8장 경계

* 이번시간에는 외부코드(오픈소스, api)를 우리 코드에 깔끔하게 통합해야만하는데 이 장에서 소프트웨어 경계를 깔끔하게 처리하는 기법과 기교를 살펴본다.
* 외부 코드 사용하기
  + 인터페이스 제공자와 인터페이스 사용자 사이에는 특유의 긴장이 존대
  + 패키지 제공자나 프레임워크 제공자는 적용성을 최대한 넓히려고 애쓴다.
  + 더많은 환경에서 돌아가야 더 많은 고객이 구매하니. 반면 사용자는 자신의 요구에 집중하는 인터페이스를 바란다.
  + 이런 긴장으로 인해 시스템 경계에서 문제가 생길 소지가 많다.
* 경계 살피고 익히기
  + 와부 코드를 사용하면 적은 시간에 더 많은 기능을 출시하기 쉬워진다. 만약 외부에서 가져온 패키지를 사용하고 싶다면 어디서 어떻게 시작해야 좋을까..외부 패키지 테스트가 우리 책임은 아니다.
  + 하지만 우리 자신을 위해 우리가 사용할 코드를 테스트하는 편이 바람직하다.
  + 외부코드를 익히기도 통합하기도 어렵다. 두가지를 동시에 하기는 두배나 어렵다. 다르게 접근하면 곧바로 우리쪽 코드를 작성해 외부 코드를 호출하는 대신 먼저 간단한 테스트 케이스를 작성해 외부 코드를 익히면 어떨까 이걸 학습 테스트라고 부른다
  + 학습 테스트는 프로그램에서 사용하려는 방식대로 외부 API를 호출한다. 통제된 환경에서 API를 제대로 이해하는지 확인하는 셈이다.
* Log4j 익히기
  + 로깅 기능을 직접 구현하는 대신 아파치의 log4j 패키지를 사용하려고 한다고 가정하자.
* 학습 테스트는 공짜 이상이다.
  + 학슴 테스트에 드는 비용은 없다. 어찌됐든 api를 배워야하기 떄문이다.
  + 오히려 필요한 지식만 확보하는 손쉬운 방법이다. 학습 테스트는 이해도를 높여주느 정확한 실험이다.
  + 학습 테스트는 공짜 이상이다. 투자하는 노력보다 얻는 성과가 더 크다. 패키지 새 버전이 나온다면 학습 테스트를 돌려 차이가 있는지 확인한다.
  + 학습테스트는 패키지가 예상대로 도는지 검증한다. 일단 통합한 이후라고 하더라도 패키지가 우리 코드와 호환되리라는 보장은 없다. 패키지 작성자에게 코드를 변경할 필요가 생길지도 모른다.
  + 학습 테스트를 이용한 학습이 필요하든 그렇지 않든 실제 코드와 동일한 방식으로 인터페이스를 사용하는 테스트 케이스가 쉬워진다. 이런 경계 테스트가 있다면 패키지의 새 버전을 이전하기 쉬워진다. 그렇지 않다면 낡은 버전을 필요 이상으로 오랬동안 사용하려는 유혹에 빠지기 쉽다.
* 아직 존재하지 않은 코드를 사용하기
  + 경계와 관련해 또 다른 유형은 아는 코드와 모르는 코드를 분리하는 경계이다.
  + 가령 지정한 주파수를 이용해 스트림에서 들어오는 자료를 아날로그, 신호로 전송하라는 기능을 구현하라는 프로젝트를 받았다는 가정을 한다면 아직 저쪽 팀이 api를 설계하지 않았으므로 구체적인 방법은 몰랐다. 그래서 우리는 구현을 나중에 미루고 자체적으로 인터페이스를 정의하여 Transmitter라는 간단한 클래스를 만든 후 transmit라는 메서드를 추가하여 transmit인터페이스로 주파수와 자료 스트림을 받아 우리가 바라는 인터페이스를 가져오는 것이다.
  + 이와 같은 설계는 테스트도 아주 편하다. 적절한 FakeTransmitter 클래스를 사용하면 CommunicationController클래스를 테스트할 수 있기 때문이다.
* 깨끗한 경계
  + 경계에 위치하는 코드는 깔끔히 분리한다. 또한 기대치를 정의하는 테스트 케이스도 작성한다. 이쪽 코드에서 외부 패키지를 세세하게 알아야 할 필요가 없다. 통제가 불가능한 외부 패키지에서 의존하는 대신 통제가 가능한 우리 코드에 의존하는 편이 훨씬 좋다.
  + 외부 패키지를 호출하는 코드를 가능한 줄여 경계를 관리하자. Map에서 봤듯이 새로운 클래스로 경계를 감싸거나 아니면 Adapter 패턴을 사용해 우리가 원하는 인터페이스 패키지가 제공하는 인터페이스로 변환하다.