개발자를 위한 단위 테스트

목차

테스트 선행 방식의 프로그래밍 장점

- 자동화된 테스트
  - 단순한 오류 예방
  - 코딩 전에 그 코드에 기대하는 동작을 정의하는 설계 보조 수단
  - 모든 구현 후의 검증이 아닌 설계 단계부터 검증 적용

Junit이란?

컴퓨터 프로그래밍에서 소스 코드의
특정 모듈이 의도된 대로 정확히 작동하는지
검증하는 절차

Java 기반의 기능 단위 테스팅(Unit Testing) 도구
(Framework)

Test 코드의 필요성

단위 test는 개발 단계 초기의 문제를 발견하게 도와줌

개발자가 나중에 코드를 리팩토링 하거나 library 업그레이드 등에서 기존 기능이 올바르게 작동하는지 확인할 수 있음(예, 회귀 테스트)

단위 테스트는 기능에 대한 불확실성을 감소 시킬 수 있음

단위 test는 시스템에 대한 실제 문서를 제공, 즉 단위 테스트 자체가 문서로 사용 될 수도 있음

Unit Test 원칙

- F.I.R.S.T 원칙
  - 일반적으로 단위 테스트 코드를 작성할 때의 5가지 원칙 강조

F – Fast (테스트 코드를 실행하는 일은 오래 시간을 소요하지 말아야 함)

I – Independent (독립적으로 실행이 되어야 함)

R – Repeatable (반복 가능해야 함)

S – Self Validating (메뉴얼 없이 테스트 코드만 실행해도 성공, 실패 여부를 알 수 있어야 함)

T – Timely (바로 사용 가능해야 함)

단위테스트 작성 방법

하나의 테스트케이스에 최소한의 기능만 검증하고, 최대한 간결하게 작성한다.

입력값에 대한 결과 값을 검증하는 방식으로 작성하라

최소한 같은 팀이라면 테스트의 중요성과 코드 품질을 지키기 위한 노력에 대해 충분히 공감하고 있어야 한다

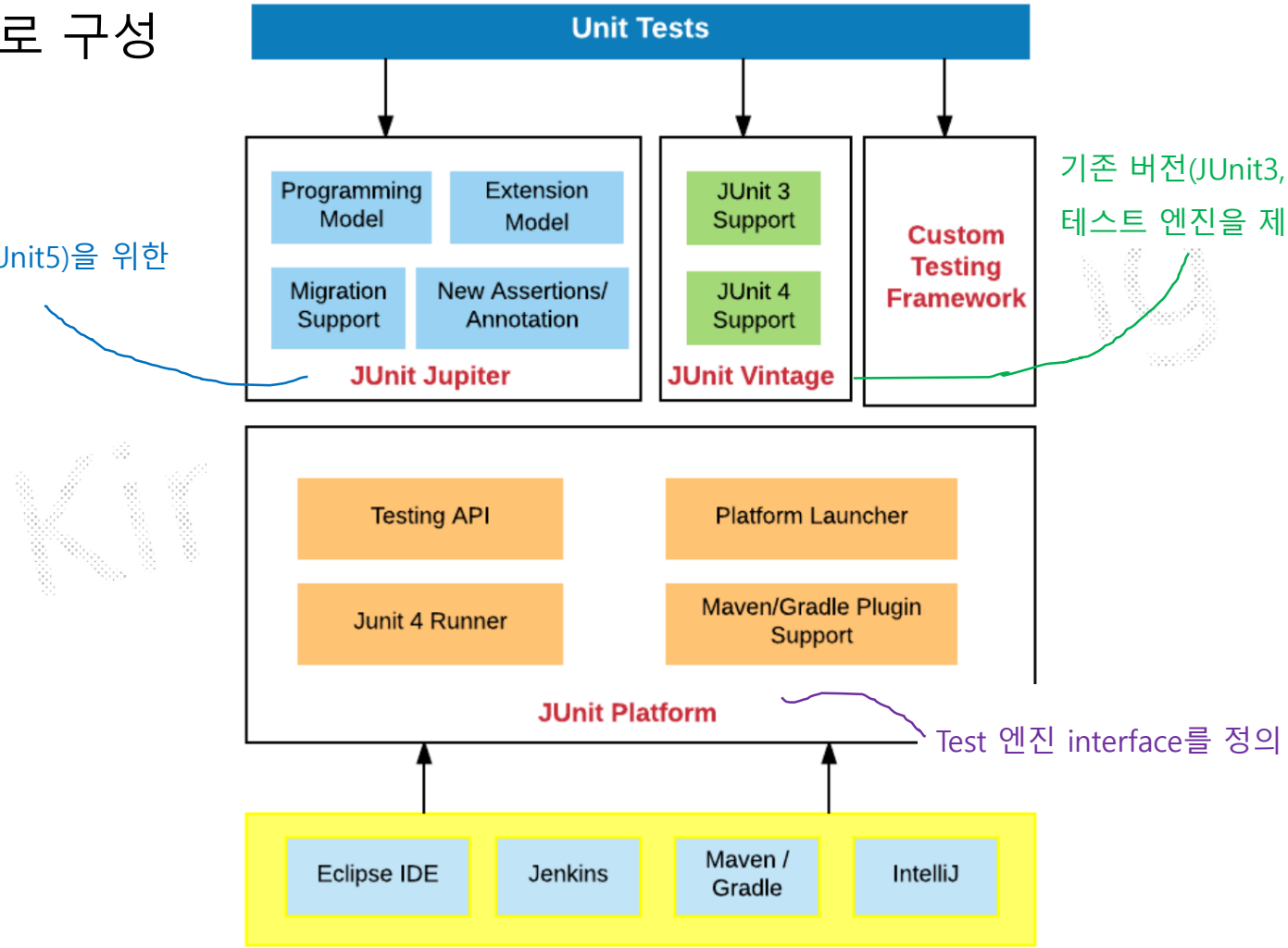
각 테스트는 독립적이어야 한다.

테스트를 위한 코드는 제품 코드에서 분리되어야 한다

....

JUnit5 컴포넌트

- 3개의 컴포넌트로 구성



새로운 프로그래밍 모델(JUnit5)을 위한 테스트 엔진 제공

기존 버전(JUnit3, JUnit4)을 위한 테스트 엔진을 제공

Test 엔진 interface를 정의

JUnit5 사용을 위한 maven기반의 설정

- pom.xml

```xml
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-runner</artifactId>
    <version>1.0.1</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.0.1</version>
</dependency>

<!-- parameter 적용을 위한 설정 -->
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-params</artifactId>
    <version>5.0.1</version>
    <scope>test</scope>
</dependency>
```

JUnit5 애노테이션

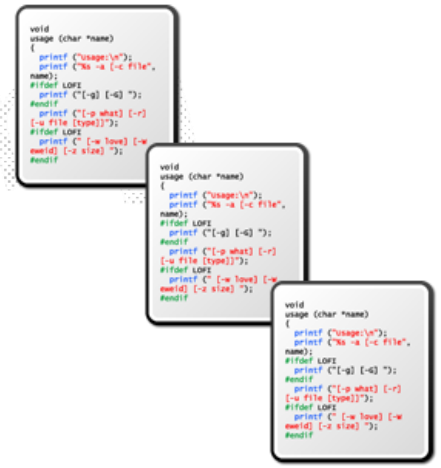
애노테이션	특징	JUnit4 vs JUnit5
@BeforeAll	모든 test 실행 전 최초 한번 실행 static 메소드 여야만 함	JUnit4의 @BeforeClass
@BeforeEach	test 실행할 때마다 test 전에 실행	JUnit4의 @Before
@Test	test 실행 static 메소드에 적용 불가	JUnit4의 @Test
@AfterEach	test 종료할 때마다 test 이후 실행	JUnit4의 @After
@AfterAll	모든 test 종류 후 마지막 실행 static 메소드 여야만 함	JUnit4의 @AfterClass
@Disabled	test를 수행하지 않고 path	JUnit4의 @Ignore
@DiplayName	test 이름 설정(메소드명 대신 출력)	

@BeforeAll -> @BeforeEach -> @Test -> @AfterEach -> @AfterAll

Assertion(단정문)

true 일거야...란 가정하에 true 여부 검증

false 일거야...란 가정하에 false 여부 검증



프로그래머가 맞다는 가정하에 상황에 대한 체크로 true/false로 검증

Assertion(단정문)

- 이미 구현된 프로그램의 **적합성을 논리적으로 테스트**하는 기능
  - boolean값을 가지는 표현식을 포함하는 하나의 명령문
    - 실행 로직이 정확하게 실행되었을 경우 true
    - 그렇지 않을 경우 false 반환
- 예외와의 차이점
  - 예외 : 실행 중 발생하는 예외를 단순히 지정된 핸들러로 처리
  - assertion : 프로그래머가 맞다는 가정하에 상황에 대한 체크로 true/false로 검증
- 한계
  - 부적절하게 사용되는 경우 개발자 or 테스터들의 일을 돕는 대신 오히려 방해되는 요인이기도 함

Junit & Assertion 적용 방식

- Assert(단정) 메소드로 Test 케이스의 수행 결과를 판별
  - 예 : assertEquals(예상값, 실제값)

- 원리
  - @Test 애노테이션이 선언된 메소드가 호출될 때마다 새로운 instance를 생성하여 독립적인 test 진행

Assertion(단정문) 적용 방법

메소드명	특 징
assertArrayEquals	두 배열이 똑같은 값을 가진 똑같은 크기의 배열인지 확인
assertEquals	두 값이 똑같은 값인지 확인
assertTrue	해당 값이 True인지 확인
assertFalse	해당 값이 False인지 확인
assertNull	해당 값이 Null인지 확인
assertNotNull	해당 값이 null이 아닌지 확인
assertSame	두 개체가 같은 개체인지 확인
assertNotSame	두 개체가 같은 개체가 아닌지 확인
fail	무조건 실패
assertThat	해당 개체가 특정 상황을 만족하는지 확인