JAVA E JUNIT

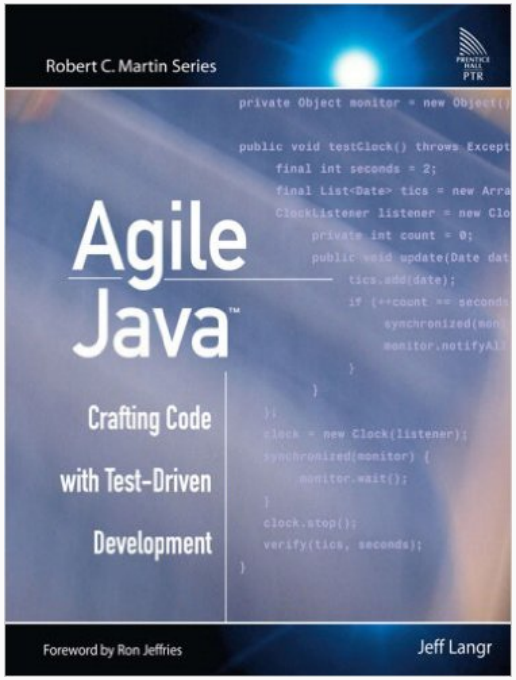
ACH 2003 — COMPUTAÇÃO ORIENTADA A OBJETOS

Daniel Cordeiro

24 de fevereiro de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

BIBLIOGRAFIA



Agile Java™: Crafting Code with Test-Driven Development
Jeff Langr

Disponível na rede da USP em:
http://proquest.safaribooksonline.com/book/programming/java/0131482394

Introduction to Test Driven Development (TDD) por Scott W. Ambler

http://www.agiledata.org/essays/tdd.html

TESTES

- Quão importante é testar seu código?
- Você já escreveu testes para o seu código?
- Como você escreve seus testes?

TESTES

- Quão importante é testar seu código?
- Você já escreveu testes para o seu código?
- Como você escreve seus testes?
- Você já ficou feliz por ter escrito testes para seu código? :)

“Se vale a pena ser construído, então vale a pena ser testado.

Se não vale a pena ser testado, então porque você está desperdiçando seu tempo trabalhando nisso?” — Scott W. Ambler

Como você escreve um teste?

1. Anote um método com

   ```
   @org.junit.Test
   ```

2. Para verificar um valor

   ```
   import org.junit.Assert.*
   ```

   e chame

   `assertTrue()`

   passando como parâmetro um valor booleano que deve ser ser verdadeiro se o teste passou; ou

   `assertEquals()`

   com os dois valores que devem ser iguais

EXEMPLO:

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 1;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}

$ javac Calculator.java
```

EXEMPLO: COMPILAÇÃO DO TESTE

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

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

```
$ javac -cp .:junit-4.XX.jar CalculatorTest.java
```

RESULTADO DO TESTE

```
$ java -cp .:junit-4.XX.jar:hamcrest-core-1.3.jar
org.junit.runner.JUnitCore CalculatorTest

JUnit version 4.12
.E
Time: 0,008
There was 1 failure:
1) evaluatesExpression(CalculatorTest)
java.lang.AssertionError: expected:<6> but was:<7>
  at org.junit.Assert.fail(Assert.java:88)
  at org.junit.Assert.failNotEquals(Assert.java:834)
  at org.junit.Assert.assertEquals(Assert.java:645)
  at org.junit.Assert.assertEquals(Assert.java:631)
  at CalculatorTest.evaluatesExpression(
                                CalculatorTest.java:9)
```

EXEMPLO:

```java
public class Calculator {
  public int evaluate(String expression) {
    int sum = 1;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

CONSERTANDO O ERRO

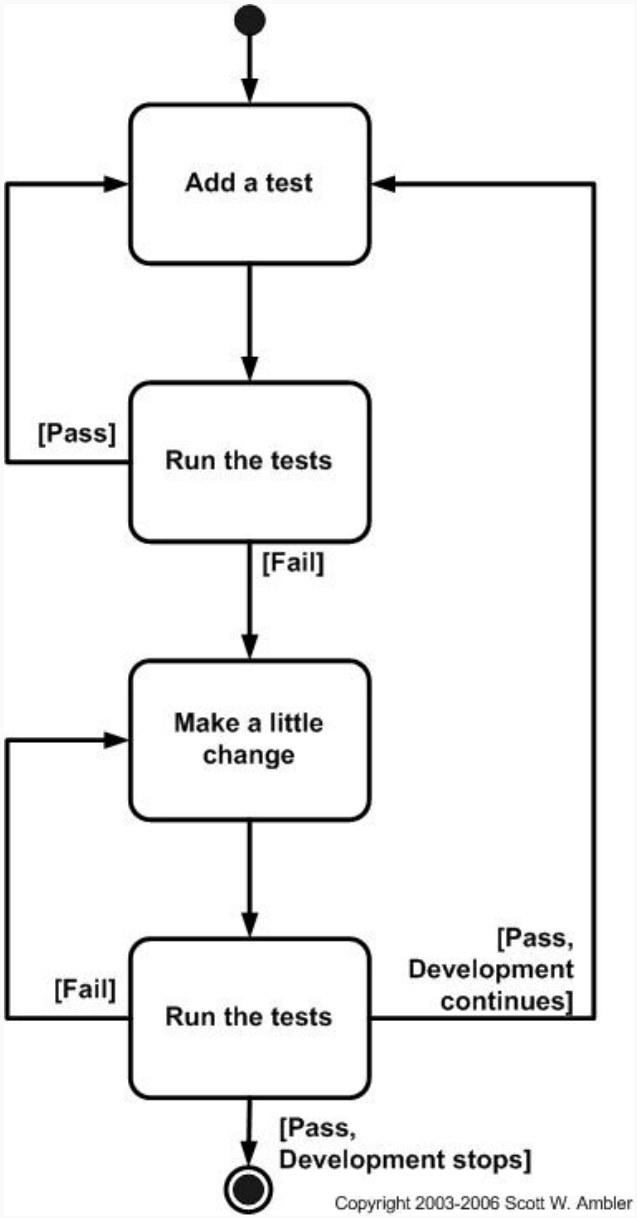
```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

Resultado:

```
JUnit version 4.12
.
Time: 0,005

OK (1 test)
```

Test-driven development (TDD)



- Desenvolvimento guiado por testes é um processo evolucionário de desenvolvimento
- A ideia é escrever primeiro o teste e só então escrever o mínimo de código necessário para satisfazer o teste
- Objetivo principal de TDD: escrever código limpo e que funciona, pensando nos requisitos e no design antes de escrever código funcional

TDD: REGRAS FUNDAMENTAIS

2 regras simples:

1. Você só deve escrever código de negócio novo quando tiver um teste automatizado falhando;
2. Você deve eliminar todas as duplicações que encontrar.

Bons testes de unidade:

- Rodam rapidamente
- Rodam isoladamente (você deve poder reordenar os testes)
- Usa dados que os tornem fáceis de ler e de entender
- Usam dados reais (ex: dados de produção) quando precisar deles
- Representam um passo em direção ao seu objetivo geral

UMA SESSÃO DE TDD

PROBLEMA

Implemente uma Calculadora[1] que atenda os seguintes requisitos:

1. Crie uma calculadora simples com um método `int Add(String numbers)`:
   - 1.1 O método pode receber 0, 1 ou 2 números positivos e devolverá sua soma (para uma String vazia, devolverá 0). Ex: `"", "1" ou "1,2"`.
   - 1.2 Inicie com o caso de teste mais simples possível com uma string vazia e só depois passe para 1 ou 2 números
   - 1.3 Resolva os problemas da forma mais simples possível para forçá-lo a escrever testes que você não tinha pensado antes
   - 1.4 Lembre-se de melhorar o código após fazer os testes passarem
2. Modifique o método `Add` para aceitar qualquer quantidade de inteiros positivos
3. Modifique o método `Add` para que ele permita quebras de linhas entre os números além de vírgulas (`"1\n2,3"` está OK, `"1,\n"` não)
4. Permita diferentes delimitadores. Para mudar o delimitador, o início da string deve contar uma linha separada que se parece com isso: `"//[delimitador]\n[números ...]"`

[1] Baseado em um Kata de Roy Osherove — `http://osherove.com/tdd-kata-1/`