



II Beljungle

SEPAI 2005

JUnit

Implementando Testes
Unitários em Java



br.groups.yahoo.com/group/xpnorte



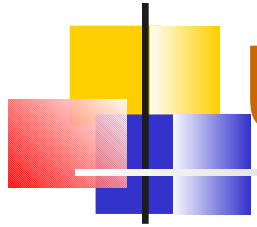
Palestrante: Manoel Pimentel Medeiros

- É Engenheiro de Software, com mais de 15 anos na área de TI, atualmente trabalha com projetos pela Rhealeza(SP). É Diretor Editorial da Revista Visão Ágil, Membro da Agile Alliance e foi um dos pioneiros na utilização e divulgação de métodos ágeis no Brasil. Já escreveu artigos para importantes revistas e portais especializados no Brasil e no exterior. Possui as certificações CSM e CSP da Scrum Alliance. Já participou do time de Desenvolvimento do NetBeans(Sun), foi criador do projeto BoxSQL, fundador do grupo XPNorte e do NUG-BR e frequentemente palestra em eventos sobre processos e tecnologias. Maiores informações em: <http://manoelpimentel.blogspot.com>



Agenda:

- Um pouco de XP.
- Como programar guiado a testes?
- Teste Unitário (O que?, por que?, quando?, quem?, como?).
- JUnit(O que?, por que?, quando?, quem?, como?).
- JUnit(Planejamento e arquitetura das classes).
- JUnit(Funcionamento e Análise do resultado).
- Implementado testes em JUnit usando o Eclipse.
- Outros métodos e técnicas complementares.
- Conclusão.



Um pouco de XP:

- **XP** é um apelido carinhoso de uma **metodologia ágil** de desenvolvimento designada **Extreme Programming**, com foco em agilidade de equipes e qualidade de projetos, apoiada em valores como **simplicidade, comunicação, feedback e coragem.**



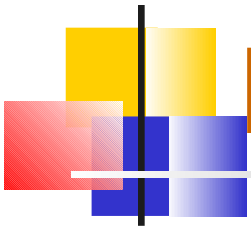
Você desenvolve “O” programa com o cliente?

- XP é metodologia muito comportamental, onde prima mudanças de atitudes e práticas.
- Sua principal mudança está na máxima integração entre pessoas e principalmente, estimulando uma participação maior do cliente. Portanto, literalmente, temos que **FAZER O PROGRAMA COM O CLIENTE.**



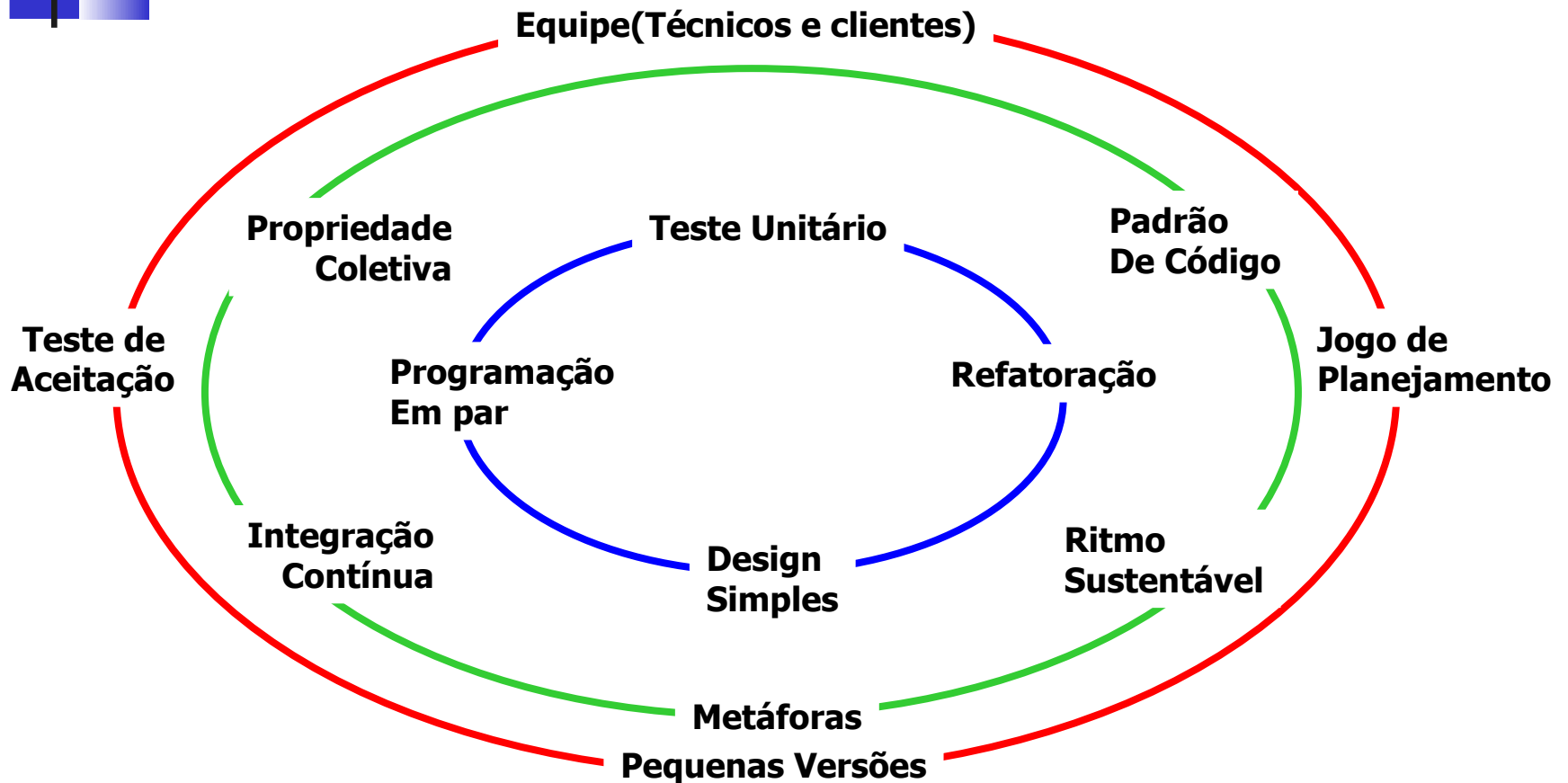
Práticas XP

- XP **sugere** um conjunto de **boas práticas** que melhoram o **planejamento, execução,** e **gerenciamento** de seu projeto de software.
- Essas práticas melhoram sua **eficiência e eficácia**, diminuindo o **retrabalho**, garantindo dessa forma a **qualidade** em seu projeto.



Práticas XP

- Práticas organizacionais
- Práticas de equipe
- Práticas de pares

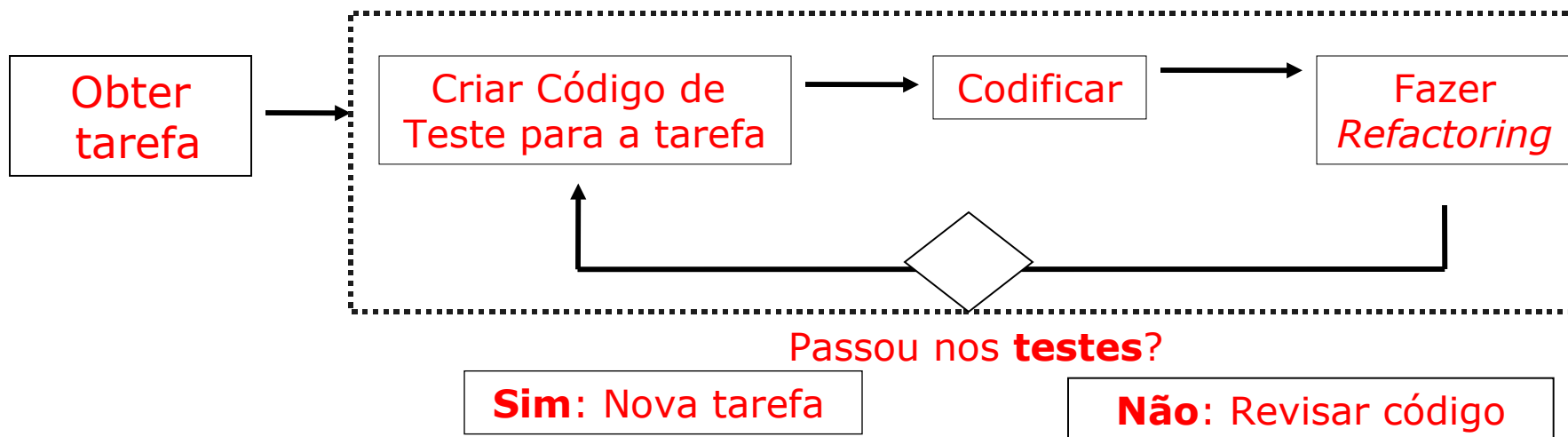


TDD

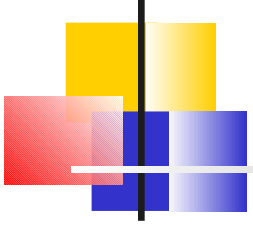
Test Driven Development

Desenvolvimento Guiado por Testes, define que antes de criarmos um código novo, devemos escrever um teste para ele.

- E testes serão usados como métrica em todo o tempo de vida do projeto.



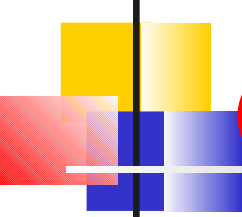
Teste Unitário



Imagine se um avião só fosse testado após a conclusão de sua construção....

Seria um desastre....



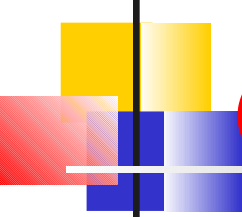


Teste Unitário

(O que é?)

O **teste unitário** é uma modalidade de testes que se concentra na verificação da menor unidade do projeto de software. É realizado o teste de uma **unidade lógica**, com uso de dados suficientes para se testar apenas a lógica da unidade em questão.

Em sistemas construídos com uso de linguagens orientadas a objetos, essa unidade pode ser identificada como um **método**, uma **classe** ou mesmo um **objeto**.



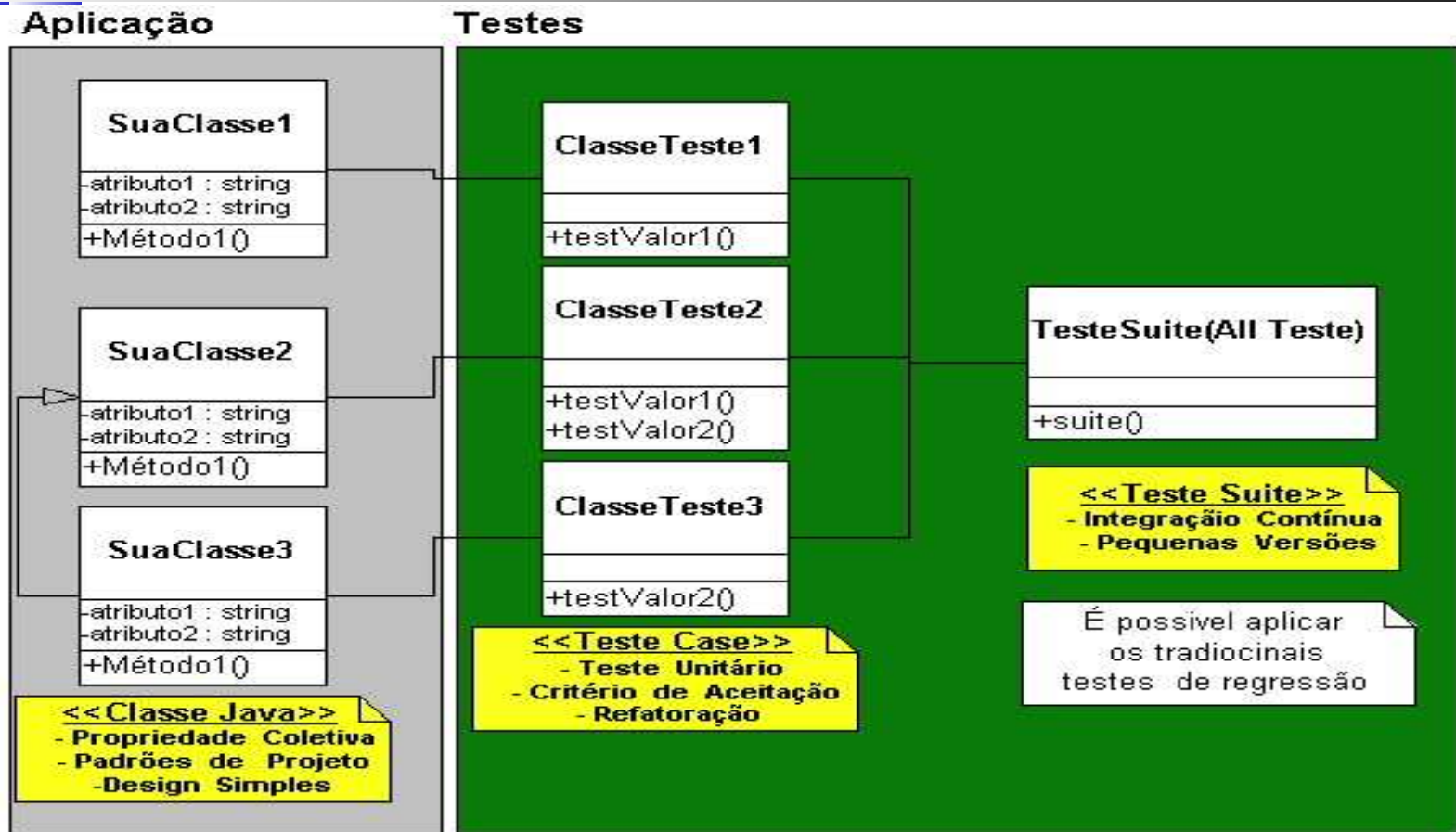
Teste Unitário

(Por que?)

- **Previne** contra o aparecimento de “BUG’S” oriundos de códigos mal escritos.
- Código testado é mais confiável.
- Permite alterações sem medo(**coragem**)
- Testa situações de sucesso e de falha.
- Resulta em outras práticas XP como : **Código coletivo, refatoração, integração contínua.**
- Serve como métrica do projeto (*teste == requisitos*)
- Gera e preserva um “conhecimento” sobre o projeto.

Teste Unitário

(Organização dos testes e práticas XP)





Teste Unitário

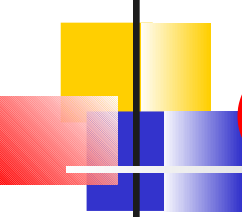
(Quando fazer?)

- **No início**

Primeiro projetar e escrever as classes de testes, **depois** as classes com regra de negócios

- **Diariamente**

È SUGERIDO que seja rodado os testes várias vezes ao dia (é fácil corrigir **pequenos** problemas do que corrigir um “**problemão**” somente no final do projeto.



Teste Unitário

(Quem faz?)

- **Test Case**(para cada classe)
Desenvolvedor(Projeta, escreve e roda)
- **Test Suite**(Rodas vários test cases)
Coordenador e Desenvolvedor
(Projeta, escreve e roda)

* *Teste de aceitação(homologação) é feito junto ao cliente.*



Teste Unitário

(Que Testar?)

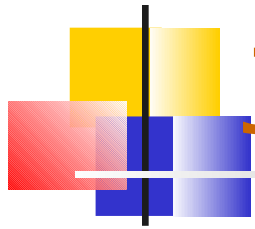
- A principal regra para saber o que testar é: “Tenha **criatividade** para imaginar as **possibilidades** de testes”.
- Comece pelas mais simples e deixe os testes “complexos” para o final.
- Use apenas dados suficientes (não teste 10 condições se três forem suficientes)
- Não teste métodos triviais, tipo get e set.
- No caso de um método set, só faça o teste caso haja validação de dados.
- Achou um bug? Não conserte sem antes escrever um teste que o pegue (se você não o fizer, ele volta)!



Exercício de Imaginação

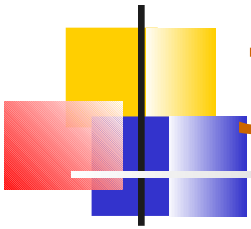
- Ache as possibilidades de testes neste diagrama de classe





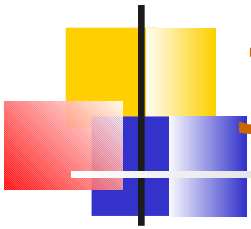
JUnit – O que é?

- Um **framework** que facilita o desenvolvimento e execução de **testes de unidade** em código **Java**
- Fornece Uma API para construir os testes e Aplicações para executar testes



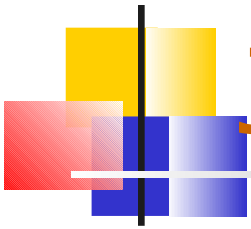
JUnit – Por que?

- JUnit pode verificar se cada unidade de código funciona da forma esperada.
- Facilita a **criação, execução** automática de testes e a **apresentação** dos resultados.
- É Orientado a Objeto
- É Free e pode ser baixado em:
www.junit.org



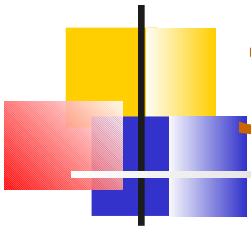
JUnit – Como instalar?

- Incluir o arquivo **junit.jar** no **classpath** para compilar e rodar os programas de teste
- Já vem configurado nas versões recentes de IDE's como Eclipse, JBuilder, BlueJ e outros.



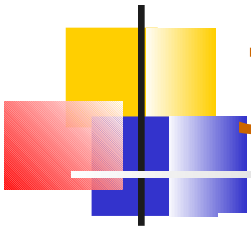
JUnit – Planejando os testes

- 1. Defina uma **lista** de tarefas a implementar(o que testar)
- 2. Escreva uma classe (**test case**) e implemente um método de teste para uma tarefa da lista.
- 3. Rode o **JUnit** e certifique-se que o teste falha
- 4. Implemente o código mais **simples** que rode o teste

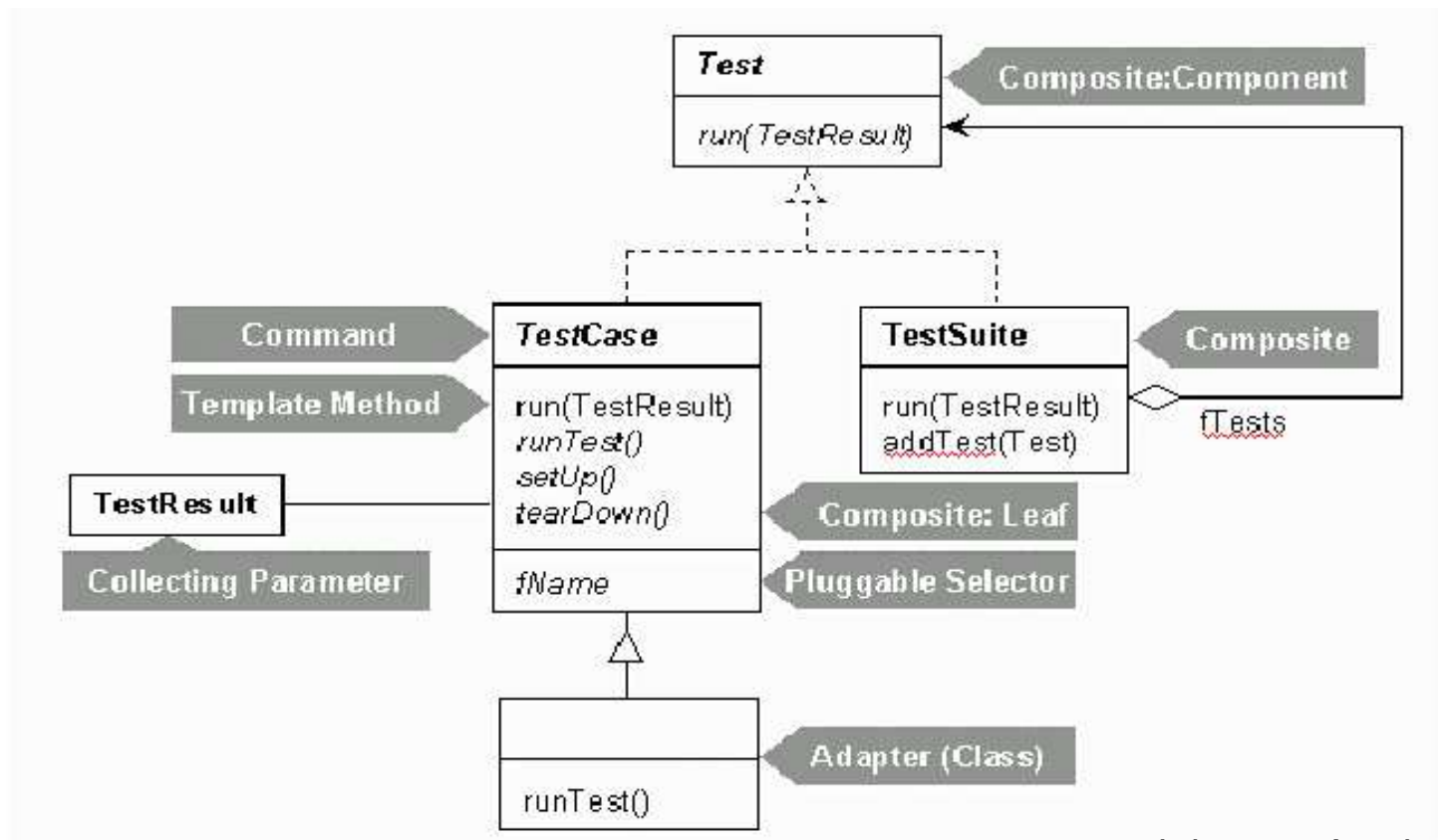


JUnit – Planejando os testes

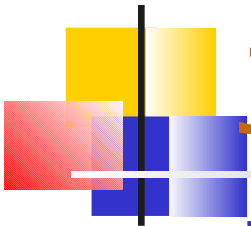
- 5. **Refatore** o código para remover a duplicação de dados
- 6. Caso necessário, escreva mais um teste ou **refine** o existente
- 7. Faça esses passos para toda a lista de tarefas.



JUnit- Arquitetura das Classes



Fonte: Manual do JUnit (Cooks Tour)



JUnit – Como implementar

- 1. Crie uma classe que estenda **junit.framework.TestCase** para cada classe a ser testada

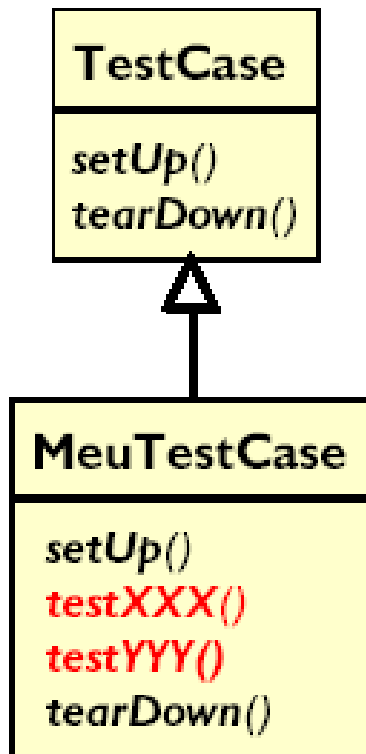
```
import junit.framework.*;
class SuaClasseTest extends TestCase
{...
}
```
- 2. Para cada método a ser testado defina um método **public void test???()** no test case
- SuaClasse:

```
public int Soma(Object o ... )
{ ...
}
```
- SuaClasseTest:

```
public void testSoma()
```



JUnit – Funcionamento



- O TestRunner recebe uma subclasse de `junit.framework.TestCase`
- Cada método `testXXX()`, executa:
 - 1. o método `setUp()` */* Opcional */*
 - 2. o próprio método `testXXX()`
 - 3. o método `tearDown()` */* Opcional */*

JUnit – Analisando o Resultado

- Em modo gráfico, os métodos testados podem apresentar o seguintes resultados:



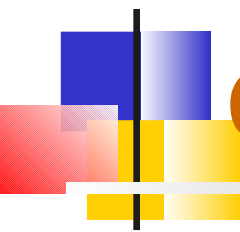
- *Sucesso*



- *Falha*



- *exceção*



Criando a **classe** de **teste** no Eclipse



CalculoTest.java X Calculo.java

```
package processos;
import junit.framework.TestCase;

public class CalculoTest extends TestCase {
    public void testExecutaCalculo() {
        //Define os valores a serem calculados e testados
        float PassaValor1 = 10;
        float PassaValor2 = 5;
        float RetornoEsperado = 15;
        //Dispara o método "ExecutaCalculo" da classe "Calculo" e armazena
        //resultado em um variável
        float RetornoFeito = Calculo.ExecutaCalculo(PassaValor1,PassaValor2);
        //Compara o valor retornado com o que era esperado
        assertEquals(RetornoEsperado,RetornoFeito,0);
    }
}
```

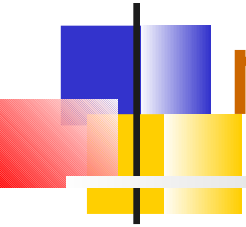


Sua Classe a
ser testada

```
package processos;

public class Calculo {
    public static float ExecutaCalculo(float Valor1, float Valor2)
    {
        float Soma = Valor1 + Valor2;
        return Soma;
    }
}
```

Rodando o teste em
modo **gráfico**



CalculoTest.java X Calculo.java

```
package processos;  
import junit.framework.Tes
```

```
public class CalculoTest e  
    public void testExecut  
        //Define os valore  
        float PassaValor1  
        float PassaValor2  
        float RetornoEsper  
        //Dispara o método  
        //resultado em um  
        float RetornoFeito  
        //Compara o valor  
        assertEquals(Retor  
    }  
}
```

Toggle Line Breakpoint Ctrl+Shift+B

Toggle Method Breakpoint

Toggle Watchpoint

Skip All Breakpoints

Add Java Exception Breakpoint...

Add Class Load Breakpoint...

Run Last Launched Ctrl+F11

Debug Last Launched F11

Run History

Run As

Run...

Debug History

Debug As

Debug...

Inspect Ctrl+Shift+I

Display Ctrl+Shift+D

Execute Ctrl+U

Step into Selection

External Tools

1 Java Applet

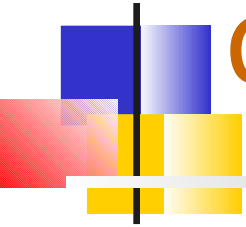
2 Java Application

3 JUnit Plug-in Test

4 JUnit Test

5 Run-time Workbench

Resultado em caso
de **SUCESSO**





JUnit Package Explorer

Finished after 0,016 seconds



Runs: 1/1 Errors: 0 Failures: 0

Failures Hierarchy

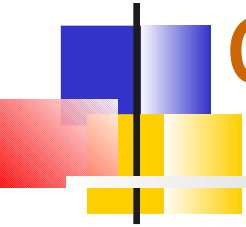
Failure Trace

CalculoTest.java Calculo.java

```
package processos;
import junit.framework.TestCase;

public class CalculoTest extends TestCase {
    public void testExecutaCalculo() {
        //Define os valores a serem calculados e testados
        float PassaValor1 = 10;
        float PassaValor2 = 5;
        float RetornoEsperado = 15;
        //Dispara o método "ExecutaCalculo" da classe "Calculo" e armazena
        //resultado em um variável
        float RetornoFeito = Calculo.ExecutaCalculo(PassaValor1,PassaValor2);
        //Compara o valor retornado com o que era esperado
        assertEquals(RetornoEsperado,RetornoFeito,0);
    }
}
```

Resultado em caso
de **FALHA**





JUnit Package Explorer

Finished after 0,016 seconds

Runs: 1/1 Errors: 0 Failures: 1

Failures Hierarchy

testExecutaCalculo - processos.CalculoTest

Failure Trace

```
junit.framework.AssertionFailedError: expected: <15.0> but was: <16.0>
at processos.CalculoTest.testExecutaCalculo(CalculoTest.java:14)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcce
```

CalculoTest.java Calculo.java

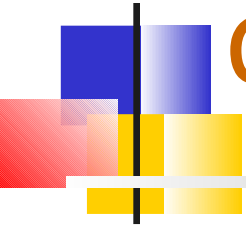
```
package processos;
import junit.framework.TestCase;

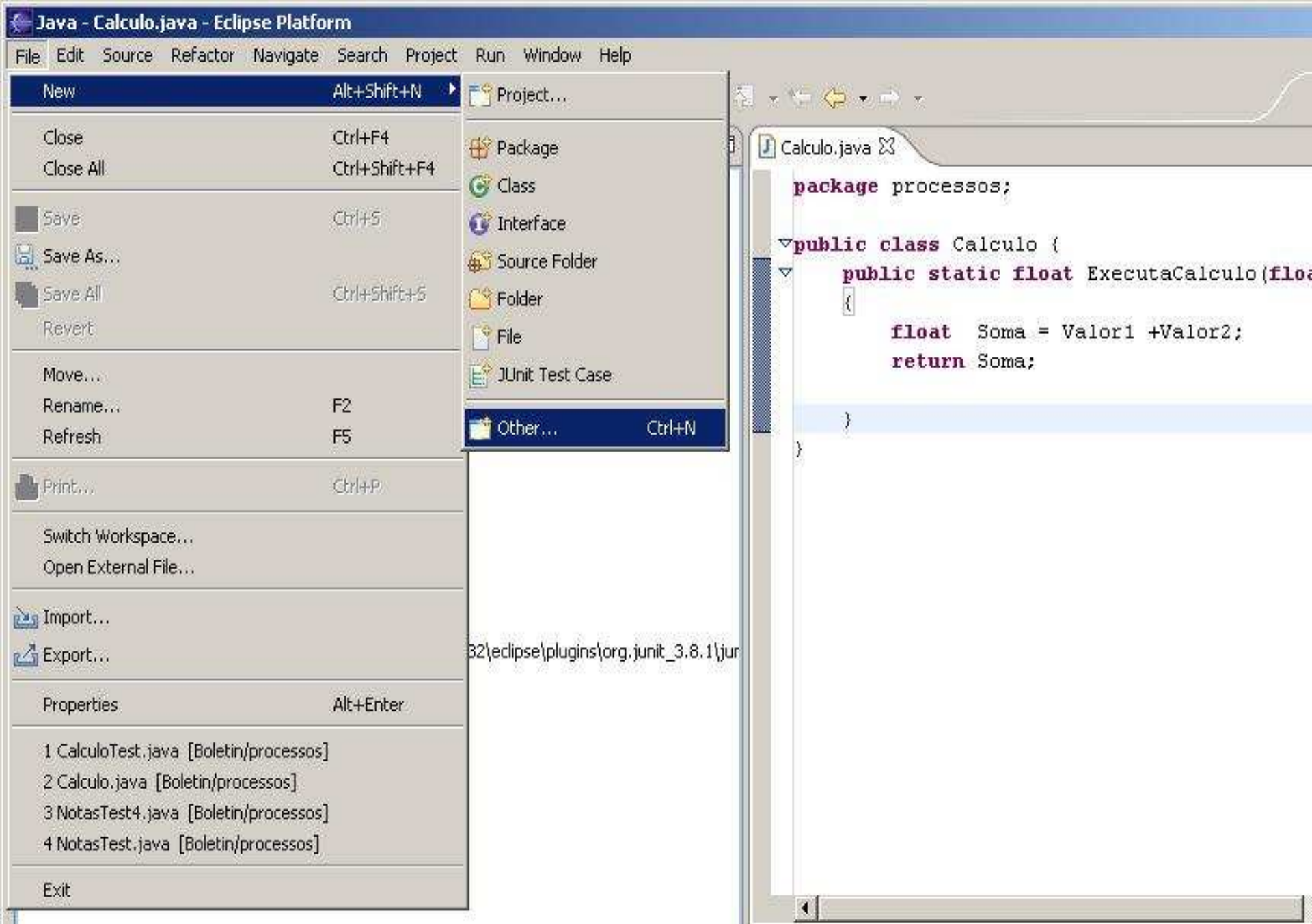
public class CalculoTest extends TestCase {
    public void testExecutaCalculo() {
        //Define os valores a serem calculados e testados
        float PassaValor1 = 10;
        float PassaValor2 = 6;
        float RetornoEsperado = 15;
        //Dispara o método "ExecutaCalculo" da classe "Calculo"
        //resultado em um variável
        float RetornoFeito = Calculo.ExecutaCalculo(PassaValor1
        //Compara o valor retornado com o que era esperado
        assertEquals(RetornoEsperado, RetornoFeito, 0);
    }
}
```

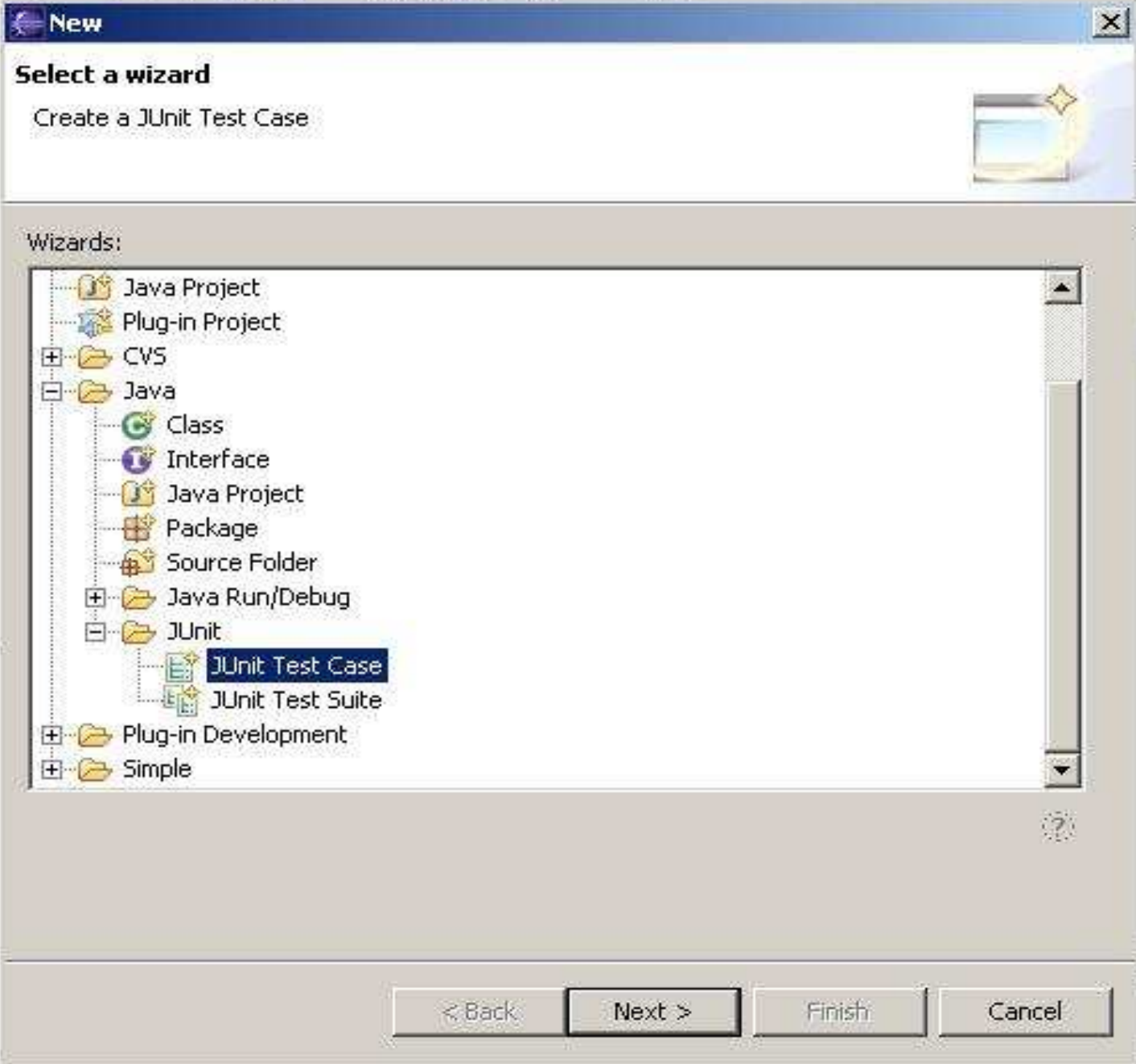
Console

<terminated> CalculoTest [JUnit] C:\Borland\JBuilder2005\jdk1.4\bin\javaw.exe (11/09/2005 23:38:49)

Automatizando a criação dos Test Cases







New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

Source Folder:

Boletin

Browse...

Package:

processos

Browse...

Name:

Calculo2Test

Superclass:

junit.framework.TestCase

Browse...

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Add TestRunner statement for: swing ui

☐ setUp()

☐ tearDown()

☐ constructor()

Class Under Test:

processos.Calculo

Browse...

< Back

Next >

Finish

Cancel

```
culo(float Valo  
r2;
```




New JUnit Test Case

Test Methods

Select methods for which test method stubs should be created.



Available methods:

- ☒  Calculo
 - ☒  ExecutaCalculo(float, float)
- ☐  Object

Select All

Deselect All

1 method selected.

- ☐ Create final method stubs
- ☐ Create tasks for generated test methods

< Back

Next >

Finish

Cancel



Calculo.java *Calculo2Test.java x

```
/*
 * Created on 16/09/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
package processos;

import junit.framework.TestCase;

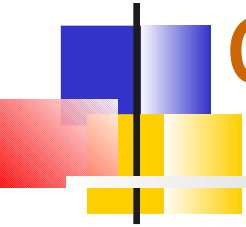
/**
 * @author Manoel Pimentel
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Calculo2Test extends TestCase {

    public void testExecutaCalculo() {

    }

}
```

Outros Exemplo de teste





Calculo.java X

```
package processos;

public class Calculo {
    public static boolean NomeValido(String vNome) {
        if (vNome.length() > 10) {
            return true;
        }
        else {
            return false;
        }
    }

    public static float ExecutaCalculo(float Valor1, float Valor2) {
        float Soma = Valor1 + Valor2;
        return Soma;
    }

    public static float MultilicaValores(float Valor1, float Valor2) {
        float Produto = Valor1 * Valor2;
        return Produto;
    }
}
```

Java - VerificaNomeTest.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

JUnit

Package Explorer

Finished after 0,016 seconds

Runs: 1/1 Errors: 0 Failures: 1

Failures

Hierarchy

processos.VerificaNomeTest

testNomeValido

Failure Trace

junit.framework.AssertionFailedError: Nome Inválido
at processos.VerificaNomeTest.testNomeValido(VerificaM
at sun.reflect.NativeMethodAccessorImpl.invoke0(Nativ
at sun.reflect.NativeMethodAccessorImpl.invoke(Native
at sun.reflect.DelegatingMethodAccessorImpl.invoke(De

Calculo.java

VerificaNomeTest.java

CalculoTesteMultiplicacao.java

```
package processos;

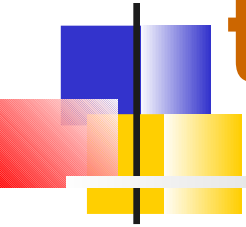
import junit.framework.TestCase;

public class VerificaNomeTest extends TestCase {

    public void testNomeValido() {
        boolean b = Calculo.NomeValido("Manoel");
        assertTrue("Nome Inválido",b);
    }

}
```

Criando **Test Suite** para rodar vários **test cases**



New

Select a wizard

Create a JUnit Test Suite

Wizards:

- Java Project
- Plug-in Project
- CVS
- Java
 - Class
 - Interface
 - Java Project
 - Package
 - Source Folder
 - Java Run/Debug
 - JUnit
 - JUnit Test Case
 - JUnit Test Suite
- Plug-in Development
- Simple

< Back

Next >

Finish

Cancel

New JUnit Test Suite

JUnit Test Suite

Create a new JUnit Test Suite class for a package

Source Folder:

Boletin

Browse...

Package:




processos

Browse...

Test suite:

AllTests

Test Classes to include in Suite:

- ☒  CalculoTest
- ☒  CalculoTesteMultiplicacao
- ☒  VerificaNomeTest

Select All

Deselect All

3 classes selected

Would you like to create a method stub for main?

☐ public static void main(String[] args)

☐ Add TestRunner statement for: text.ui

< Back

Next >

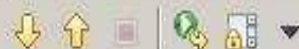
Finish

Cancel



JUnit Package Explorer

Finished after 0,062 seconds



Runs: 3/3 Errors: 0 Failures: 2

Failures Hierarchy

- Test for processos
 - processos.CalculoTest
 - testExecutaCalculo
 - processos.CalculoTesteMultiplicacao
 - testExecutaCalculo
 - processos.VerificaNomeTest
 - testNomeValido

Failure Trace

```
junit.framework.AssertionFailedError: expected: <15.0> but  
at processos.CalculoTest.testExecutaCalculo(CalculoTest.java:15)  
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
at sun.reflect.NativeMethodAccessorImpl.invoke(Native Method)
```

Calculo.java AllTests.java

```
package processos;  
import junit.framework.Test;  
  
public class AllTests {  
  
    public static Test suite() {  
        TestSuite suite = new TestSuite("Test for processos");  
        //$JUnit-BEGIN$  
        suite.addTestSuite(CalculoTest.class);  
        suite.addTestSuite(CalculoTesteMultiplicacao.class);  
        suite.addTestSuite(VerificaNomeTest.class);  
        //$JUnit-END$  
        return suite;  
    }  
}
```




JUnit - Outros Métodos de Testes

- **assertEquals**
 - Testa igualdade entre dois objetos(esperado x retornado)
- **assertFalse()**
 - Testa Retorno booleano FALSO
- **assertTrue()**
 - Testa Retorno booleano VERDADEIRO
- **assertNotNull()**
 - Testa se um valor de um objeto NÃO está NULO
- **assertNull()**
 - Testa se um valor de um objeto está NULO

JUnit – métodos **setUp()** e **tearDown()**

São os dados **reutilizados** por vários testes, **Inicializados** no **setUp()** e **destruídos** no **tearDown()** (se necessário)

```
package processos;
import junit.framework.TestCase;
public class CalculoVariosTest extends TestCase {
    float PassaValor1;
    float PassaValor2;
    protected void setUp(){
        PassaValor1 = 10;
        PassaValor2 = 5; }
    public void testExecutaCalculo() {
        float RetornoEsperado = 15;
        float RetornoFeito = Calculo.ExecutaCalculo(PassaValor1,PassaValor2);
        assertEquals(RetornoEsperado,RetornoFeito,0); }
    public void testMultiplicaValores() {
        float RetornoEsperado = 60;
        float RetornoFeito = Calculo.MultiplicaValores(PassaValor1,PassaValor2);
        assertEquals(RetornoEsperado,RetornoFeito,0);}
}
```



Técnicas complementares

- É importante também, ser aplicado tipos de testes como:
 - Teste de Performance,
 - Teste de Carga,
 - Teste de estresse,
 - Teste de aceitação, etc.



Dúvidas?

E-mail: manoelp@gmail.com

Ou

br.groups.yahoo.com/group/xpnorte