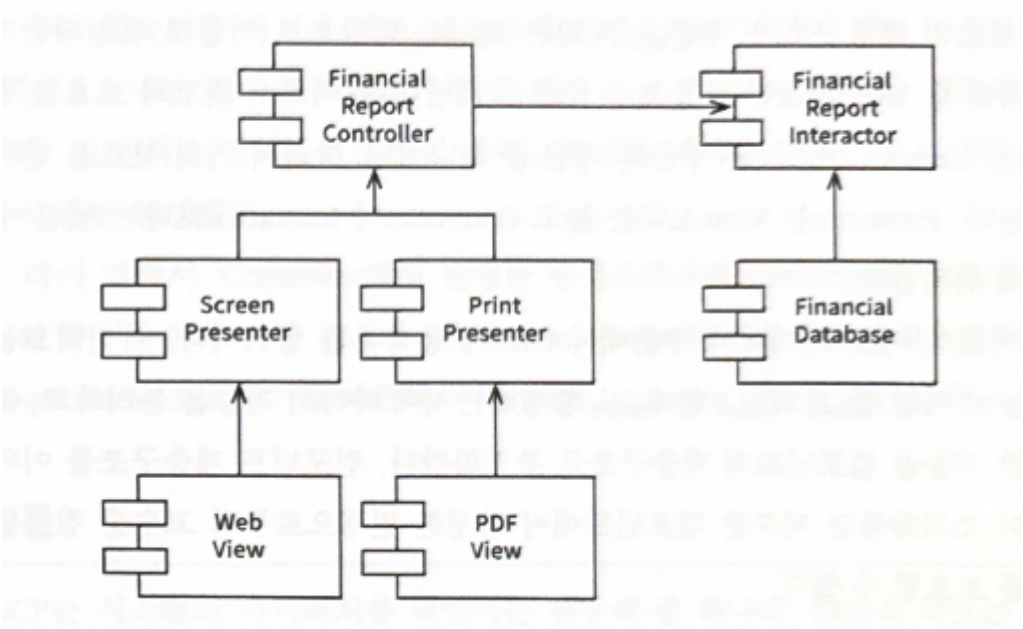
08장. OCP: 개방-폐쇄 원칙

- 소프트웨어 개체는 확장에는 열려 있어야 하고, 변경에는 닫혀 있어야 한다.

사고 실험

- 새로운 이해관계자가 데이터의 새로운 출력 양식을 기대할 때, 변경되는 코드의 양이 가장 적을 수록 좋다.
  - 서로 다른 목적으로 변경되는 요소를 적절하게 분리(단일 책임 원칙)
  - 요소 사이의 의존성을 체계화(의존성 역전 원칙)



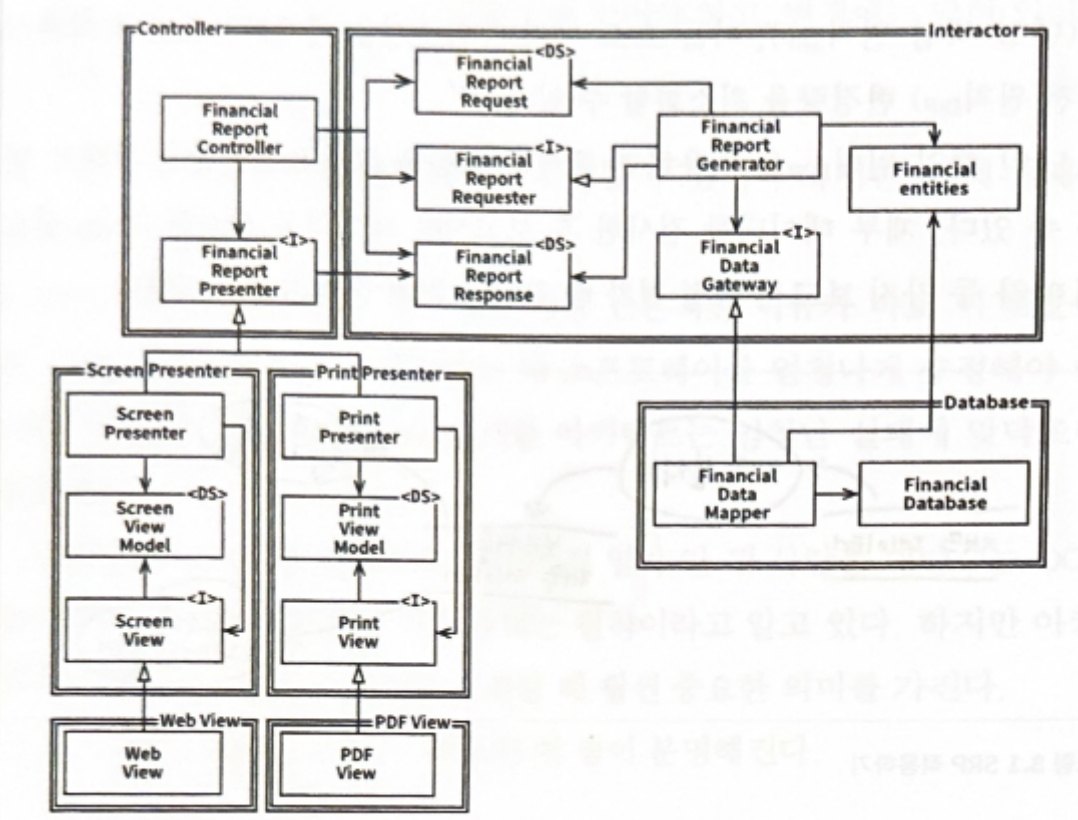
- 화살표가 A 클래스에서 B 클래스로 향하면 A에서는 B를 호출하지만, B에서는 A를 호출하지 않는 다는 것이다. 화살표는 변경으로부터 보호하려는 컴포넌트를 향하도록 그려진다. 그리고 A 컴포넌트에서 발생한 변경으로부터 B 컴포넌트를 보호하려면 반드시 A 컴포넌트가 B컴포넌트에 의존해야한다.
- 위 예제의 경우 presenter에서 발생한 변경으로부터 controller를 보호하고자 한다. interactor는 다른 모든 것에서 발생한 변경으로부터 보호하고자 하는데 가장 중요한 업무 규칙을 호함하고 있기 때문이다.
- 보호의 계층 구조가 ‘수준(level)’이라는 개념을 바탕으로 어떻게 생성되는지 주목하자. interactor는 가장 높은 수준의 개념 중 하나이며 최고의 보호를 받는다. 이것이 OCP가 동작하는 방식이다.

- 기능이 어떻게, 왜, 언제 발생하는 지에 따라서 기능을 분리하고 분리한 기능을 컴포넌트의 계층구조로 조직화 한다.

방향성 제어

- 아키텍트가 방향을 정함으로써 보호할 주체를 선택할 수 있다.

정보 은닉



- FinancialReportRequester 인터페이스는 controller가 interactor 내부를 알지 못하도록 막기 위해 존재한다. 이 인터페이스가 없었다면 controller는 financialEntities에 대해 추이 종속성을 가지게 된다.
- 추이 종속성을 가지게 되면, 소프트웨어 엔티티는 '자신이 직접 사용하지 않는 요소에는 절대로 의존해서는 안 된다'는 원칙을 위반하게 된다.
- controller에서 발생한 변경으로부터 interactor를 보호하는 것이 가장 우선순위가 높지만, interactor에서의 변경으로부터 controller를 보호하기 위해 은닉시킨다.

결론

OCP는 시스템의 아키텍처를 떠받치는 원동력 중 하나이다.