Teste de Unidade - JUnit em Java

Fábio Tadeu Paiva da Silva[1], André Teixeira[2]

[1] Curso de Bacharelado em Informática – Universidade Católica do Salvador (UCSAL)

[2] Curso de Bacharelado em Informática – Universidade Católica do Salvador (UCSAL)

fabiotadeupaiva@hotmail.com, andré_wasilva@hotmail.com

Resumo. *Este artigo descreve sobre o framework JUnit seu padrão de uso e aplicação.*

1. Introdução

Com objetivo principal de reduzir a ocorrência de erros em sistemas informatizados criou-se metodologia e tecnologia para testar software´s garantindo mais qualidade aos programas. Dentre os testes existentes o primeiro e mais importante é o teste de unidade, que é a forma de testar a menor parte de um programa. No caso dos programas funcionais a menor unidade são as funções, já para as linguagens OO são as Classes, objetos e métodos.

2. Objetivo

Neste tutorial abordamos o conceito de testes unitário de software, critério de avaliação e execução de testes com a ferramenta JUnit da linguagem Java.

3. Conceito de Testa de Unidades

Teste de Unidade é o teste realizado na menor unidade dos programas. Assim, temos o teste das funções de um programa funcional e o teste de classes, objetos e métodos em uma linguagem OO como teste unitário.
O conceito de teste unitário parte da análise de entradas esperadas com saídas desejadas em trechos específicos do código, isolados do restante do programa. Nestas saídas temos a analise dos resultados que podem ser falha ou sucesso. Na ocorrência de falhas é atribuído ao programador à responsabilidade da correção. O TESTE DE UNIDADE NÃO CORRIGE ERRO. APENAS OS ENCONTRA.

3.1. Avaliação de Casos de Teste

A) Para as Entradas:

A.1) “Casos-limite de teste são aqueles se encontram no limite das entradas aceitáveis.”

B) Para as Saídas:

B.1) “Análise Simples aplicada quando temos conhecimento correto das saídas esperadas, sendo pela simplicidade dos cálculos ou pela implementação de métodos.”

B.2) “Calculo reverso onde pode ser escrito um testador que verifique as saídas com determinadas propriedades, como o calculo no sentido oposto. Como exemplo a raiz de um numero exato pode ser validada pela multiplicação deste numero com ele mesmo.”

B.3) “Oráculo que pode ser calculado em conjunto com o processo, mais lento, e compara as respostas obtidas. “

3.2 Teste

A) “Teste da Caixa-Preta descreve um método de teste que não leva em conta a estrutura da implementação.”

B) “Teste da Caixa-Branca utiliza as informações sobre a estrutura de um programa.”

C) “Teste de abrangência é uma medida do número de partes testadas de um programa.”

D) “Teste de Regressão envolve a repetição da execução de testes anteriores para garantir que falhas conhecidas de versões anteriores não apareçam em novas versões do software.”

E) “Suíte de Teste é um conjunto de testes repetidos.”

Referencia: Os conceitos apresentados entre aspas foram retirados do livro Big Java referenciado ao final deste artigo.

3.3 Implementação de Teste usando conceitos e a linguagem Java

Problema: Dado dois números inteiros calcular o valor absoluto da diferença de dois números inteiros da e retornar um valor inteiro não negativo.

```
1  public class DiferencaValorAbsoluto{
2     private int valor1, valor2;
3     public DiferencaValorAbsoluto(int parcela1, int parcela2){
4         this.valor1 = parcela1;
5         this.valor2 = parcela2;
6     }
7     public int ResultaDiferencaValorAbsoluto(){
8          if(this.valor1<0)
9             this.valor1 = ((this.valor1) * -1);
10         if(this.valor2<0)
11            this.valor2 = ((this.valor12 * -1);
12         return (-1 * (this.valor1 - this.valor2));
13    }
14    public static void main(String args[]){
15    DiferencaValorAbsoluto p = new DiferencaValorAbsoluto(2,2);
16          System.out.println("O valor Absoluto da diferença é : "+
```

Fig. 1 – Código para resolver problema apresentado com uso dos conceitos de teste

3.4 Analise do Resultado

- É um código com teste embutido – Não aconselhável
- Sem padronização
- Sem documentação
- Sem automatização

4.0 JUnit

Criado para padronizar as tarefas de testes em unidades o JUnit é um frameword java, open-source, na sua versão 4.x, que pode ser baixado do site www.junit.org um arquivo junit.rar e deve ser extraído sua estrutura de diretórios na pasta de plugins de sua IDE, conforme Fig. 2 apresentada como exemplo abaixo. Algumas IDE´s como Eclipse já traz incorporados este framework não sendo necessário tal procedimento.

Fig. 2 – Exemplo do local onde deve ser descompactado o arquivo junit.jar, no eclipse

Referencia: walfredo.lsd.ufcg.edu.br/cursos/leda20011/Usando-**JUnit**.ppt

4.1 Características:

- **Padronização para testes em Java**
- **Facilidade de visualização de forma mais simples**
- **Verifica funcionamento esperado do código**
- **Agrupamento para rodar múltiplos testes em conjunto**
- **Causa exibida dos erros em cada teste**

4.2 Arquitetura Completa

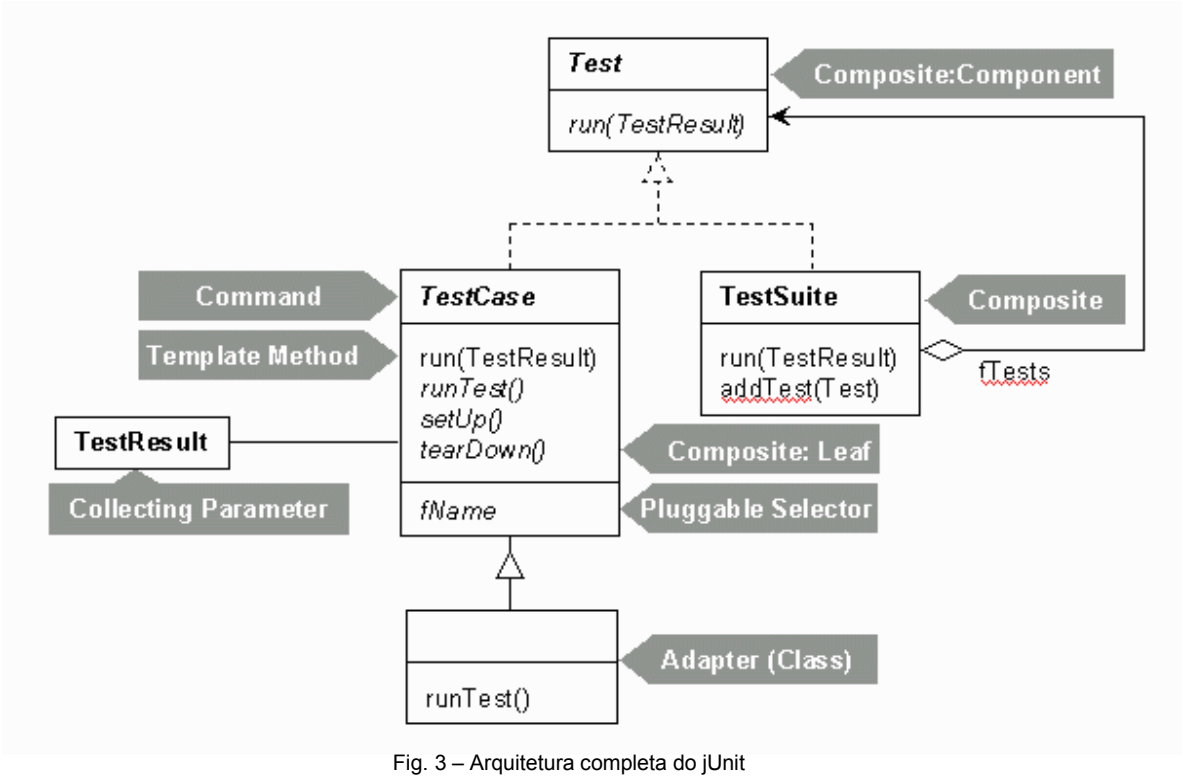


Fig. 3 – Arquitetura completa do jUnit

Referencia: www.lbd.dcc.ufmg.br:8080/colecoes/sbes/2006/003.pdf

Nesta figura devemos abstrair o fato de todas as classes testes que criarmos sempre serão subclasses da classe TestCase. Ou seja estas sempre irão herdar métodos da classe TestCase. E que o TestSuite será um conjuntos de TestCase, adicionado pelo método addTest.

4.3 Estrutura Básica de um TESTE

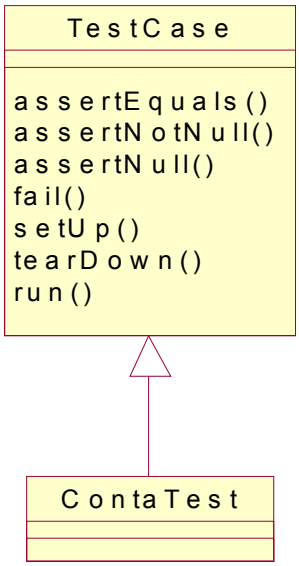


Fig. 4 – Estrutura TestCase

Referencia : www.inf.ufsc.br/~cybis/ine5322/Aula13_Teste_de_SW_cont.pdf

Desta figura podemos observar que todas as classes testes devem terminar seu nome com a palavra “Test”. Ex: “xxxTest”. Já os métodos devem ter seus nomes iniciados pela palavra “test”. Ex: “testExecutaMetodo()”.

4.4 Métodos da Classe TestCase

- assertEquals (esperado,retornado) testa a igualdade entre os valores esperado e retornado.
- assertTrue (boolean b) testa retorno booleano VERDADEIRO
- assertFalse (boolean b) testa retorno boolean FALSO
- assertNull (object o) testa se um valor de um objeto esta nulo.
- assertNotNull (object o) testa se um valor de um objeto não esta nulo.
- assertSame (object a, object b) testa a igualdade entre dois objetos.
- asserNotSame (Objetc a, Objetc b) testa a diferença ente dois objetos
- setUp() cria ambiente para cada caso de teste.
- tearDown() destrói ambiente para cada teste.

5 Implementação com JUnit

Passo: Criar uma classe com nome **DiferencaValorAbsoluto.java** e copiar ou digitar o conteúdo abaixo, apagando os índices (ex: 1, 2,3,etc):

```
1   public class DiferencaValorAbsoluto{
2     private int valor1, valor2;
3     public DiferencaValorAbsoluto(int parcela1, int parcela2){
4         this.valor1 = parcela1;
5         this.valor2 = parcela2;
6    }
7     public int ResultaDiferencaValorAbsoluto(){
8           if(this.valor1<0)
9                 this.valor1 = (-1 * (this.valor1));
10          if(this.valor1<0)
11                this.valor1 = (-1 * (this.valor1));
12
13          return (-1 * (this.valor1 - this.valor2));
14          //return this.valor1 - this.valor2;
15    }
16   } // Fim de Classe
```

Fig. 5 – Código padronizado para Junit

Passo: Criar uma classe teste com o **DiferencaValorAbsolutoTest.java** e digitar o conteúdo da imagem abaixo.

Observando a primeira linha o import para podermos importar as classes do JUnit.

ResultaDiferencaValorAbsoluto()

Outra forma de criarmos é clicando com botão direito do mouse sobre a classe DiferencaValorAbsoluto e selecionando a opção *New , Other.* Será aberta a janela NEW, selecione a opção JUNIT , JUNIT TESTECASE. Na classe aberta copie o conteúdo da classe baixo sem o import, public extends e a ultima chaves.

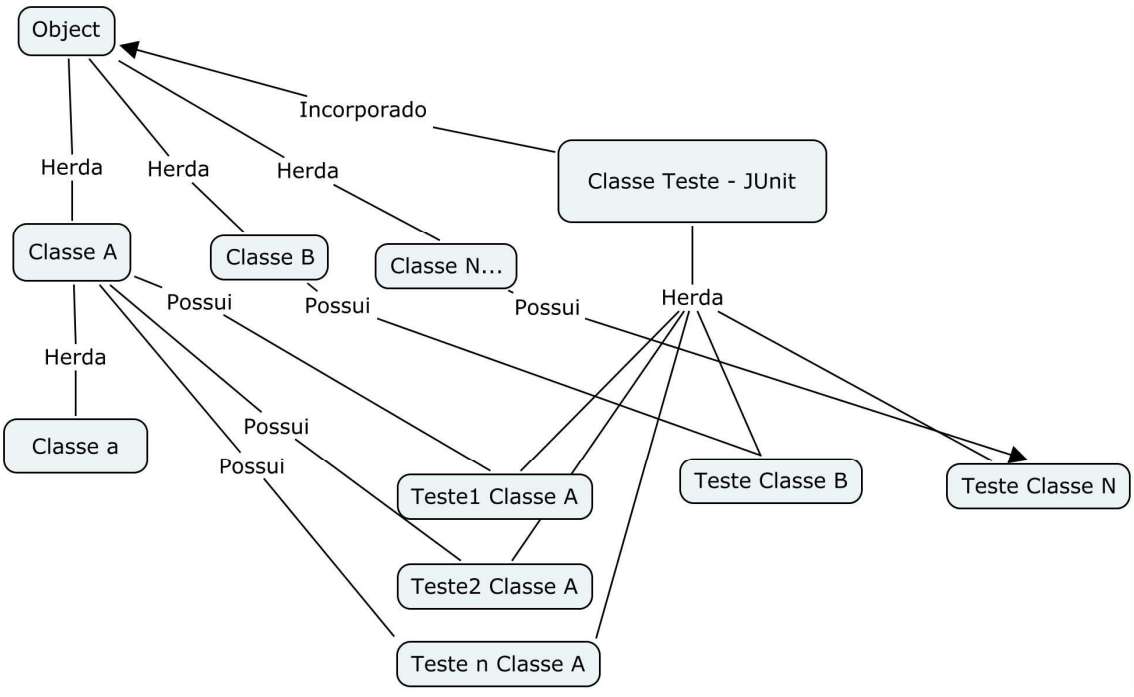
```
import junit.framework.TestCase;

public class DiferencaValorAbsolutoTest extends TestCase {

 public void testResultaDiferencaValorAbsoluto() {
// atribui valores manualmente
          int primeiro = 1;
          int segundo = 2;
          int esperado = 1;
            // executa metodo e
       DiferencaValorAbsoluto p = new
DiferencaValorAbsoluto(primeiro,segundo);
       int resultadoocorrido = p.ResultaDiferencaValorAbsoluto();
       assertEquals(esperado,resultadoocorrido);
}
} // Fim de classe

// DiferencaValorAbsolutoTest extends TestCase
```

Fig. 5 – Código padronizado para Classe Teste Junit

Passo: Click no botão run do Java ou acesse a opção run de menu Run. Será apresentada a Interface Gráfica de Teste do Junit (TRunner). Se o teste for ok apenas será apresentado o tempo gasto para execução e uma barra verde. Se não for validado será exibida uma tarja vermelha com o erro ocorrido na guia do junit.

Sucesso -
Falha -
Exceção -

6 MAPA CONCEITUAL



7 REFERENCIAS BIBLIOGRAFICAS

Livro Big Java – Capitulo IV – Teste de Unidade
3º edição – Editora Makron

8 SITES PESQUISADOS

- www.junit.org.br acessado em 20/10/2008.
- <http://walfredo.dsc.ufpb.br/cursos/2002/progII20021/aulas/testes.htm>, acessado em 20/10/2008
- <http://web.teccomm.les.inf.puc-rio.br:8668/eclipse/space/Eclipse>, acessado em 20/10/2008
- http://www.mhavila.com.br/link/prog/soft-eng.html, acessado em 20/10/2008.
- <http://www.inf.furb.br/~jomi/java/apresentacoes/JavaBasico/sld009.htm>,