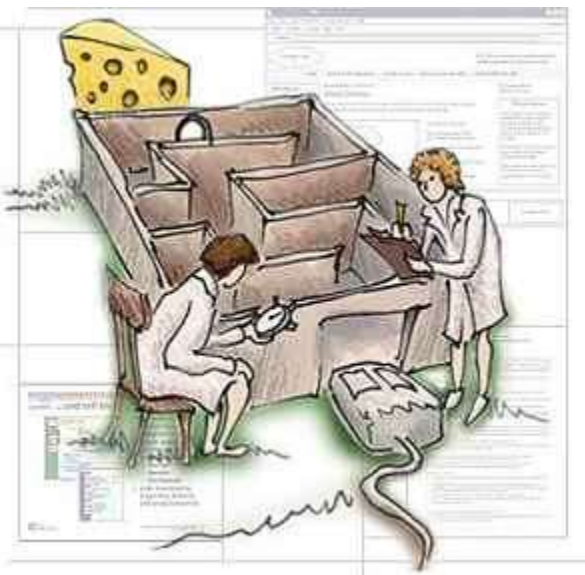


Testes de Unidade com JUnit

Prof. MSc. Álvaro d'Arce
alvaro@darce.com.br



Tópicos

Testes de Programas

JUnit – Introdução

JUnit – Prática

Testes

- Você viajaria em um avião que nunca saiu do chão?



- Você entrega software sem testar?

Testes de Programas [1/4]

“Qualquer recurso de programa sem um teste automatizado simplesmente não existe.”

Kent Beck. Extreme Programming Explained. p 56

Testes de Programas [2/4]

- Defeitos de programa: problemas
 - Custos enormes
 - Tempo, dinheiro, frustrações...
- Como amenizar esses problemas?
 - Criação e execução de casos de teste (de maneira contínua) para programas
 - Abordagem prática e comum para lidar com defeitos de programas
 - Antes que sejam “deixados para trás” no ambiente de desenvolvimento

Testes de Programas [3/4]

- Importância dos testes
 - Um produto de software deve passar por várias fases de teste:
 - Teste de unidade, de integração, de sistema, de aceitação...
- Função de um teste:
 - Certificar que uma determinada entrada **sempre** produz uma **mesma saída**

Testes de Programas [4/4]

- Testes devem ser escritos
- Poucos o fazem...
 - “Falta” de tempo...
- Resultado: ciclo vicioso



- Quebrando o ciclo:
 - Criar um ambiente simples de testes

Testes de Programas

Testes de Unidade (Unitário)

- Testam as menores unidades de programa desenvolvidas
 - POO: unidade pode ser método/classe/objeto
- Objetivo
 - Prevenir defeitos
 - Permitir um nível de qualidade de produto durante o desenvolvimento do software

Testes de Programas

Importância dos Testes [1/5]

- Dados não mentem – **Erros existem!**
- 1/3 poderiam ser evitados
- 50% são detectados em produção
- Prejuízo de US\$ ~60 bilhões/ano

[http://www.nist.gov/public_affairs/releases/n02-10.htm]

Testes de Programas

Importância dos Testes [2/5]

- Defeitos são caros!
 - Quanto mais tarde são encontrados, mais caros serão.
- Conclusão:
 - É melhor encontrar defeitos o mais cedo possível.

Testes de Programas

Importância dos Testes [3/5]

- Seja **profissional**, garanta seu trabalho!
- **Você** é o único **responsável** pela **qualidade** do seu trabalho.
 - Ninguém melhora se **você** não melhorar primeiro.
- Desenvolvedores **profissionais** escrevem **testes**
- **Teste seu software!**



Testes de Programas

Importância dos Testes [4/5]

- **Jamais** entregue ao seu cliente um produto **sem** qualidade!



Testes de Programas

Importância dos Testes [5/5]

- Escreva seus testes antes de terminar sua programação.
- Você vai:
 - Amar seus códigos
 - Programar melhor
 - Pensar antes de codificar
 - Reduzir código inútil
 - Ganhar com qualidade

Testes de Programas

Testes de Unidade (Unitário) [1/2]

- Pergunta:
 - Ao criarmos métodos na programação OO, temos a certeza de que ele retorna o valor correto?
 - Ou em caso mais geral, será que eles estão fazendo o que deveriam fazer?
- Resposta:
 - Vamos aplicar testes unitários

Testes de Programas

Testes de Unidade (Unitário) [2/2]

- Testes
 - Responsabilidade do próprio desenvolvedor
 - Comumente testam um método individualmente
 - Comparação de uma saída conhecida após o processamento da mesma
 - Não testam todo o programa

Testes de Programas

Java: Teste pelo método `main()` [1/3]

- Criação do método *main()* na classe a ser testada
 - Instância da classe
 - Execução de uma série de checagens
 - Certificar que o objeto possui o comportamento desejado

Testes de Programas

Java: Teste pelo método `main()` [2/3]

- Questões: eficiência como ambiente de teste
 - Não há conceito explícito de teste aprovado ou reprovado
 - Normalmente, o programa gera mensagens com `System.out.println()`
 - Desenvolvedor decide se a mensagem está correta ou não
 - `main()` tem acesso a itens *protected* e *private*
 - Enquanto desenvolvedor pode querer testar o funcionamento interno de uma classe, muitos testes se referem à interface de um objeto ao mundo externo

Testes de Programas

Java: Teste pelo método main() [3/3]

- Muito código a ser escrito
- Testa-se apenas 1 método por vez
- Não há mecanismos para coletar resultados de maneira estruturada
 - Mais código ainda para se verificar o retorno dos métodos de forma automática
- Não há replicabilidade
 - Após cada teste, o desenvolvedor tem que examinar e interpretar os resultados

Testes de Programas

Junit [1/2]

“Sempre que você estiver tentando escrever um print() ou uma expressão de depuração, escreva um teste.”

Martin Fowler

Testes de Programas

JUnit [2/2]

- Framework de criação de testes automatizados para desenvolvimento Java
- Possui API que habilita o desenvolvedor a facilmente criar casos de teste em Java
- Provê abrangente facilidade de asserção
 - Verificar resultados esperados x resultados reais
- Utiliza um princípio fundamental da programação XP:
 - Criação e execução de testes deve ser fácil

Tópicos

Testes de Programas

JUnit – Introdução

JUnit – Prática

JUnit

Introdução

- Facilita criação de código para automação de testes com apresentação dos resultados
 - Dirigido a Testes de unidade – Caixa Branca
 - Componentes de um sistema (classes/métodos) testados de maneira isolada
- Verifica se cada método de uma classe funciona da maneira esperada
 - Exibindo possíveis erros ou falhas

JUnit

Vantagens [1/2]

- Permite rápida criação de códigos de teste
 - Possibilitando aumento da qualidade do sistema sendo desenvolvido e testado
- Não é necessário escrever o próprio framework
 - Framework Caixa Preta
- Amplamente utilizado pelos desenvolvedores da comunidade open-source
- Uma vez escritos, os testes são executados rapidamente
 - Sem a interrupção do processo de desenvolvimento

JUnit

Vantagens [2/2]

- Checa os resultados dos testes e fornece uma resposta imediata
- Pode-se criar uma hierarquia de testes que permitirá testar apenas uma parte ou todo o sistema
- Permite que o programador perca menos tempo depurando seu código
- Integração com as principais IDEs
 - NetBeans, Eclipse, JDeveloper...

Testes Manuais x JUnit

- Calcular imposto com base no salário – calcular(Float salario)
 - Imposto 27,5% a partir de 3.743
 - Imposto 22,5% a partir de 2.995 até 3.743
 - O método deve retornar o valor do imposto a ser pago

Testes Manuais x JUnit

Método a ser testado

```
public static double calcular( double salario ) {  
    if ( salario > 3743 ) {  
        return salario * 0.275;  
    } else if ( salario > 2995 && salario < 3743) {  
        return salario * 0.225;  
    }  
    return -1;  
}
```

Testes Manuais x JUnit

Teste manual pelo método main

```
public static void main( String args[] ){  
  
    double resultado = calcular( 3800.0);  
    if ( resultado == 1045) {  
        System.out.println("Passou no teste");  
    } else {  
        System.out.println("Teste falhou");  
    }  
}
```

Testes Manuais x JUnit

Classe de teste automatizado com JUnit [1/2]

```
import org.junit.Assert;  
import org.junit.Test;
```

```
public class FiscalRendaTest {
```

```
    @Test
```

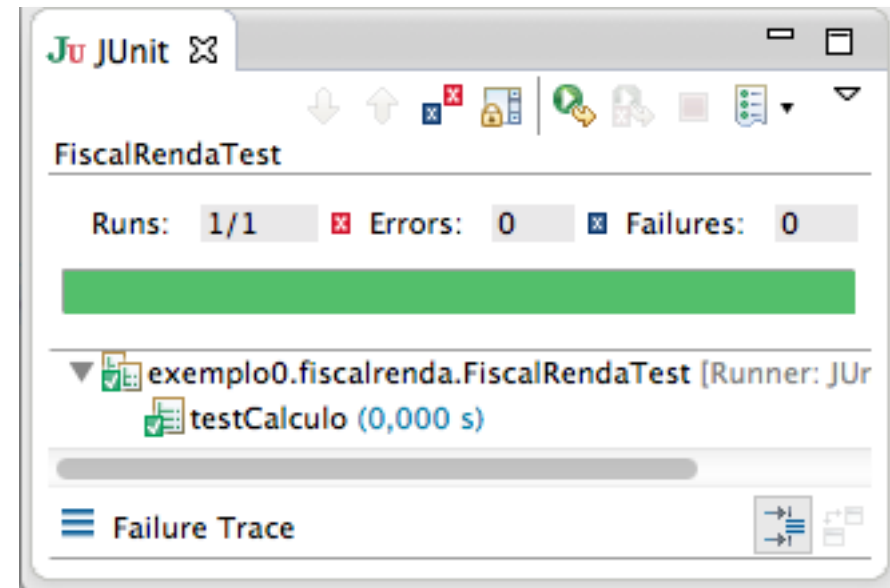
```
    public void testCalculo(){  
        double resultado =
```

```
FiscalRenda.calcular(3800.0);
```

```
        Assert.assertEquals(1045, resultado, 0.1);
```

```
    }
```

```
}
```



Testes Manuais x JUnit

Classe de teste automatizado com JUnit [2/2]

Erro

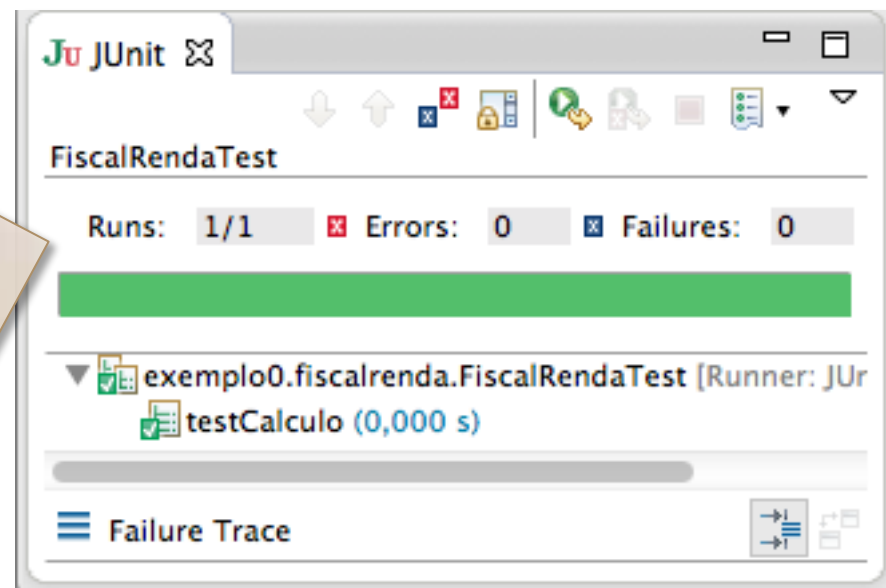
Quando o método de teste produziu um resultado inesperado (exceção)

Falha

Quando há algum problema relacionado ao código testado

Sucesso

Quando o método produziu o resultado esperado



```
Assert.assertEquals(1045, resultado, 0.1);
```

```
}
```

```
}
```

JUnit

Utilização

1. Criar uma classe de testes JUnit
2. Imports as classes e métodos necessários
3. Criar métodos para realizar os testes
4. Comparar o retorno do método testado com o resultado esperado

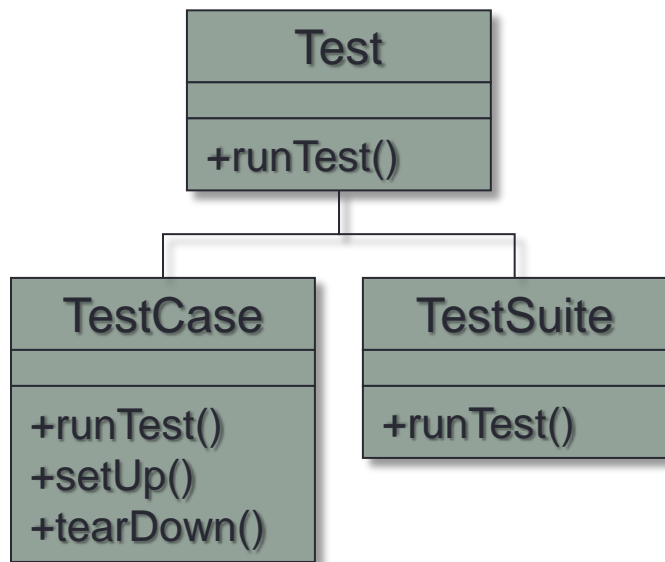
JUnit

Questões

- Casos de teste são definidos em classes separadas
 - Sem acesso a partes encapsuladas
- Testes são realizados a partir da interface de um objeto ao mundo externo
- Hábito:
 1. Codifique um pouco...
 2. Teste um pouco...
 3. Codifique um pouco...
 4. Teste um pouco...
 - Resumo: Objeto pronto, teste-o

JUnit

Arquitetura



até o JUnit 3.x

a partir do JUnit 4

API do JUnit

- JUnit 3
 - Classe *Test*
 - *runTest()*: controla execução de testes particulares
 - Classe *TestCase*: testa os resultados de um método
 - *setUp()*: chamado antes de cada método de teste*
 - *tearDown()*: chamado depois de cada método de teste*
 - Classe *TestSuite*: define um conjunto de testes
- JUnit 4
 - *import static* na API

*defasados – substituídos por anotações no JUnit 4

JUnit

Considerações Finais

- Testes de unidade:
 - Importantes na construção de métodos
 - Permite ao desenvolvedor testá-los durante a construção
 - Viabilizando a implementação de métodos livres de erros
- JUnit:
 - Possibilita criar testes antes da conclusão do sistema
 - Testando métodos separadamente assim que estiverem prontos
 - Evita percorrer todo o código para descobrir defeitos que possivelmente apareceriam quando o sistema estivesse pronto
 - Viabiliza criação de sistemas mais estáveis

Dúvidas?



Tópicos

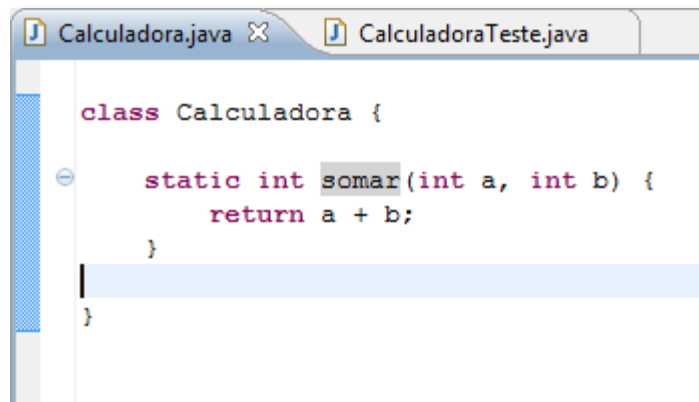
Testes de Programas

JUnit – Introdução

JUnit – Prática

JUnit no Eclipse

Exemplo 1: Calculadora



```
Calculadora.java X CalculadoraTeste.java

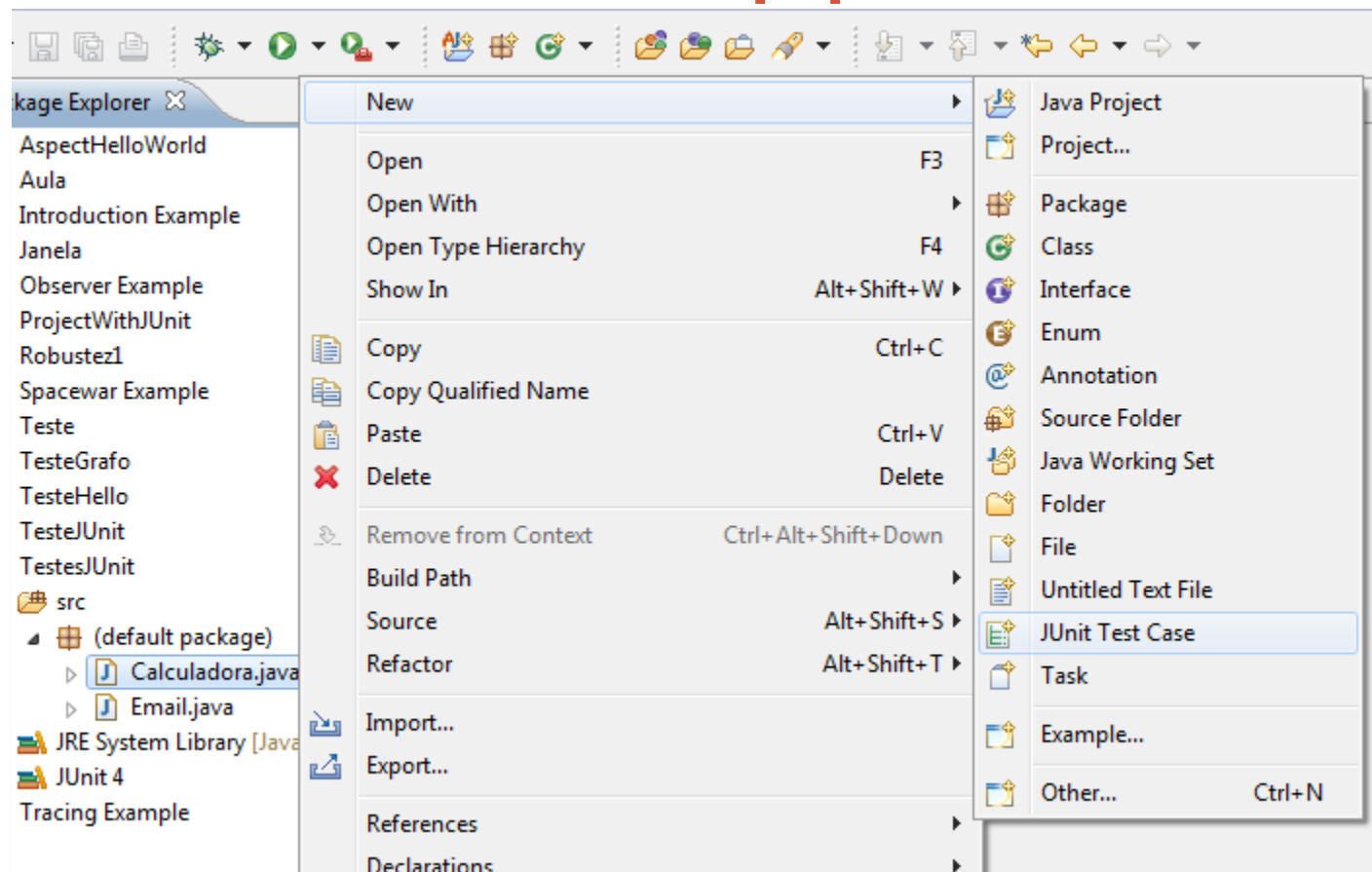
class Calculadora {

    static int somar(int a, int b) {
        return a + b;
    }
}
```

Classe a ser testada

JUnit no Eclipse

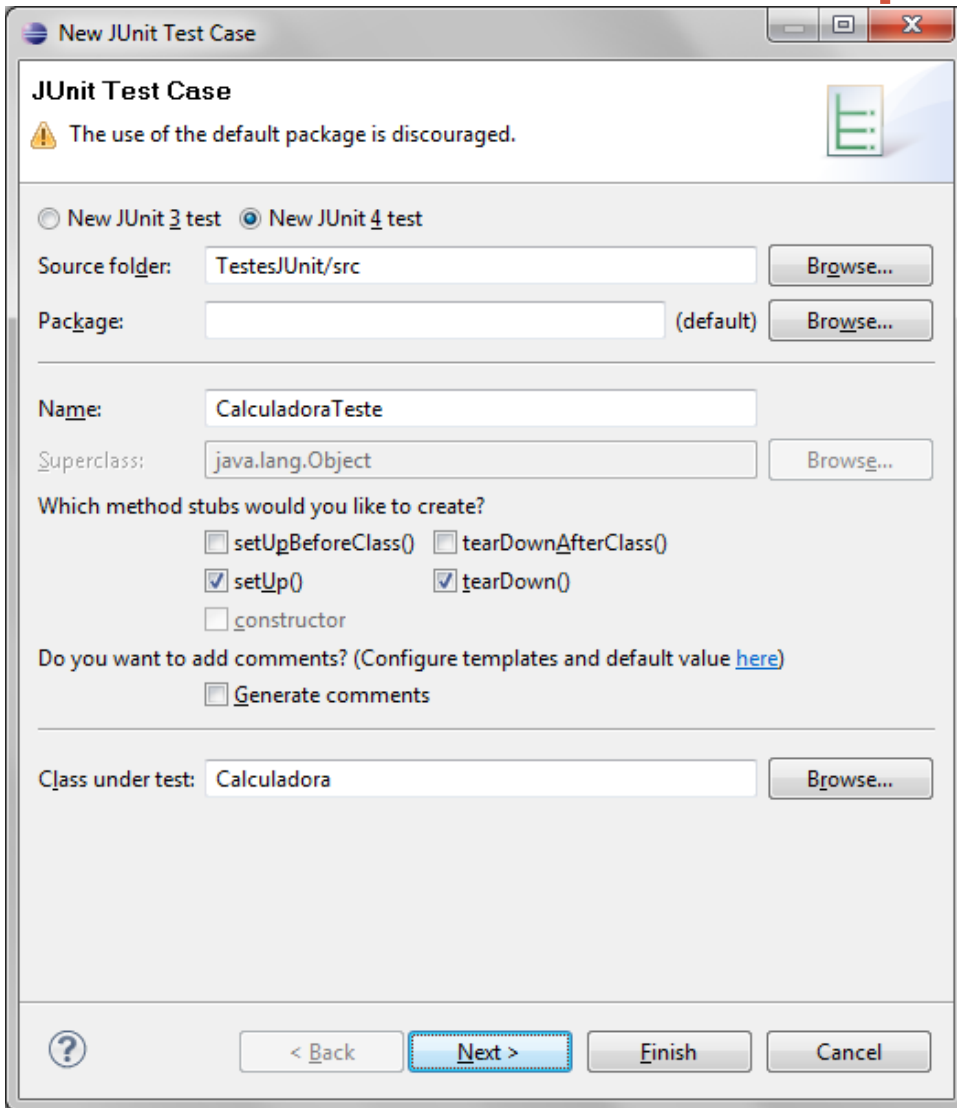
Criando um Caso de Teste [1/5]



- Botão direito na classe a ser testada (Calculadora)
- *New*
- *JUnit Test Case*

JUnit no Eclipse

Criando um Caso de Teste [2/5]



The screenshot shows the 'New JUnit Test Case' dialog box in Eclipse. The title bar says 'New JUnit Test Case'. Inside, there's a warning icon and text: 'The use of the default package is discouraged.' Below this, there are two radio buttons: 'New JUnit 3 test' and 'New JUnit 4 test', with 'New JUnit 4 test' selected. The 'Source folder' is 'TestesJUnit/src' with a 'Browse...' button. The 'Package' is '(default)' with a 'Browse...' button. The 'Name' is 'CalculadoraTeste'. The 'Superclass' is 'java.lang.Object' with a 'Browse...' button. Under 'Which method stubs would you like to create?', there are checkboxes for 'setUpBeforeClass()', 'tearDownAfterClass()', 'setUp()' (checked), 'tearDown()' (checked), and 'constructor'. Below this, it asks 'Do you want to add comments? (Configure templates and default value [here](#))' with a checkbox for 'Generate comments'. At the bottom, 'Class under test' is 'Calculadora' with a 'Browse...' button. The bottom of the dialog has a help icon, '< Back', 'Next >' (highlighted), 'Finish', and 'Cancel' buttons.

New JUnit Test Case

⚠ The use of the default package is discouraged.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder: Browse...

Package: Browse...

Name:

Superclass: Browse...

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☒ setUp() ☒ tearDown()
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

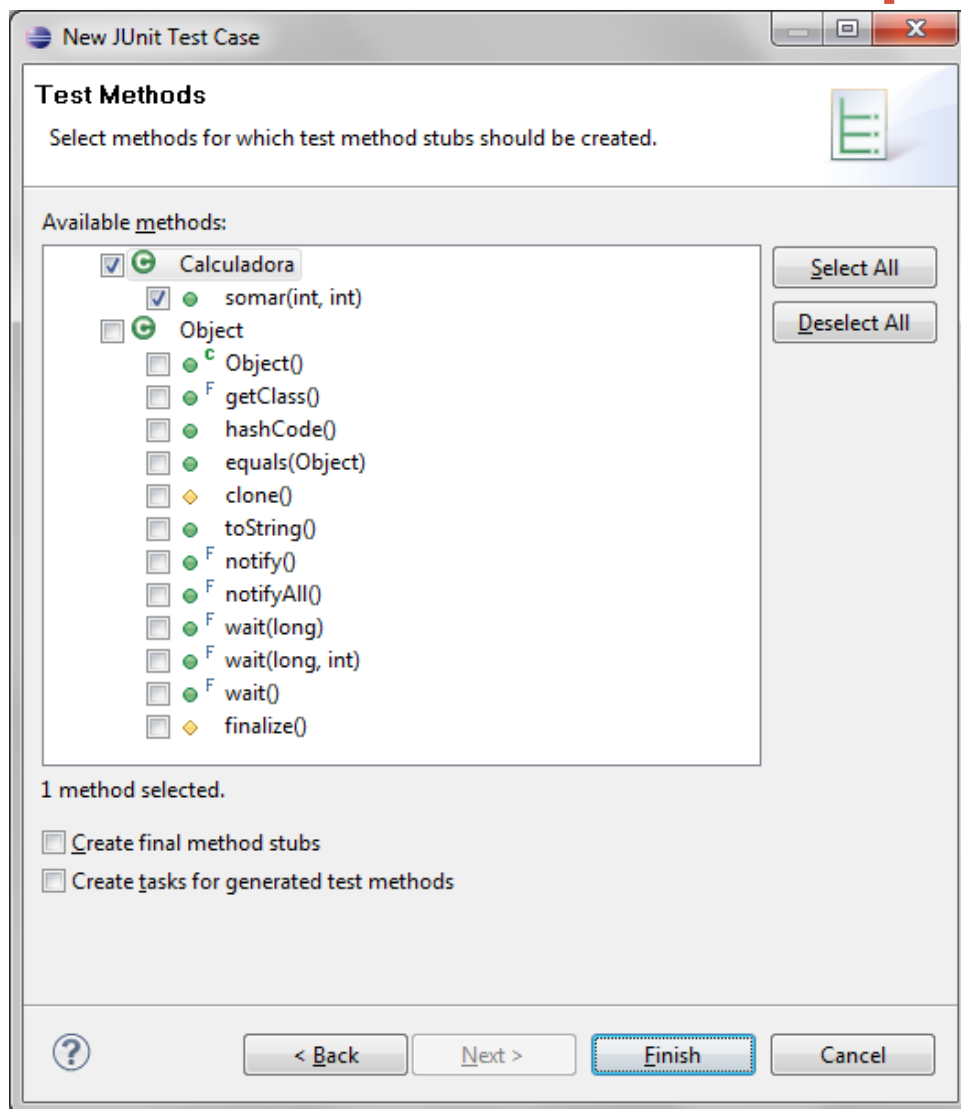
Class under test: Browse...

? < Back Next > Finish Cancel

- Nome do caso de teste
- Classe a ser testada
- *Next*

JUnit no Eclipse

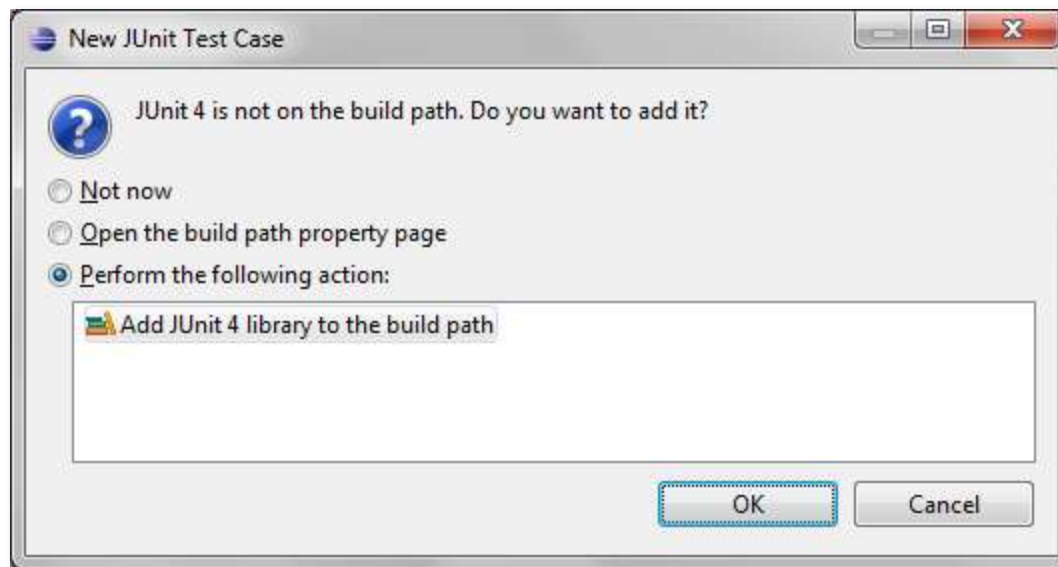
Criando um Caso de Teste [3/5]



- Selecionar métodos a serem testados
- *Finish*

JUnit no Eclipse

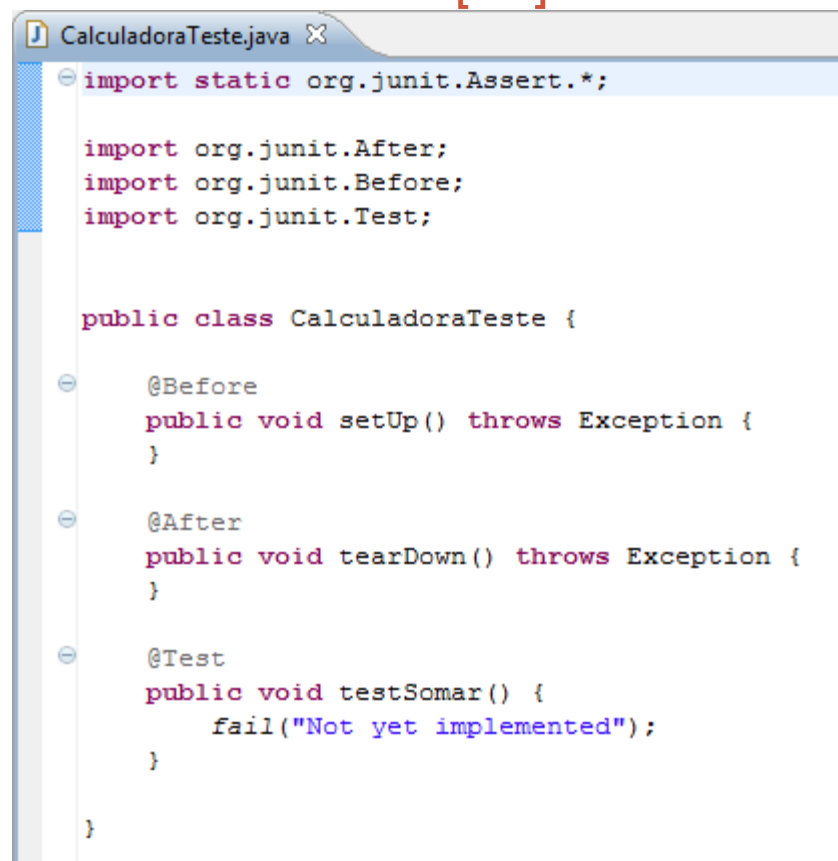
Criando um Caso de Teste [4/5]



- *Perform the following action...*
- *Ok*

JUnit no Eclipse

Criando um Caso de Teste [5/5]



```
CalculadoraTeste.java X
import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CalculadoraTeste {

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

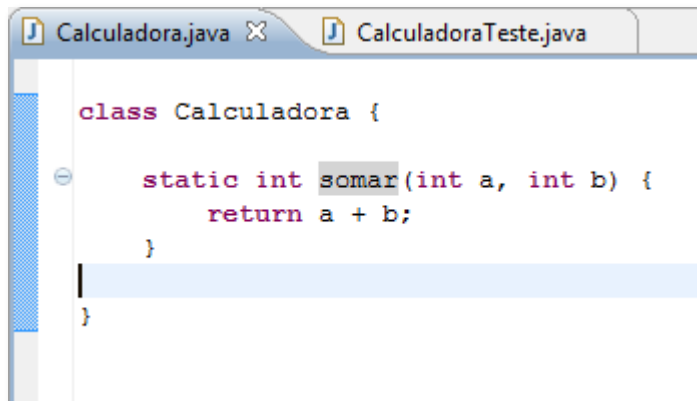
    @Test
    public void testSomar() {
        fail("Not yet implemented");
    }

}
```

- Esqueleto do caso de teste
- *testSomar()*:
 - Testes para o método somar()

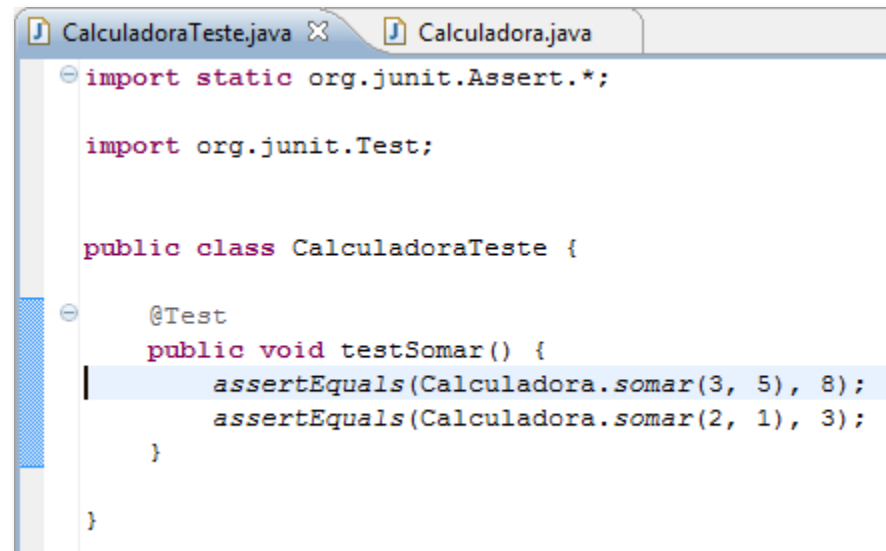
Exemplo 1: Calculadora

Implementando o Caso de Teste criado



```
class Calculadora {  
    static int somar(int a, int b) {  
        return a + b;  
    }  
}
```

Classe a ser testada



```
import static org.junit.Assert.*;  
  
import org.junit.Test;  
  
public class CalculadoraTeste {  
    @Test  
    public void testSomar() {  
        assertEquals(Calculadora.somar(3, 5), 8);  
        assertEquals(Calculadora.somar(2, 1), 3);  
    }  
}
```

Caso de teste

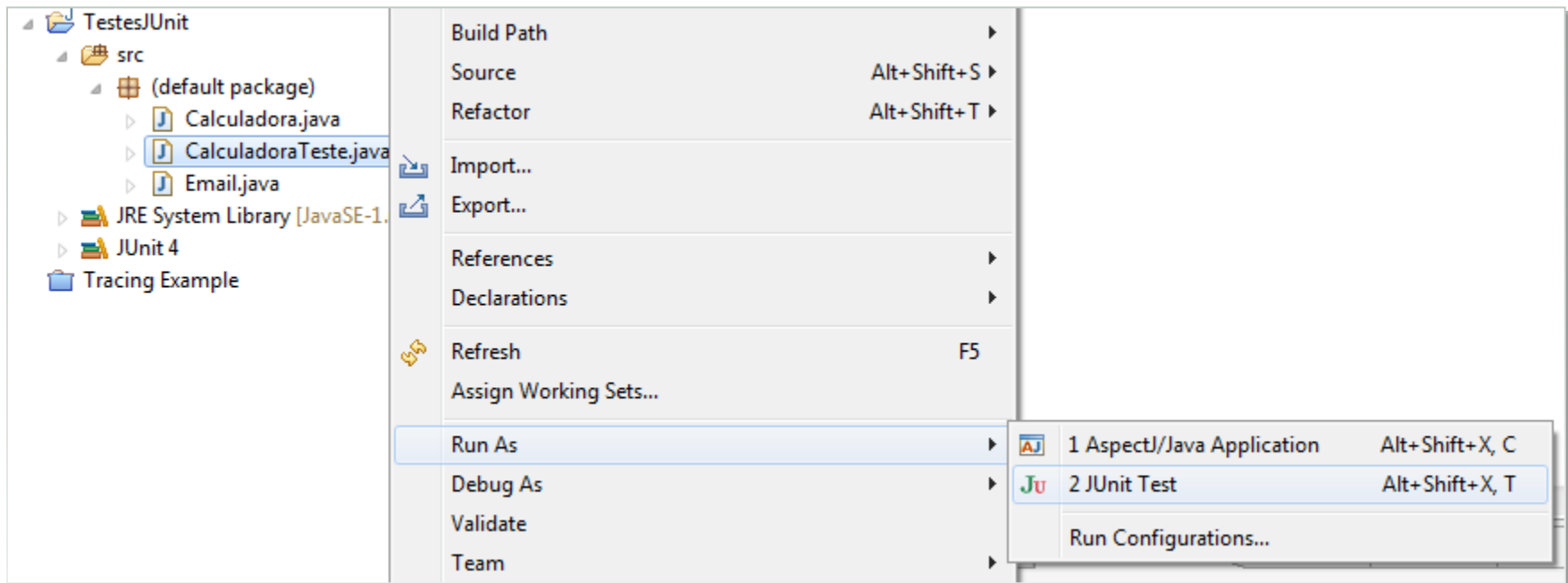
JUnit no Eclipse

Execução de um Caso de Teste

- Por plugin
 - Interface pronta
 - Recursos:
 - Visualização de falhas e erros (exceções)
 - Debug
 - Histórico de testes
- Por código
 - Desenvolvedor cria a interface
 - Método conhecido como *TestRunner*

Exemplo 1: Calculadora

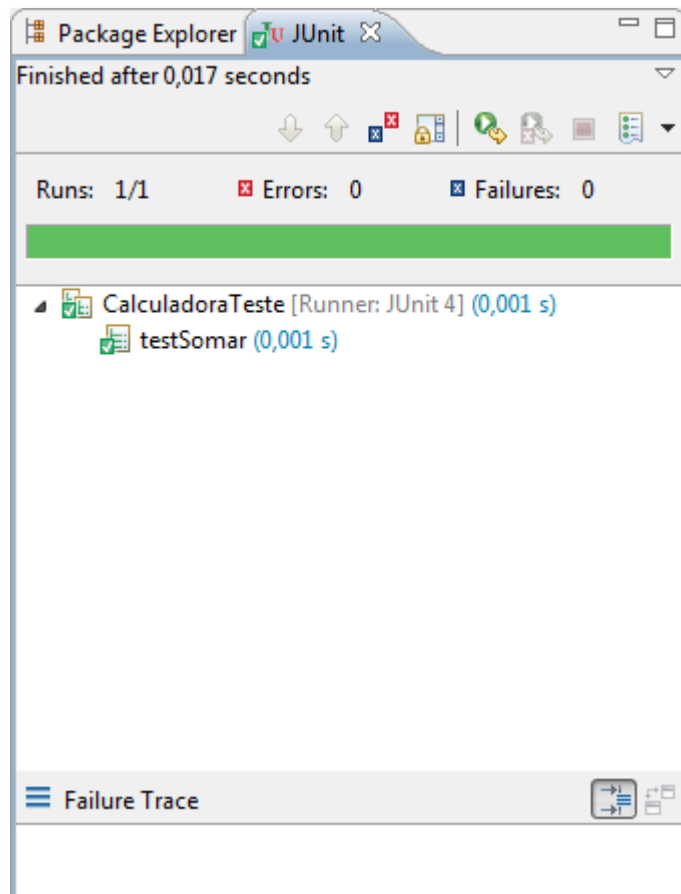
Executando o Caso de Teste - Plugin



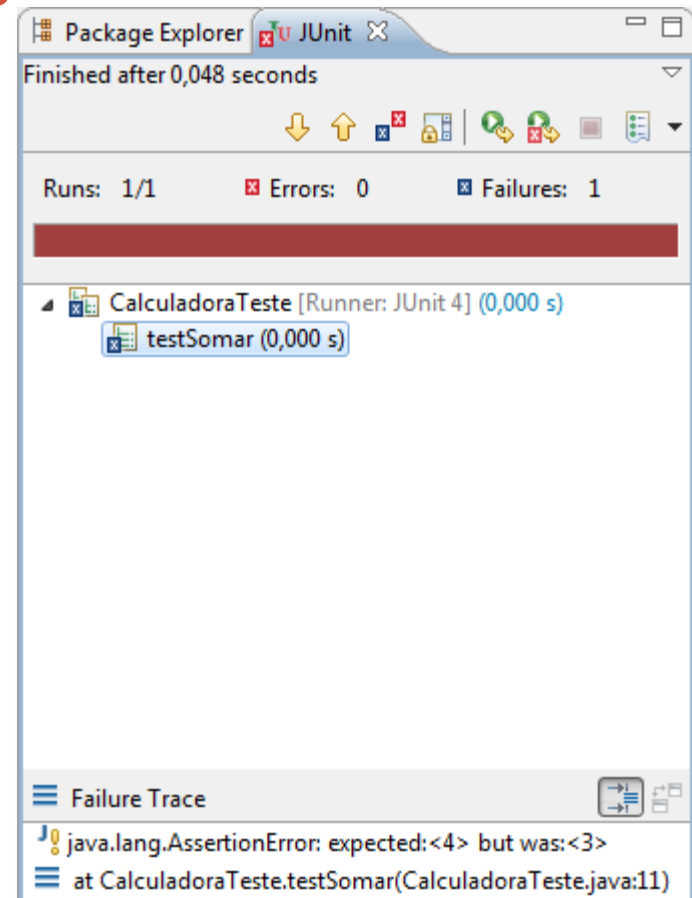
- Botão direito no caso de teste
- *Run As*
- *JUnit Test*

Exemplo 1: Calculadora

Resultado dos testes - Plugin



Resultado sem falhas



Exemplo de resultado com falhas

- **Falha**: quando uma “afirmação” (assertion) falha
- **Erro**: quando ocorre uma exceção
 - Ex.: *NullPointerException*, *ArrayIndexOutOfBoundsException* ...

Exemplo 1: Calculadora

Executando o Caso de Teste - Código

```
1 import org.junit.runner.JUnitCore;
2 import org.junit.runner.Result;
3 import org.junit.runner.notification.Failure;
4
5 public class MeuTestRunner {
6     public static void main(String[] args) {
7         // executa caso de teste e retorna dados sobre testes e falhas
8         Result resultado = JUnitCore.runClasses(CalculadoraTeste.class);
9         System.out.println("Número de falhas: " + resultado.getFailureCount());
10        for (Failure failure : resultado.getFailures()) {
11            System.out.println("Falha: " + failure.toString() + "\n" + failure.getTrace());
12        }
13    }
14 }
```

Console Problems Javadoc Declaration Progress Properties Cross Ref

<terminated> MeuTestRunner (1) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (20/09/2011 0

Número de falhas: 1
Falha: testSomar(CalculadoraTeste): expected:<3> but was:<2>
java.lang.AssertionError: expected:<3> but was:<2>
at org.junit.Assert.fail(Assert.java:91)
at org.junit.Assert.failNotEquals(Assert.java:645)
at org.junit.Assert.assertEquals(Assert.java:126)
at org.junit.Assert.assertEquals(Assert.java:470)
at org.junit.Assert.assertEquals(Assert.java:454)
at CalculadoraTeste.testSomar(CalculadoraTeste.java:11)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)

JUnit

Assertions

Sintaxe do Método	Descrição	Teste passa se
assertEquals ([String mensErro,] <i>esperado, atual</i>)	Compara dois valores	<i>esperado.equals</i> (<i>teste</i>)
assertFalse ([String mensErro,] boolean <i>condicao</i>)	Avalia uma expressão booleana	<i>condicao</i> == false
assertTrue ([String mensErro,] boolean <i>condicao</i>)		<i>condicao</i> == true
assertNotNull ([String mensErro,] Object <i>objeto</i>)	Compara um objeto com nulo	<i>objeto</i> != null
assertNull ([String mensErro,] Object <i>objeto</i>)		<i>objeto</i> == null
assertNotSame ([String mensErro,] Object <i>esperado</i> , Object <i>atual</i>)	Compara dois objetos	<i>esperado</i> != <i>atual</i>
assertSame ([String mensErro,] Object <i>esperado</i> , Object <i>atual</i>)		<i>esperado</i> == <i>atual</i>
fail ([String mensErro])	Causa uma falha no teste atual (comumente usado em manipulação de exceções)	

JUnit

Diferença entre versões [1/2]

- Até o JUnit 3.x
 - Classes de teste herdam classe *TestCase*
 - Declaração do construtor da classe
 - Métodos de teste iniciavam com a palavra test
 - testMetodo(), testGravacao(), testSoma() etc
 - Método *setUp()*
 - Método *tearDown()*

JUnit

Diferença entre versões [2/2]

- A partir do JUnit 4
 - Herança de `TestCase` substituída por import estático
 - *`import static org.junit.Assert.*`*
 - Não precisa declarar o construtor da classe
 - Métodos de teste identificados com a anotação `@Test`
 - Métodos *`setUp()`* e *`tearDown()`* substituídos pelas anotações *`@Before`* e *`@After`* respectivamente

JUnit

Anotações do JUnit 4 [1/2]

- **@Test**
 - Identifica que o método é um método de teste.
- **@Before**
 - Executa o método antes de cada teste. Este método pode ser usado para preparar o ambiente de teste (Ex: ler dados do usuário). Substituto do setUp().
- **@After**
 - Executa o método após cada teste. Substituto do tearDown().
- **@BeforeClass**
 - Executa o método antes do início de todos os testes
 - Ex: conectar base de dados

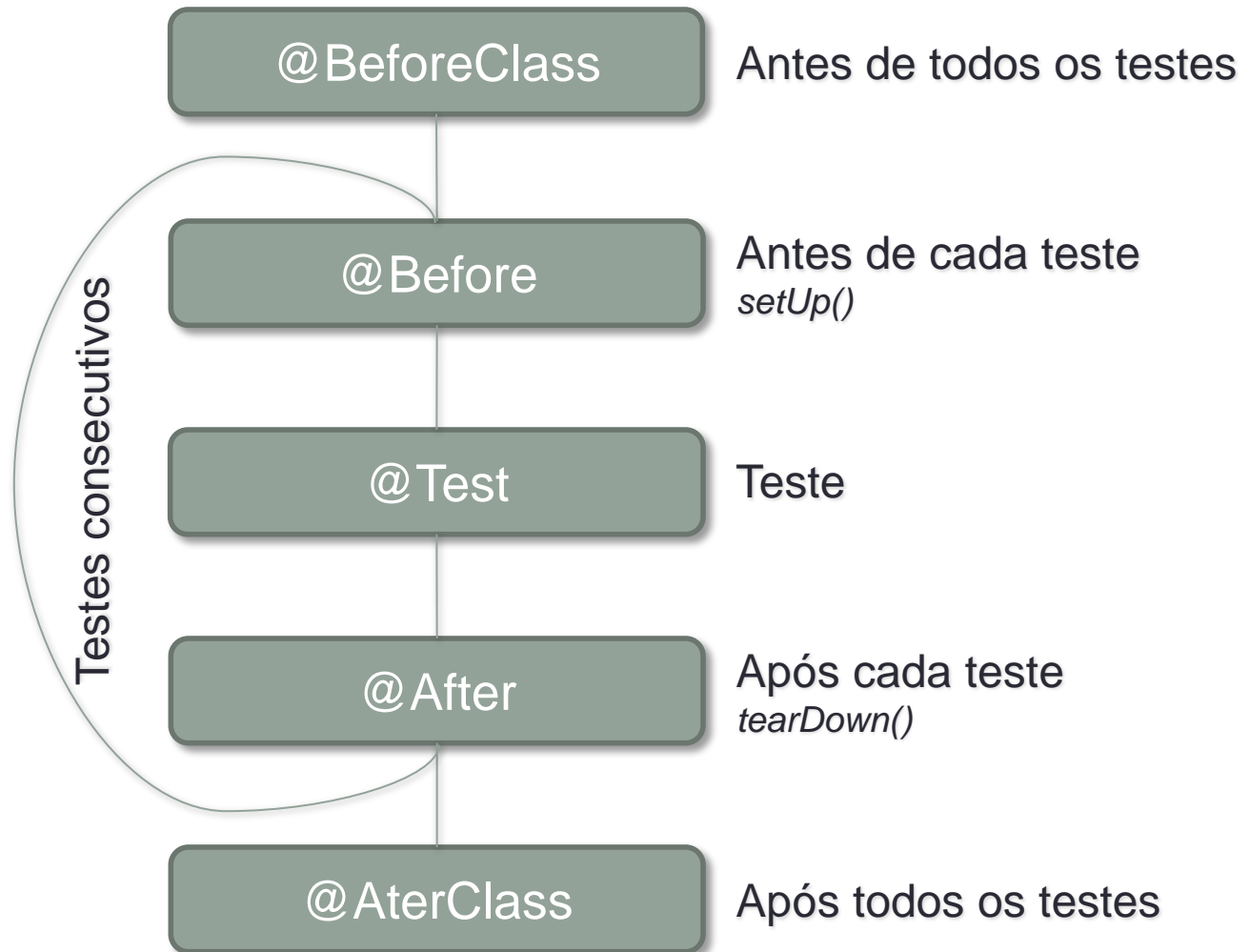
JUnit

Anotações do JUnit 4 [2/2]

- **@AfterClass**
 - Executa o método após todos os testes finalizarem (Ex: desconectar base de dados).
- **@Ignore[("Comentário")]**
 - O método é ignorado.
- **@Test (*expected=IllegalArgumentException.class*)**
 - Testa se o método levanta a exceção especificada.
- **@Test (*timeout=100*)**
 - Falha se o teste demorar mais que 100 milissegundos

JUnit

Sequência de Desenvolvimento



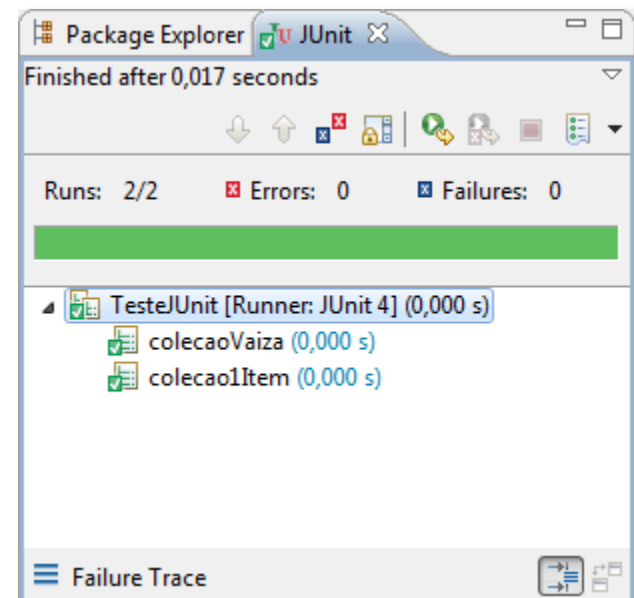
Exemplo 2

Sequência dos métodos

```

5 public class TesteJUnit {
6     private Collection colecao;
7     @BeforeClass
8     public static void inicializacaoGeral() {
9         System.out.println("@BeforeClass: inicializacaoGeral()");
10    }
11    @AfterClass
12    public static void finalizacaoGeral() {
13        System.out.println("@AfterClass: finalizacaoGeral()");
14    }
15    @Before
16    public void antes() {
17        colecao = new ArrayList();
18        System.out.println("@Before: antes()");
19    }
20    @After
21    public void apos() {
22        colecao.clear();
23        System.out.println("@After: apos()");
24    }
25    @Test
26    public void colecaoVazia() {
27        assertTrue(colecao.isEmpty());
28        System.out.println("@Test: colecaoVazia()");
29    }
30    @Test
31    public void colecao1Item() {
32        colecao.add("Primeiro");
33        assertEquals(1, colecao.size());
34        System.out.println("@Test colecao1Item");
35    }
36 }

```



The screenshot shows the Console window with the following output:

```

<terminated> TesteJUnit [JUnit] C:\Program Files\Jav
@BeforeClass: inicializacaoGeral()
@Before: antes()
@Test: colecaoVazia()
@After: apos()
@Before: antes()
@Test colecao1Item
@After: apos()
@AfterClass: finalizacaoGeral()

```

Exemplo 2

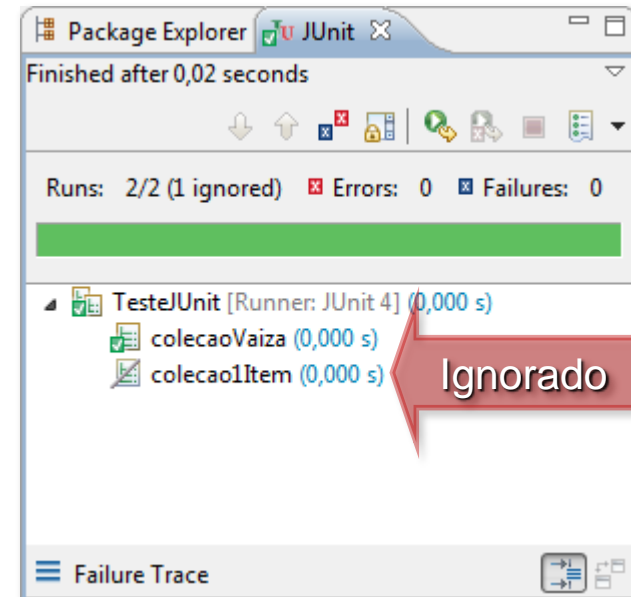
Ignorando um teste

```

5 public class TesteJUnit {
6     private Collection colecao;
7     @BeforeClass
8     public static void inicializacaoGeral() {
9         System.out.println("@BeforeClass: inicializacaoGeral()");
10    }
11    @AfterClass
12    public static void finalizacaoGeral() {
13        System.out.println("@AfterClass: finalizacaoGeral()");
14    }
15    @Before
16    public void antes() {
17        colecao = new ArrayList();
18        System.out.println("@Before: antes()");
19    }
20    @After
21    public void apos() {
22        colecao.clear();
23        System.out.println("@After: apos()");
24    }
25    @Test
26    public void colecaoVaiza() {
27        assertTrue(colecao.isEmpty());
28        System.out.println("@Test: colecaoVazia()");
29    }
30    @Ignore("Não está pronto...")
31    @Test
32    public void colecao1Item() {
33        colecao.add("Primeiro");
34        assertEquals(1, colecao.size());
35        System.out.println("@Test colecao1Item");
36    }
37 }

```

Ignorado



Console

<terminated> TesteJUnit [JUnit] C:\Program Files\Java\

```

@BeforeClass: inicializacaoGeral()
@Before: antes()
@Test: colecaoVazia()
@After: apos()
@AfterClass: finalizacaoGeral()

```

Exemplo 3: Janela

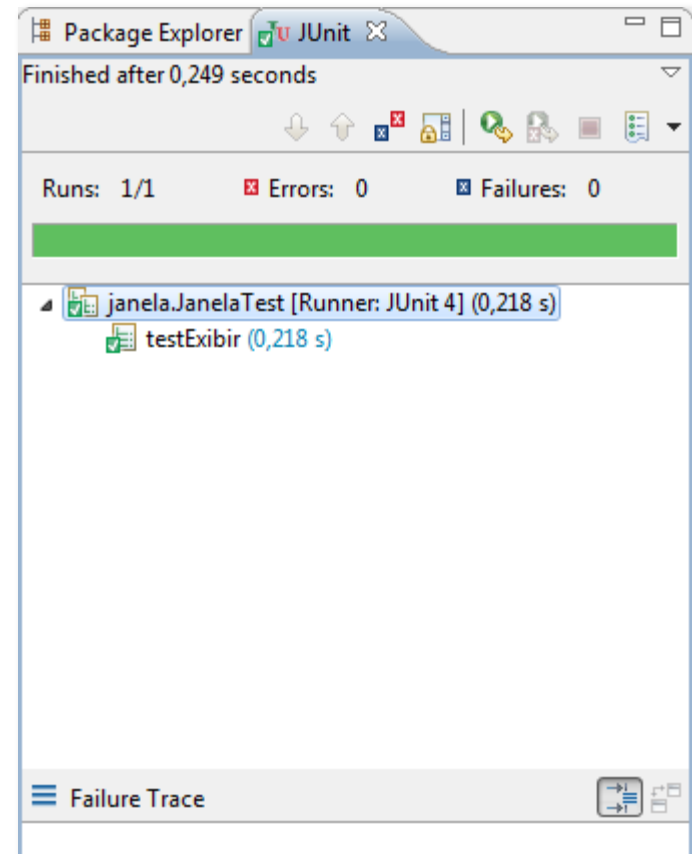
Classe Janela

```
1 //esta janela não poderia mudar de posição
2 package janela;
3
4 import java.awt.Point;
5 import javax.swing.JFrame;
6
7 public class Janela extends JFrame{
8     private static final long serialVersionUID = 1L;
9     private Point xy;
10
11     public Janela(int x, int y) {
12         super(x + ", " + y);
13         xy = new Point(x, y);
14     }
15
16     public void exibir() {
17         this.setLocation(xy);
18         this.setVisible(true);
19     }
20 }
```

Exemplo 3: Janela

Caso de Teste: JanelaTest [1/2]

```
1 package janela;  
2  
3 import static org.junit.Assert.*;  
4  
5 import org.junit.After;  
6 import org.junit.Before;  
7 import org.junit.Test;  
8  
9 public class JanelaTest {  
10     private Janela jan;  
11  
12     @Before  
13     public void setUp() throws Exception {  
14         jan = new Janela(30, 20);  
15     }  
16  
17     @After  
18     public void tearDown() throws Exception {  
19         jan = null;  
20     }  
21  
22     @Test  
23     public void testExibir() {  
24         jan.exibir();  
25         assertEquals(jan.getLocation().x, 30);  
26         assertEquals(jan.getLocation().y, 20);  
27         assertEquals(jan.getTitle(), "30,20");  
28     }  
29 }
```



- A classe *Janela* está correta?

Exemplo 3: Janela

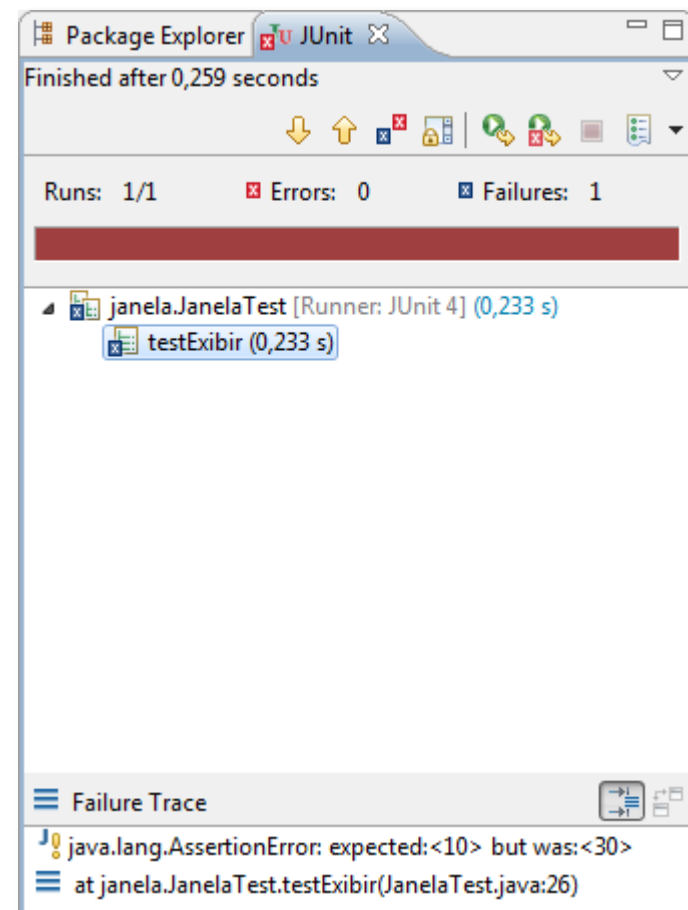
Caso de Teste: JanelaTest [2/2]

```

1 package janela;
2
3 import static org.junit.Assert.*;
4
5 import org.junit.After;
6 import org.junit.Before;
7 import org.junit.Test;
8
9 public class JanelaTest {
10     private Janela jan;
11
12     @Before
13     public void setUp() throws Exception {
14         jan = new Janela(30, 20);
15     }
16
17     @After
18     public void tearDown() throws Exception {
19         jan = null;
20     }
21
22     @Test
23     public void testExibir() {
24         jan.exibir();
25         jan.setLocation(10, 10);
26         assertEquals(jan.getLocation().x, 30);
27         assertEquals(jan.getLocation().y, 20);
28         assertEquals(jan.getTitle(), "30,20");
29     }
30 }

```

Mudar
posição



- A classe *Janela* está correta?

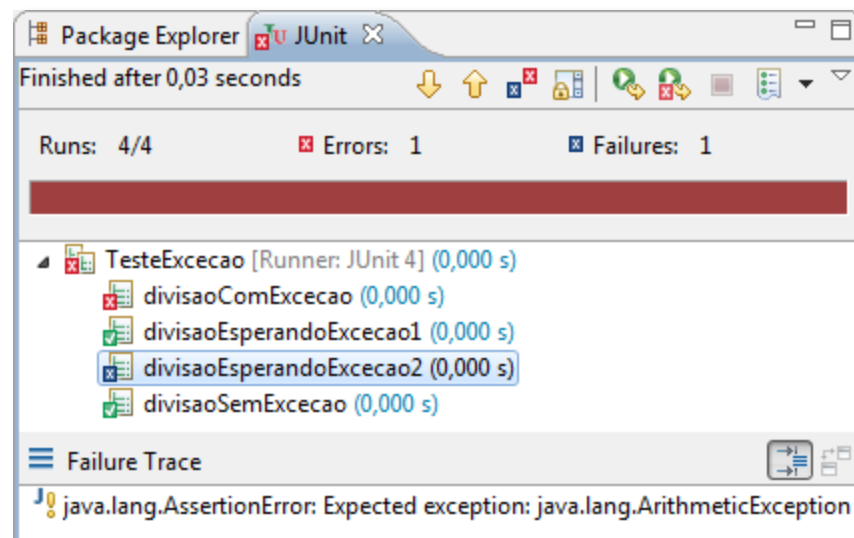
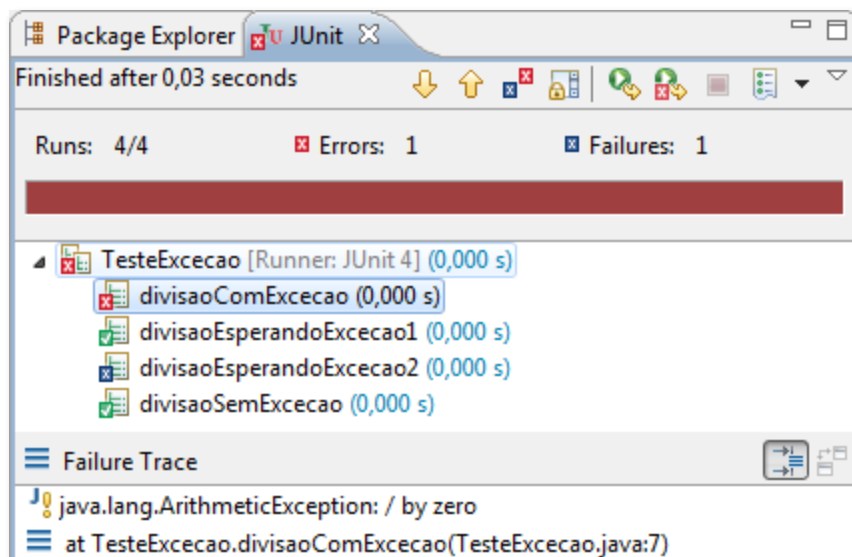
Exemplo 4: Esperando exceções

Caso de Teste

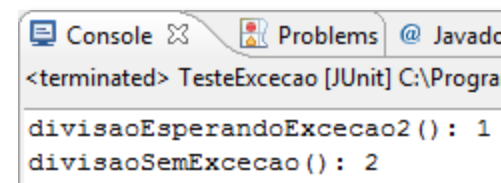
```
3 public class TesteExcecao {
4
5     @Test
6     public void divisaoComExcecao() {
7         int i = 4 / 0; //teste resulta em erro por causa da exceção
8         System.out.println("divisaoComExcecao(): " + i);
9     }
10
11     @Test(expected = ArithmeticException.class)
12     public void divisaoEsperandoExcecao1() {
13         int i = 4 / 0; //teste passa, pois espera a exceção
14         System.out.println("divisaoEsperandoExcecao1(): " + i);
15     }
16
17     @Test(expected = ArithmeticException.class)
18     public void divisaoEsperandoExcecao2() {
19         int i = 4 / 4; //teste falha, pois não ocorre a exceção esperada
20         System.out.println("divisaoEsperandoExcecao2(): " + i);
21     }
22
23     @Test
24     public void divisaoSemExcecao() {
25         int i = 4 / 2;
26         System.out.println("divisaoSemExcecao(): " + i);
27     }
28 }
```

Exemplo 4: Esperando exceções

Resultado

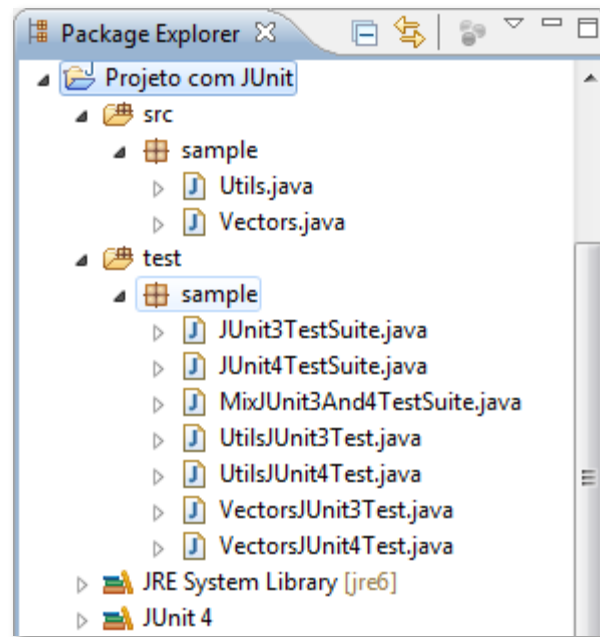


- ***divisaoComExcecao()***: erro
 - Exceção ocorrida (divisão por zero)
- ***divisaoEsperandoExcecao1()***: passou
 - Ocorreu exceção esperada
- ***divisaoEsperandoExcecao2()***: falhou
 - Não ocorreu exceção esperada
- ***divisaoSemExcecao()***: passou



Dicas

Organização



- Organize os casos de teste em uma pasta separada dos códigos fonte do sistema
 - Organize os casos de teste dentro dos mesmos pacotes que os códigos fonte

Dicas

Desenvolvimento Dirigido a Testes

- Testes devem ser
 - escritos assim que possível
 - adaptados de acordo com mudanças
- Testes antigos:
 - Deixar em execução
- Surgimento de novas ideias:
 1. Criar testes
 2. Verificar se funcionam
 3. Se necessário, altere o código do programa

Dicas

Granularidade dos Testes

- Cada teste deve verificar uma porção específica de uma funcionalidade do sistema
- Não combine testes não relacionados em um único método de teste

Dicas

Quais e quantos testes escrever [1/2]

- Regra principal:
 - Tenha criatividade para imaginar as possibilidades de testes
- Comece pelo teste mais simples
 - deixe os mais complexos para o final
- Use apenas dados suficientes
 - Ex.: não teste 10 condições se apenas 3 são suficientes

Dicas

Quais e quantos testes escrever [2/2]

- Não teste métodos triviais
 - *gets* e *sets*
 - Métodos *set*: Somente crie testes se houver validação de dados
- Achou um bug?
 - Não conserte sem antes escrever um teste que o detecte
 - Caso contrário, ele pode voltar

Exercício

Criatividade na criação de testes

MovimentoFinanceiro
<ul style="list-style-type: none">-ID: int-TipoES: char-DataEmissao: Date-Cliente: int-DataVencimento: Date-DataPagamento: Date-ValorOriginal: float-ValorJuros: float-ValorPagamento: float
<ul style="list-style-type: none">+geralD(valorIncremento: int): int+geraVencimento(dataEmissao: Date): Date+calculaJuros(valorOriginal: float): float+calculaValorPagamento(valorOriginal: float, valorJuros: float): float

**Exercício**

Imagine e crie testes para esta classe de movimentação financeira

Exemplo 5: Memória

Classe MemoriaS_[1/3]

```

3 public abstract class MemoriaS {
4     public static final int BYTE=1; //define unidade como BYTE
5     public static final int KB=2;   //define unidade como KB
6     public static final int MB=3;   //define unidade como MB
7     public static final int GB=4;   //define unidade como GB
8     protected double total;
9     protected double utilizadoKB;
10    protected int unidade;
11
12    //Construtor. Recebe como parâmetro o total a ser gravado e a unidade de dados
13    public MemoriaS(int newTotal, int newUnidade){
14        this.total=newTotal;
15        this.unidade=newUnidade;
16        this.utilizadoKB=0;
17    }
18
19    //Construtor. Recebe como parâmetro o total a ser gravado e define a unidade como KB
20    public MemoriaS(int newTotal){
21        this(newTotal, KB);
22    }
23
24    //Busca valor gravado
25    public double getUtilizadoKB(){
26        return this.utilizadoKB;
27    }
28
29    //Retorna perda na gravação
30    public abstract double getPerda();

```

Exemplo 5: Memória

Classe MemoriaS_[2/3]

```
32 //Retorna espaço disponível real em KB
33 public abstract double getEspacoDisponivelRealKB();
34
35 //Retorna espaço disponível em KB
36 public double getEspacoDisponivelKB() {
37     return this.getConverteKB(this.total);
38 }
39
40 //Utilizado na gravação
41 public boolean GravaKB(int newTamanho) {
42     if (this.getConverteKB(this.total) - this.utilizadoKB >= newTamanho) {
43         this.utilizadoKB = this.utilizadoKB + newTamanho;
44         return true;
45     }
46     return false;
47 }
48
49 //Converte unidade para KB
50 public double getConverteKB(double valor) {
51     if (this.unidade == BYTE) {
52         return valor / 1024;
53     } else if (this.unidade == MB) {
54         return valor * 1024;
55     } else if (this.unidade == GB) {
56         return valor * 1024 * 1024;
57     }
58     else return valor;
59 }
```

Exemplo 5: Memoria

Classe MemoriaS_[3/3]

```

61 //Busca unidade
62 public String getUnidade() {
63     if (this.unidade == BYTE) {
64         return "BYTE";
65     } else if (this.unidade == KB) {
66         return "KB";
67     } else if (this.unidade == MB) {
68         return "MB";
69     } else return "GB";
70 }
71
72 //Busca percentual disponível
73 public double getPercentualDisponivel() {
74     return (this.getConverteKB(this.total) - this.utilizadoKB)
75         / this.getConverteKB(this.total) * 100;
76 }
77
78 public String toString() {
79     return "Percentual Disponivel: " + getPercentualDisponivel() +
80         "%\nEspaço Total: " + getEspacoDisponivelKB() + "KB\nEspaço Disponivel Real: " +
81         getEspacoDisponivelRealKB() + "KB\nPerda: " + getPerda() + "KB";
82 }
83
84 public static void main(String args[]) {
85     MemoriaS hd = new HD("46327", 10, MemoriaS.MB);
86     MemoriaS cd = new CD(650, MemoriaS.MB);
87     hd.GravaKB(10242);
88     cd.GravaKB(665602);
89     System.out.println(hd);
90     System.out.println(cd);
91 }
92 }

```

Exemplo 5: Memória

Classe CD [1/2]

```

3 public class CD extends MemoriaS {
4     public static final int ABERTO =1; //define o estado como aberto
5     public static final int FECHADO=2; //define o estado como fechado
6     protected int estado;
7
8     //Construtor. Recebe o total a ser gravado e a unidade
9     public CD(int newTotal, int newUnidade){
10         super(newTotal, newUnidade);
11         this.estado = ABERTO;
12     }
13
14     //Reescreve método abstrato. Retorna espaço real disponível em KB
15     public double getEspacoDisponivelRealKB() {
16         return this.getEspacoDisponivelKB()-super.getUtilizadoKB();
17     }
18
19     //Reescreve método abstrato. Retorna perda na gravação
20     public double getPerda() {
21         return this.getEspacoDisponivelKB()*0.02;
22     }
23
24     //Reescreve método abstrato. Utilizado para gravar, recebendo o tamanho
25     public boolean GravaKB(int newTamanho) {
26         if (this.estado == ABERTO) {
27             if (super.GravaKB(newTamanho)) {
28                 this.estado = 2;
29                 return true;
30             } else return false;
31         }
32         else return false;
33     }

```

Exemplo 5: Memoria

Classe CD [2/2]

```
35 //Busca estado (aberto ou fechado)
36 public String getEstado(){
37     if (this.estado == ABERTO)
38         return "ABERTO";
39     else
40         return "FECHADO";
41 }
42
43 public String toString(){
44     return "CD Estado: " + this.getEstado() + "\n" + super.toString();
45 }
46 }
```

Exemplo 5: Memória

Classe HD

```
3 public class HD extends MemoriaS {
4     protected String numeroSerie;
5
6     //Construtor. Recebe o numero da série, o tamanho do HD e a unidade
7     public HD(String newNumeroSerie, int newTotal, int newUnidade){
8         super(newTotal, newUnidade);
9         this.numeroSerie=newNumeroSerie;
10    }
11
12    //Reescreve método abstrato. Retorna o espaço real disponível em KB
13    public double getEspacoDisponivelRealKB() {
14        return this.getEspacoDisponivelKB() - super.getUtilizadoKB();
15    }
16
17    //Reescreve método abstrato. Retorna a perda na gravação
18    public double getPerda() {
19        return this.getEspacoDisponivelKB() / 1024 / 100;
20    }
21
22    //Busca o número de série do HD
23    public String getNumeroSerie() {
24        return this.numeroSerie;
25    }
26
27    public String toString(){
28        return "HD Numero de Serie: " + this.getNumeroSerie() + "\n" + super.toString();
29    }
30
31 }
```

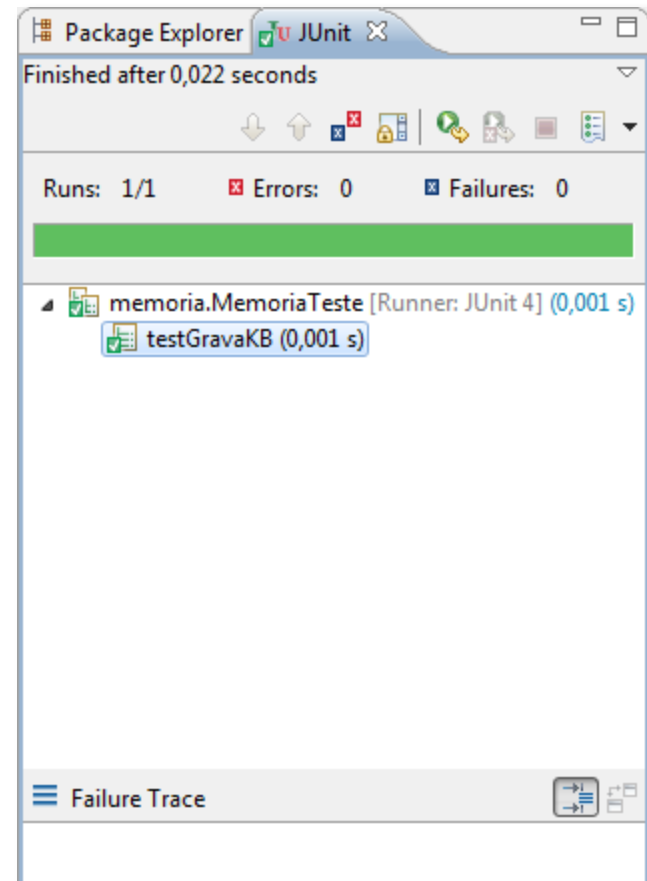
Exemplo 5: Memoria

Caso de Teste MemoriaTeste

```

3 import static org.junit.Assert.*;
4 import org.junit.After;
5 import org.junit.Before;
6 import org.junit.Test;
7
8 public class MemoriaTeste {
9     CD cd;
10    HD hd;
11
12    @Before
13    public void setUp() throws Exception {
14        cd = new CD(10000, 2);
15        hd = new HD("01", 1000, 2);
16    }
17
18    @Test
19    public void testGravaKB() {
20        assertEquals(cd.GravaKB(10000), true);
21        assertEquals(cd.GravaKB(10001), false);
22
23        assertEquals(hd.GravaKB(1000), true);
24        assertEquals(hd.GravaKB(1001), false);
25    }
26
27    @After
28    public void tearDown() throws Exception {
29        cd = null;
30        hd = null;
31    }
32 }

```



JUnit

Suítes de Testes

- Crescimento do número de testes de unidade:
 - Necessário uma suíte de testes
 - Gerenciamento de uma coleção de testes
- Conjunto de testes
 - Executa uma coleção de Casos de Teste

Suíte de Testes

Exemplo 6: Classe Utils

```
3 import java.math.BigInteger;
4
5 public class Utils {
6
7     public static String concatenaPalavras(String... palavras) {
8         StringBuilder buf = new StringBuilder();
9         for (String palavra : palavras)
10             buf.append(palavra);
11         return buf.toString();
12     }
13
14     public static String calculaFatorial(int numero) throws IllegalArgumentException {
15         if (numero < 1)
16             throw new IllegalArgumentException("parâmetro zero or negativo (" + numero + ')');
17         BigInteger fatorial = new BigInteger("1");
18         for (int i = 2; i <= numero; i++)
19             fatorial = fatorial.multiply(new BigInteger(String.valueOf(i)));
20         return fatorial.toString();
21     }
22 }
```

Suíte de Testes

Exemplo 6: Caso de Teste TesteUtils

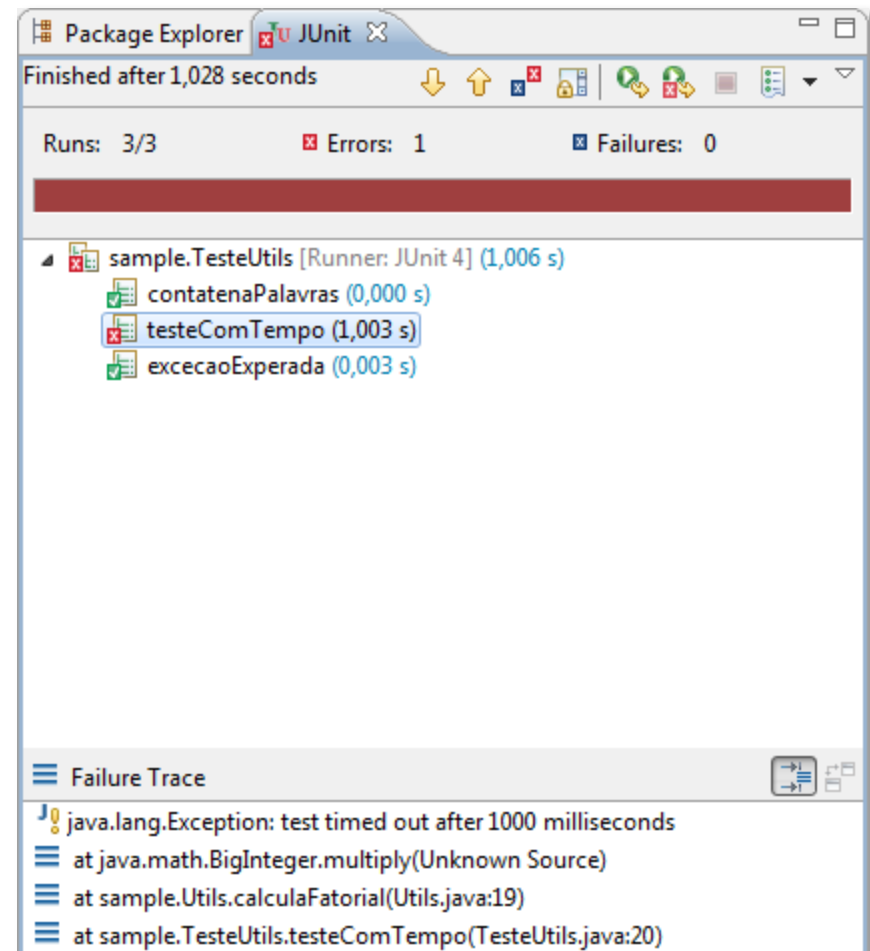
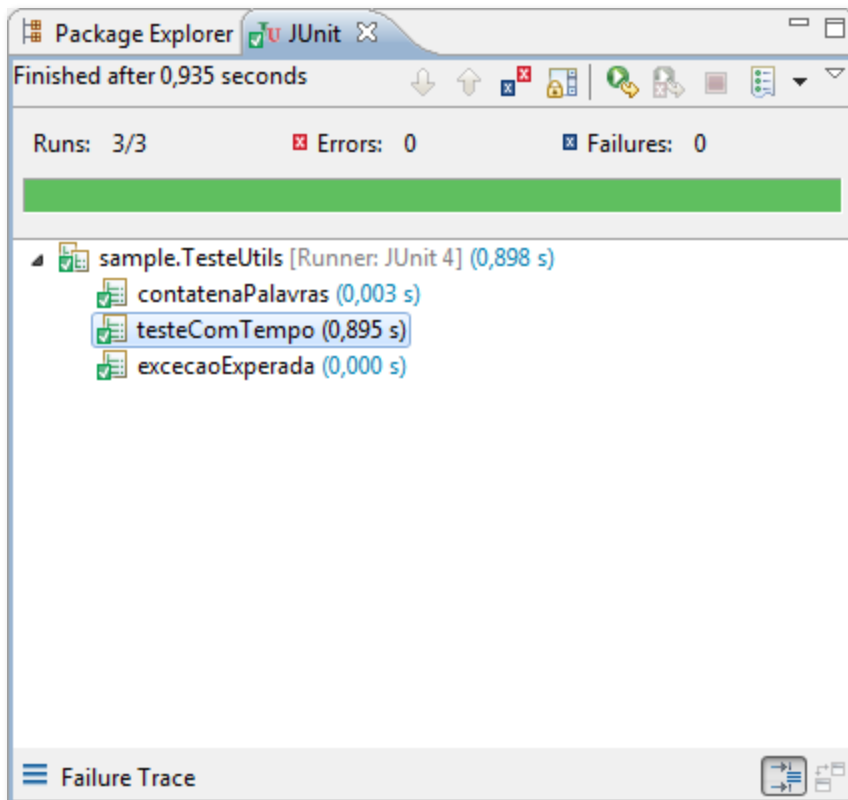
```

3 import junit.framework.JUnit4TestAdapter;
4 import org.junit.Test;
5 import static org.junit.Assert.*;
6
7 public class TesteUtils {
8
9     public static junit.framework.Test suite() {
10         return new JUnit4TestAdapter(TesteUtils.class);
11     } //Necessário para criação de suítes via wizzard - bug do JUnit 4
12
13     @Test
14     public void concatenaPalavras() {
15         System.out.println("*** concatenaPalavras() ***");
16         assertEquals("Hello, world!", Utils.concatenaPalavras("Hello", " ", " ", "world", "!"));
17     }
18
19     @Test(timeout = 1000)
20     //Se o teste não terminar em 1 segundo, ele é interrompido e falha
21     public void testeComTempo() {
22         System.out.println("*** testeComTempo() ***");
23         final int fatorialDe = 1 + (int) (20000 * Math.random()); // sorteio entre 1 e 20000
24         System.out.println("Computando fatorial de " + fatorialDe);
25         System.out.println(fatorialDe + "! = " + Utils.calculaFatorial(fatorialDe));
26     }
27
28     @Test(expected = IllegalArgumentException.class)
29     public void excecaoEsperada() {
30         System.out.println("*** excecaoEsperada() ***");
31         final int fatorialDe = -5; //número inválido para calculaFatorial()
32         System.out.println(fatorialDe + "! = " + Utils.calculaFatorial(fatorialDe));
33     }
34 }

```

Suíte de Testes

Exemplo 6: Resultados de TesteUtils



Suíte de Testes

Exemplo 6: Classe Vetores

```
3 public final class Vetores {
4
5     //verifica se 2 vetores são iguais
6     public static boolean vetoresIguais(int[] a, int[] b) {
7         if ((a == null) || (b == null))
8             throw new IllegalArgumentException("argumento nulo");
9         if (a.length != b.length)
10            return false;
11        for (int i = 0; i < a.length; i++)
12            if (a[i] != b[i])
13                return false;
14        return true;
15    }
16
17    //produto interno de 2 vetores
18    public static int produtoInternoVetorial(int[] a, int[] b) {
19        if ((a == null) || (b == null))
20            throw new IllegalArgumentException("argumento nulo");
21        if (a.length != b.length)
22            throw new IllegalArgumentException(
23                "vetores de tamanhos diferentes ("
24                + a.length + ", " + b.length + ')');
25        int sum = 0;
26        for (int i = 0; i < a.length; i++)
27            sum += a[i] * b[i];
28        return sum;
29    }
30 }
```

Suíte de Testes

Exemplo 6: Caso de Teste TesteVetores

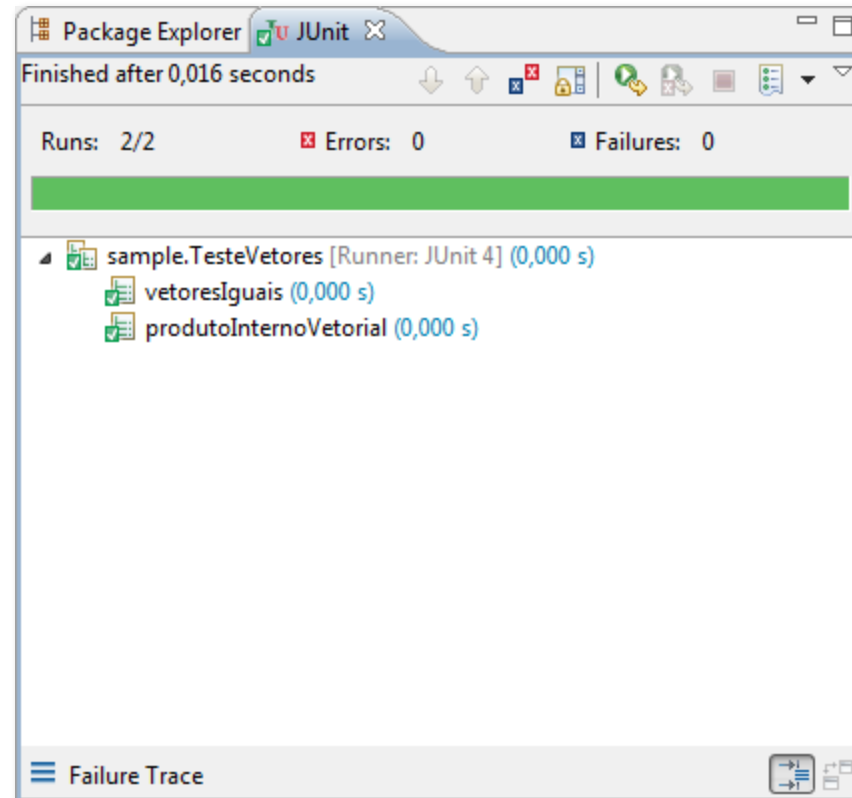
```

3 import junit.framework.JUnit4TestAdapter;
4 import org.junit.Test;
5 import static org.junit.Assert.*;
6
7 public class TesteVetores {
8
9     public static junit.framework.Test suite() {
10         return new JUnit4TestAdapter(TesteVetores.class);
11     } //Necessário para criação de suítes via wizzard - bug do JUnit 4
12
13     @Test
14     public void vetoresIguais() {
15         System.out.println("*** vetoresIguais() ***");
16         assertTrue(Vetores.vetoresIguais(new int[] {}, new int[] {}));
17         assertTrue(Vetores.vetoresIguais(new int[] {0}, new int[] {0}));
18         assertTrue(Vetores.vetoresIguais(new int[] {5, 6, 7}, new int[] {5, 6, 7}));
19
20         assertFalse(Vetores.vetoresIguais(new int[] {}, new int[] {0}));
21         assertFalse(Vetores.vetoresIguais(new int[] {0}, new int[] {0, 0}));
22         assertFalse(Vetores.vetoresIguais(new int[] {0}, new int[] {}));
23
24         assertFalse(Vetores.vetoresIguais(new int[] {0, 0, 0}, new int[] {0, 0, 1}));
25         assertFalse(Vetores.vetoresIguais(new int[] {0, 0, 0}, new int[] {0, 1, 0}));
26         assertFalse(Vetores.vetoresIguais(new int[] {0, 0, 0}, new int[] {1, 0, 0}));
27     }
28
29     @Test
30     public void produtoInternoVetorial() {
31         System.out.println("*** produtoInternoVetorial() ***");
32         assertEquals(0, Vetores.produtoInternoVetorial(new int[] {0, 0}, new int[] {0, 0}));
33         assertEquals(39, Vetores.produtoInternoVetorial(new int[] {3, 4}, new int[] {5, 6}));
34         assertEquals(-39, Vetores.produtoInternoVetorial(new int[] {-3, 4}, new int[] {5, -6}));
35     }
36 }

```

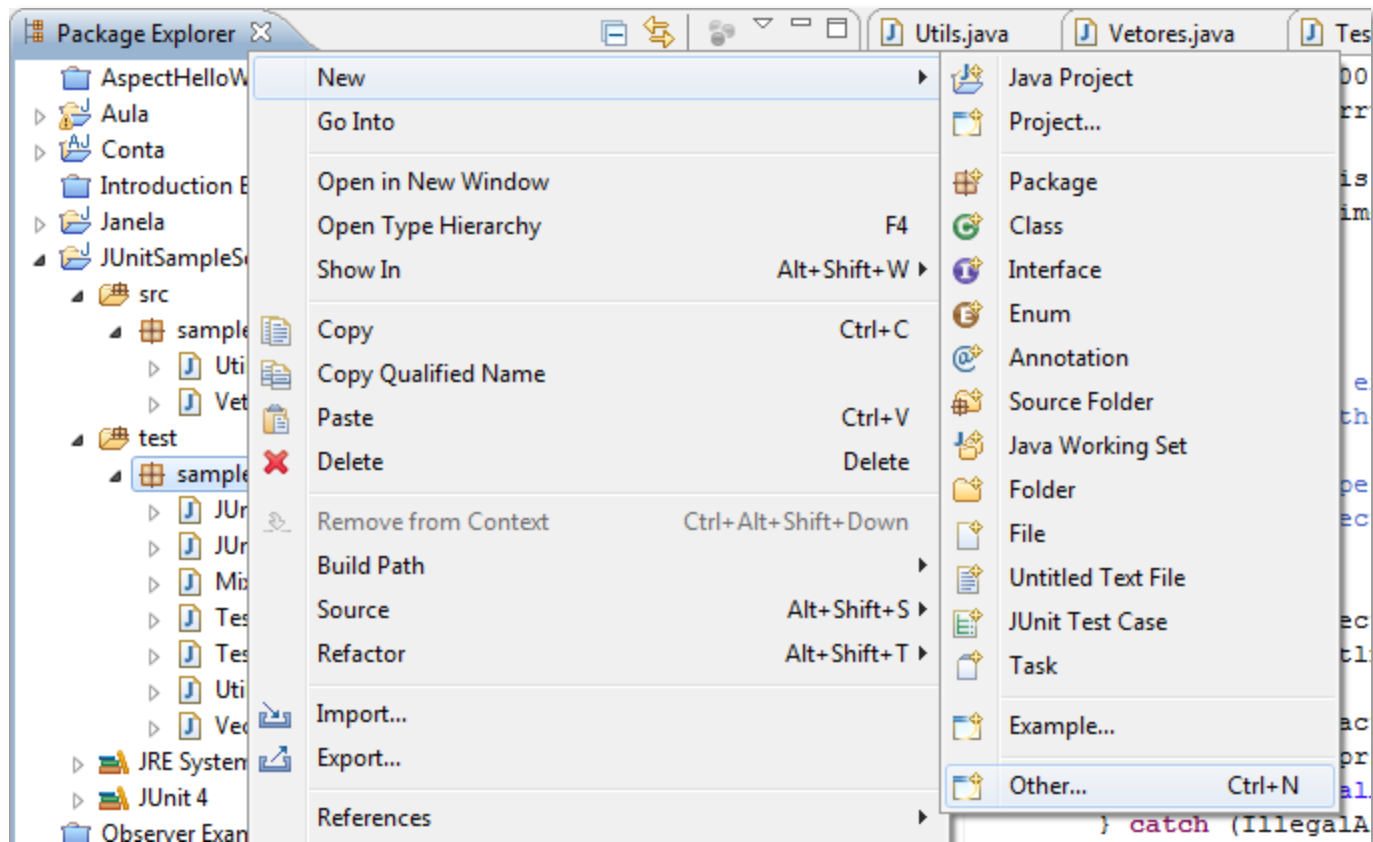
Suíte de Testes

Exemplo 6: Resultado de TesteVetores



Suíte de Testes

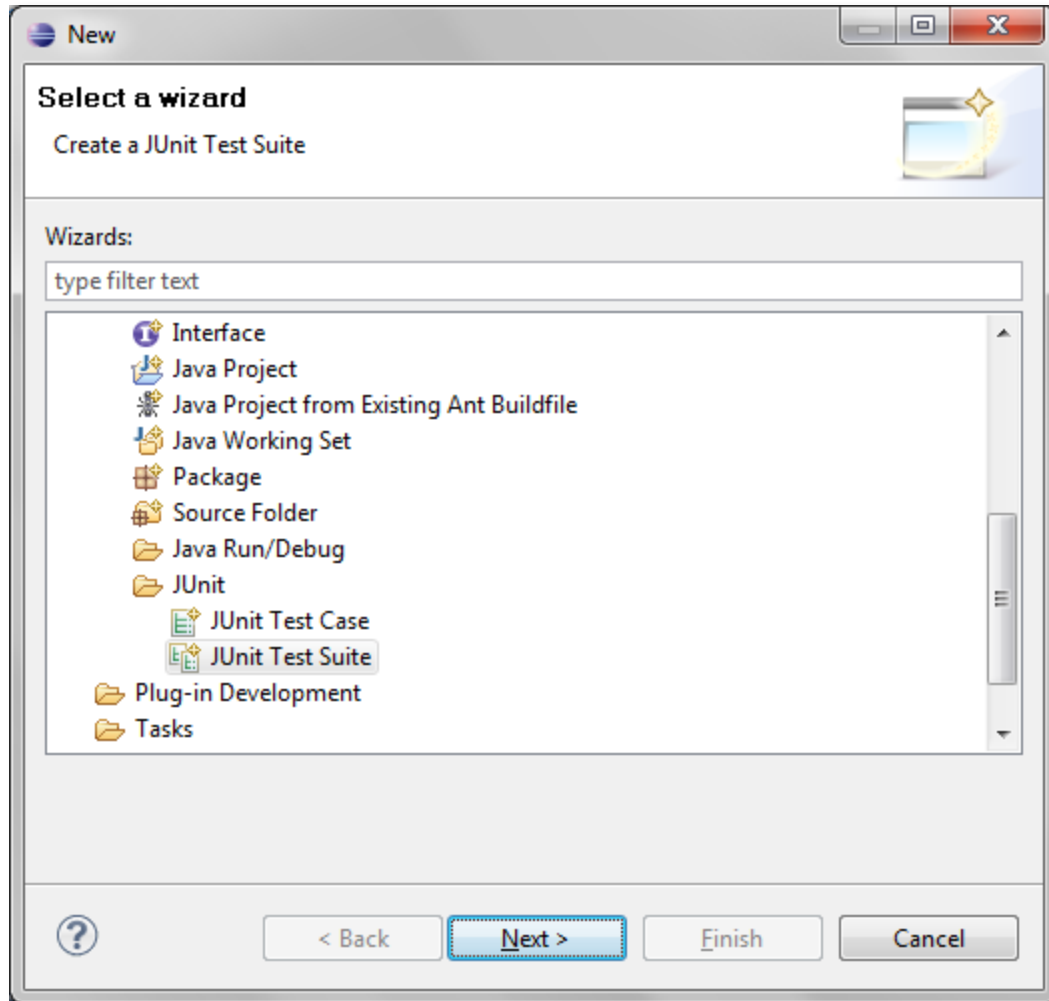
Criando a Suíte



- Botão no pacote dos testes

Suíte de Testes

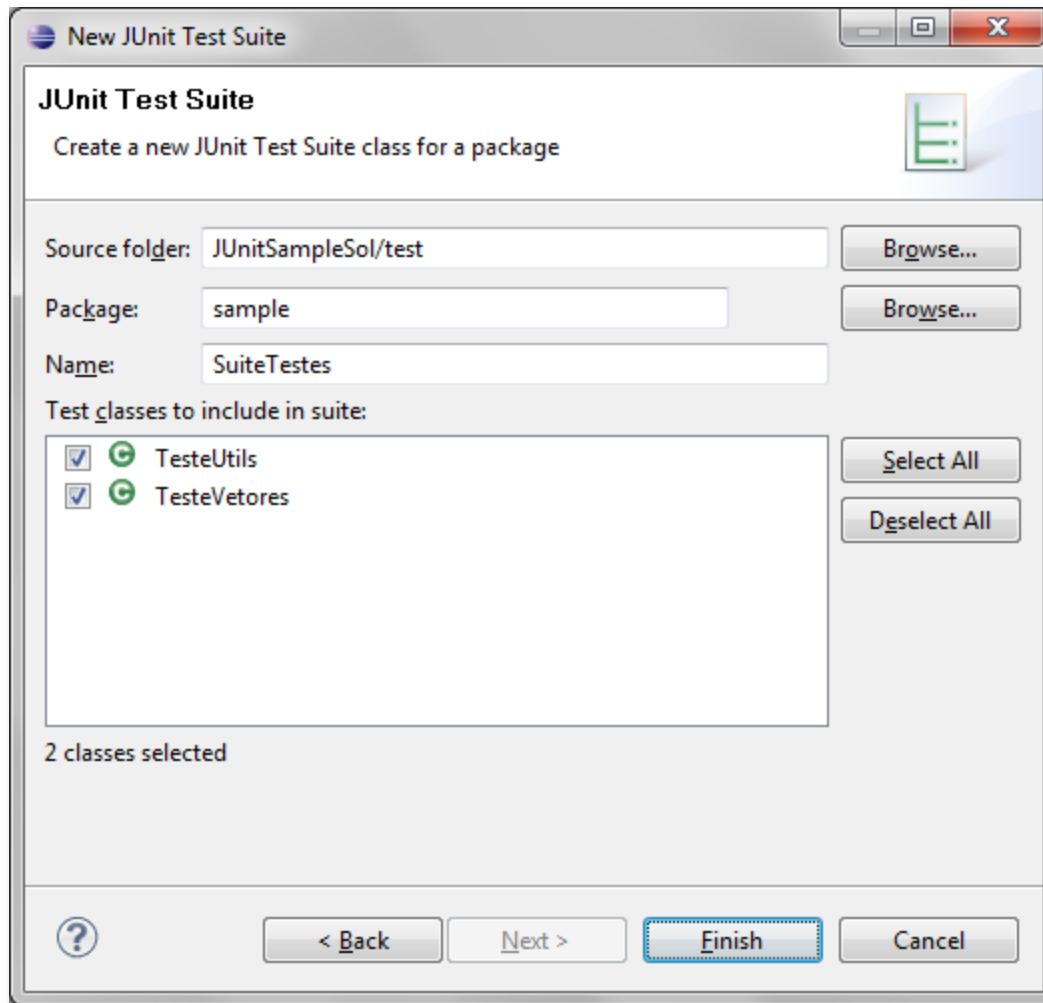
Criando a Suíte



- *JUnit Test Suite*
- *Next*

Suíte de Testes

Criando a Suíte



- Definir nome da suíte
- Selecionar casos de teste
- *Finish*

Suíte de Testes

Exemplo 6: Criando a Suíte

```
1 package sample;
2
3 import junit.framework.Test;
4 import junit.framework.TestSuite;
5
6 public class SuiteTestes {
7
8     public static Test suite() {
9         TestSuite suite = new TestSuite(SuiteTestes.class.getName());
10        //$JUnit-BEGIN$
11        suite.addTest(TesteUtils.suite());
12        suite.addTest(TesteVetores.suite());
13        //$JUnit-END$
14        return suite;
15    }
16
17 }
```

- Suíte de Testes criada, padrão JUnit 3
- Execute-a como um caso de teste

Suíte de Testes

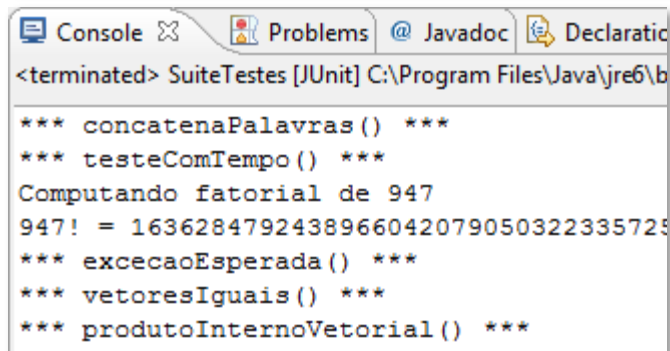
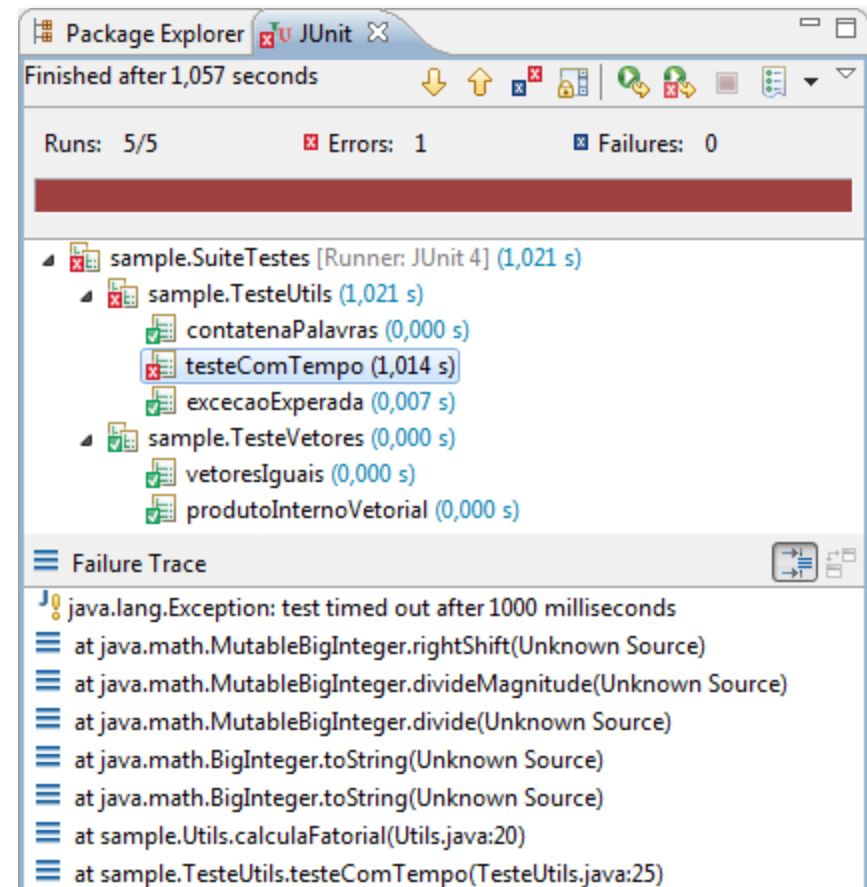
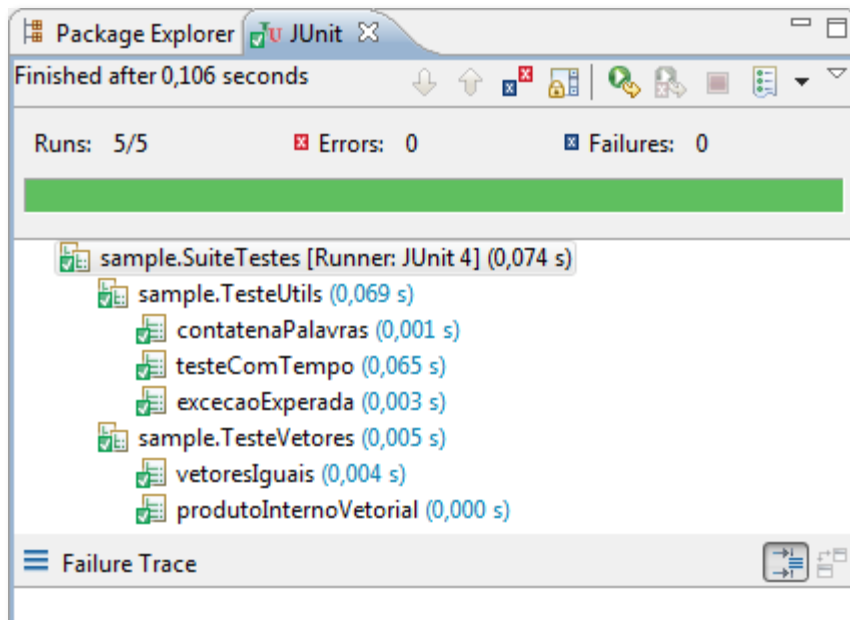
Suíte em JUnit 4

```
1 package sample;
2
3 import org.junit.runner.RunWith;
4 import org.junit.runners.Suite;
5
6 @RunWith(Suite.class)
7 @Suite.SuiteClasses( { TesteUtils.class, TesteVetores.class })
8 public class SuiteTestes {
9 }
```

- Suíte de Testes modificada para padrão JUnit 4
- Execute-a como um caso de teste

Suíte de Testes

Resultado dos testes da suíte



Dúvidas?



Exercícios [1/2]

- Criar testes unitários em JUnit para:
 - Método para realizar soma
 - Recebe como entrada dois números
 - Retorna o resultado da soma
 - Método para realizar divisão
 - Lançar exceção quando receber 0 no denominador
 - Método para realizar depósito em uma conta bancária
 - O método deve receber um flutuante com o valor a ser depositado
 - O método deve retornar o valor contido na conta

Exercícios [2/2]

- Criar testes unitários em JUnit para:
 - Método para fazer o saque em uma conta
 - O método aceita dois valores: valor_saque, valor_saldo
 - O método deve retornar um valor referente ao saldo da conta após o saque ou -1 em situação de erro
 - Método para liberar ou não a prova do Detran
 - O método aceita um valor do tipo Integer com a idade do aluno
 - A liberação da prova será mediante ao atendimento dos critérios
 - Idade do aluno entre 18 e 90 anos
 - O método deve retornar um boolean true caso a prova seja liberada ou false em caso contrário

Exercícios para Casa

- Treinamento:
 - Implementar cada exemplo da aula
- Criatividade:
 - Implementar 1 classe e alguns casos de teste para:
 - Registro de ligações
 - Fila de impressão
 - Bilheteria de cinema
 - Estacionamento com ticket