

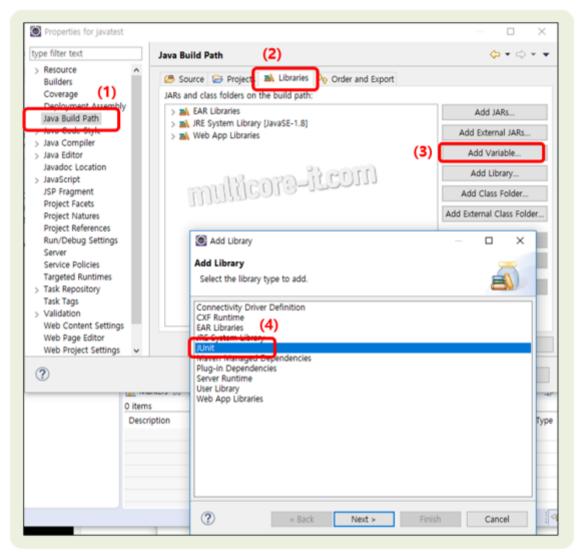
## 대표적인 단위테스트 도구 JUnit

2018.11.19 15:25

#JUnit

## ☐ 대표적인 단위테스트 도구 JUnit

JUnit은 이클립스로 개발하는 자바 환경에서 가장 많이 사용되는 단위 테스트 도구이다. 테스트 함수를 단일 패키지로 묶을 수 있으며 다양한 검증 함수를 지원해 프로그램이 반복적으로 수정되는 프로그램 개발 과정에서 테스트 케이스를 반복적으로 적용할 수 있다. 단위 테스트 시간을 획기적으로 줄일 수 있을 뿐만 아니라 테스트 품질도향상시킬 수 있다.



JUnit, 설정

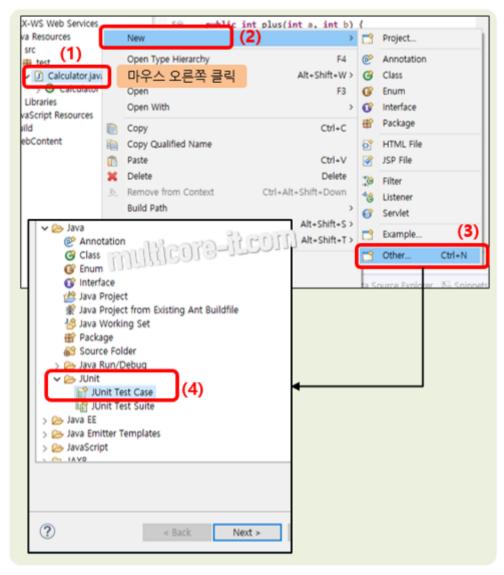
JUnit을 이클립스에 기본적으로 내장되어 있지만 사용을 위해서는 라이브러리를 추가해 줘야 한다. 이클립스 프로젝트 환경설정에서 Java Build Path > Library > Ad d Library에서 JUnit을 추가해주면 된다.

```
🗾 Calculator.java 🖂
    package test;
    public class Calculator {
 5⊝
        public int plus(int a, int b) {
            return a+b;
 6
 8
 9⊝
        public int minus(int a, int b) {
10
            return a-b;
11
12
13⊖
        public int multiple(int a, int b) {
14
            return a*b;
15
16
        public float divide(int a, int b) {
17⊝
18
            return a/b;
19
 20
 21
22
 23
```

=

## 사칙연산 프로그램 개발

본격적으로 JUnit을 사용하기 전에 테스트를 위해 사칙 연산하는 클래스 Calculator.java를 만들어보자. Calculator.java는 plus, minus, multiple, divide 이렇게 모두 4 개의 메소드를 가지고 있다.



JUnit 클래스 생성 1/3

이제 JUnit 테스트 클래스를 만들어보자. 좌측 Project Explorer 영역에서 테스트할 클래스를 선택하고 마우스 오른쪽 버튼을 클릭해 [New > Other] 메뉴를 선택하면 새로운 팝업 창이 새로 나온다. 아래로 좀 내려오면 JUnit 영역이 나오고 한번 더 클릭해 JUnit Test Case를 선택하자.



JUnit 클래스 생성 2/3

JUnit 테스트 케이스 생성화면에서 파일이 생성될 패키지 이름, 클래스 이름 그리고 어떤 클래스를 테스트할 지 정보를 확인할 수 있다. 실제로 JUnit을 사용할 때는 별도의 패키지를 만들어서 JUnit 테스트 케이스 클래스만 별도로 모아두는 것이 좋다. 운영에 적용할 때 JUnit 테스트 케이스 클래스는 필요가 없기 때문에 삭제하거나 운영으로 기관이 안되도록 설정하는 것이 좋다. 별도의 패키지에 파일을 모아 놓으면 이런 설정을 쉽게 할 수 있다. 이 책에서는 편의를 위해 테스트 대상 클래스와 같은 패키지 플레 JUnit 클래스를 만들도록 한다.

(1)



```
Test Methods
 Select methods for which test method stubs should be created.
Available methods:

☑ CalculatorTest.java 
☒
                                                              Calculator.java

✓ ✓ ✓ O Calculator

                                                              1 package test;
        plus(int, int)
        minus(int, int)
                                                                3@ import static org.junit.jupiter.api.Assertions.*;[]
        multiple(int, int)
        divide(int, int)
                                                                7 class CalculatorTest {
     ☐ ⊙ Object
        Object()
                                                                         void testPlus() {
    fail("Not yet implemented");
        ☐ d<sup>NF</sup> getClass()
☐ o N hashCode()
                                                               11
                                                               1100
        equals(Object)
        O N clone()
                                                                         void testMinus() {
    fail("Not yet implemented");
                                                               15
        □ o toString()
□ o<sup>NF</sup> notify()
□ o<sup>NF</sup> notifyAll()
                                                               16
                                                               17
                                                               18
        ☐ d wait(long)
                                                               198
                                                                         void testMultiple() {
4 methods selected.
                                                               21
                                                                             fail("Not yet implemented");
                                                               22
Create final method stubs
                                                               248
Create tasks for generated test methods
                                                                         void testDivide() {
    fail("Not yet implemented");
                                                               25
                                                               27
```

JUnit 클래스 생성 3/3

다음 버튼을 누르면 테스트할 메소드를 선택할 수 있다. 사칙 연산을 위한 모든 메소드를 선택하자. CalculatorTest.java 클래스가 생성됐고 사칙연산과 관련된 테스트 메소드 4개가 생성된 것을 확인할 수 있다. 아직 세부적인 테스트 케이스가 지정되지 않았기 때문에 "Not yet implemented"라고 지정되어 있다.

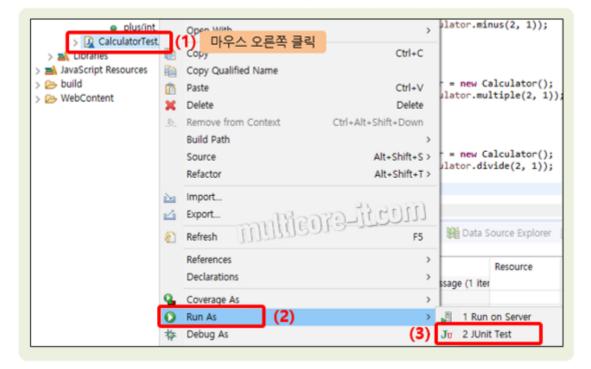
```
Calculator.java
                package test;
30 import static org.junit.jupiter.api.Assertions.*;
7
   class CalculatorTest {
8
9⊝
       void testPlus() {
10
           Calculator calculator = new Calculator();
           assertEquals(3, calculator.plus(1, 2));
12
13
14
15⊖
       @Test
       void testMinus() {
16
17
           Calculator calculator = new Calculator();
18
           assertEquals(3, calculator.minus(2, 1));
19
20
21⊖
       @Test
22
       void testMultiple() {
23
           Calculator calculator = new Calculator();
24
           assertEquals(2, calculator.multiple(2, 1));
25
26
27⊝
       void testDivide() {
28
29
           Calculator calculator = new Calculator();
30
           assertEquals(3, calculator.divide(2, 1));
31
32
33 }
```

JUnit 테스트 케이스 생성

이제 테스트 함수에 테스트 케이스를 포함하고 있는 검증 함수를 작성해서 JUnit 클래스를 완성해 보자. assertEqual은 두 개의 인자의 결과가 같으면 테스트 성공을 의미한다. JUnit은 다양한 검증 함수를 제공하며 JUnit에서 제공하는 API 문서에 다양한 함수에 대한 설명을 찾을 수 있다. http://JUnit.sourceforge.net/javadoc/org/JUnit/Assert.html 사이트를 참고해 보자.

≡





JUnit 단위테스트 실행

검증 함수까지 개발이 완료됐으면 이제 단위 테스트를 실행할 차례다. JUnit 클래스를 선택하고 마우스 오른쪽 버튼을 클릭하면 새 창이 나오는데 Run As > JUnit Test 를 선택하면 단위 테스트를 실행할 수 있다.

```
☑ CalculatorTest.java 

☒
 Calculator.java
  package test;
3⊕ import static org.junit.jupiter.api.Assertions.*;
  class CalculatorTest {
       void testPlus() {
           Calculator calculator = new Calculator();
           assertEquals(3, calculator.plus(1, 2));
       void testMinus() {
           assertEquals(3, calculator.minus(2, 1));
       @Test
       void testMultiple() {
           Calculator calculator = new Calculator();
           assertEquals(2, calculator.multiple(2, 1));
       void testDivide() {
           assertEquals(3, calculator.divide(2, 1));
Markers 🔲 Properties 🚜 Servers 🙀 Data Source Explorer 🔝 Snippets 🞳 Junit 🗯
ished after 0.172 seconds
                Errors: 0

□ Failures: 2

CalculatorTest [Runner: JUnit 5] (0.047 s)
                                                                       testMinus() (0.031 s)
  testDivide() (0.000 s)
```

JUnit 단위테스트 결과 분석

단위 테스트를 실행하면 화면 하단에 JUnit 이라는 이름의 탭이 하나 생기고 결과가 표시된다. 단위 테스트에는 모두 2개의 실패 (Failures)가 나왔는데 아래쪽으로 좀 더 내려와보면 testMinus와 testDivide 함수 앞에 x 표시가 된 것을 확인할 수 있다. 더블 클릭하면 화면 상단에 소스코드에 실패가 발생한 위치로 바로 이동한다. 실패가 발생한 코드를 살펴보고 프로그램을 수정하고 다시 CalculatorTest 클래스를 실행해 오류가 정정됐는지 확인할 수 있다.

JUnit을 사용하는 이유는 테스트 케이스를 함수로 작성해서 관리할 수 있고 별도의 클래스에 테스트 케이스를 보관해 프로그램이 변경 되더라도 기존의 테스트 케이스를 그대로 활용할 수 있기 때문이다. 또한 다양한 검증 함수를 지원하기 때문에 복잡한 테스트 케이스를 적용할 수 있다.





트더 괜찮은 개발자가 되기위한 > 시스템 개발하기' 카테고리의 다른 글

SECRE WRITE

**;;** 

PREV 1 2 3 4 5 6 7 NEXT

+ Recent posts

9. 카트폴 DQN 모델 실행 8. 카트폴 DQN 강화학습 7. 텐서플로우 심층 신경망 6. 텐서플로우 단일 신경망

Rss Feed and Twitter, Facebook, Youtube, Google+

Powered by Tistory, Designed by wallel

