**항목 7. 다형성을 가진 기본 클래스에서는 소멸자를 반드시 가상 소멸자로 선언하자.**

**이슈**

TimeKeeper 클래스를 만들어 기본 클래스로 지정한 후 적절한 용도로 파생

|  |
| --- |
| class TimeKeeper  {  public:  TimeKeeper() {}  ~TimeKeeper() {}  // ...  };  class AtomicClock : public TimeKeeper {};  class WaterClock : public TimeKeeper {};  class WristWatch : public TimeKeeper {}; |

어떤 시각 기록 객체에 대한 포인터를 손에 넣는 용도로 팩토리 함수를 선언

|  |
| --- |
| TimeKeeper\* getTimeKepper(); // Timekeeper에서 파생된 클래스를  // 통해 동적으로 할당된 객체의 포인터를 반환 |

getTimeKeeper 함수에서 반환되는 개체는 힙에 있으므로 **메모리 누수방지를 위해 delete 호출**

|  |
| --- |
| TimeKeeper \*ptk = getTimeKepper(); // TimeKeeper 클래스 계통으로부터  // 동적으로 할당된 객체를 얻습니다.  delete ptk; //자원 누수를 방지하기 위해 메모리 해제 |

getTimeKeeper 함수가 반환하는 포인터가 파생 클래스 (AtomicClock) 객체애 대한 포인터이며, 이 포인터 객체가 삭제될 때는 기본 클래스 포인터(TimeKeeper)를 통해서 삭제가 된다.

하지만 기본클래스에 들어 있는 소멸자가 비가상 소멸자(non-virtual destructor) 이므로 파생 클래스 객체인 AtomicClock의 소멸 과정은 호출되지 않으므로, 부분 소멸이 이루어짐

\* C++ 규정에 의하면, 기본 클래스 포인터를 통해 파생 클래스 객체가 삭제될 때, 그 기본 클래스에 비 가상 소멸자가 들어 있으면 프로그램 동작은 미정의 사항이라는 내용 존재

**해결 1.**

기본 클래스의 소멸자를 가상 소멸자로 변경

|  |
| --- |
| class Timekeeper  {  public:  TimeKeeper();  virtual ~TimeKeeper();  // ...  };  TimeKeeper \*ptk = getTimeKeeper();  delete ptk; // 정상 동작 |

**효과 1.**

**정상적인 메모리 삭제가 이루어짐**

가상 소멸자가 없는 클래스를 보면 ‘**저 클래스는 기본 클래스로 쓰일 의지가 없는 클래스**’로 간주

**이슈 2.**

**기본 클래스로 의도하지 않는 클래스에 대해 소멸자를 가상으로 선언 하는건 좋지 않다.**

1. 가상 함수를 C++에서 구현하려면 클래스에 별도의 자료구조 하나가 들어 가야함 (객체가 어떤 가상 함수를 호출 해야 하는지 결정하는데 쓰이는 정보)

2. 대게 vptr (가상 함수 테이블 포인터) 이라는 이름으로 불림

3. vptr은 가상함수의 주소, 포인터 들의 배열을 가리킴

4. 가상 함수 테이블 포인터의 배열은 vtbl(가상 함수 테이블) 이라 함

5. 어떤 객체에 대해 어떤 가상 함수가 호출되려면 호출되는 실제 함수는 그 객체의 bptr이 가리키는 vtbl에 따라 결정

결과적으로 가상함수가 들어가는 **클래스의 타입 객체의 크기가 커짐.**

해결 2.

1. **소멸자를 전부 virtual로 선언하는 것은 virtual로 절대 선언하지 않는 것만큼 부적절함**

2. 가상 소멸자를 선언하는 건 클래스에 **가상함수가 하나라도 있는 경우 사용**

3. **가상 소멸자가 없는 클래스를 기본 클래스로 사용하는 것을 주의**

|  |
| --- |
| class SpecialString : public std::string  {  }; |

std::string에는 가상 소멸자가 없으며, 그밖의 STL 컨테이너 타입 전부가 비가상 소멸자 클래스

해결 3.

**순수 가상 소멸자를 두어 편하게 쓸 수 있다**. 순수 가상함수는 해당 클래스를 추상 클래스(자체적으로 객체를 만들지 못하는 클래스)로 만드는데, 이 추상 클래스로 만들고 싶은 클래스에 순수 가상 소멸자를 선언하는 것

|  |
| --- |
| class AWOV  {  public:  virtual ~AWOV() = 0; // 순수 가상 소멸자를 선언  }; |

소멸자의 동작순서는 **가장 말단의 파생 클래스를 시작으로 기본 클래스로 차례로 올라가는 순서**를 가지며, 각 기본 클래스의 소멸자를 거치면 실행 된다.

**요약**

\* 다형성을 가진 기본 클래스에는 반드시 가상 소멸자를 선언해야 합니다. 즉, 어떤 클래스가 가상 함수를 하나라도 갖고 있으면, 이 클래스의 소멸자도 가상 소멸자이어야 합니다.

\* 기본 클래스로 설계되지 않았거나 다형성을 갖도록 설계되지 않은 클래스에는 가상 소멸자를 선언하지 말아야 합니다.