Testes com JUnit

Danilo Toshiaki Sato
dtsato@ime.usp.br

Treinamento ALESP – SPL

Agenda

1. Introdução
2. Por que usar JUnit?
3. Quando escrever um teste?
4. Como escrever um teste?
5. Como rodar um teste?
6. Conclusão
7. Workshop

Introdução

- Cenário:
  - Defeitos são caros
  - Quanto mais tarde são encontrados, mais caros
  - Conclusão: É melhor encontrar defeitos o mais cedo possível
- Teste cedo e freqüentemente

Introdução - Junit

- Arcabouço livre para testes automatizados escrito em Java
- Escrito originalmente por Kent Beck e Erich Gamma
- Parte de uma família de arquitetura para testes conhecida como xUnit
- Utilizado principalmente no desenvolvimento de testes de unidade
- **http://www.junit.org**

Por que usar JUnit?

- Facilita a escrita de testes automatizados
- Funcionalidades inclusas:
  - Asserções para testar resultados esperados
  - *Fixtures* para reutilização de dados para teste
  - *Test Suites* para organizar e executar conjuntos de testes
  - Interface gráfica e textual para execução de testes
- Integração com as principais IDEs
- Grande comunidade de usuários

Quando escrever um teste?

> *"Sempre que estiver tentado a escrever um `print()` ou uma expressão de depuração, escreva um teste"*
>
> -- Martin Fowler

Momentos em que é bom investir em testes:

- Durante o desenvolvimento
  - Crie testes para as classes que está desenvolvendo
- Durante a correção de defeitos
  - Crie um teste que reproduza o erro antes de corrigí-lo

Como escrever um teste?

- Mais simples:
  1. Crie uma subclasse de TestCase

```
public class TesteSimples extends TestCase {
    (…)
}
```

  2. Crie um método de teste (que comece com `test`) que verifica os resultados esperados

```
public void testColecaoVazia() {
    Collection colecao = new ArrayList();
    assertTrue(colecao.isEmpty());
}
```

Como escrever um teste?

- *Fixture*: Conjunto de dados de teste e objetos utilizados na execução de um ou mais testes
- Para reaproveitar uma *Fixture* em mais de um teste:
  1. Sobrescreva o método `setUp()` (inicialização)

```
protected void setUp() {
    colecao = new ArrayList();
}
```

  2. Sobrescreva o método `tearDown()` (limpeza)

```
protected void tearDown() {
    colecao.clear();
}
```

Como escrever um teste?

```
public class TesteSimples extends TestCase {
    private Collection colecao;
    protected void setUp() {
        colecao = new ArrayList();
    }
    protected void tearDown() {
        colecao.clear();
    }
    public void testColecaoVazia() {
        assertTrue(colecao.isEmpty());
    }
    public void testColecaoComUmItem() {
        colecao.add("itemA");
        assertEquals(1, colecao.size());
    }
}
```

Como escrever um teste?

- Possível ordem de execução:
  1. `setUp()`
  2. `testColecaoComUmItem()`
  3. `tearDown()`
  4. `setUp()`
  5. `testColecaoVazia()`
  6. `tearDown()`
- Como os testes são chamados por reflexão, a ordem de execução dos testes pode não seguir o mesmo fluxo do código
- Garantia: `setUp()` será executado antes e `tearDown()` será executado depois

Como escrever um teste?

- Testando uma exceção esperada (cenário de erro)
  1. Capture a exceção num bloco try/catch e falhe o teste caso ela não seja lançada

```
public void testIndexOutOfBoundsException() {
    ArrayList listaVazia = new ArrayList();
    try {
        Object o = listaVazia.get(0);
        fail("Não lançou exceção esperada.");
    } catch (IndexOutOfBoundsException e) {
        assertTrue(true);
    }
}
```

Como escrever um teste?

- Testando uma exceção não esperada

1. Declare a exceção na assinatura do método e não capture-a no código do teste

```
public void testFalhaIndexOutOfBoundsException()
    throws IndexOutOfBoundsException {

    ArrayList listaVazia = new ArrayList();
    Object o = listaVazia.get(0);
}
```

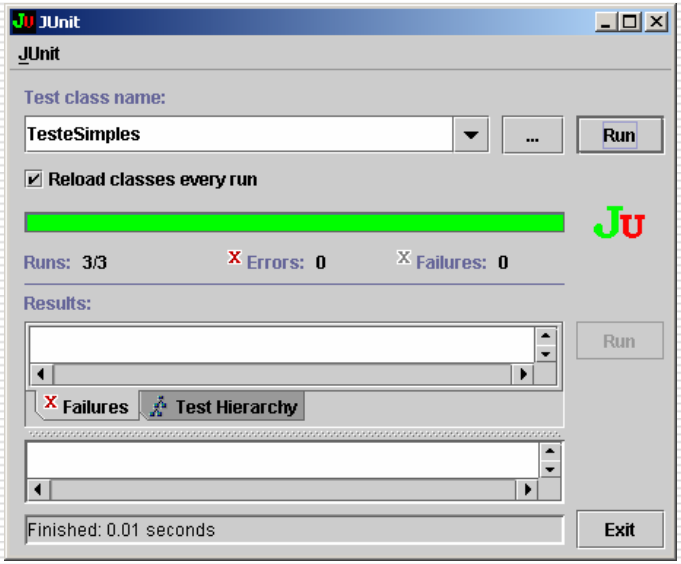
Obs: Esse teste irá falhar

Como escrever um teste?

- Algumas considerações:
  - Testes de unidade devem exercitar o comportamento isolado de uma classe
  - Geralmente o comportamento de um objeto depende da interação com outros objetos
  - Nesse caso, é comum utilizar objetos "dublês" para isolar o comportamento
  - Alguns tipos de objetos "dublês":
    - Dummy
    - Fake
    - Stubs
    - Mocks

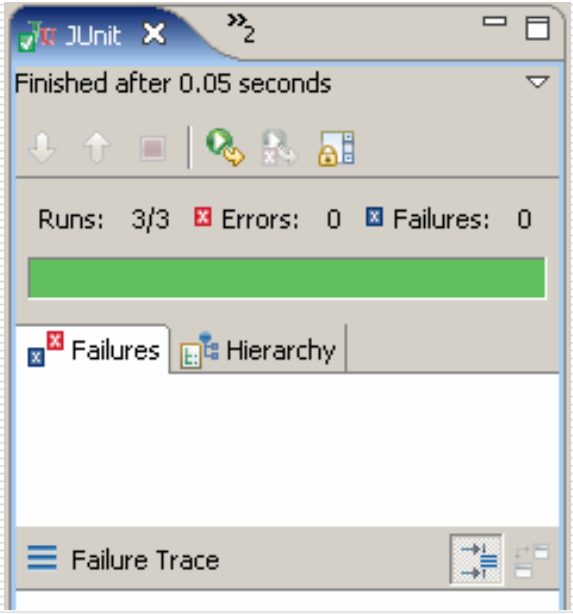
Como rodar um teste?

- JUnit vem com dois *TestRunners*:
  - Textual: utilizado na linha de comando
  - Gráfico: interface gráfica simples para execução e acompanhamento do progresso dos testes



Como rodar um teste?

- Eclipse
  1. Clicar com o botão direito na classe de teste e escolher “Run As > JUnit Test”



Conclusão

- O mais importante é:
  1. Testar cedo
  2. Testar freqüentemente
  3. Testar de forma automatizada
- JUnit ajuda com o item 3
- O resto é com você!

Referências

- http://www.junit.org
- K. Beck and E. Gamma. *Test Infected: Programmers Love Writing Tests*. Java Report, July 1998, Volume 3, Number 7