JUnit

개요

- xUnit
  - 유닛 테스트 프레임워크(Unit test framework)를 통틀어서 칭하는 명칭

- 유닛 테스트 (Unit Test)
  - 컴퓨터 프로그램의 특종 모듈이 의도된 대로 정확히 작동하는지 검증하는 테스트
  - 모듈의 모든 함수 및 메소드에 대한 테스트 케이스를 수행

- xUnit 종류

xUnit	언어	관련 사이트
CUnit	C	http://cunit.sourceforge.net/
CppUnit	C++	https://sourceforge.net/projects/cppunit/
PHPUnit	PHP	https://phpunit.de/
PyUnit	Python	http://pyunit.sourceforge.net/
JUnit	Java	http://junit.org/

JUnit 단정 메소드

1. 단정 메소드 (Assert method)
   - 구현된 메소드들이 올바르게 작동하는지 확인하는 메소드
   - 테스트 케이스의 수행 결과를 판별

2. 단정 메소드 종류

assert	설명
assertEquals(expected, actual) assertEquals(expected, actual, delta) assertEquals(String message, expected, actual) assertEquals(String message, expected, actual, delta)	- actual의 값이 expected와 동일한지 테스트 - actual과 expected가 실수이면, delta보다 작은 차이는 같은 값으로 간주 - actual과 expected가 사용자 정의 데이터 형이면, 그것들의 equals 메소드를 호출하여 비교
assertNull(Object obj) assertNull(String message, Object obj)	- obj가 null을 참조하는지 테스트

assertNotNull(Object obj) assertNotNull(String message, Object obj)	- obj가 실제 객체를 참조하는지 테스트
assertTrue(boolean condition) assertTrue(String message, boolean condition)	- 조건식(condition)의 계산 결과가 참인지 테스트
assertFalse(boolean condition) assertFalse(String message, boolean condition)	- 조건식(condition)의 계산 결과가 거짓인지 테스트
assertSame(Object expected, Object actual) assertSame(String message, Object expected, Object actual)	- expected와 actual가 참조하는 객체가 같은지 테스트
assertNotSame(Object expected, Object actual) assertNotSame(String message, Object expected, Object actual)	- expected와 actual가 참조하는 객체가 다른지 테스트
fail() fail(String message)	- 테스트 결과를 실패로 처리

예시) JUnit 테스트

※ 여기서는 JUnit을 Java 패키지에 어떻게 Install 하는지는 알려주지 않는다.

1. 간단한 계산기 클래스 구현 (.../src/main/java/Calcurator.java)

```
public class Calcurator {
   public double sum(double a, doubleb){
      return a + b;
   }
}
```

2. JUnit 테스트 클래스 구현 (.../src/test/java/CaluratorTest.java)

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class CaluratorTest {                    ← ①
   @Test                                        ← ②
   public void testSum(){
      Calcurator c = new Calcurator();
      double result = c.sum(10, 50);            ← ③
      assertEquals(60, result, 0);              ← ④
   }
}
```

① 테스트 클래스는 반드시 public으로 선언, 테스트 클래스는 이름은 관례로 클래스명+Test로 작성

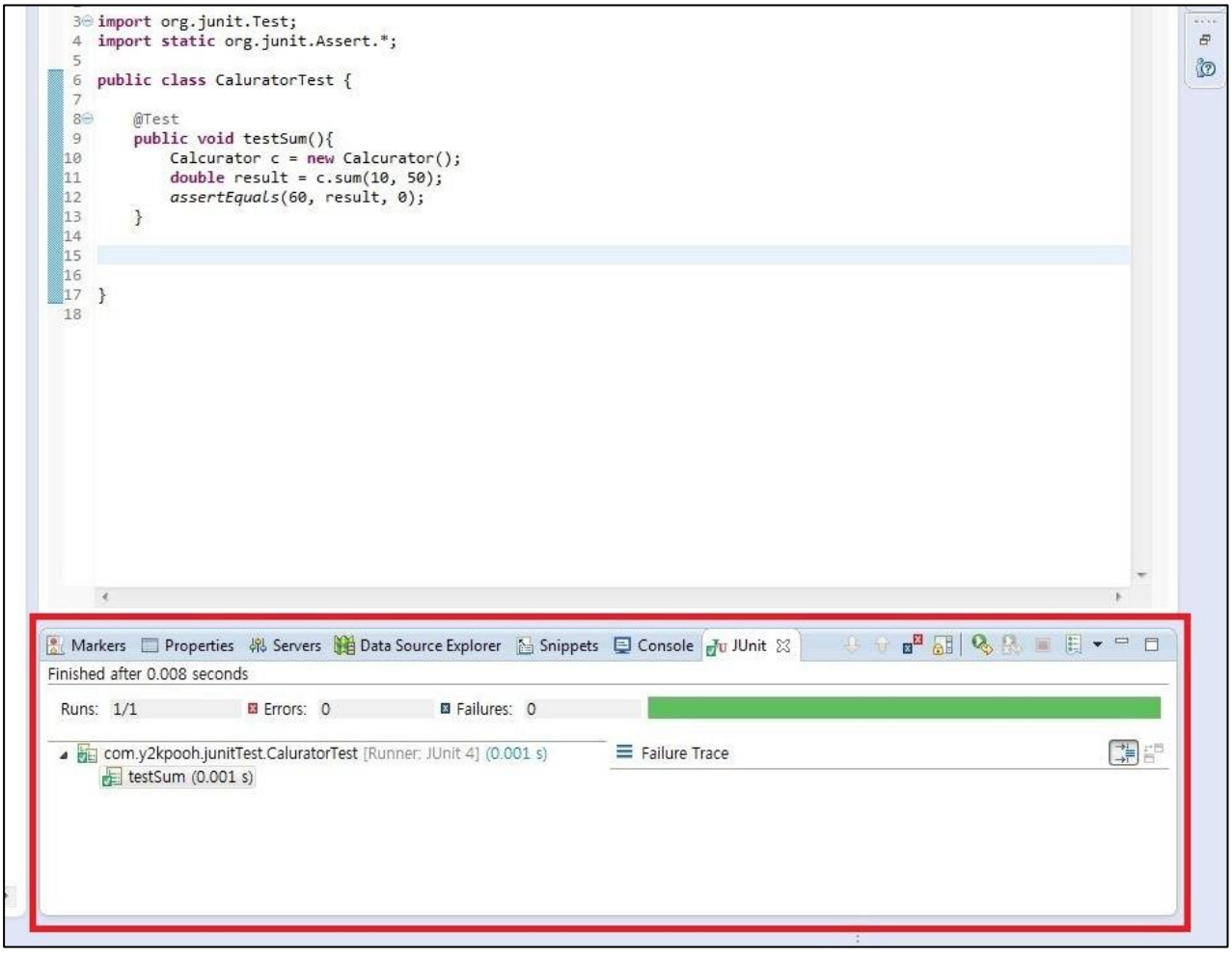
② @Test 어노테이션을 testSum() 메소드에 선언하여 testSum()이 단위 테스트 메서드임을 선언

③ result 변수에 Calcurator클래스의 sum() 메소드의 결과값을 반환

④ JUnit의 assert클래스의 정적 메소드인 assertEquals()를 이용하여 테스트 결과값을 확인

✓ c.sum()을 이용해 10과 50을 더한 값이 result에 담겼으며 assertEquals()를 통해 result에 저장된 값과 60이 동일 값인지 확인하는 단위 테스트이다.

3. 테스트 결과확인 (테스트를 성공한 화면)



JUnit 어노테이션

- @Test
  - 이 어노테이션이 특정 메소드 위에 선언되면, 해당 메소드는 테스트 대상 메소드임을 의미
  - @Test(timeout=5000)로 특정 메소드 위에 선언 시 해당 메소드의 테스트 결과가 5,000ms를 초과하면 테스트 실패
  - @Test(expected=RuntimeException.class)로 특정 메소드 위에 선언 시 해당 메소드의 테스트 결과로 RuntimeException이 발생해야 테스트 성공, 그렇지 않으면 실패

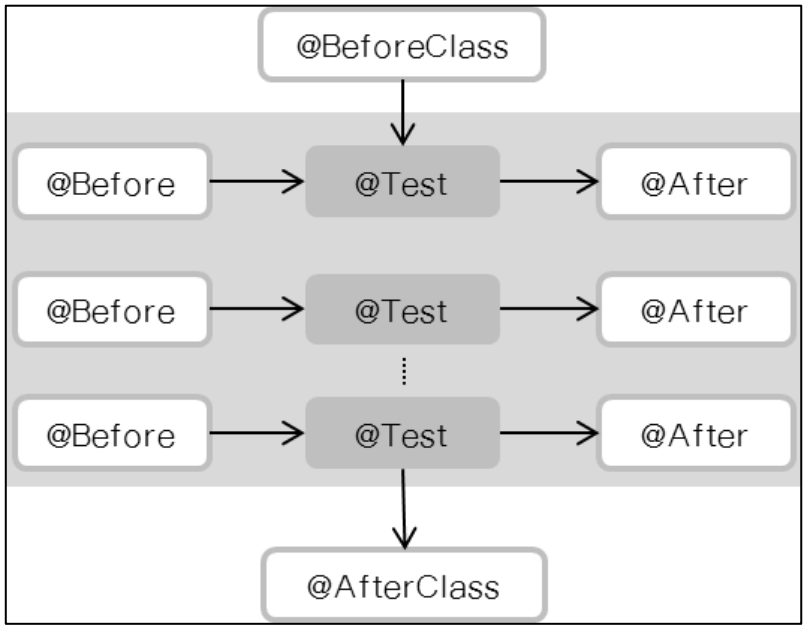
- @Before
  - 이 어노테이션이 특정 메소드 위에 선언되면, 해당 메소드는 특정 테스트 메소드를 **'실행'**전에 실행되는 메소드임을 의미
  - 주로 테스트 메소드를 실행 전에 초기화 작업을 수행할 때 사용

- @After
  - 이 어노테이션이 특정 메소드 위에 선언되면, 해당 메소드는 특정 테스트 메소드를 **'실행'**후에 실행되는 메소드임을 의미
  - 주로 테스트 메소드를 실행 후에 해제 작업을 수행할 때 사용

- @BeforeClass
  - 이 어노테이션이 특정 메소드 위에 선언되면, 해당 메소드는 단위(Unit) 테스트 **'시작'**전 딱 한번 실행되는 메소드임을 의미
  - 주로 테스트 실행 전 Network 및 DB 연결 등의 초기화 작업을 수행할 메소드에 사용

- @AfterClass
  - 이 어노테이션이 특정 메소드 위에 선언되면, 해당 메소드는 단위(Unit) 테스트가 **'끝난 뒤'** 딱 한번 실행되는 메소드임을 의미
  - 주로 테스트가 끝난 뒤 Network 및 DB 연결해제 등의 해제 작업에 사용

✓ JUnit Test Flow



참고문헌

- 도리안 (2013). https://m.blog.naver.com/PostView.nhn?blogId=netrance&logNo=110175973047&proxyReferer=https:%2F%2Fwww.google.com%2F
- wallel (2018). https://epthffh.tistory.com/entry/Junit%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-%EB%8B%A8%EC%9C%84%ED%85%8C%EC%8A%A4%ED%8A%B8
- 권영재 (2017). https://nesoy.github.io/articles/2017-02/JUnit
- Nextree (2013). http://www.nextree.co.kr/p11104/