**항목 9. 객체 생성 및 소멸 과정 중에는 절대로 가상 함수를 호출하지 말자**

이슈.

**객체의 생성 및 소멸 동안 가상 함수들을 호출해서는 안된다**, 왜냐면 이 호출들은 사용자가 생각하는 것처럼 동작하지 않을 것이며, **설사 동작하더라고 예상치 못한 결과가 될 수 있기 때문**

매수 오더, 매도 오더들과 같은 주식 트랜잭션들을 모델링하기 위한 클래스 계층을 가지고 있다고 가정, 이러한 트랜잭션 들은 감사가 가능 해야 함이 중요하므로, 트랜잭션 객체가 생성될 때마다, 감사 로그에 적절한 엔트리가 생성될 필요가 있습니다.

|  |
| --- |
| class Transaction { // 모든 거래에 대한 기본 클래스  public:  Transaction();  virtual void logTransaction() const = 0; // 타입에 따라 달라지는 로그 기록을 만듬  //...  };  Transaction::Transaction() // 기본 클래스의 생성자 구현  {  //...  logTransaction(); // 이 거래를 로그 기록을 만들기 시작  }  class BuyTransaction : public Transaction { // Transaction의 파생 클래스  public:  virtual void logTransaction() const; // 이 타입에 따른 거래내역 로깅을 구현  //...  };  class SellTransaction : public Transaction { // Transaction의 파생 클래스  public:  virtual void logTransaction() const; // 이 타입에 따른 거래내역 로깅을 구현  //...  }; |

위 코드는 BuyTransaction b; 코드가 실행 될 때 어떤 일이 발생하는지 볼 필요가 있다.

BuyTransaction 생성자가 호출될 것이지만, 먼저 Transaction 생성자가 호출되어야 한다. 파생 클래스 부분이 생성되기 전에 파생 클래스 객체들이 생성된다. Transaction 생성자의 마지막 문장은 **가상 함수 logTransaction을 호출하지만, 여기서 문제가 발생**한다. 여기서 호출되는 **logTransaction의 버전은 BuyTransaction에 있는 버전이 아닌 Transaction에 있는 버전입니다**.

지금 생성되는 객체가 BuyTransaction 이라도 말이다.

\* **기본 클래스의 생성자가 호출될 동안에는 가상 함수는 절대로 파생 클래스 쪽으로 내려가지 않는다. 그 대신, 객체 자신이 기본 클래스 타입인 것처럼 동작한다**. 즉, **기본 클래스 생성 과정에서는 가상 함수가 먹히지 않는다.**

**이유**.

기본 클래스 생성자는 파생 클래스 생성자보다 앞서서 실행되기 때문에, 기본 클래스 생성자가 돌아가고 있을 시점에 **파생 클래스 데이터 멤버는 아직 초기화된 상태가 아니다.**

이때, 기본 클래스 생성자에서 어쩌다 호출된 가상 함수가 파생 클래스 쪽을 내려 간다면 치명적인 위험이 있을 수 있다. 그러므로 C++은 이런 실수를 하지 못하도록 막아 둔 것 이다.

**핵심**.

파생 클래스 객체의 기본 클래스 부분이 생성되는 동안은, **그 객체의 타입은 바로 기본 클래스**이며, 호출되는 가상 함수는 모두 기본 클래스의 것으로 결정될 뿐 아니라, **런타입 타입 정보를 사용하는 언어 요소 (ex dynamic\_cast) 를 사용한다해도 모두 기본 클래스 타입의 객체를 취급한다.**

그러므로, BuyTransaction 클래스만의 데이터는 아직 초기화된 상태가 아니므로, **아예 없던 것처럼 취급하는 편이 가장 안전**

**객체가 소멸될 때에도 동일한 원리가 적용된다**. 파생 클래스의 소멸자가 일단 호출되고 나면 파생 클래스만의 데이터 멤버는 정의되지 않은 값으로 가정하기 때문에, 이제부터 C++은 이들을 없는 것처럼 취급하며, 진행한다. 기본 클래스 소멸자에 진입할 당시의 객체는 기본 클래스 객체가 되며, 모든 C++ 기능들 역시 기본 클래스 객체를 기준으로 처리합니다.

생성자 혹은 소멸자 안에서 가상 함수가 호출 되는 지를 찾는 것은 쉬운 것이 아닙니다.

|  |
| --- |
| class Transaction {  public:  Transaction()  {  init(); // 비 가상 함수를 호출  }  virtual void logTransaction() const = 0;  //...  void init()  {  // ...  logTransaction(); // 가상 함수가 내포되어 있다.  }  }; |

위 코드는 컴파일에서 문제가 없으며, 링크에서도 이상이 없다는 결과를 뱉는다. 하지만 이 코드가 실행된다면, logTransaction은 Transaction 클래스 안에서 순수 가상 함수이기 때문에, 대부분의 시스템은 순수 가상 함수가 호출될 때 프로그램을 바로 끝내 버린다. 그렇지 않은 경우라도 logTransaction 함수가 호출 될 때 Transaction의 버전이 호출될 것이므로 예상치 못한 결과를 낼 수 있다.

**해결** 1.

logTransaction을 Transaction 클래스의 **비가상 멤버 함수로 변경하는 것**이다. 그러고 나서 파생클래스의 생성자들로 하여금 필요한 로그 정보를 Transaction의 생성자로 넘겨야 한다는 규칙을 만든다. **logTransaction이 비가상 함수 이기 때문에 Transaction의 생성자는 이 함수를 안전하게 호출할 수 있다.**

|  |
| --- |
| class Transaction {  public:  explicit Transaction(const std::string& logInfo);  void logTransaction(const std::string& logInfo) const; // 비가상 함수  ...  };  Transaction::Transaction(const std::string& logInfo)  {  ...  logTransaction(logInfo); // 비가상 함수  }  class BuyTransaction : public Transaction {  public:  BuyTransaction(parameters)  : Transaction(createLogString(parameters)) // 로그 정보를 기본 클래스 생성자로 넘김  {  ...  }  ...  private:  static std::string createLogString(parameters);  }; |

기본 클래스 부분이 생성될 때는 가상 함수를 호출한다 해도 기본 클래스의 울타리를 넘어 내려갈 수 없기 때문에, 필요한 초기화 정보를 파생 클래스 쪽에서 기본 **클래스 생성자로 ‘올려’주도록 만듦으로써 부족한 부분을 채울 수 있다는 것**

**요약**

\* 생성자 혹은 소멸자 안에서 가상 함수를 호출하지 마세요. 가상 함수라고 해도, 지금 실행중인 생성자나 소멸자에 해당되는 클래스의 파생 클래스 쪽으로는 내려가지 않습니다.