JUnit 프레임워크

목차 정리

단위 테스트(Unit Test) vs 통합 테스트(Integration Test)

- **단위 테스트 (Unit Test)**

: 하나의 모듈(애플리케이션에서 작동하는 하나의 기능 또는 메소드)을 기준으로 독립적으로 진행되는 가장 작은 단위의 테스트

→ 애플리케이션을 구성하는 하나의 기능이 올바르게 동작하는지 독립적으로 테스트하는 것으로, “어떤 기능이 실행되면 어떤 결과가 나온다” 정도로 테스트하는 것

- **통합 테스트 (Integration Test)**

: 모듈을 통합하는 과정에서 모듈 간의 호환성을 확인하기 위해 수행되는 테스트

→ 웹 페이지로부터 API를 호출하여 통합된 모듈들이 올바르게 연계되어 동작하는지 검증하는 것

TDD (Test Driven Development) : 테스트 주도 개발

테스트케이스 작성 후 실제 코드 개발 -> 리팩토링 진행 : Test First Development

JUnit -> TDD -> 애자일

[애자일 (Agile) 방법론]

```
Agile = '기민한, 날렵한'

: 소프트웨어 개발 방식 중 하나
작업 계획을 짧은 단위로 세워 고쳐 나가는 사이클을 지속적으로 반복 -> 고객의 요구 변화에 유연하며 신속하게 대응 가능한 개발 방법론

  # 애자일 프레임워크
  : 애자일 방법론을 따르는 개발 기법
  ex) Scrum, kanban, XP(eXtreme Programming) 등


애자일은 워터폴(waterfall; 폭포수) 방식의 단점을 보완한 방식

워터폴(waterfall) : 주문 -> 디자인 -> 기능구현 -> 테스팅 -> 배포
긴 계획을 짜고 그 안에서 순서대로 체계적으로 진행되는 전통 방식
=> 지나치게 계획과 절차에 의존하여 시간과 비용의 낭비 발생 가능

- 애자일(Agile) : (주문 -> 디자인 -> 기능구현 -> 테스팅 -> 배포) X 무한대
기능을 축소하여 주기를 짧게하고 한 주기를 돈 후 중간 테스트(피드백)을 많이 가지는 방식
  # 짧은 주기 : 스프린트(sprint)
```

좋은 단위 테스트의 특징

1) 1개의 테스트 함수에 대해 assert 최소화

2) 1개의 테스트 함수는 1가지 개념만 테스트

[FIRST 규칙] - CleanCode 책

(1) Fast : 테스트는 빠르게 동작하여 자주 돌릴 수 있어야 함

(2) Independent : 각각의 테스트는 독립적이며 서로 의존 X

(3) Repeatable : 어느 환경에서든 반복 가능

(4) Self-Validating : 테스트는 성공 또는 실패로 Boolean값을 결과로 내어 자체적으로 검증되어야함

(5) Timely : 테스트는 테스트하려는 실제 코드를 구현하기 직전에 구현되어야 함

JUnit 기본 개념

Java에서 주로 사용되는 독립된 단위 테스트 프레임워크 중에 하나로 보이지 않고 숨겨진 단위 테스트를 끌어내어 정형화시켜 단위 테스트를 쉽게 해주는 테스트용 Framework

(**TestNG** 라는 프레임워크도 존재)

해당 부분에 대한 부가설명은 블로그 참고 후 설명 보태기

[JUnit 의 특징]

- @Test 메서드가 호출할 때마다 새로운 인스턴스가 생성되어 독립적인 테스트 가능
- 단위 테스트 Framework 중 하나
- 문자 혹은 GUI 기반으로 실행됨
- 단정문으로 테스트케이스의 수행결과를 판별
- 결과는 성공(녹색), 실패(붉은색) 표시
- 테스트 결과를 확인하는 것 이외에 최적화된 코드를 유추하는 기능 제공

JUnit 사용법

필요한 라이브러리 : JUnit 5, AssertJ

- **given/when/then 패턴**

  : 1개의 단위 테스트를 3가지 단계로 나누어 처리하는 패턴

  - **given(준비) : 어떠한 데이터가 준비되었을 때**
  - **when(실행) : 어떠한 함수를 실행하면**
  - **then(검증) : 어떠한 결과가 나와야 한다.**

JUnit 의 Stub

사전적 의미 : 토막, 꽁초, 남은 부분

개발 Test에서의 의미 : 실제로 준비는 되지 않았지만 원활한 테스트 동작을 위해 미리 정해진 답(하드코딩한 값)으로 반환하는 구현체

Stub은 주로 External Service(Database, Web Service)를 사용하는 code 테스트 시 사용된다.

- **사용 이유**

- 우리가 JUnit을 이용한 코드 테스트 시, Database와 connect하거나 Web Service이 call 할 수 있는지 테스트하려고 하지 않음. 이런 부분을 하드코딩으로 대체하고 business logic에 집중하는 것의 stub의 사용 목적
- 단위 테스트는 해당 모듈에 대한 독립적인 테스트이지만 다른 객체와 메시지를 주고 받는 모듈일 경우, 다른 객체 대신 가짜 객체(Mock Object)를 주입하여야 함. 그게 바로 Stub

Ex)

구글 API를 사용해야하는 business logic 존재

해당 구글 API의 반환값을 아예 하드코딩하는 것이 **Stub**

자세한 부분은 블로그 다시 참고하기

https://minholee93.tistory.com/entry/JUnit-Stub

[참고]

JAVA 에서 잘 쓰이는 단위테스트 : JUnit / TestNG

JUnit + Mockito

==========

230921_세미나 : Spring JUnit

TDD에 대해서 간략하게 설명

java spring JUnit

JUnit 구현

이번 API를 이용해서 JUnit 어떻게 다루는지(구현하는지)

==========

[참고자료]

기본 https://mangkyu.tistory.com/143

JUnit Stub https://minholee93.tistory.com/entry/JUnit-Stub

https://kimcoder.tistory.com/418

https://okky.kr/questions/269348