Testes unitários

Prof. Marcel Hugo

Departamento de Sistemas e Computação
Universidade Regional de Blumenau - FURB

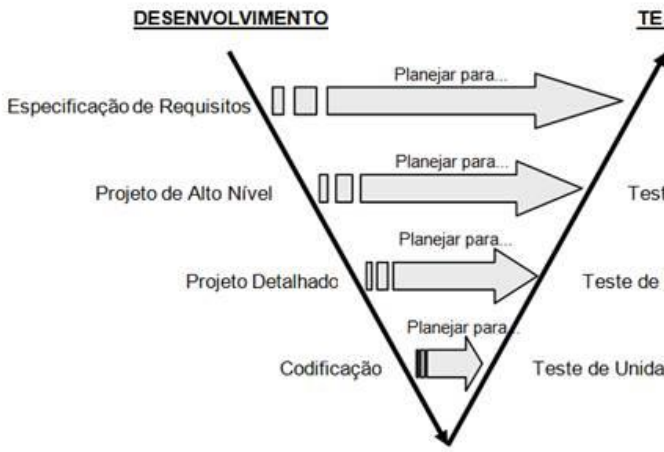
1

Teste de software

- Teste é a execução do software de maneira controlada para avaliar se ele se comporta ou não conforme o especificado.
- “Teste é o processo de executar um programa com o objetivo de encontrar erros” (Myers, 1979).
- Testamos software para melhorar a qualidade do software

2

Níveis do teste de software



Alvo do teste

- Sistema
- Agrupamento de módulos
- Módulo

3

Testes unitários

- Verificar uma funcionalidade **pequena e específica** do software – nível de "**rotinas**";
- Realizado por desenvolvedores, para verificar se a rotina faz o que deve fazer;
- Proposta:
  - Se partes individuais (rotinas) estão funcionando corretamente, pode-se continuar com o desenvolvimento do sistema.
- Questão que todo teste unitário deve se propor a responder:
  - O código faz o que deveria?
  - O código sempre faz o que deveria?

5

5

Testes unitários

- Testes Unitários deveriam:
  - Testar todas as operações dos objetos
  - Definir e verificar todos os atributos dos objetos
  - Colocar o objeto em todos os estados possíveis

6

6

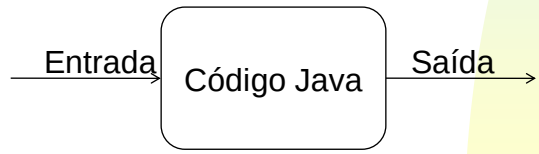
Caso de teste

- Caso de teste – cada teste que contém um conjunto de entrada diferente é chamado de “caso de teste”.
- Tupla – entradas desejadas e saídas esperadas
- Quando possível, deve-se abranger o maior número de situações possíveis
- “(...) geralmente é impossível garantir a ausência de erros em um programa. A complexidade combinatória ultrapassa amplamente a possibilidade de realizar um teste completo. Por este motivo é preciso organizar o teste escolhendo corretamente os casos de teste de maneira que seja eficiente.“ (KOSCIANSKI)

7

Categoria dos testes

- Caixa-branca (estrutural)
  - Avalia o comportamento interno do sistema;
  - São avaliados códigos, configurações, modelos de dados, ...;
  - Tudo o que auxilia na definição das entradas de teste e resultados esperados.



8

Categoria dos testes

- **Caixa-preta (funcional)**
  - Avalia o comportamento externo do sistema (funcionalidades e usabilidade).

9

Testes Unitários com JUnit: uma visão geral

junit.org

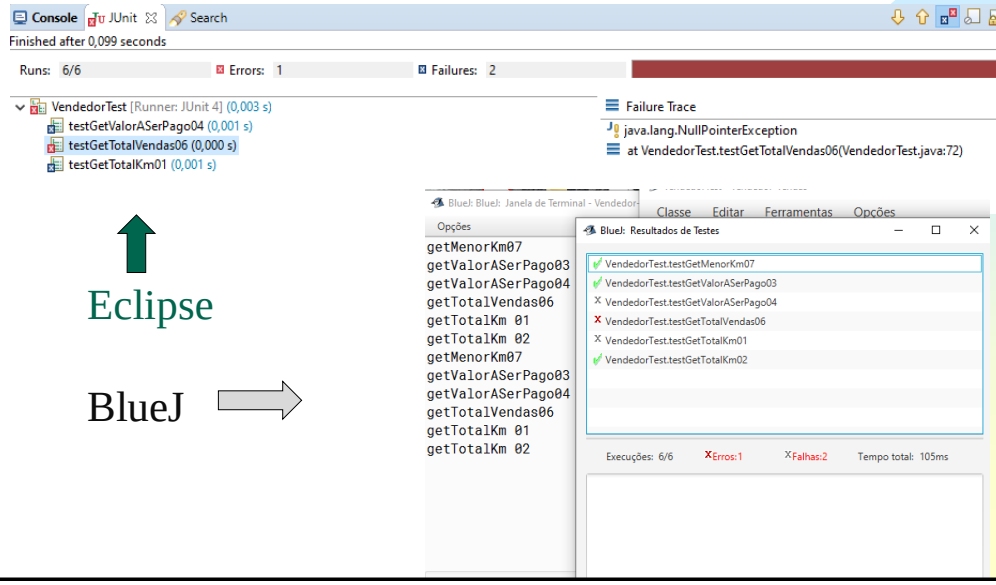
10

Sobre o JUnit

- JUnit é um framework para execução de teste unitários de forma automatizada em Java
- Criado em 1997 por Erich Gamma e Kent Beck
- Hospedado no GitHub
- Distribuído sob a licença IBM Common Public License
- Modelo adotado por outras linguagens
- Possui versões:
  - Versão 3: primeira de grande uso (integrada com ambientes)
  - Versão 4: para versões do Java 4 e superior
  - Versão 5 (Jupiter): para Java 8, 9 e 10

11

Execução dos testes em IDE



12

Testes simples com JUnit 5.0

- Criar uma classe nova para implementar os “casos de teste” do plano de testes de uma classe
- Acrescentar as duas cláusulas abaixo de importação:
  - import org.junit.jupiter.api.Test
  - import static org.junit.jupiter.api.Assertions.*;
- Para cada “caso de teste”, criar um método e introduzir a anotação *@Test*
  - O método não pode ser privado
  - O método não pode retornar dados (void)
  - Dentro deste método, deve-se utilizar um comando *assert*, para validar uma situação

13

Testes simples com JUnit 5.0

- Exemplo:

```java
import static org.junit.jupiter.api.Assertions.assertEquals;

import example.util.Calculator;

import org.junit.jupiter.api.Test;

class MyFirstJUnitJupiterTests {

    private final Calculator calculator = new Calculator();

    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1));
    }

}
```

14

Métodos *assert*

- Os métodos *assert* são utilizados para verificar se uma condição é verdadeira. Caso a verificação não seja bem sucedida, é lançada a exceção *AssertionError* .

Método	Descrição
assertEquals(long esperado, long real)	Verifica se os dois números inteiros são iguais
assertEquals(double esperado, double real, double tolerancia)	Verifica se dois números reais são iguais até o limite de tolerância informado.
assertEquals(Object esperado, Object real)	Verifica se dois objetos são iguais (*equals*)
assertTrue(boolean condição)	Verifica se a condição é verdadeira
assertFalse(boolean condição)	Verifica se a condição é falsa

15

Métodos *assert*

Método	Descrição
assertNotNull(Object)	Verifica se o objeto não é nulo
assertNull(Object)	Verifica se o objeto é nulo
assertSame(Object esperado, Object real)	Verifica se os objetos são o mesmo
assertNotSame(Object esperado, Object real)	Verifica se os objetos não são o mesmo
assertArrayEquals(Object[] esperado, Object[] real)	Verifica se os vetores possuem o mesmo conteúdo. Os vetores podem ser de tipos primitivos.
assertThrows(*ClasseExceçãoEsperada.class*, métodoExecutado);	Verifica se a execução gerou a exceção esperada

16

Método *fail*

- O método *fail* pode ser utilizado para lançar a exceção *AssertionError*

Registro de erros e falhas

- As exceções *AssertionError* são contabilizadas como *Falhas*
- Qualquer outra exceção é contabilizada como *Erro*.

17

Considerações

- Bons testes unitários não testam necessariamente todas as possíveis combinações.
- Testes unitários não testam métodos muito simples.
- Desenvolvedores podem consultar um teste para descobrir o uso pretendido de uma classe ou método.
- São boas práticas:
  - Adotar nomes significativos para os testes;
  - Utilizar os métodos assert que tem uma String como último parâmetro indicando a mensagem de problema;
  - Um teste deveria conter apenas um assert

19