();
};

class WristClock        : public TimeKeeper
{
        void design() {
                cout << "WristClock" << endl;
        }
};

// Factory 구성
class TimeKeeperFactory() {
public:
        void newClock(const string& name) {
                TimeKeeper *tk = getTimeKeeper();
                objPool[name] = tk;
        }

        virtual TimeKeeper* getTimeKeeper() = 0;

private:
        map<string, Timekeeper *> objPool;
};

class WristTimeKeeperFactory() : public TimeKeeperFactory {
        TimeKeeper* getTimeKeeper() {
                return new WristClock;
        }
};

int main() {
        WristTimeKeeperFactory wtkf;
        wtkf.newClock("apple watch");
}
```

위와 같이 구성된 코드에서 아래 코드가 수행된다면 상속받아 생성된 파생 class의 소멸자가 호출되지 않아 문제가 발생하게 됩니다.

```
...

TimeKeeper* getTimeKeeper() {
        return new WristClock;
}
...

// TimeKeeper 클래스 계통으로 동적할당
TimeKeeper *ptk = getTimeKeeper();

...     // 객체 사용

delete ptk;     // 객체 제거
```

기본 클래스로 의도 했는데, 가상소멸자를 안쓰면 안되는 것은 알겠음.

그렇다면 기본 클래스로 의도 하지 않았더라도, 일단 가상소멸자를 쓰고보면 어떨까?

```
class Point {
public:
        Point(int xCoord, int yCoord);
        ~Point();

private:
        int x, y;
};
```

여기서 point를 위와 같이 정의하고, int가 32bit를 차지한다고 가정하면 point 객체는 64비트 레지스터에 딱 맞게 들어갈 수 있다. 하지만 가상함수를 구현하려면 여기에 별도의 자료구조(가상함수 테이블 포인터)가 들어가야한다. 즉 64비트 레지스터에 딱 맞게 들어갈 수 없어 객체의 크기가 50%~100%까지 커지게된다.

- 32비트 아키텍처 : 64비트 → 96비트
- 64비트 아키텍쳐 : 64비트 → 128비트

그럼 가상함수를 쓰면 일단 소멸자를 virtual로 선언하면 되겠다!(가상함수가 있다는 것은 기본 클래스로 사용하기 위해 설계 되었다는 것이기 때문)

주의사항!!

STL 컨테이너 타입(vector, list, set, tr1::unodered_map 등]은 가상소멸자가 없어서 상속받아 사용하면 아래 코드에서 문제가 발생할 수 있다.

```
class SpecialString::public std::string{...};

SpecialString *pss = new SpecialString("Impending Doom");
std::string *ps;
...
ps =pss;
...
delete ps;
```

위 코드에서 SpecialString의 소멸자가 호출되지 않아 *ps의 SpecialString부분의 자원이 누출됨.

순수가상소멸자

- 추상클래스 설명 : https://hwan-shell.tistory.com/223

어떤 class가 추상클래스였으면 하는데, 마땅히 넣을만한 순수 가상함수가 없으면 순수 가상 함수를 만들자!

  - 추상클래스는 기본클래스로 쓰이기 위해 만든 것임
  - 기본 클래스는 가상소멸자를 가져야함.

위 두가지를 만족시키기 위해 순수 가상 함수를 만들면 된다. 이 때는 순수 가상 소멸자의 정의를 두지 않으면 안된다.

소멸자가 동작하는 순서 : 상속 계통 구조에서 가장 말단의 derived class의 소멸자 -> 다음 소멸자 -> .. base class 소멸자

컴파일러는 ~AWOV의 호출 코드 을 만들 것인데 이 때 링커 에러가 안날려면 정의를 꼭 만들어줘야 한다.

이것만은 잊지 말자!

- 다형성을 가진 기반 클래스에는 반드시 가상 소멸자를 선언하자. 즉, 어떤 클래스가 가상함수를 하나라도 가지고 있다면 반드시 가상소멸자를 넣어야한다.
- 기반클래스로 설계되어있지 않거나, 다형성을 갖도록 되어있지 않다면 "가상 소멸자"를 선언하지 말자!