**항목 22. 데이터 멤버가 선언될 곳은 private 영역임을 명심하자.**

public 데이터 멤버가 왜 안될까요?

**1. 문법적 일관성.**

데이터 멤버가 public이 아니라면, 사용자 쪽에서 어떤 객체를 접근할 수 있는 유일한 수단은 멤버 함수일 것입니다.   
어떤 클래스의 공개인터페이스에 있는 것들이 전부 함수뿐이라면, 그 클래스의 멤버에 접근하고 싶을 때 괄호를 붙여야 하는지 말아야 하는지를 기억하지 못해서 고민할 필요가 없어진다.

**2. 데이터 멤버의 접근성에 대해 훨씬 정교한 제어를 할 수 있다.**  
어떤 데이터 멤버를 public으로 내놨다면 모두가 이 멤버에 대해 일기/쓰기 접근 권한을 갖게 되지만, 이 값을 읽고 쓰는 함수가 있으면 **접근 불가/ 읽기 전용/ 읽기 쓰기 접근 등을 직접 구현**할 수 잇다.

**3. 캡슐화.**

함수를 통해서만 데이터 멤버에 접근할 수 있도록 구현해 두면, 데이터 멤버를 나중에 계산식으로 대체할 수도 있을 것이고, 사용자는 절대로 이 클래스를 넘볼 수 없습니다.

Protected 데이터 멤버의 경우도, 똑같습니다.

문법적 일관성과 세밀한 접근 제어에 관한 이야기라면 public 데이터 멤버처럼 protected 데이터 멤버에도 그대로 적용할 수 있습니다.

어떤 데이터 멤버를 일단 public 혹은 protected로 선언했으며 사용자가 그것을 사용하기 시작했으면, 그때부터 그 멤버는 완전히 코 꿰인 것입니다. 그 **멤버에 대해 무엇을 바꾸기란 무척 힘들어진단 말**입니다.

캡슐화의 관점에서 쓸모 있는 접근 수준은 private(캡슐화 제공)와 private이 아닌 나머지(캡슐화 없음), 이렇게 2뿐입니다.

요약.

\*데이터 멤버는 private 멤버로 선언 합시다. 이를 통해 클래스 제작자는 문법적으로 일관성 있는 데이터 접근 통로를 제공할 수 있고, 필요에 따라서는 세밀한 접근 제어도 가능하며, 클래스의 불변속성을 강화할 수 있을 뿐 아니라, 내부 구현의 융통성도 발휘할 수 있습니다.

\* protected는 public보다 더 많이 ‘보호’받고 있는 것이 절대로 아닙니다.