Programação Orientada a Objetos

Departamento de Computação
Universidade Federal de Sergipe

JUnit

Prof. Kalil Araujo Bispo
kalil@dcomp.ufs.br

Bugs

- Debugar e Modificar levam muito mais tempo do que Escrever o código
- Consertar um bug é rápido! O problema é encontrá-lo
- Uma correção de um bug pode introduzir novos bugs

Como muitos testam...

- **“Cada classe deve ter seus métodos de teste”**
  - Em Java, normalmente fazemos testes em métodos main
    - Separados
    - Muitas vezes são descartados depois que a classe está “pronta”
  - Geralmente esses testes devem ter sua saída interpretada

Filosofia dos Teste de Unidade

- **A identificação de erros nos testes deve ser automática**
  - Indicar o teste, o valor esperado e o obtido
  - Caso não tenham havido erros, uma mensagem com OK é o suficiente

- **Cada teste deve ser independente**

Filosofia dos Teste de Unidade

- **Fazer o teste antes de implementar**
  - Ponto de parada
  - Objetivos mais explícitos

- **Ferramenta para executar os testes de uma vez**
- **Os testes devem ser pequenos e rápidos**
- **Permite executá-los frequentemente**

Obstáculos

- **Convencer programadores de que:**
  - É interessante
  - Gasta-se menos tempo

- **Alguns programadores não testam**

Testes de Unidade

- **Trechos de código escritos pelo programador**
  - Executam uma funcionalidade específica no código que está sendo testado
- **Avaliam pequenas unidades de código (testes locais)**
  - Métodos e classes
- **A porcentagem de código validado por testes de unidade é tipicamente chamada de cobertura do teste (*test coverage*)**

Testes de Unidade

- **Garantem que o código funciona como previsto**
  - Após implementação e também após a modificação do código
- **Ter uma alta cobertura faz com que seja possível adicionar novas funcionalidades no seu código sem ter que fazer uma série de testes manuais**
- **Tipicamente são criados em uma pasta ou projeto separado, a fim de evitar que o código e o teste se misturem**

Testes de Unidade

- **Ferramentas**
  - http://www.junit.org (Java)
  - http://dunit.sourceforge.net (Delphi)
  - http://cppunit.sourceforge.net (C++)
  - http://cunit.sourceforge.net (C)
  - http://httpunit.sourceforge.net (Extensão do JUnit)

JUnit

- **É um framework de código aberto criado por Erich Gamma (Padrões de Projeto) e Kent Beck (XP)**
- **Usa anotações para identificar métodos que especificam um teste**
- **Tipicamente esses métodos estão contidos em uma classe que é utilizada apenas para teste**
  - Classe de teste

Métodos

- **Método de teste de unidade**
  - Pode ser criado no Eclipse através do comando
  - File → New → JUnit → JUnit Test case
- **Usa anotação @org.junit.Test**

```
@Test
public void testMultiply() {

   // MyClass is tested
   MyClass tester = new MyClass();

   // Check if multiply(10,5) returns 50
   assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));
 }
```

Métodos

- É assumido que todos os métodos podem ser executados em ordem arbitrária
- Um teste não deve depender de outro teste

Anotações

Annotation	Descrição
@Test public void method()	Identifica o método de teste.
@Before public void method()	Método a ser executado antes de cada teste.
@After public void method()	Método a ser executado depois de cada teste.
@BeforeClass public static void method()	Método executado uma vez, antes do início de todos os testes. Devem ser definidos como estáticos.

Anotações

Annotation	Descrição
@Ignore	Ignora um método de teste.
@Test (expected = Exception.class)	Falha se o método não lançar a exceção esperada.
@Test(timeout=100)	Falha se o método levar mais que o tempo definido por timeout.

Principais Métodos

- A classe Assert traz um conjunto de métodos que podem ser usados nos métodos test<nome>

Statement	Description
fail(String)	Faz o método falhar no teste.
assertTrue([message], boolean condition)	Checa se uma condição booleana é verdadeira.
assertsEquals([String message], expected, actual)	Teste se os valores esperado e atual são iguais.

Principais Métodos

assertsEquals([String message], expected, actual, tolerance)	Testa se o valor double ou float casa com o esperado, dentro da tolerância espectificada.
assertNull([message], object)	Checa se o objeto é nulo.
assertNotNull([message], object)	Checa se o objeto não é nulo.
assertSame([String], expected, actual)	Checa se ambas as variáveis referenciam o mesmo objeto.
assertNotSame([String], expected, actual)	Checa se ambas as variáveis referenciam objetos diferentes.

JUnit

- **Test Suite**
  - Reúne as classes de teste.
  - Executa todas as classes de teste que estiverem na Test Suite

```
@Suite.SuiteClasses statement.
package seu.pacote;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ MyClassTest.class,
MySecondClassTest.class })
public class AllTests { … }
```

JUnit

- **Não vou cobrar na prova**
- **Vou cobrar no projeto**
  - Seria interessante testes para todas as classes

Referências

- Lars Vogel. Junit - Tutorial. Disponível no link: http://www.vogella.com/articles/JUnit/article.html
- Homepage do JUnit: http://junit.org/