
# JUnit

## 자바 단위 테스트 도구

김기정 (bangry313@gmail.com)

# 단위 테스트 정의


## ✓ 단위 테스트 (Unit Test)

- 단위 테스트는 하나의 모듈을 기준으로 독립적으로 진행되는 가장 작은 단위의 테스트이다.
  - 모듈은 애플리케이션에서 작동하는 하나의 기능 또는 메소드를 의미한다.
  - 모듈 예 : 웹 애플리케이션에서 로그인 메소드에 대한 독립적인 테스트가 1개의 단위테스트가 될 수 있다.
- 단위 테스트는 애플리케이션을 구성하는 하나의 기능이 올바르게 동작하는지를 독립적으로 테스트하는 것으로, "어떤 기능이 실행되면 어떤 결과가 나온다" 정도로 테스트를 진행한다.

## ✓ 통합 테스트(Integration Test)

- 통합 테스트는 모듈들을 통합하는 과정에서 모듈 간의 호환성을 확인하기 위해 수행되는 테스트이다.
- 일반적으로 애플리케이션은 여러 개의 모듈들로 구성이 되고, 모듈들끼리 메세지를 주고 받으면서(메소드 호출) 기능을 수행하게 되는데 통합된 모듈들이 올바르게 연계되어 동작하는지 검증하는 것이 통합 테스트이다.

# 단위 테스트 필요성


✓ **일반적으로 실무에서 얘기하는 테스트는 거의 단위 테스트를 의미한다.**

- 통합 테스트는 메로리 캐시나 데이터베이스 등 다른 컴포넌트들과 실제 연결을 해야 하고, 시스템을 구성하는 컴포넌트들이 많아질수록 테스트를 위한 비용(시간)이 상당히 커진다.

✓ **반면에 단위 테스트는 해당 부분만 독립적으로 테스트하는 것이기 때문에 어떤 소스 코드를 리팩토링 하여도 빠르게 문제 여부를 확인할 수 있다.**

- 테스팅에 대한 시간과 비용을 절감할 수 있다.
- 새로운 기능 추가 시에 수시로 빠르게 테스트 할 수 있다.
- 리팩토링 시에 안정성을 확보할 수 있다.

✓ **요즘 트렌드인 TDD(Test-Driven Development, 테스트 주도 개발 방법) 에서 얘기하는 테스트도 단위 테스트를 의미한다.**

- 개발자가 작성한 테스트 코드를 수시로 빠르게 실행하면서 문제를 파악할 수 있다.

✓ **성공적인 TDD를 위한 선행 조건**

- 요구사항이 명확하고 구체적일수록 기능이 구체적이며 신뢰성이 증가한다.
- 서비스의 기능적 요구사항이 개발의 전부가 아니며, 비기능적(성능, 신뢰성 등) 요구사항도 동등하게 중요하다.

✓ **자바 프로젝트 단위 테스트를 진행할 때 대부분 JUnit이라는 단위 테스트 도구를 사용한다.**

- 입력과 출력의 예상값과 실제값을 비교하여, 요구사항이 명확히 수행되었는지 확인할 수 있다.

# JUnit 소개

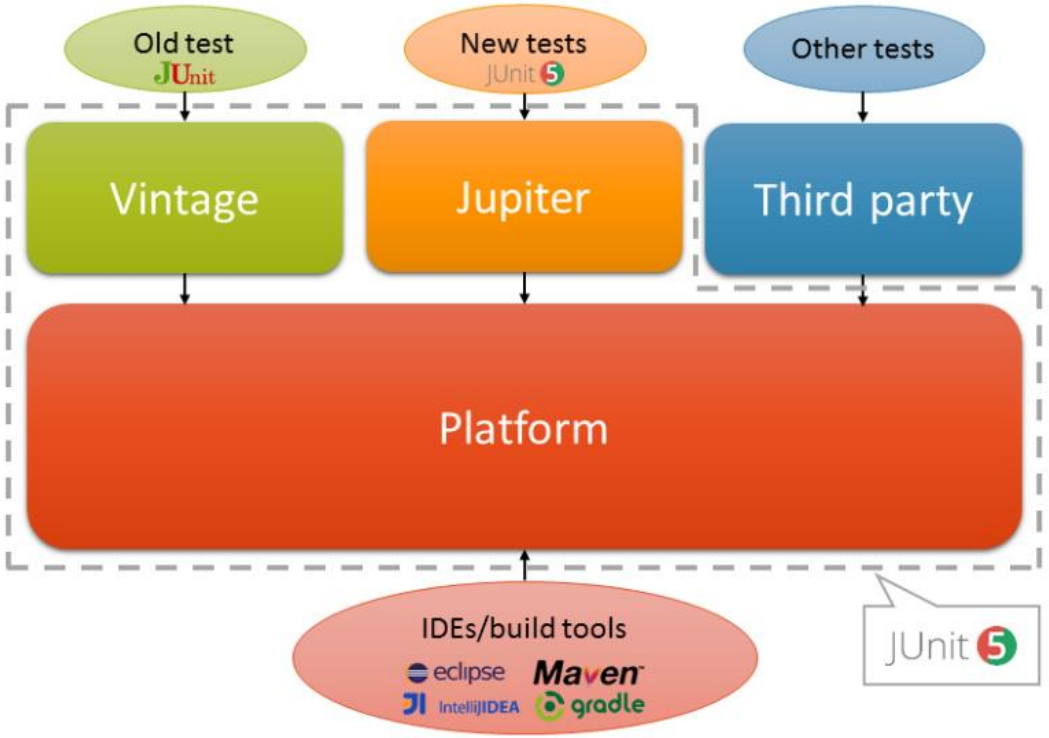
JUnit 5

✓ Junit은 자바 개발자 대부분이 사용하는 자바 프로그래밍 언어용 단위 테스트 도구(프레임워크) 이다.

- 1998년 켄트 벡과 에릭 감마에 의해 최초 개발 되었으며, 2022년 9월 발표된 5.9.1 버전이 최신 안정화 버전이다.
- 애플리케이션 테스트 클래스를 작성하여 System.out.println("로그내용") 으로 번거롭게 디버깅 하지 않아도 된다.
- 또한 프로그램 테스트 시 수행 시간도 검증할 수 있다.
- 대부분의 자바 IDE에 플러그인 형태로 기본 내장되어 있으며, Spring boot 2.2 버전 이상부터 기본 제공된다.

✓ JUnit5 구성 요소

- Platform
  - 테스트를 실행해 주는 런처와 TestEngine 인터페이스를 제공한다.
- Vintage
  - TestEngine 인터페이스 구현체로 JUnit3, 4에서 제공한다.
- Jupiter
  - Test Engine 인터페이스 구현체로 JUnit5에서 제공한다.



# JUnit 특징


✓ @Test 메소드가 호출할 때마다 새로운 객체가 생성되어 독립적인 테스트 가능하다.

✓ 단정문(assert)으로 테스트 케이스의 수행 결과를 판별할 수 있다.

- 입력과 출력의 예상값과 실제값을 비교하여, 요구사항이 명확히 수행되었는지 확인할 수 있다.
- 예) assertEquals(예상값, 실제값)

✓ JUnit4부터는 Annotation을 지원하여, 간결하게 테스트를 수행할 수 있다.

# Junit을 활용한 자바 단위 테스트 작성 준비 (1/3)

✓ 자바 단위테스트 작성에는 2가지 라이브러리가 주로 사용된다.

✓ JUnit5 : 자바 단위 테스트를 위한 기본 테스트 프레임워크

- 기본 제공하는 assertNotNull(), assertEquals() 등과 같은 테스트 메소드 만으로도 단위테스트를 충분히 작성할 수 있다.

✓ AsserJ : 자바 단위 테스트를 돕기 위해 다양한 단위 테스트 API를 지원하는 3rd Party 라이브러리

- AssertJ는 많은 Assertion 메소드와 오류 메시지 테스트를 지원하고, 테스트 코드 가독성을 향상시키며 훨씬 유연한 단위 테스트를 작성할 게 도와준다.

✓ 실무에서는 자바 애플리케이션 단위 테스트를 위해 JUnit5와 AssertJ 조합하여 많이 사용한다.

# Junit을 활용한 자바 단위 테스트 작성 준비 (2/3)

✓ 빌드 도구 (maven, gradle)의 JUnit5 의존성 (dependency) 추가

Sort: **relevance** | popular | newest

1. **JUnit Jupiter API**

13,412 usages

org.junit.jupiter » junit-jupiter-api

EPL

JUnit Jupiter is the API for writing tests using JUnit 5.

Last Release on Jul 23, 2023

- Maven

```
<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.9.0</version>
    <scope>test</scope>
</dependency>
```

- Gradle

```
// https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api
testImplementation 'org.junit.jupiter:junit-jupiter-ap:5.9.0'
```

# Junit을 활용한 자바 단위 테스트 작성 준비 (3/3)

✓ 빌드 도구 (maven, gradle)의 AssertJ 의존성 (dependency) 추가

Sort: **relevance** | popular | newest

1. **AssertJ Core**

15,731 usages

org.assertj » assertj-core

Apache

Rich and fluent assertions for testing in Java

Last Release on Jan 16, 2023

- Maven

```
<!-- https://mvnrepository.com/artifact/org.assertj/assertj-core -->
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>3.22.0</version>
    <scope>test</scope>
</dependency>
```

- Gradle

```
// https://mvnrepository.com/artifact/org.assertj/assertj-core
testImplementation 'org.assertj:assertj-core:3.22.0'
```

# JUnit 사용하기 – Annotation

✓ JUnit5 (Jupiter API)는 테스트 코드 작성을 위해 다음과 같은 Annotation을 지원한다.

- 기본 패키지 : org.junit.jupiter.api

| JUnit5 Annotation | 설명 | JUnit4 |
|---|---|---|
| @Test | 테스트 메소드 임을 선언 | |
| @BeforeEach | 각각의 테스트 메소드가 실행되기 전에 실행되어야 하는 메소드 임을 선언 | @Before |
| @AfterEach | 각각의 테스트 메소드가 실행된 후 실행되어야 하는 메소드 임을 선언 | @After |
| @BeforeAll | @BeforeEach는 각각의 테스트 메소드 마다 실행되지만,<br>@BeforeAll은 각각의 테스트 메소드가 실행되기 전 딱 한 번만 실행 | @BeforeClass |
| @AfterAll | 각각의 테스트 메소드가 실행된 후 딱 한 번만 실행 | @AfterClass |
| @DisplayName | 테스트 클래스 또는 메소드의 사용자 정의 이름을 선언 | |
| @Disabled | 테스트 클래스나, 메소드의 테스트를 비활성화 | @Ignore |
| @Timeout | 주어진 시간안에 테스트가 끝나지 않으면 실패 | |
| @TestMethodOrder | 테스트 메소드 실행 순서를 선언 | |

# JUnit 사용하기 – Annotation

✓ JUnit5 (Jupiter API)는 테스트 코드 작성을 위해 다음과 같은 Annotation을 지원한다.

```java
public class JUnitAnnotationTest {
    @BeforeAll
    static void initAll() {
        System.out.println("--- 각각의 테스트 메소드 실행 전 딱 한번 실행 ---");
    }
    @BeforeEach
    void init() {
        System.out.println("각각의 테스트 메소드 실행 전 실행");
    }
    @Test
    public void succeedingTest1() {
        System.out.println("테스트 메소드 실행1...");
    }
    @Test
    public void succeedingTest2() {
        System.out.println("테스트 메소드 실행2...");
    }
    @AfterEach
    void destroy() {
        System.out.println("각각의 테스트 메소드 실행 후 실행");
    }
    @AfterAll
    static void destroyAll() {
        System.out.println("--- 각각의 테스트 메소드 실행 후 딱 한번 실행 ---");
    }
}
```

# JUnit 사용하기 – Assertions 메소드

✓ Junit5에는 JUnit4의 여러 Assertion 메서드 외에 Java8 람다식과 함께 사용하기에 적합한 몇 가지를 추가하였다.

- 모든 JUnit5 Assertion 메소드는 org.junit.jupiter.api.Assertions 클래스의 static 메서드이다.
- static import 사용 가능

✓ 주요 Assertion 메소드

- assertEquals(x, y)
  - x, y가 일치함을 단언(주장)한다. x(예상값)와 y(실제값)가 같으면 통과
- assertNotEquals(x, y)
- assertArrayEquals(x, y)
  - 배열 x, y가 일치함을 단언(주장)한다. X와  y가 같으면 통과
- assertTrue(x)
  - x가 true임을 단언한다.
- assertNull(x)
  - x가 null임을 단언한다.
- assertSame(x, y)
  - x와 y가 같은 객체를 참조하고 있음을 단언한다.
- fail()
  - 테스트를 바로 실패 처리한다.

# JUnit 사용하기 – Assertions 메소드

✓ 주요 Assertion 메소드

```java
/**
 * JUnit5 테스트를 위한 비즈니스 객체
 */
public class SomeServiceImpl /* implements SomeService */{

    public int sum(int x, int y) {
        return x + y;
    }

    public String getMessage() {
        return "Hello. JUnit5~~~";
    }

}
```

# JUnit 사용하기 – Assertions 메소드

✓ 주요 Assertion 메소드

```java
/** JUnit 테스트 클래스 */
public class SomeServiceTest {
    SomeServiceImpl someServiceImpl;

    @BeforeEach
    void init() {
        someServiceImpl = new SomeServiceImpl();
    }

    @Test
    void sumTest() {
        assertEquals(10, 10);
        assertEquals(10, someServiceImpl.sum(5, 10));
    }

    @Test
    @DisplayName("메시지 획득")
    void getMessageTest() {
        assertNotNull(someServiceImpl);
        assertNotNull(someServiceImpl.getMessage());
    }

    @Test
    void failTest() {
        fail("아직 구현되지 않았습니다.");
    }
}
```

# JUnit 사용하기 – Assertions 메소드

✓ @TestMethodOrder Annotation

- Junit은 기본적으로 테스트 메소드의 실행 순서를 보장하지 않는다.
- 순서를 설정할 수 있는 다양한 Annotation을 지원한다.

```java
@TestMethodOrder(value = MethodOrderer.MethodName.class)
public class TestOrderTest {

    @Test
    public void aMethodTest() {
        System.out.println("aMethod()실행...");
    }

    @Test
    public void bMethodTest() {
        System.out.println("bMethod()실행...");
    }

    @Test
    public void cMethodTest() {
        System.out.println("cMethod()실행...");
    }

    @ParameterizedTest
    @ValueSource(ints = {10})
    public void dMethodTest(int x) {
        System.out.println("dMethod()실행 : " + x);
    }
}
```

# JUnit 사용하기 – Assertions 메소드

✓ @TestMethodOrder Annotation

- Junit은 기본적으로 테스트 메소드의 실행 순서를 보장하지 않는다.
- 순서를 설정할 수 있는 다양한 Annotation을 지원한다.

```java
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
//@TestMethodOrder(MethodOrderer.Random.class)
public class TestOrderTest {

    @Test
    @Order(3)
    public void aMethodTest() {
        System.out.println("aMethod()실행...");
    }

    @Test
    @Order(2)
    public void bMethodTest() {
        System.out.println("bMethod()실행...");
    }

    @Test
    @Order(1)
    public void cMethodTest() {
        System.out.println("cMethod()실행...");
    }
}
```

# AssertJ 사용하기 – Assertions 메소드

✓ AssertJ를 사용하면 많은 Assertion과 오류 메시지 테스트를 지원하고, 테스트 코드 가독성을 향상시키며 훨씬 유연한 단위 테스트를 작성할 게 도와준다.

- 기본 패키지 : org.assertj.core.api;

✓ AssertJ의 모든 테스트 코드에 assertThat() 메소드를 일관성 있게 사용할 수 있다.

- 사용 형식 : assertThat(Target).테스트메소드().테스트메소드();
- 사용될 수 있는 테스트 메소드 종류
  - isEqualTo(e), contains(e), doesNotContain(e), startsWith(e), endsWith(e), isNotEmpty(), isPositive(n), isGreaterThan(n) 등

✓ 요즘 단위 테스트는 거의 given-when-then 패턴으로 작성하는 추세이다.

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

# AssertJ 사용하기 – Assertions 메소드

✓ Assertion 주요 메소드

```java
public class AssertJTest {

    @Test
    void methodTest() {
        String message = "JUnit is unit test tool...";
        assertThat(message).isNotNull().contains("unit");
    }

    @Test
    void sumTest() {
        // given
        int x = 100, y = 200;
        SomeServiceImpl someServiceImpl = new SomeServiceImpl();
        // when
        int result = someServiceImpl.sum(x, y);
        // then
        assertThat(result).isPositive().isEven().isEqualTo(300);
    }
}
```

# End of Document

✓ Q&A


감사합니다...