**항목 17. new로 생성한 객체를 스마트 포인터에 저장하는 코드는 별도의 한 문장으로 만들자.**

이슈.

처리 우선순위를 알려주는 함수와 동적으로 할당한 객체에 대해 우선순위에 따른 처리를 적용하는 함수가 있습니다.

동적으로 할당 되는 Widget객체에 대해 스마트 포인터 (tr1::shared\_ptr) 을 사용하도록 만들어 졌습니다.

|  |
| --- |
| // 처리 우선순위를 알려주는 함수  **int** priority();  // 우선순위에 따라 처리를 적용하는 함수  void processWidget( std::tr1::shared\_ptr<Widget> pw, **int** priority); |

ProcessWidget에 대한 호출은 다음과 같다.

|  |
| --- |
| processWidget(new Widget, priority()); |

**위 코드는 컴파일 되지 않는데. new Widget 표현에서 TR1::shared\_ptr로의 암시적 변환은 수행될 수 없습니다. tr1::shared\_ptr의 생성자는 explicit로 선언되어 있기 때문입니다.**

|  |
| --- |
| processWidget(**std::tr1::shared\_ptr<Widget>(**new Widget**)**, priority()); |

위 코드는 컴파일에 이상이 없습니다.

하지만 위의 코드는 자원 관리 객체를 쓰고 있음에도 **자원의 누수의 가능성이 존재** 합니다.

컴파일러는 processWidget 호출 코드를 만들기 전에 우선 이 함수의 매개변수로 넘겨지는 인자를 평가하는 순서를 진행합니다. 두 번째 인자는 prioryti함수의 호출 문 밖에 없지만 첫 번째 인자(**std::tr1::shared\_ptr<Widget>(**new Widget**)**)는 **두 가지 부분**으로 나뉩니다.

**1. new Widget 표현식을 실행하는 부분**

**2. tr1::shared\_ptr 생성자를 호출하는 부분**

그러므로 precessWidget 함수 호출이 이루어지기 전에 컴파일러는 다음의 **세 가지 연산을 위한 코드**를 만들어야 합니다.

1. priority를 호출합니다.

2. new Widget을 실행합니다.

3. tr1::shared\_ptr 생성자를 호출합니다.

그런데 여기서 **각각의 연산이 실행되는 순서는 컴파일러 마다 다르게 설정 될 수 있습니다**. C++ 컴파일러의 경우 이들의 순서를 정하는 데 있어서 상당한 자유도를 가집니다. new Widget의 경우 tr1::shared\_ptr 생성자 호출보다 이전에 수행 되어야 합니다. 이 표현식의 결과가 tr1::shared\_ptr의 생성자 인자로 넘어가야 하기 때문이죠. 그러나 priority의 호출은 처음 호출 될 지 두 번째나, 세 번째에 호출 될 수 있습니다. 만일 두 번째에 정해 졌다면 다음과 같이 실행됩니다.

1. new Widget을 실행합니다.

2. priority를 호출합니다.

3. tr1::shared\_ptr 생성자를 호출합니다.

하지만 **priority 호출에서 예외가 발생한다면 어떻게 될까요**. new Widget으로 만들어진 포인터가 유실 될 수 있습니다. 자원 관리 객체인 tr1::shared\_ptr 에 저장 되기도 전에 예외가 발생했기 때문입니다.

**해결법**

이 문제를 피해가는 방법은 간단합니다. Widget을 생성해서 스마트 포인터에 저장하는 **코드를 별도의 문장 하나로 만들고, 그 스마트 포인터를 processWidget에 넘기는 것**입니다.

|  |
| --- |
| std::tr1::shared\_ptr<Widget> pw(new Widget);  processWidget(pw, priority()); |

**요약.**

\* new로 생성한 객체를 스마트 포인터로 넣는 코드는 별도의 한 문장으로 만듭시다. 이것이 안되어 있으면, 예외가 발생될 때 디버깅하기 힘든 자원 누출이 초래될 수 있습니다.