

JUnit入門

テストコードを書く

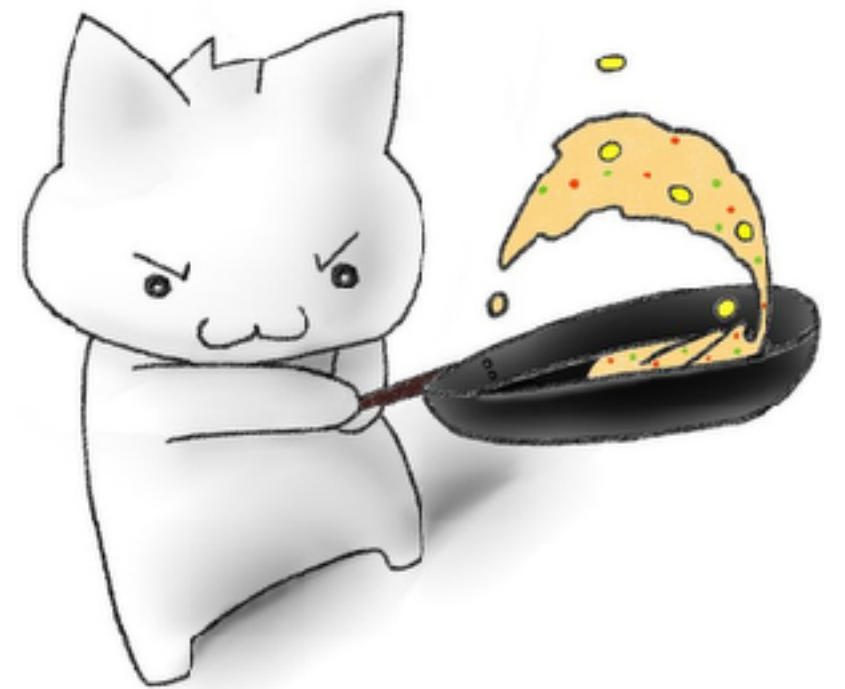
参考文献：現場で使えるソフトウェアテスト -Java編-

発表者 吉富 楓雅

目次

- Junitでテストコードを作る
- @Parametersを使ったテスト
- その他にテストコードに記述する内容

JUnitでテストコード を作る



使用するコード1

```
1 package sampleTest;↓
2 ↓
3 public class Calc {↓
4 ^ public int sum(int a, int b) {↓
5 ^ ^ return a + b;↓
6 ^ }↓
7 ↓
8 H↓
9
```

JUnitテストを作る

- eclipseで作成したJavaコードに対するテストコードを作成する場合
 - * 【新規】 → 【JUnitテスト・ケース】
- これにより作成したJUnitテストにコードを書く

テストコード

```
1 package sampleTest;↓
2 ↓
3 import org.junit.Test;↓
4 ↓
5 import junit.framework.TestCase;↓
6 ↓
7 public class CalcTest extends TestCase{↓
8     @Test↓
9     public void testSum() throws Exception {↓
10         int input1 = 1;↓
11         int input2 = 2;↓
12         Calc calc = new Calc();↓
13         int result = calc.sum(input1, input2);↓
14         assertEquals(3, result);↓
15     }↓
16 ↓
17 }↓
18
```

Assertクラスのメソッド

の利用方法

メソッド	説明
<code>assertArrayEquals(arrays expected, arrays actual)</code>	配列同士を比較、イコールである場合true
<code>assertEquals(Object expected, Object actual)</code>	オブジェクト同士を比較、イコールである場合true
<code>assertFalse(boolean condition)</code>	条件がfalseである事を検証、falseの場合true
<code>assertNotNull(Object obj)</code>	オブジェクトがNullで無いことを検証、Nullでない場合true
<code>assertNotSame(Object expected, Object actual)</code>	<code>expected != actual</code> の場合true
<code>assertNull(Object obj)</code>	オブジェクトがNullである事を検証、Nullの場合true
<code>assertSame(Object expected, Object actual)</code>	<code>expected == actual</code> の場合true
<code>assertTrue(boolean condition)</code>	条件がtrueである事を検証、trueの場合true
<code>assertThat([value], [matcher statement])</code>	汎用的な比較を許可するアサーションメソッド
<code>fail()</code>	常に失敗させる。『ここに来たらアカン(テスト失敗)』箇所に配置

参考： <http://d.hatena.ne.jp/absj31/20120811/1344759958>

org.junitパッケージの アノテーション

org.junitパッケージのアノテーション

@Test	テストメソッド用の <u>アノテーション</u>
@Before	JUnit3 の setup() に相当。各テストメソッドの実行前に実行される
@After	JUnit3 の tearDown() に相当。各テストメソッドの実行後に実行される
@BeforeClass	テストクラスの実行前に一度だけ実行される
@AfterClass	テストクラスの実行後に一度だけ実行される
@Ignore	該当のテストメソッドをスキップする。コメントを付与することもできる

参考: <http://etc9.hatenablog.com/entry/20090823/1251019936>

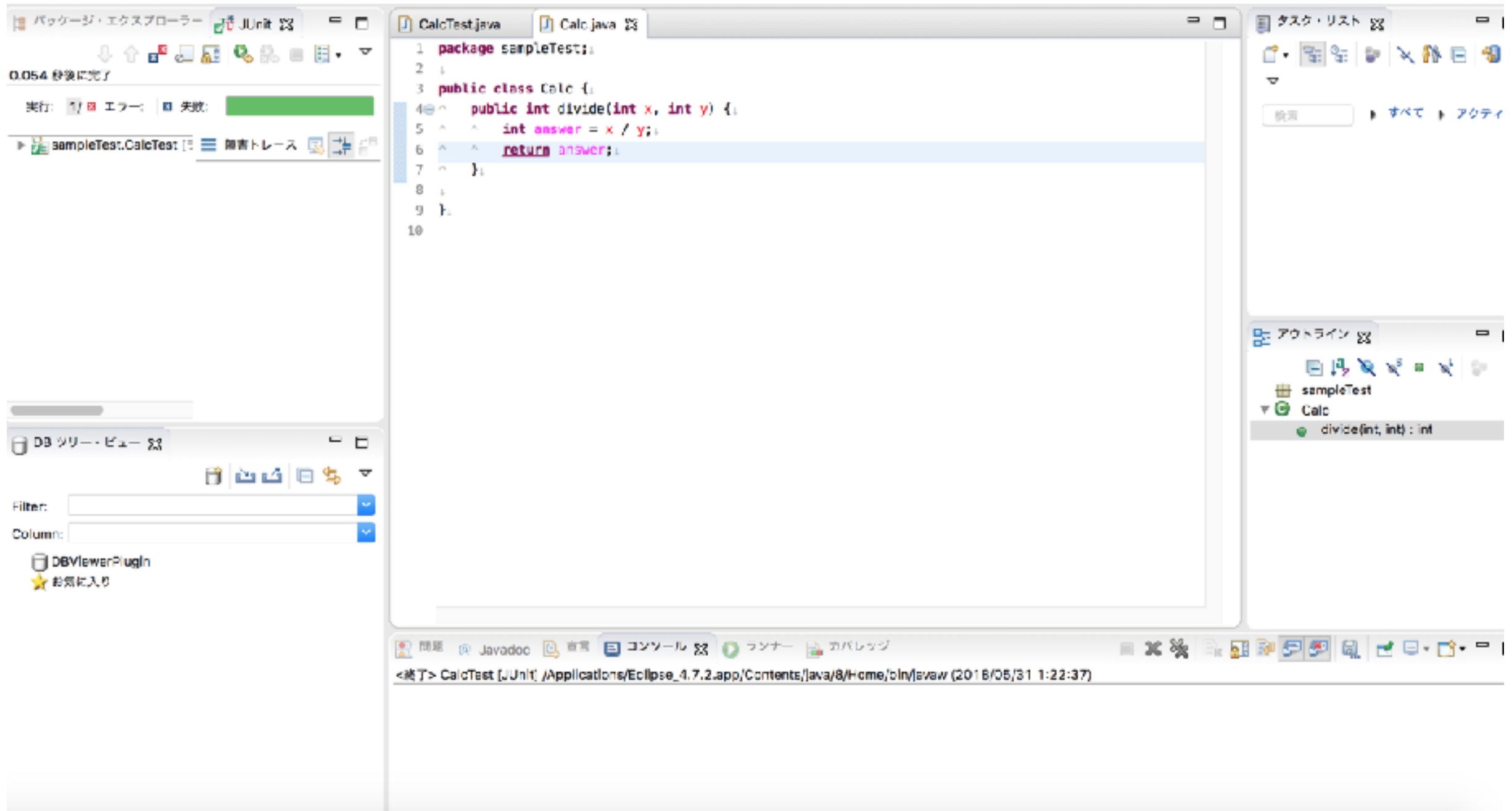
org.junit.runnerパッケージの アノテーション

org.junit.runnerとorg.junit.runnersパッケージの[アノテーション](#)

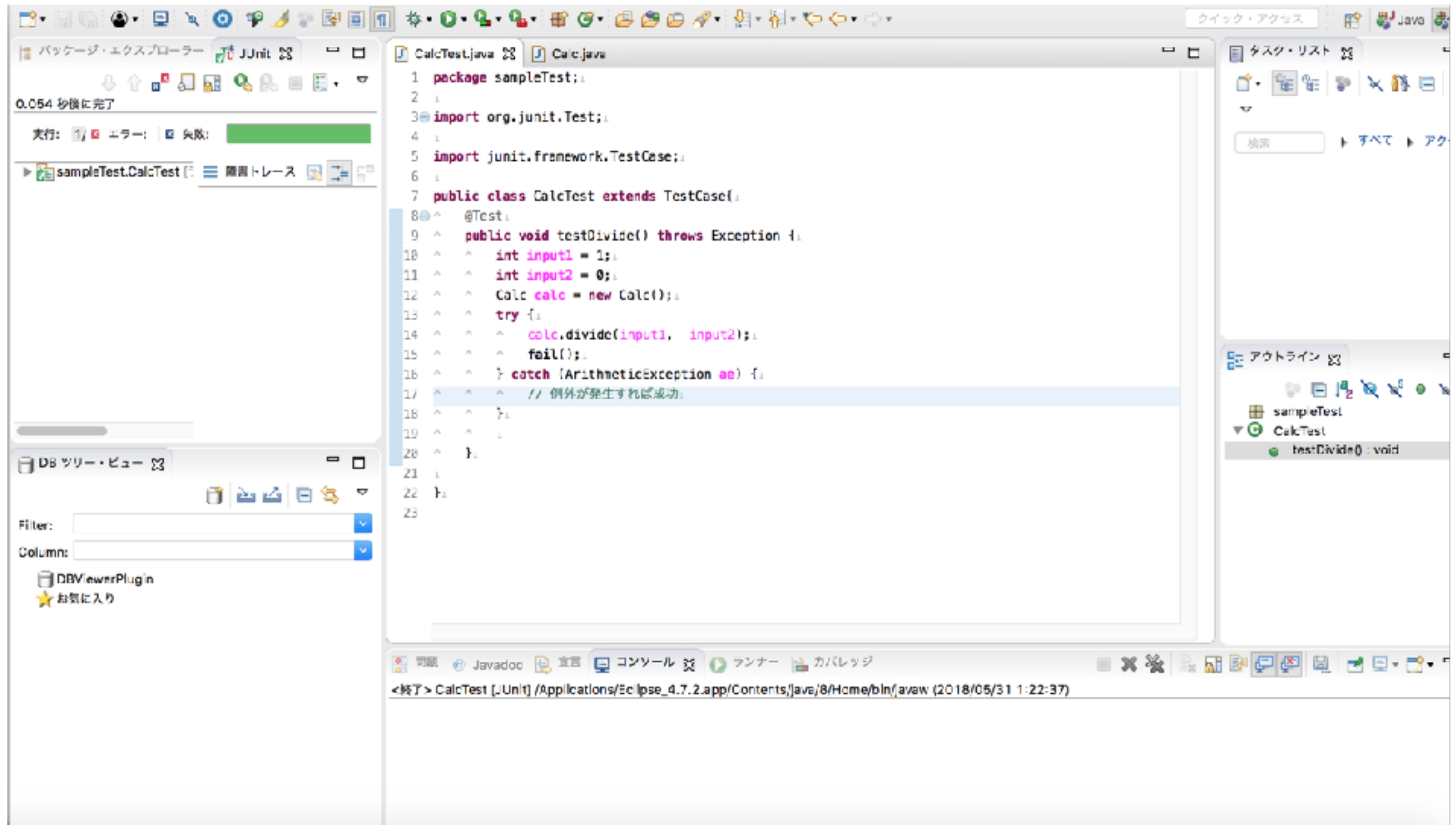
@RunWith	テストランナを指定する
@SuiteClasses	テストスイートクラスを指定する
@Parameters	パラメータを生成するメソッドを指定する

参考: <http://etc9.hatenablog.com/entry/20090823/1251019936>

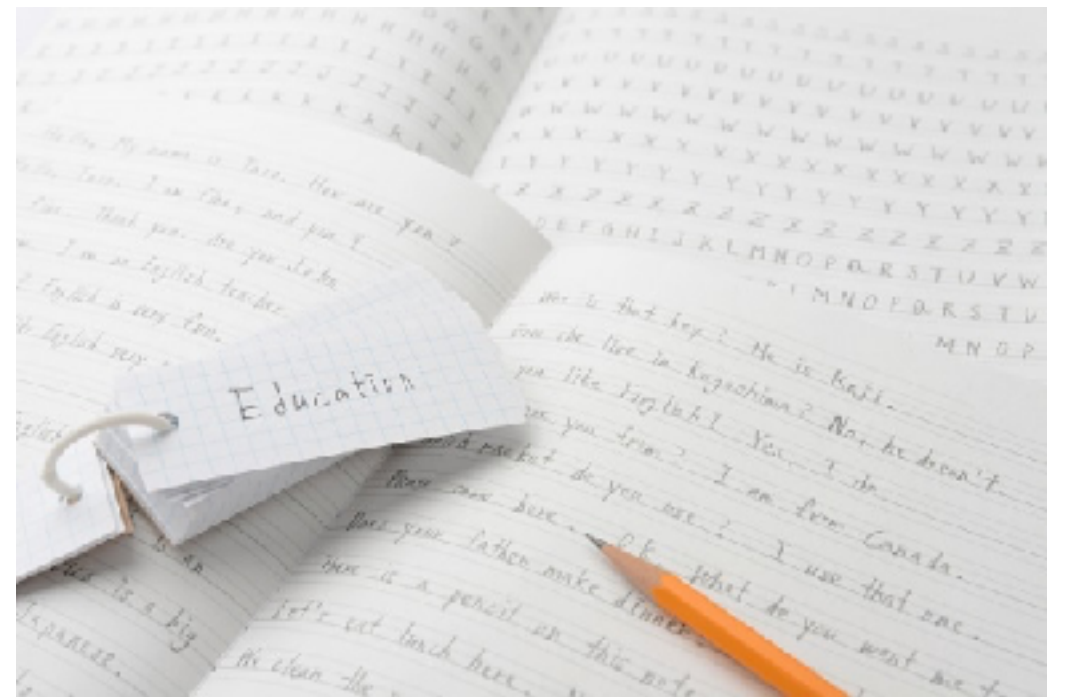
使用するコード2



テストコード



@Parametersを 使ったテスト



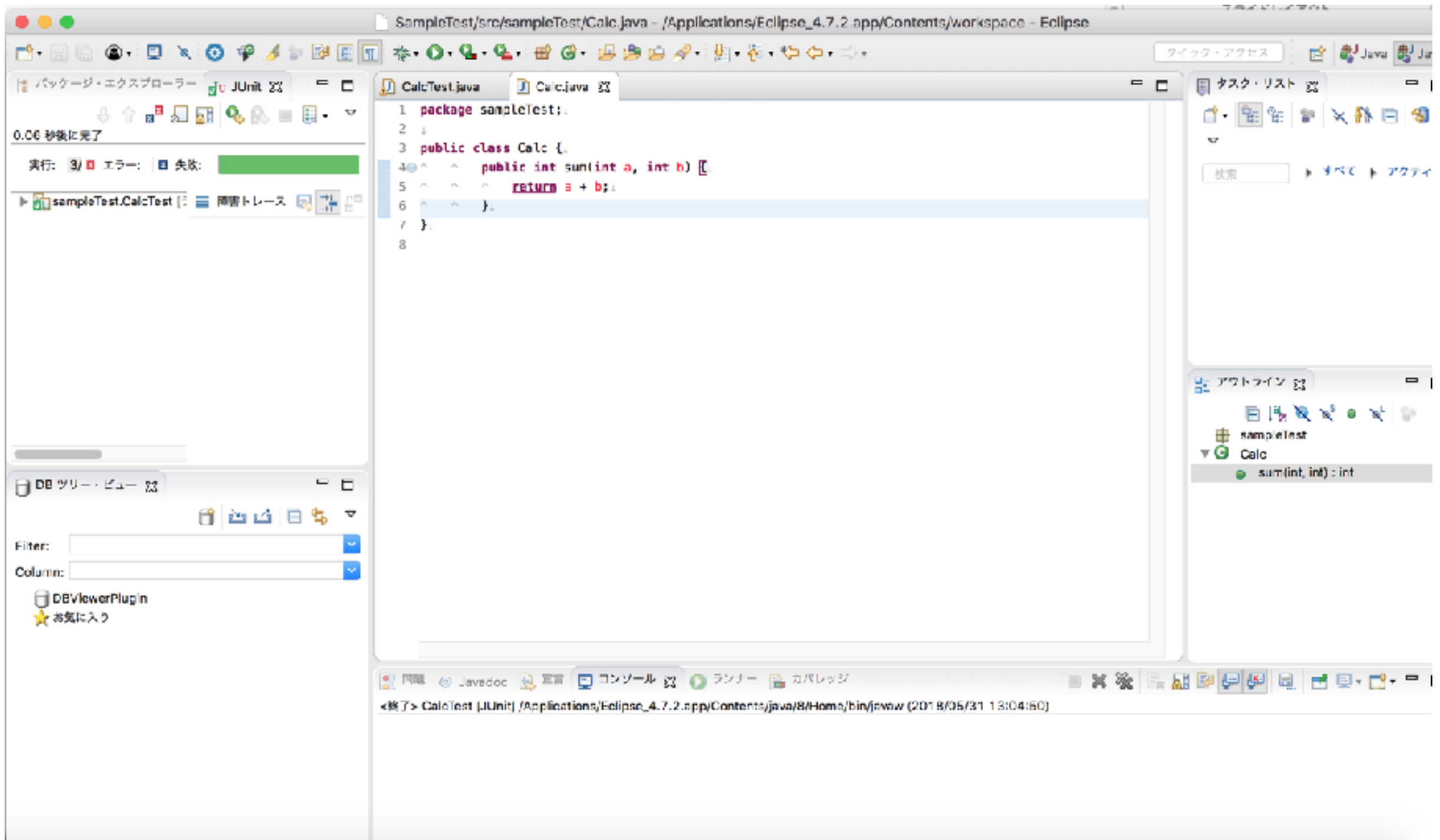
複数作成は面倒

- Junit4以降にはアノテーション@Parametersが導入されている
- これによりテスト対象メソッドに渡す引数と期待値だけが異なるテストを複数回実行できる

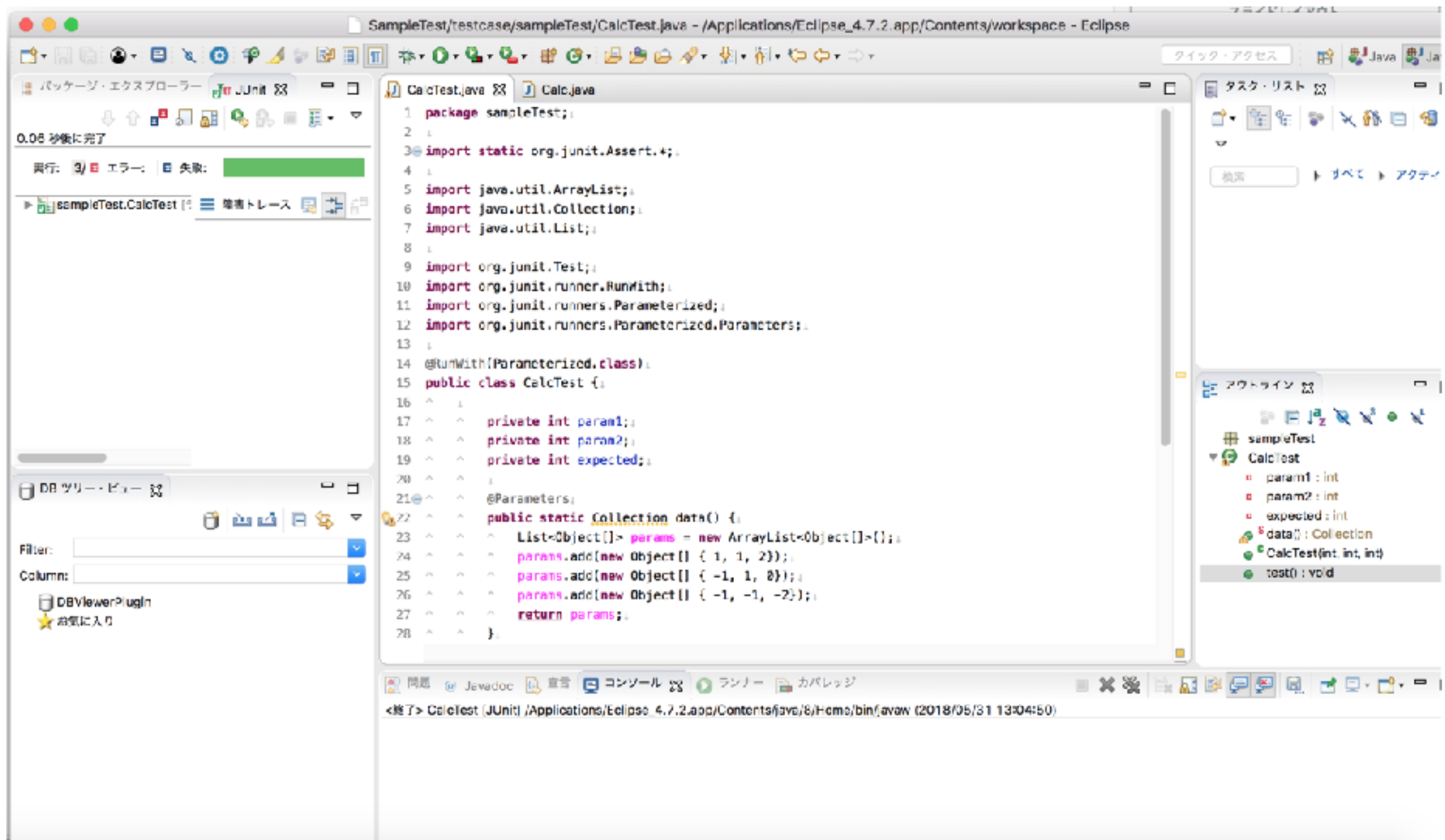
複数作成は面倒

- テストコードを作成する上でのポイント
 - * テストクラスに@RunWith(Parameterized.class)を指定する
 - * テスト別のパラメータを返却するメソッド(次の例ではdata()メソッドを作成する)
 1. アノテーション@Parametersを指定する
 2. public static で定義する
 3. 戻り値はCollectionで, Collectionの要素はObject配列とする
 - * パラメータを受け取るコンストラクタを作成する
 - * テストメソッドを作成する

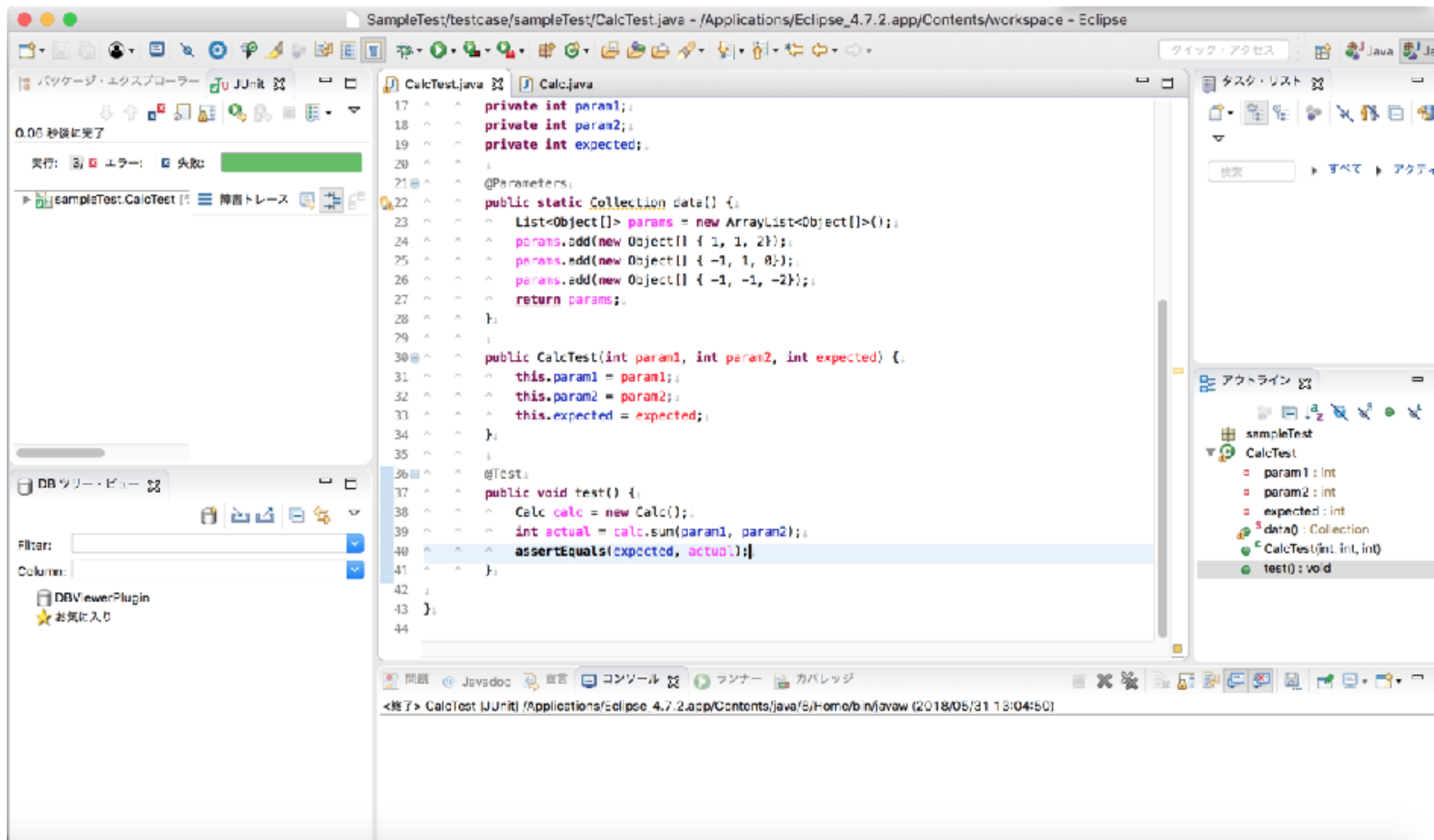
使用するコード3



テストコード(1)



テストコード(2)



その他にテストコード
に記述する内容



テストメソッドの 前処理と後処理

- テストメソッドの実行前後に毎回実行される処理がある場合、複数のテストメソッドに同じ処理を毎回記述するのではなく、共通化をしておくと便利なおことがある
 - * ネットワークへの接続・接続
 - * データベースの初期化
 - * 外部ファイルの初期化
- テストメソッドの前後に共通処理を実行する方法は2通りある

方法1; setUp()メソッド, tearDown()メソッドを使う

- これらのメソッドはjunit.framework.TestCaseで宣言されている

```
import junit.framework.TestCase;↓
↓
public class SampleTest extends TestCase {↓
^   ↓
⊖ ^   protected void setUp() throws Exception {↓
^   ^   //↓
^   }↓
^   ↓
⊖ ^   protected void tearDown() throws Exception {↓
^   ^   //↓
^   }↓
}↓
```

方法2: アノテーション

@Before, @Afterを使う

- Junit4以降で利用可能
- org.junit.Beforeとorg.junit.Afterをインポートする必要がある
- この方法を使うと複数のメソッドを前処理や後処理として利用できる(方法1では1つしか使えない)
- ただしクラスを継承してテストクラスを作った場合, つまり方法1と方法2を同時にプログラミングした場合は, 方法1が実行される
 - * アノテーションよりもメソッド名が優先されるという性質がある

方法2: アノテーション

@Before, @Afterを使う

```
import org.junit.After;↓
import org.junit.Before;↓
import org.junit.Test;↓
↓
public class SampleTest {↓
    ^
    ↓
    @Before↓
    ^
    public void initialize() throws Exception {↓
    ^
    ^    //↓
    ^
    }↓
    ^
    ↓
    @After↓
    ^
    public void disconnect() throws Exception {↓
    ^
    ^    //↓
    ^
    }↓
}
```