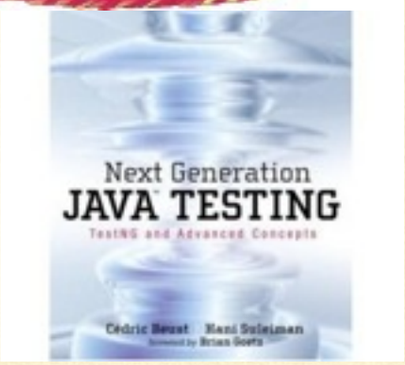
Javaツール勉強会@福岡 2016/02

～テストツールとか編～

JUnit

Apache JMeter™



この勉強会について

- Javaに関連するツールについて、いろいろ勉強していきたくて始めた勉強会です！
- 今回が２回目です！
- 今回は私の独断でテストツールをサブタイトルにしました

『Javaに関連するツール』って？

- Javaが絡んでるツールなら割となんでもありでいこうかと
- ライブラリからフレームワーク、JVM言語もありとか
- ぶっちゃけ、JavadocのようなJava SE同梱も含めてよいかと
- 単にJavaの文法の話は別でやろうねくらいの意味です

今後について

- 発表形式で今後も続けていきたいと思います
- とりあえずゆるゆると。

JUnitについて

Javaツール勉強会@福岡 2016/02 吉村 武志

- 自己紹介
- 吉村 武志
- 福岡周辺の勉強会に
  ちょくちょく参加して
  togetterまとめたりしてる人

- Javaの勉強会やってますが、
  仕事はClassic ASP & VB.Net
- 趣味は音ゲーとかアナログゲーム（ボードゲーム・TRPG）

先にお断り

JUnit

まともに使ってない人の発表です。

ツッコミ求む！

JUnitについて(教科書知識)

- Javaで開発されたプログラムにおいてユニットテスト（単体テスト）の自動化を行うためのフレームワーク

- http://junit.org/
- 最新版はVersion: 4.12（2015-12-03）

  （※JUnit 5は現在5.0.0-SNAPSHOT：https://junit-team.github.io/junit5/）

- ３系(Java1.4以前）と４系（Java5.0以降対応）がある

さて、まずは簡単に

JUnitを使ってみます！

JUnit

まずはテスト対象のClassを準備

```
1   package fizzbuzzjunit;
2
3   /**
4    * FizzBuzzプログラムのはずだが・・・？
5    */
6   public class FizzBuzz {
7       public static String fizzBuzz(int input) {
8           if (input % 3 == 0) {
9               return "Fizz";
10          } else if (input % 5 == 0) {
11              return "Buzz";
12          } else if (input % 3 == 0 && input % 5 == 0) {
13              return "FizzBuzz";
14          } else {
15              return String.valueOf(input);
16          }
17      }
18  }
19
```

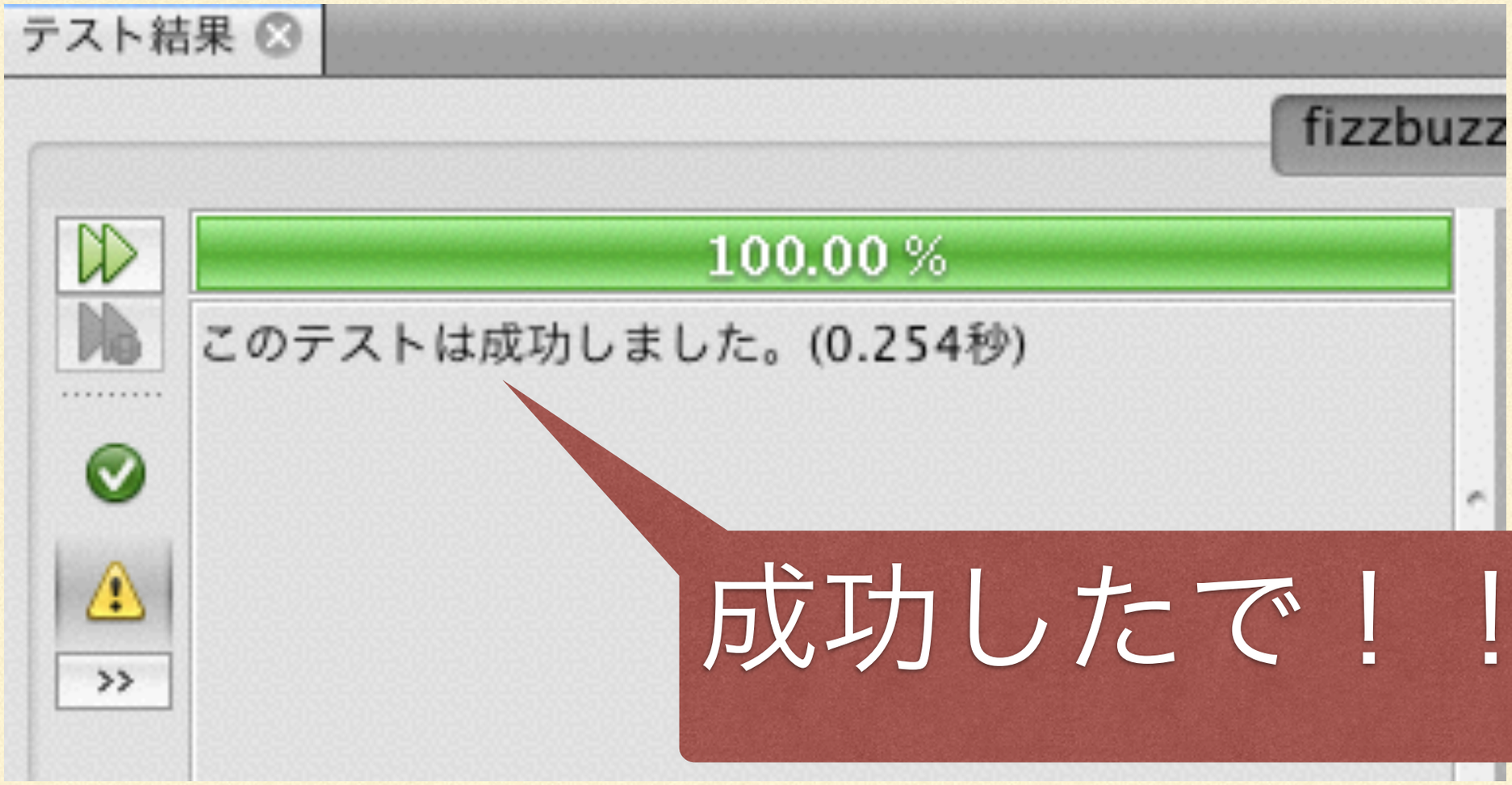
JUnit

そしてテストするクラスを作成

```
1   package fizzbuzzjunit;
2
3   import org.junit.Test;
4   import static org.junit.Assert.*;
5   import static org.hamcrest.CoreMatchers.*;
6
7   public class FizzBuzzTest {
8
9       @Test
10      public void 入力が３で割り切れる時() {
11          int input = 6;
12
13          String actualResult = FizzBuzz.fizzBuzz(input);
14
15          assertThat(actualResult, equalTo("Fizz"));
16      }
17
```

JUnit

で、テスト実行！！



で、何が嬉しいの？

これは多分、

簡単すぎる例では伝わらない。

JUnit

ちょっと想像してみましょう…

- 数百万行はある、巨大なプログラム
- ユニットテストプログラムが無い
- 仕様変更が発生
- 変更が正しいかどうか、**手動テストで全部確認する**
- 動かしてみたらNullPointerException
  そこをを解決したらArrayIndexOutOfBoundsException
  さらにその次は・・・・・・

テストをしないまま、

機能を作り込む

↓

修正の大変なバグを作り込む

JUnit

でも、毎回手動テストは非効率！！

面倒！！

面倒なテストはやりたくない。

後でまとめて・・・

そしてやっかいなバグ・・・

つらい想像はちょっと、

脇によけておきましょう・・・

JUnit

ちょっと想像してみましょう…

- 数百万行はある、巨大なプログラム
- ユニットテストプログラムがあり、５秒以内に終わる
- 仕様変更が発生
- どうせ５秒で終わるからと、
  とりあえずユニットテストプログラム起動→もちろんエラー無し
- プログラムを修正して、テストプログラムも作って、
  ユニットテストプログラム起動→エラーはあるかな？どうかな？
- うっかりミスしてバグを作ったけれど、５秒で発見→修正完了

JUnit

小さな機能を作るたびに

しょっちゅうユニットテスト

↓

バグを作り込んだとしても修正が容易

テストプログラムの起動で

ユニットテストが実施できるならば、

テスト実施のハードルが下がる！

それが多分、

ユニットテストプログラムを書く理由

そして、

テストツールを使う理由

参考までに、
JUnitの価値と言われているもの…

- 一度作成すればすばやくテスト可能である。
- その後はテストコードを標本とすることでバグ訂正が容易となる。
- テストコードを見れば仕様が一目瞭然となる。
- 誰でも同じテストを行えるようになる。
- 独自のテストコードによるテスト作成の手間を省ける。

さて改めまして、

もうちょっとJUnitについて調べた事

テストケースクラスの作成

- テストケース：テストケースとは、ソフトウェアテストを実施する際に用意する、実行条件や入力データ、期待される出力や結果などの組み合わせ。

JUnit

テストケースクラスの作成

- テストケースクラスは「○○○○Test」といったクラス名で作成する事が多い。

- テストケースクラスには、１つ以上のテストメソッドを作成する。テストメソッドには「@Test」アノテーションを付与する（JUnit4系）

  （※JUnit3以前の場合は「testXxxxx」といった命名規則になってたハズ）

@Testアノテーション

- http://junit.sourceforge.net/javadoc/org/junit/Test.html
- JUnit4系において、テストケースとして実行するテストメソッドに付与するアノテーション
  - expected属性：期待する例外を記述する。
    指定した例外が発生すると成功として扱われる。発生しないと失敗として扱われる。
  - timeout属性：オプションでミリ秒の数値を指定する。
    テスト実行が指定ミリ秒を超えると、テストが失敗する。
- @Ignreアノテーションというものもあり、@Testの一時的な無効化が可能。

JUnit

テストメソッドの作成

- テストメソッドはArrange-Act-Assertを行うように作成する
  - Arrange：セットアップ。操作開始前の準備を行う。
  - Act：操作。テストしたい処理を呼び出す。
  - Assert：アサーション。期待通りの値が得られたか確認する。
- テストメソッドには日本語を使うのもいいかも？

JUnit

Arrangeをまとめたい

- @Beforeアノテーションを使うと、全てのテストメソッドで事前に行う処理を記述できる。

- @Beforeに対応して、@Afterアノテーションも存在する。
  @Beforeで確保したリソースをテストメソッドの終了後に解放したい場合等に使用する。

- なお、テスト開始前に１回だけ実行したい場合、およびテスト終了後に１回だけ実行したい処理がある場合は、@BeforeClassアノテーションおよび、@AfterClassアノテーションを使用するのが良い。

アサーションいろいろ

- org.junit.Assertにアサートメソッドがいくつかある

  http://junit.sourceforge.net/javadoc/org/junit/Assert.html

- <T> void assertThat(T actual, Matcher<T> matcher)
  <T> void assertThat(String reason, T actual, Matcher<T> matcher)
  …実行結果がmatcherを満たすかどうかを確認する
  …これを使う事をオススメ。
  …matcherはorg.hamcrest.CoreMatchersを使うと便利らしい。

- void fail() … テストが失敗する。例外を想定するテスト用（多分）

JUnit

アサーションいろいろ（古いもの）

多分もう使わなくてよい奴ら

- void assertTrue(boolean condition) … conditionがtrueか否かを確認
- void assertFalse(boolean condition) … conditionがfalseか否かを確認
- void assertEquals(expected, actual) … 二つが等しいか否かを確認
- void assertArrayEquals(expected, actual) … 二つの配列が等しいか否かを確認
- void assertNotNull(object) … objectが非nullか否かを確認
- void assertNull(object) … objectがnullか否かを確認
- void assertSame(expected, actual) … 二つが同じものを参照しているか否かを確認
- void assertNotSame(unexpected, actual) … 二つが別のものを参照しているか否かを確認

JUnit

assertThatで使うMatcherいくつか

HamcrestのCoreMatcherより

- equalTo() … Object.equalsで等しいかどうかのMatcher
- is() … 他のMacherのデコレーション用
- not() … Macherを反転させる
  例）assertThat(cheese, is(not(equalTo(smelly))))
- nullValue() … nullであるか否かのMatcher
- notNullValue() … 非nullであるか否かのMatcher

JUnit

assertThatで使うMatcherいくつか

HamcrestのCoreMatcherより

- sameInstance() … 同一インスタンスであるか否かのMatcher
- instanceOf() … 指定クラスであるか否かのMatcher
- startsWith() … 指定文字列ではじまる文字列か否かのMatcher
- entsWith() … 指定文字列で終わる文字列か否かのMatcher
- containsString() … 指定文字列を含むか否かのMatcher

……CoreMatcherだけでもまだまだ…

JUnit

assertThatで使うMatcherいくつか

その他のHamcrestのMatcherより

- org.hamcrest.number.IsCloseTo.closeTo()
  … 浮動小数点数について、許容誤差を含めて比較するMatcher

- org.hamcrest.text.IsEqualIgnoringCase.equalToIgnoringCase()
  … 大文字小文字を無視して文字列を比較するMatcher

……まだまだいっぱい

……自前で作る事も可能っぽいけれど、

既にいろいろあるという事を頭に入れておくと幸せになれるかも。

JUnit

例外の発生はどうテストする？

- @Testアノテーションのexpected属性を使う
  期待した例外が発生した場合を成功とする。

- try/catchで書く（古い方法ぽい）
  最後まで行ったらfail()とすれば実装可能。

- org.junit.rules.ExpectedExceptionを使う

ここまでのざっくりまとめ

- ユニットテストプログラムを書くと幸せになれる（多分）
- JUnitのテストメソッドが書けてさえいれば、使うのは簡単
- @Testアノテーション付ければテストメソッドになる
- 実行結果のアサートはassertThatでHamcrestのMacherってのを使うと便利っぽいぽい

でも・・・

私が知りたい・・・

- テストしやすい実装になっていないときは、どうやってテストを作るの？
  - データベース操作やファイルI/Oをどうテストする？
  - グローバル設定に依存しまくりのレガシー実装
  - そもそもMVCみたいな３層構造になってない場合
- 「テストプログラムの正しさはどう検証するの？」とか言われる

辛い環境の話はさておき、

可能ならばみなさんユニットテスト

頑張りましょう