**항목 23. 멤버 함수보다는 비멤버 프렌드 함수와 더 가까워지자.**

**이슈**

웹 브라우저를 나타내는 클래스

|  |
| --- |
| class WebBrowser {  public:  ...  void clearCache();  void clearHistory();  void removeCookies();  ...  }; |

하지만 사용자 중에는 이 3 동작을 한 번에 하고 싶은 분들도 있기 때문에, 3함수를 모아서 불러주는 함수도 준비해 둘 수 있을 것이다.

|  |
| --- |
| class WebBrowser {  public:  ...  **void clearEverything();** // clearCache, clearHistory, removeCookies를 호출.  ...  }; |

물론 이 기능은 비 멤버 함수로 제공해도 된다.  
웹 브라우저 객체의 멤버함수를 순서대로 불러주기만 하면 되는 것이다.

|  |
| --- |
| **void clearBrowser(WebBrowser& wb)**  {  wb.clearCache();  wb.clearHistory();  wb.removeCookies();  } |

어느 쪽이 더 괜찮을까요? 멤버버전인 clearEverything ? 아니면 비 멤버 버전인 clearBrowser ?

객체 지향 법칙에 관련된 이야기를 찾아보면 데이터와 그 데이터를 기반으로 동작하는 함수는 한 데 묶여 있어야 하며, 멤버 함수가 더 낫다고들 합니다. 하지만 불행히도 이 제안은 틀렸습니다.  
객체 지향 방법이 무엇인가에 대해 제대로 이해하지 못한 상태에서 나온 제안입니다. 분명히 객체 지향 법칙은 할 수 있는 만큼 데이터를 **캡슐화**하라고 주장하고 있지요.  
그러나 멤버 버전이니 clearEverything은 비멤버 버전인 clearBrowser보다 캡슐화 정도에서 오히려 형편 없습니다.   
비 멤버 함수를 사용하면 WebBrowser관련 기능을 구성하는 데 있어서 패키징 유연성(packaging flexibility)이 높아지는 장점이 있는데다가, 이로 인해 얻게 되는 추가적인 이점으로 컴파일 의존도도 낮추고 WebBrowser의 확장성도 높일 수 있습니다.   
그래서 **비 멤버 방법이 멤버 함수보다 여러모로 낫다는 이야기가 나오는 것**입니다.

**캡슐화**  
어떤 것을 캡슐화하면, 우선 외부에서 이것을 볼 수 없게 됩니다.   
캡슐화하는 것이 늘어나면 그만큼 밖에서 볼 수 있는 것들이 줄어듭니다.  
밖에서 볼 수 있는 것들이 줄어들면, 그것들을 바꿀 때 필요한 유연성이 커집니다.   
변경 자체가 영향을 줄 수 있는 범위가 ‘변경된 것을 볼 수 있는 것들’로 한정되기 때문에…  
캡슐화되는 것들이 많아지면, 그것들을 변경할 수 있는 여유도 많아집니다.   
바로 이것 때문에 우선 캡슐화에 가치를 두는 것이지요.  
즉, 이미 있는 코드를 바꾸더라도 제한된 사용자들밖에 영향을 주지 않는 융통성을 확보할 수 있다는 뜻입니다.

똑같은 기능을 제공하는데 멤버함수 (그 클래스의 private 데이터 멤버뿐만 아니라 private 멤버로 되어 있는 다른 함수, 나열자, typedef 타입 등등을 모두 접근할 수 있는)를 쓸 것이냐,   
아니면 비멤버 프렌드 함수(어느 것도 접근할 수 없는)를 쓸 것이냐?

캡슐화의 정도가 더 높은 쪽을 고른다면 단연 후자일 것입니다.   
왜냐하면 **비 멤버 비프렌드 함수는 어떤 클래스의 private 멤버 부분을 접근할 수 있는 함수의 개수를 늘리지 않으니까요.**  
이것으로 clearBrowser(비멤버 비프랜드 함수)가 clearEverything(멤버함수)보다 어째서 더 바람직한지에 대한 이유가 설명될 것 입니다. WebBrowser 클래스에 대한 캡슐화 정도가 더 높은 것을 고른 것이죠.

여기서 주의할 점 2가지.

1.     **비 멤버 비프렌드 함수에만 적용된다는 것.**

프렌드 함수는 private 멤버에 대한 접근권한이, 해당 클래스의 멤버함수가 가진 접근권한과 똑같기 때문에, 캡슐화에 대한 영향 역시 같습니다.

캡슐화라는 관점에서 보았을 때, 위의 선택은 멤버 함수와 비 멤버 함수 사이의 선택이 아닙니다. 멤버함수와 비 멤버 ‘비 프렌드’ 함수 사이의 선택입니다 (물론 캡슐화만이 관점은 아닙니다. (24)를 보면 암시적 타입 변환의 관점에서 멤버 함수 및 비멤버 함수 중 하나를 고르는 이야기가 나옴).

**2.     캡슐화에 대한 이야기 때문에 “함수는 어떤 클래스의 비 멤버가 되어야 한다”라는 주장이 “그 함수는 다른 클래스의 멤버가 될 수 없다”라는 의미가 아니라는 점입니다.**

즉, clearBrowser 를 다른 유틸리티 클래스 같은 정적 멤버 함수로 만들어도 된다는 것.  
어쨌든 이 함수가 WebBrowser 클래스의 멤버(혹은 프렌드)가 아니기만 하면 됩니다.  
WebBrowser가 가진 private 멤버의 캡슐화에 영향을 주지 않는다는 점이 중요.

C++로는 더 자연스런 방법을 구사할 수 있음.  
clearBrowser를 비멤버 함수로 두되, WebBrowserStuff 같은 네임스페이스 안에 두는 것.

|  |
| --- |
| **namespace WebBrowserStuff {**             class WebBrowser {...};             void clearBrowser(WebBrowser& wb);             ...  } |

사실 이건 자연스러움보다 몇 걸음 더 나아간 방법이라고 볼 수 있음.  
왜냐하면 네임스페이스는 클래스와 달리 여러 개의 소스 파일에 흩어질 수 있기 때문.

이 부분은 굉장히 중요한데, clearBrowser 같은 함수는 편의상 준비한 함수들(이하 편의함수)이기 때문입니다. 멤버도 아니고 프렌드도 아니기에, WebBrowser 사용자 수준에서 아무리 애를 써도 얻어낼 수 없는 기능은 이들도 제공할 수 없습니다.   
예를 들어 clearBrowser가 없다고 해도, 사용자는 그냥 clearCache, clearHistory, removeCookies를 알아서 불러주면 되는 것입니다.

WebBrowser처럼 응용도가 높은 클래스는 이런 종류의 편의 함수가 꽤 많이 생길 수 있습니다.  
즐겨찾기, 인쇄, 쿠키 함수도 가능하지요. 즐겨찾기 기능에만 관심있는 사용자가 다른 함수들(쿠키관련 함수)에 대한 컴파일 의존성을 고민할 이유가 없다는 것이죠.  
이것들을 나누어 놓는 쉽고 깔끔한 방법은, 즐겨찾기 관련 편의 함수를 하나의 헤더파일에 몰아서 선언하고, 쿠키 관련 편의함수는 다른 헤더 파일에 몰아서 선언하는 것입니다.

|  |
| --- |
| // **WebBrowser.h 헤더** - WebBrowser 클래스 자체에 대한 헤더  // WebBrowser에 과련된 '핵심' 기능들이 선언되어 있음.  namespace WebBrowserStuff {             class WebBrowser {...};             ... // '핵심' 관련 기능, 이를 테면 거의 모든 사용자가 써야 하는 비멤버 함수들이 여기에 들어갑니다.  }    **// WebBrowserBookmark.h 헤더**  namespace WebBrowserStuff {             ... // 즐겨찾기 관련 편의 함수들  }    **// WebBrowserCookies.h 헤더**  namespace WebBrowserStuff {             ... // 쿠키 관련 편의 함수들  } |

표준 C++ 라이브러리가 이러한 구조로 구성되어 있답니다.  
std 네임스페이스에 속한 모든 것들이 <C++ StandardLibrary> 헤더 같은 것에 모조리 들어가 한 통으로 섞여 있지 않고, 몇 개의 기능과 관련된 함수들이 수십 개의 헤더(<vector>, <algorithm>, <memory> 등)에 흩어져 선언되어 있습니다.  
이렇게 하면, 사용자가 실제로 사용하는 구성요소에 대해서만 컴파일 의존성을 고려할 수 있게 되는 거죠 (컴파일 의존성을 줄이는 방법으로 다른 것들도 있는데 (31)을 참고)  
반면 클래스 멤버 함수로 오게 되면 이야기가 암울해 집니다. 이런 식으로 기능을 쪼개는 것 자체가 불가능합니다. 하나의 클래스는 그 전체가 통으로 정의되어야 하고 여러 조각으로 나눌 수가 없기 때문입니다.

편의 함수 전체를 여러 개의 헤더 파일에 (그러나 하나의 네임스페이스에) 나누어 놓으면 편의 함수 집합의 **확장(extend)**도 손쉬워집니다.   
해당 네임스페이스에 비멤버 비프렌드 함수를 원하는 만큼 추가해 주기만 하면 그게 확장입니다.  
예를 들어, WebBrowser 에 다운로드 관련 편의 함수를 추가한다면, 헤더 파일 하나를 추가한 후에 WebBrowserStuff 네임스페이스를 만들고 그 안에 관련 함수의 선언문만 끼워 넣으면 끝이라는 거죠. 이렇게 새로 추가된 함수는 기존의 다른 편의 함수들처럼 바로 사용할 수 있으며, WebBrowserStuff의 구성요소로 바로 합쳐집니다.   
이런 부분은 클래스로는 제공이 불가능한 기능입니다. 클래스 정의 자체를 사용자가 확장할 수는 없으니까요. 물론 새로운 클래스를 파생시킬 수 있기는 하지요. 하지만 파생 클래스는 기본 클래스 안에 캡슐화된 (즉, private) 멤버에 대한 접근권한이 없기 때문에, 이런 식의 ‘확장 기능’은 이등석 티켓 정도입니다. 게다가 (7)에서 언급했듯이 모든 클래스가 기본 클래스로 쓸 용도로 설계된 것도 아니죠.

**요약.**

- 멤버 함수보다는 비멤버 프렌드 함수를 자주 쓰도록 합시다. 캡슐화 정도가 높아지고, 패키징 유연성도 커지며, 기능적인 확장성도 늘어납니다.