소프트웨어 테스트

- 소프트웨어 테스트에는 여러가지가 있습니다.
- 각 테스트는 목적, 방법에 따라 차이점을 가집니다.
- 단위 테스트, 통합 테스트, 인수 테스트, 시스템 테스트 등이 있습니다.
- 이 중 단위 테스트에 대해서 알아보도록 하겠습니다.

목차

단위 테스트 (UNIT TEST)

단위 테스트는 하나의 모듈을 기준으로 독립적으로 진행되는 가장 작은 단위의 테스트입니다.

여기서 말하는 모듈은, 애플리케이션에서 작동하는 하나의 기능이나 메서드가 될 수 있습니다.

예를 들면, 저희 앱에서 로그인을 할 때 해당 유저에 대한 검증 이라는 것이 1개의 단위 테스트가 될 수 있습니다.

즉, 단위 테스트는 애플리케이션을 구성하는 하나하나의 기능들이 온전하게 동작하는지를 독립적으로 테스트 하는 것으로써,

"어떠한 메소드를 실행하면 이러한 결과가 나온다" 라고 테스트를 진행하면 되겠습니다.

단위 테스트 작성의 필요성

일반적으로 실무에서 테스트 코드를 작성한다고 하면 바로 이 **단위 테스트** 입니다.

통합 테스트의 경우 외부 의존성에 대한 연결들을 실제로 진행을 해주어야 하고, 시스템을 구성하는 컴포넌트들이 많으면 많을 수록 테스트에 들어가는 비용이 상당해집니다.

반면 단위테스트는?

해당 부분만 독립적으로 테스트 하기 때문에 코드를 리팩토링 하더라도 빠르게 확인할 수 있는 장점이 존재합니다.

장점

- 테스트에 대한 시간과 비용을 절감할 수 있다.
- 새로운 기능 추가 시에 수시로 빠르게 테스트 할 수 있다.
- 리팩토링시, 안정성을 확보할 수 있다.
- 코드에 대한 문서가 될 수 있다.

이러한 이유들로 인해 단위테스트가 필요하고 흔히 TDD(Test-Driven Development, 테스트 주도 개발)에서 얘기하는 테스트도 바로 이 단위 테스트입니다.

우리는 우리가 작성한 테스트를 수시로 빠르게 돌리면서 문제를 파악할 수 있습니다.

단위 테스트의 단점

- 처음 테스트를 작성하는 경우 무엇부터 시작해야되는지 모를 수 있습니다.
  (즉, 진입 장벽이 높다)
  - 테스트 케이스를 늘려나가면서 깨달아야 한다.
- 테스트 코드 작성에 시간이 소요된다.
  - 짧게 보면 당장은 비용이 많이 들어간다고 볼 수 있습니다.
  - 하지만, 장기적으로 보게 되면 이 테스트 코드로 기능 개발 비용이 절감될 수 있습니다.

좋은 단위테스트란?

테스트 역시 코드이기 때문에 아무런 규칙없이 작성하다 보면 가독성이 떨어질 수 밖에 없습니다.

그렇게 시간이 지나게 되면 쓰레기 코드로 남아있을 가능성이 존재합니다.

그래서 테스트 코드에서도 규칙을 따르는 편이 좋습니다.

각각의 첫 글자를 따서 **F.I.R.S.T** 법칙이라고 합니다.

빠르게(Fast)

테스트는 빨라야 한다. 테스트는 빨리 돌아야 한다는 말입니다. 테스트가 느리면 자주 돌릴 엄두를 못 내게 되고, 자주 돌리지 않으면 초반에 문제를 찾아내 고치지 못하게 됩니다. 이렇게 되면 연쇄작용으로 코드를 마음껏 정리할 수도 없게되고 결국 코드 품질이 망가지기 시작합니다. 😭

독립적으로(Independent)

각 테스트를 서로 의존해서는 안됩니다. 한 테스트가 다음 테스트가 실행될 환경을 준비해서는 안 된다. 각 테스트는 독립적으로 그리고 어떤 순서로 실행해도 괜찮아야 합니다. 테스트가 서로에게 의존하면 하나가 실패할 때 나머지도 잇달아 실패하므로 원인을 진단하기 어려워지며 후반 테스트가 찾아내야 할 결함이 숨겨지게 되기 때문입니다.

반복가능하게(Repeatable)

테스트는 어떤 환경에서도 반복 가능해야 합니다. 실제 환경, QA 환경, 극단적인 예로 버스를 타고 집으로 가는 길에 사용하는 노트북 환경(네트워크가 연결되지 않은)에서도 실행할 수 있어야 합니다. 항상 실행가능한 코드로 만들어주는 것이 최선입니다!!

FIRST 법칙

자가검증하는(Self-Validating)
테스트는 boolean값으로 결과를 내야 합니다. 성공 또는 실패로 통과 여부를 알리고 로그 파일을 읽게 만들어서는 안 됩니다. 통과 여부를 보려고 텍스트 파일 두 개를 수작업으로 비교하게 만들어서도 안됩니다. 테스트가 스스로 성공과 실패를 가늠하지 못한다면 판단은 주관적이 되며 지루한 수작업 평가가 필요하게 됩니다. 😇

적시(Timely)
테스트는 적시에 작성해야 합니다. 단위 테스트는 테스트하려는 실제 코드를 구현하기 직전에 구현한다. 실제 코드를 구현한 다음에 테스트 코드를 만들면 실제 코드가 테스트하기 어렵다는 사실을 발견하게 될것입니다. 어떤 실제 코드는 테스트하기 너무 어렵다고 결론이 날 것입니다. 지금은 테스트가 불가능하도록 실제 코드를 짜고있을 수 있습니다.

GIVEN-WHEN-THEN PATTERN

일반적인 테스트 패턴

```
public class FooTest {
    @Test
    void test() {
        //given
        //... 여러개의 테스트 데이터셋을 준비한다.
        //when
        //... 후에 동작을 수행한다.
        //then
        //... 후에 검증한다
    }
}
```

구현 예시 보시고
세미나를 마치도록 하겠습니다.