JUnit概要

2015/4/16版　今泉 俊幸

目次

1. 手動テストと自動テスト
2. JUnitの機能
3. 検証用メソッド
4. 基本的なJUnitテストケース
5. 実践的なJUnitテストケース
6. よく使う検証用メソッド
7. テストクラスの命名、配置など

手動テストと自動テスト

- 手動テスト
  - テスト仕様書に基づいて、人手で値を入力、結果を検証する
  - プログラム修正の度に実施するのはコストが高い
- 自動テスト
  - システムの動作を検証するプログラム(テストコード)を作成し、それを実行する
  - テストコード作成のコストが高いが、一度作成すれば、プログラムを修正しても再テストが容易

自動テストの必要性

- プログラムは作成、テスト実施後も、度々修正が入る。修正の度に手動テストは避けたい
  - 修正が入る場合の例
    - 仕様変更による機能修正
    - 不具合対応
    - 新規機能と重複する処理の共通化

- 仕様をよく知らない人でも、テストコードを見ればクラスやメソッドの使い方が分かるようになる

自動テストの種類

- シナリオテスト
  - システム全体の動きを検証する
    - 例：ログインして、カートに商品を入れて、注文ができる
  - プログラム中のクラスやメソッドの動きは考えず、
    実際のユーザが行うであろう操作に対し、
    最終的な結果が正しいかのみを検証する
  - Webアプリ用ツールとして、Seleniumが有名
- ユニットテスト
  - クラス、メソッド単位で仕様通りに動くかを検証する
    - 例：与えられた金額に対する税額の計算が正しいか
  - JUnitが有名

ユニットテスト用フレームワーク

- JUnit
  - オープンソースのJavaのユニットテスト用フレームワーク
- 提供する機能
  - 一連のテストメソッドの実行
  - テスト結果の検証用メソッドの提供
  - テストに必要なデータの初期化
  - テスト結果のレポート化
- 手動テストと同じようにテストケースを作成し、テストケースと対応するテストメソッドを実装し動作を検証する

検証用メソッド 1/2

- メソッドの結果を検証するassert～系のメソッドが用意されている
- assertEquals(Object expected,Object actual)
  - expected：期待する結果
  - actual：実際のテスト結果
  - expectedとactualが異なる場合は、エラーを記録して次のテストへ進む
- 例
  - assertEquals(3, add(1,2));

検証用メソッド 2/2

- assertThat(T actual,
  Matcher<? super T> matcher)
  - actual: 実際のテスト結果
  - matcher: 検証方法
  - actualがmatcherによる検証に失敗した場合は、エラーを記録して次のテストへ進む
  - assert~よりもテストコードが読みやすくなる(後述)
- 例
  - assertThat(add(1,2), is(equalTo(3)));

assertThatを使う利点

- 英文を読むように、何を検証するか理解できる
  - assertThat(actual, is(equalTo(expected)));

  ⇒Assert that actual is equal to expected.

  Assert that add 1,2 is equal to 3.
  - assert~より長くなるが、検証コードをコメントのように読むことが出来る
- equalToであれば、次のように書くことも可能
  - assertThat(actual, equalTo(expected));
  - assertThat(actual, is(expected));
- equalTo以外に、isSameInstance,between等があり、自分で定義することも可能

最も基本的なJUnitテストケース 1/2

- テスト対象のメソッド
  - 入力された値を加算して返す

```
class Calculator{
  public int add(int a,int b){
    return a + b;
  }
}
```

最も基本的なJUnitテストケース 2/2

- テストコード

```
class CalculatorTest{
  @Test
  public int addTest(){
    Calculator calculator = new Calculator();
    assertThat(calculator.add(1,2), equalTo(3));
  }
}
```

- @Testを付けたメソッドがテストメソッドになり、JUnitから呼ばれる

[実践的な例]除算を行うメソッド

```
public class Calculator {
    /**
     * 除算結果を小数点以下3桁を四捨五入して返す
     * @param dividendVal 割られる数
     * @param divisorVal 割る数
     * @throws IllegalArgumentException 割られる数が0の場合
     */
    public BigDecimal divide(int dividendVal ,int divisorVal)
      throws IllegalArgumentException{
        BigDecimal dividend = BigDecimal.valueOf(dividendVal);
        BigDecimal divisor = BigDecimal.valueOf(divisorVal);
        return dividend.divide(divisor).setScale(2);
    }
}
```

[実践的な例]テストメソッド(割り切れる場合)

```
@Test
public void 割り切れる数の除算(){
        // 6/2 = 3をテスト
        Calculator calculator = new Calculator();
        BigDecimal expected = new BigDecimal("3.00");
        BigDecimal actual = calculator.divide(6, 2);
        assertThat(actual, equalTo(expected));
}
```

[実践的な例]テストメソッド(割り切れない場合)

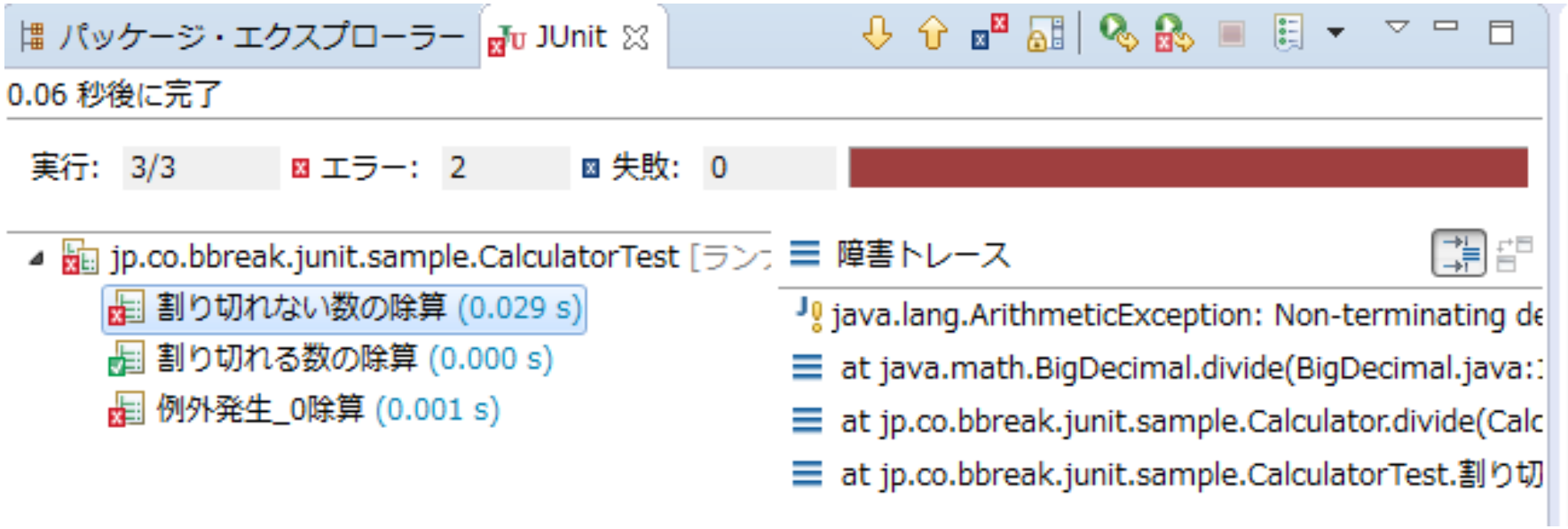
```
@Test
public void 割り切れない数の除算(){
        // 1/3 = 0.33をテスト
        Calculator calculator = new Calculator();
        BigDecimal expected = new BigDecimal("0.33");
        BigDecimal actual = calculator.divide(1, 3);
        assertThat(actual, equalTo(expected));
}
```

[実践的な例]テストメソッド(0除算)

```
@Test(expected = IllegalArgumentException.class)
public void 例外発生_0除算(){
        // 1/0 = 例外発生をテスト
         Calculator calculator = new Calculator();
         calculator.divide(1, 0);
}
```

- @Test(expected=例外クラス)を指定すると
  - テストメソッド内で指定した例外が発生すればテスト成功
  - 違う例外が発生したり、例外が発生しないとテスト失敗

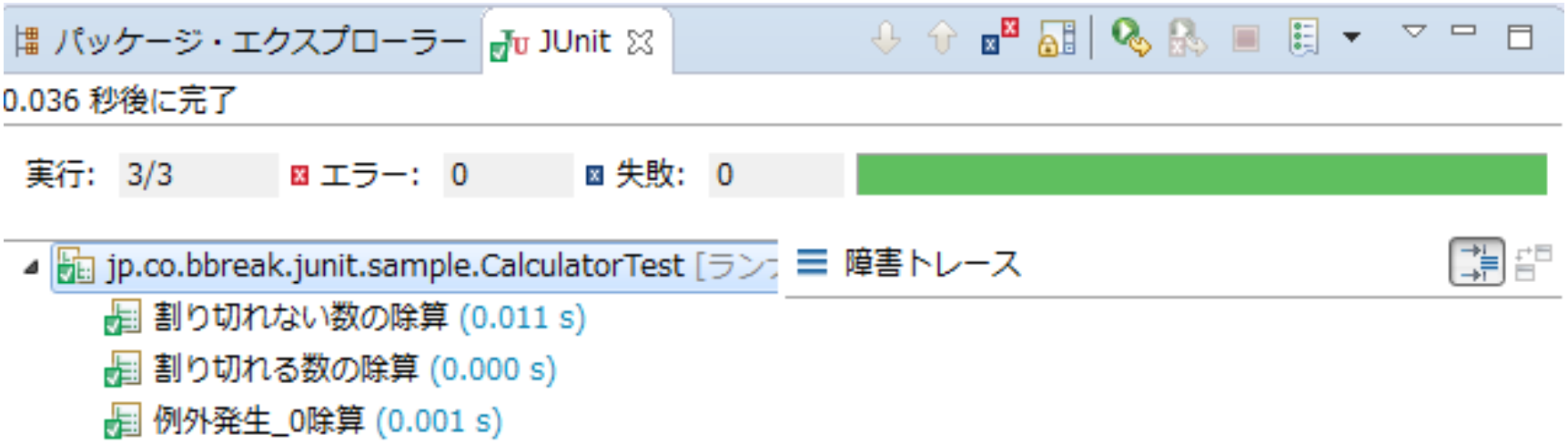
[実践的な例]テスト結果



[実践的な例]テスト結果をふまえ修正

```
public BigDecimal divide(int dividendVal ,int divisorVal)
  throws IllegalArgumentException{
    if(divisorVal == 0){
       throw new IllegalArgumentException("割られる数が0です");
    }
    BigDecimal dividend = BigDecimal.valueOf(dividendVal);
    BigDecimal divisor = BigDecimal.valueOf(divisorVal);
    return dividend.divide(divisor, 2 , RoundingMode.HALF_UP);
}
```

[実践的な例]再テスト結果



[実践的な例]補足

- BigDecimalの比較はequalsでは行わない。
  - scaleも比較してしまうため、以下はfalseになる
    - new BigDecimal(“3”).equals(new BigDecimal(“3.0”))
  - 値が一致するかどうかはcomapreToを使う
    - compareToの返り値が0ならば同じ数値である

- 割り切れない場合のテストケースは四捨五入されるのかどうかもテストするべき
  - 小数点以下3桁目が5以上の場合と、4以下の場合の2つのテストケースを作る必要がある
  - 今回であれば2/3=0.67となることもテスト

よく使うassertメソッド

- 真偽値の判定
  - assertTrue,assertFalse
- nullかどうかの判定
  - assertNull,assertNotNull
- 例外が起きるかどうかの判定
  - @Test(expected=例外クラス)

よく使うMatchers

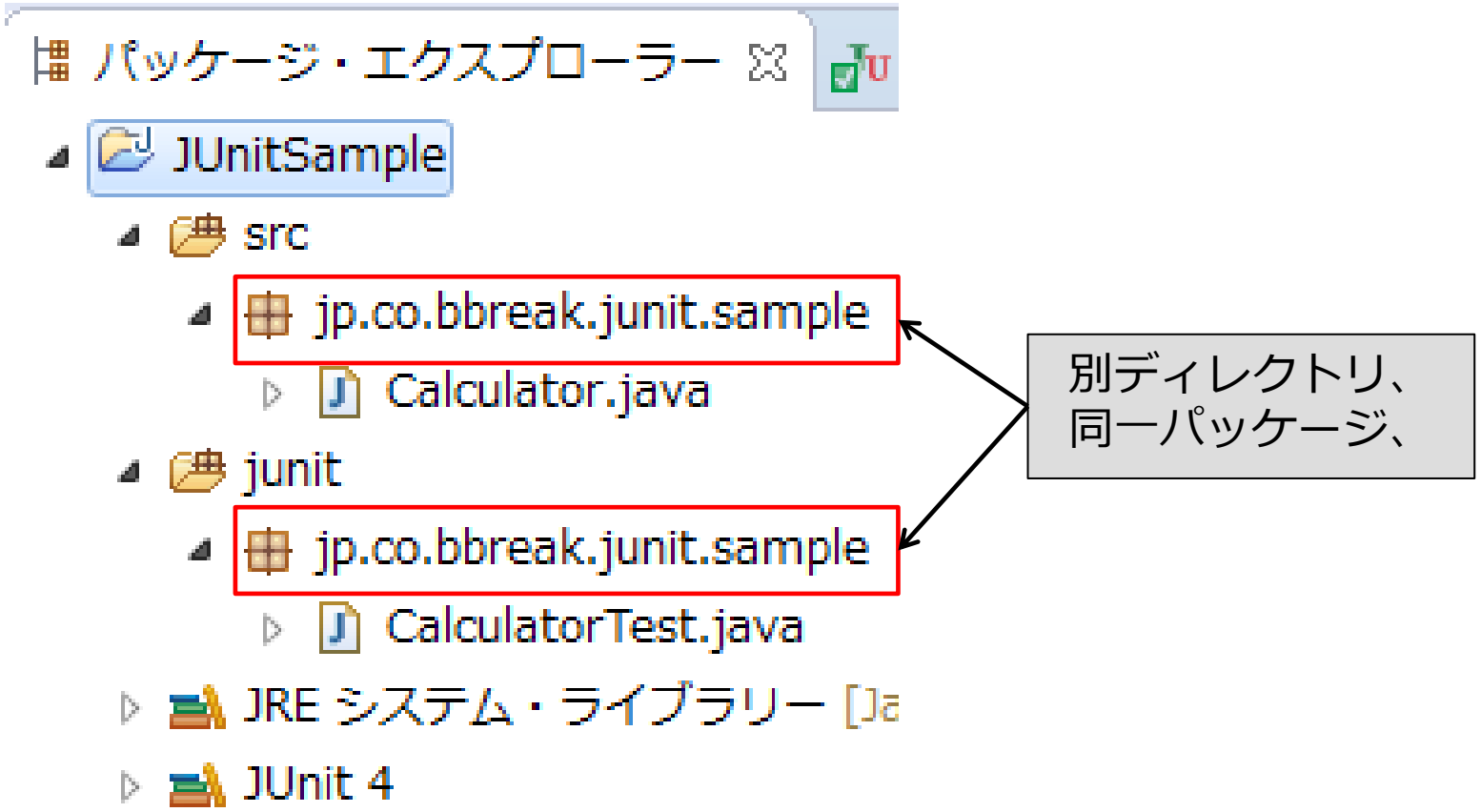
- 真偽値の判定
  - assertThat(actual, is(true));
  - assertThat(actual, is(not(true)); // falseなら成功
- nullかどうかの判定
  - assertThat(actual, is(nullValue(expected));
  - assertThat(actual, notNullValue(expected));
- 同じインスタンス
  - assertThat(actual, sameInstance(expected));
- BigDecimalの値
  - assertThat(actual, closeTo(expected, error))
  - error: 許容される誤差

※その他はhttp://hamcrest.org/JavaHamcrest/javadoc/1.3/を参照

テストクラスの命名、配置

- テストクラスの命名
  - テストクラスの名称は「テスト対象のクラス名+Test」とすることが一般的
    - Calcurator→CalcuratorTest
  - テストメソッドの名前は日本語でもよい
- テストクラスの配置
  - テストクラスはテスト用のソースフォルダを作成してそこに入れ、パッケージはテスト対象のクラスと同一パッケージとする
    - パッケージプライベートのメソッドが呼べるように

パッケージ構成例



テスト対象のメソッド

- privateなメソッドはテストしない。
  publicなメソッドをテストする。
  - privateなメソッドは必ずpublicなメソッドから呼ばれる。publicなメソッドのテストが正しければよい
  - privateなメソッドはそのクラスに閉じており、修正が行われても、外部に影響しないし、自由に修正されるべき