* 9장 단위테스트
  + 우리 분야는 지금까지 눈부신 성장을 이루었지만 앞으로 갈길은 여전히 멀다. 에자일과 tdd 덕분에 단위 테스트를 자동화하는 프로그래머들이 이미 많아졌다. 하지만 우리 분야에 테스트를 추가하려고 급하게 서두른다면 많은 프로그래미들이 제대로 된 태스트 케이스를 작성해야 한다는 좀 더 미묘한 사실을 놓치게 된다.
  + TDD의 법칙 세가지
    - 첫번째 : 실패하는 단위 테스트를 장성할 때까지 실제 코드를 작성하지 않는다.
    - 두번쨰 : 컴파일은 실패하지 않으면서 실행이 실패하는 정도로만 단위테스트를 작성한다.
    - 세번쨰 : 현재 실패하는 테스트를 통과할 정도로만 실제 코드를 작성한다.
  + 깨끗한 테스트 코드 유지하기
    - 테스트는 유연성, 유지보수성, 제사용성을 제공한다.
      * 코드에 유연성, 유지보수성, 제사용성을 제공하는 버팀목이 바로 단위 테스트이다. 이유는 단순하게 테스트케이스가 있으면 변경이 두렵지 않기 때문이다.
      * 실제 코드를 점검하는 자동화된 단위 테스트 슈트는 설계와 아키텍처를 최대한 깨끗하게 보존하는 열쇠이다.
    - 깨끗한 테스트 코드
      * 깨끗한 테스트 코드를 만들려면 가독성,과 가독성, 가독성이 필요하다.
      * 테스트 코드에 가독성을 높이려면 여느 코드와 마찬가지다. 명료성, 단순성 풍부한 표현력이 필요하다.
      * 잡다하고 세세한 코드를 거의 다 없앴다는 사실에 주목하자. 테스트 코드는 본론에 돌입해 진짜 필요한 자료 유형과 함수만 사용한다. 그러므로 코드를 읽는 사람은 온갖 잡다하고 세세한 코드에 주눅들고 헷갈릴 필요 없이 코드가 수행하는 기능을 재빨리 이해한다.
    - 도메인에 특화된 테스트 언어
      * 흔히 쓰는 시스템 조작 api를 사용하는 대신 api 위에다 함수와 유틸리티를 구현한 후 그 함수와 유틸리니를 사용하므로 테스트 코드를 짜기도 읽기도 쉬워진다.
      * 이렇게 구현한 함수와 유틸리티는 테스트 코드에서 사용하는 특수 api가 된다. 즉 테스트를 구현하는 당사자와 나중에 테스트를 읽어볼 독자를 도와주는 테스트 언어이다.
      * 이런 테스트 api는 처음부터 설계된 api가 아니다. 잡다하고 세세한 사항으로 범벅된 코드를 계속 리펙터링 하다가 진화된 api이다.
    - 이중 표준
      * 테스트 api 코드에 적용하는 표준은 실제 코드에 적요하는 표준과 확실히 다르다,
      * 단순하고 간결하고 표현력이 풍부해야 하지만 실제 코드만큼 효율적일 필요는 없다. 실제 환경이 아니라 테스트 환경에서 돌아가는 코드이기 때문이다.
  + 테스트당 assert 하나
    - junit으로 테스트 코드를 짤 떄 함수마다 assert문을 단 하나만 사용해야 한다고 주장하는 학파가 있다. 가혹한 규칙이라 생각할지라도 확실히 장점이 있다.
    - assert문이 하나인 함수는 결론이 하나라서 코드를 이해하기 쉽고 빠르다.
    - 단일assert문이라는 규칙이 훌륭한 지침이라고 필자는 생각한다. 대체로 단일 assert를 지원하는 해당 분야 테스트 언어를 만들려 노력한다. 하지만 떄로는 주저 없이 함수 하나에 여러 assert문을 넣기도 한다. 단지 assert문 개수는 최대한 줄여야 좋다는 생각.
    - 테스트당 개념 하나
      * 테스트 함수마다 한 개념만 테스트 하ㅏ라 라는 규칙이 더 나을 것
      * 이것저것 잡다한 개념을 연속으로 테스트하는 긴 함수는 피한다.
  + F.I.R.S.T
    - 꺠끗한 테스트는 다음 다섯가지 규칙을 따르는데 각 규칙에서 첫번쨰 글자를 따면 FIRST이다
    - 빠르게 : 테스트는 빨라야 한다. 테스트는 빨리 돌아야 한다는 ㅁ라이다. 테스트가 느리면 자주 돌릴 엄두도 못내기 때문에 자주 돌리지 않으면 초반에 문제를 찾아내 고치지 못한다.
    - 독립적으로 : 각 테스트는 서로 의존해선 안되고, 다음 테스트가 실행될 환경을 준비해서도 안된다.
    - 반복 가능하게 : 테스트는 어떤 환경에서도 반복 가능해야 한다. 실제 환경 QA 환경 버스를 타고 집으로 가는 길에 사용하는 노트북 환경에서도 실행할 수 있어야 한다. 테스트가 돌아가지 않는 환경이 하나라도 있다면 테스트가 실패한 이유를 둘러댈 변명이 생긴다. 게다가 환경이 지원되지 않기 때문에 테스트를 수행하지 못하는 상황에 직면한다.
    - 자가검증하는 : 테스트는 부울값으로 결과를 내야하기 때문에 통과 여부를 보려고 택스트 파일 두개르 수작업으로 비교하게 만들어서도 안된다.
    - 적시에 : 테스트는 적시에 작성해야 한다. 단위 테스트는 테스트하려는 실제 코드를 구현하기 직전에 구현한다. 실제 코드를 구현한 다음에 테스트 코드를 만들면 실제 코드가 테스트하기 어렵다는 사실을 발견할지도 모른다.
  + 결론 : 사실상 꺠끗한 테스트 코드라는 주제는 책 한 권을 할애해도 모자란 주제이다. 어떠면 실제 코드보다 더 중요할지도 모르겠다. 테스트 코드는 실제 코드의 유연성, 유지보수성, 재사용성을 보존하고 강화하기떄문에 그러므로 테스트 코드는 지속적으로 꺠끗하게 관리하자.