**항목 12. 객체의 모든 부분을 빠짐없이 복사하자.**

**이슈 1.**

**객체를 복사하는 함수를 복사 생성자와 복사 대입 연산자라고 부르며, 이 둘을 통틀어 객체 복사 함수라고 부른다**.

객체 복사함수는 컴파일러가 생성할 때도 있는데 저절로 만들어 져있는 함수라 할 지라도 기본적인 요구조건인 복사되는 개체가 갖고 있는 **데이터를 빠짐없이 복사한다는 것**이다.

객체 복사 함수를 사용자가 선언한다는 것은, 컴파일러가 만든 객체 복사 함수를 대신할 수 있어야 합니다.

고객을 나타내는 클래스가 있으며, 이 클래스의 복사 함수는 사용자가 직접 구현하였으며, 복사 함수를 호출할 때마다 로그를 남기도록 작성되었습니다.

|  |
| --- |
| void logCall(const std::string& funcName); // 로그 기록 내용을 만듬  class Customer {  public:  Customer(const Customer& rhs) : name(rhs.name)  {  logCall("customer copy constructor");  }  Customer& operator= (const Customer& rhs)  {  logCall("customer copy assignment operator");  name = rhs.name;  return \*this;  }  private:  std::string name;  }; |

문제가 없이 잘 동작하는 객체 복사 함수 들이지만 데이터 멤버 하나를 Customer클래스에 추가 한다면, 문제가 생길 수 있다.

|  |
| --- |
| void logCall(const std::string& funcName); // 로그 기록 내용을 만듬  class Customer {  public:  Customer(const Customer& rhs) : name(rhs.name)  {  logCall("customer copy constructor");  }  Customer& operator= (const Customer& rhs)  {  logCall("customer copy assignment operator");  name = rhs.name;  return \*this;  }  private:  std::string name;  Date lasTransaction; // 추가된 멤버 변수  }; |

이렇게 되면 복사 함수의 동작은 완전 복사가 아닌 부분 복사가 된다. **Customer의 name은 복사를 하지만 lastTransaction은 복사하지 않는다.**

**해결법 1.**

컴파일러는 경고의 메시지를 보내어 주지 않으며, **사용자가 추가한 객체 복사 함수는 사용자가 직접 추가한 데이터 멤버에 대해 복사 처리를 추가**해 주어야 한다.

이슈 2.

**클래스 상속의 경우 문제는 더욱 복잡해 질 수 있다.**

|  |
| --- |
| class PriorityCustomer : public Customer // 파생 클래스  {  public:  PriorityCustomer(const PriorityCustomer& rhs) : priority(rhs.priority)  {  logCall("PriorityCustomer copy constructor");  }  PriorityCustomer& operator=(const PriorityCustomer& rhs)  {  logCall("PriorityCustomer copy assignment operator");  priority = rhs.priority;  return \*this;  }  private:  int priority;  }; |

PriorityCustomer 클래스 복사 함수는 언뜻 보기엔 PriorityCustomer 의 모든 것을 복사하는 것처럼 보이나, 자세히 보면 기본 클래스인 **Customer로부터 상속받은 데이터 멤버들의 사본은 복사가 이루어 지지 않고 있습니다.**

이 경우 **Customer 의 멤버 변수들은 기본 초기화**가 되어 있는 상태가 된다.

**해결법 2.**

역시 사용자가 직접 선언한 객체 멤버 함수는 기본 클래스에 대한 부분이 빠져 있다면 **기본 클래스의 복사도 이루어 질 수 있도록 해주어야 한다.**

하지만 기본 클래스의 멤버는 private로 선언되어있을 가능성이 매우 크므로, 파생 클래스의 복사 함수안에서 기본 클래스의 복사 함수를 호출하도록 해야 한다.

|  |
| --- |
| class PriorityCustomer : public Customer // 파생 클래스  {  public:  PriorityCustomer(const PriorityCustomer& rhs) : priority(rhs.priority),  Customer(rhs) // 기본 클래스의 복사 생성자를 호출  {  logCall("PriorityCustomer copy constructor");  }  PriorityCustomer& operator=(const PriorityCustomer& rhs)  {  logCall("PriorityCustomer copy assignment operator");  Customer::operator=(rhs); // 기본 클래스의 복사 대입 연산자를 호출  priority = rhs.priority;  return \*this;  }  private:  int priority;  }; |

**객체의 복사 함수를 작성할 때는 다음의 두 가지를 꼭 명심**해야 한다.

1. **해당 클래스의 데이터 멤버를 모두 복사하고**

2. **이 클래스가 상속한 기본 클래스의 복사 함수도 호출해 주어야 한다.**

**추가.**

클래스의 **양대 복사 함수(복사 생성자와 복사 대입 연산자)는 코드 중복을 피하려는 생각으로 한쪽이 다른 한쪽을 호출하는 식으로 두 개의 복사 함수를 만드려는 경우가 종종 있다.**

결과부터 말하자면, 절대 안된다.

1. 복사 대입 연산자 에서 복사 생성자를 호출하는 것부터 말도 안 되는 발상일 수 잇다. **존재하는 객체를 다시 생성**하는 과정일 수 있으며, 특정한 조건에서 객체의 데이터가 손실 될 수 도 있음. **‘태어난 아이에게 엄마 배속으로 다시 들어 가라는 꼴’**

2. **복사 생성자에서 복사 대입연산자를 호출하는 것 또한 넌센스,** 생성자의 역할은 새로 만들어진 객체를 초기화 하는 것이지만 대입 연산자의 역할은 이미 초기화가 끝난 객체에게 값을 주는 것. **초기화된 객체에만 적용된다는 이야기**이다.

대신, **복사 생성자와 복사 대입 연산자의 코드 본문이 비슷한 부분을 별도의 멤버 함수로 만든 후 양쪽 모두에서 호출하여 중복 코드를 줄이는 방법을 사용하며, 함수는 private 멤버로 두자.**

**요약.**

**\*** 객체 복사 함수는 주어진 객체의 모든 데이터 멤버 및 모든 기본 클래스 부분을 빠뜨리지 말고 복사하자.

\* 클래스의 복사 함수 두 개를 구현할 때, 한쪽을 이용해서 다른 쪽을 구현하려는 시도는 절대로 하지 마세요. 그대신, 공통된 동작을 제 3의 함수에다 분리해 놓고 양쪽에서 이것을 호출하게 만들어서 해결합시다.