**항목 8. 예외가 소멸자를 떠나지 못하도록 붙들어 놓자**

**이슈.**

예 1.

|  |
| --- |
| class Widget  {  public:  // ...  ~Widget()  {  // 이 함수로 부터 예외가 발생  }  };  void doSomthing()  {  std::vector<Widget> v;  // ...  } |

vector 타입의 객체인 v는 소멸될 때, Widget 들 전부를 소멸하려고 한다.

**첫 번째 요소에서 예외가 발생하며, 나머지 소멸자를 호출하면 완전하지 못한 프로그램 종료나 미정의 동작이 있을 수 있음**

예 2.

데이터베이스 연결을 나타내는 클래스를 쓰고 있다고 가정

|  |
| --- |
| class DBConnection  {  public:  // ...  static DBConnection create(); // DBConnection 객체를 반환하는 함수  void close() // 연결을 닫음, **열결이 실패하면 예외**  }; |

사용자의 close함수의 누락을 사전에 차단하기 위해 DBConnection에 대한 자원 관리 클래스를 만듭니다.

|  |
| --- |
| class DBConn // DBConnection 객체를 관리하는 클래스  {  public:  ~DBConn() // 데이터베이스 연결이 닫히도록 하는 함수  {  db.close();  }  private:  DBConnection db;  }; |

위 와 같은 디자인은 다음과 같은 프로그래밍이 가능하도록 한다.

|  |
| --- |
| void main()  {  DBConn dbc(DBConnection::create()); // DBConnection 객체를 생성하고,  // DBConn 객체에 넘겨 관리를 맡김  // ... // DBConn 인터페이스를 통해  // DBConnection의 객체를 사용    // 블록이 끝나면 DBConn 객체가 소멸  // DBConnection 객체에 대한 close가 이루어짐  } |

**그러나 close를 호출했을 때, 예외가 발생한다면?**

DBConn의 소멸자는 예외를 전파할 것이며, **소멸자의 예외가 나가도록 내버려 둔다는 것이 문제의 요점**

이 문제를 피하는 방법은 두 가지가 존재 하는데

**1. 예외가 발생하면 프로그램을 바로 종료하는 것**

|  |
| --- |
| DBConn::~DBConn()  {  try { db.close(); }  catch (...)  {  // 로그 출력  **std::abort();**  }  } |

소멸자에서 생긴 예외를 그대로 흘려 보냈다가 정의 되지 않은 동작에까지 이를 수 있다면, abort를 호출하여 **못 볼꼴을 방지하는 것**

**2. 예외를 삼켜 버리는 것**

|  |
| --- |
| DBConn::~DBConn()  {  try { db.close(); }  catch (...)  {  // 로그 출력  }  } |

때에 따라서는 불완전한 프로그램 종료 혹은 미정의 동작으로 인해 입는 위험을 감수 하는 것보다 그냥 **예외를 먹어 버리는 게 나을 수 있다.**

**개선 1.**

DBConn을 개선하여 발생할 소지가 있는 문제에 대처할 기회를 사용자가 가질 수 있도록 수정

Close 함수를 직접 제공하여 함수의 실행 중 발생할 예외를 사용자가 직접 처리 할 수 있도록 DBConnection 이 닫혔는지의 여부를 유지 했다가 닫히지 않았다면 DBConnection의 소멸자에서 닫을 수 있게 되어, 데이터베이스 연결 누수를 방지 할 수 있음.

하지만 **소멸자에서 호출하는 close마저 실패 한다면 다시 끝내거나, 삼키기 모드로 돌아와야 한다.**

|  |
| --- |
| class DBConn // DBConnection 객체를 관리하는 클래스  {  public:  void Close() // 사용자가 직접 호출할 수 있도록 만든 함수  {  db.close();  closed = true;  }  ~DBConn()  {  if (closed == false)  {  try { db.close(); }  catch (...)  {  // 로그 출력  // 실패를 알린 후 끝내거나 삼키기를 수행  }  }  }  private:  DBConnection db;  bool closed;  }; |

**코드의 요점은 발생한 예외가 소멸자가 아닌 다른 함수에서 비롯된 것이어야 한다는 것이 포인트**

**요약**

\* 소멸자에서 예외가 빠져나가면 안됩니다. 만약 소멸자 안에서 호출된 함수가 예외를 던질 가능성이 있다면, 어떤 예외이든지 소멸자에서 모두 받아낸 후에 삼켜 버리든지 프로그램을 끝내던지 해야 합니다.

\* 어떤 클래스의 연산이 진행되다가 던진 예외에 대해 사용자가 반응해야 할 필요가 있다면, 해당 연산을 제공하는 함수는 반드시 보통의 함수(소멸자가 아닌 함수)이어야 합니다.