객체 지향의 주요 목표는 사람들이 생각하는 방식과
비슷한 방식으로 현실 세계의 시스템을 모델링 하는 것이다.

인터페이스를 사용해 행위를 추가하는 것이
상속을 이용해 행위를 추가하는 것보다 좋다.

즉 행위를 바꿀 수 있게하는 가장 좋은 방법은
인터페이스와 컴포지션을 사용하는 것이다.
/
상속이 아니라
합성을 이용하는 클래스

생성자 주입: 서비스 클래스를 클래스 내에 두는 것이 아니라
객체를 생성할 때 생성자에 주입한다는 것

객체는 다른 시스템에서 재사용할 수 있으며, 재사용을
염두해 코드를 작성해야 한다.

정적 메서드는 도우미 클래스나 부작용을 일으키지 않는
확장 메서드를 사용하는 경우 뿐이다.

이식하기 어려운 코드(네이티브 코드)를 사용해야 하는
시스템을 설계하는 경우에 코드를 클래스에서 뽑아내어
추상화해야 한다.
추상화 한다는 것은 이식하기 어려운 코드를 자체 클래스나
최소한 자체 메서드(오버라이드할 수 있는 메서드)로 분리

가능한 많은 속성과 행위를 지역화 하면
추상화와 구현부 모두이 자연스럽게 이루어진다.

클래스를 설계할 때에는 코드를 관리하기 쉬운 여러 조각으로
작성해 둔 다음에 합성할 수 있게 설계하는게 바람직하다.

서로 강하게 묶인 (의존성이 높은) 클래스 들은 어느 클래스를
변경하면 다른 클래스도 변경해야 한다.

올바르게 설계한 클래스는 시스템을 변경할 때는
객체의 구현부만 변경해야 한다.
유지 보수성을 높이려면 클래스 간에 서로 묶이게 되는 정도를
가능한 한 낮게 유지한다.

스텁: 인터페이스를 최소한으로 구현한 것

스텁을 이용하여 테스트를 진행한 뒤 완료가 되면
삭제를 하거나 남겨둔다.

지속성: 객체의 상태를 유지한다는 개념

고려해야 할 기본 저장 장치 3가지

① 플랫 파일 시스템

② 관계형 데이터 베이스

③ NoSQL 데이터 베이스

직렬화 : 객체의 구성을 해체해서 전송 후 재구성하는 과정

마샬링 : 통신선을 통해 객체를 보내는 행위

결론

이번 장에서는 클래스 설계에 도움이 되는 많은 지침을 제공하였다. 그렇다고 해서 모든 클래스 설계 지침을 완벽하게 제시한 건 아니다. 객체지향 설계로 여행하는 데 필요한 지침이 그 밖에도 더 있다는 점은 의심할 여지가 없다.

이번 장에서는 개별 클래스와 관련된 설계 문제를 다루었다. 그러나 우리는 클래스라는 게 단독으로 쓰이는 게 아니라는 점을 이미 보았다. 클래스는 다른 클래스와 상호 작용을 하도록 설계되어야 한다. 상호 작용을 하는 클래스 그룹은 시스템의 일부다. 궁극적으로 이러한 시스템은 최종 사용자에게 가치를 제공한다. 6장 '객체를 사용해 설계하기'에서는 완전한 시스템을 설계하는 주제를 다룬다.