Criando testes com JUnit

Aristides Vicente de Paula Neto

Centro de Informática – Universidade Federal de Pernambuco – Av. Professor Luiz Freire, S/N CEP: 50.740-540 – Recife – PE

Este artigo tem como objetivo apresentar o JUnit e suas vantagens, caracterizando a importância dos testes unitários e exemplificando (passo a passo) o uso deste framework em um projeto de desenvolvimento Java.

1. O que JUnit? E a importância dos testes unitários.

O JUnit é um framework open-source, criado por Eric Gamma e Kent Beck, com suporte à criação de testes automatizados na linguagem de programação Java.

Esse framework facilita a criação de código para a automação de testes unitários com apresentação dos resultados. Com ele, pode ser verificado se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas podendo ser utilizado tanto para a execução de baterias de testes como para extensão.

E qual a importância da execução dos testes? O produto de software, atualmente, deve passar por várias fases de testes: o teste unitário, de integração, de sistema, de aceitação, entre outros, antes de serem disponibilizados para o usuário final.

Entre estas fases, o teste unitário, também conhecido como teste de unidade, é a fase do processo de teste em que se testam as menores unidades de software desenvolvidas e tem como objetivo prevenir o aparecimento de *bug's* oriundo de códigos mal escritos e garantir um nível de qualidade de produto durante o processo de desenvolvimento de software.

2. Vantagens do JUnit

- Permite a criação rápida de código de teste possibilitando um aumento na qualidade do desenvolvimento e teste;
- Amplamente utilizado pelos desenvolvedores da comunidade código-aberto, possuindo um grande número de exemplos;
- Uma vez escritos, os testes são executados rapidamente sem que, para isso, seja interrompido o processo de desenvolvimento;
- JUnit checa os resultados dos testes e fornece uma resposta imediata;
- JUnit é livre e orientado a objetos.

3. Como configurar?

Para utilizar o JUnit, é necessário a utilização do .jar do JUnit que pode ser encontrado na página principal do próprio framework (http://www.junit.org). Para configurar o JUnit em seu ambiente, é necessário adicionar o .jar do JUnit ao caminho do seu projeto.

Se você estiver trabalhando com o Eclipse, você deve em: [Seu Projeto]/ Properties / Java Build Path/ Add External JARs Externo, conforme a figura 3.1 – Incluir .jar no path.

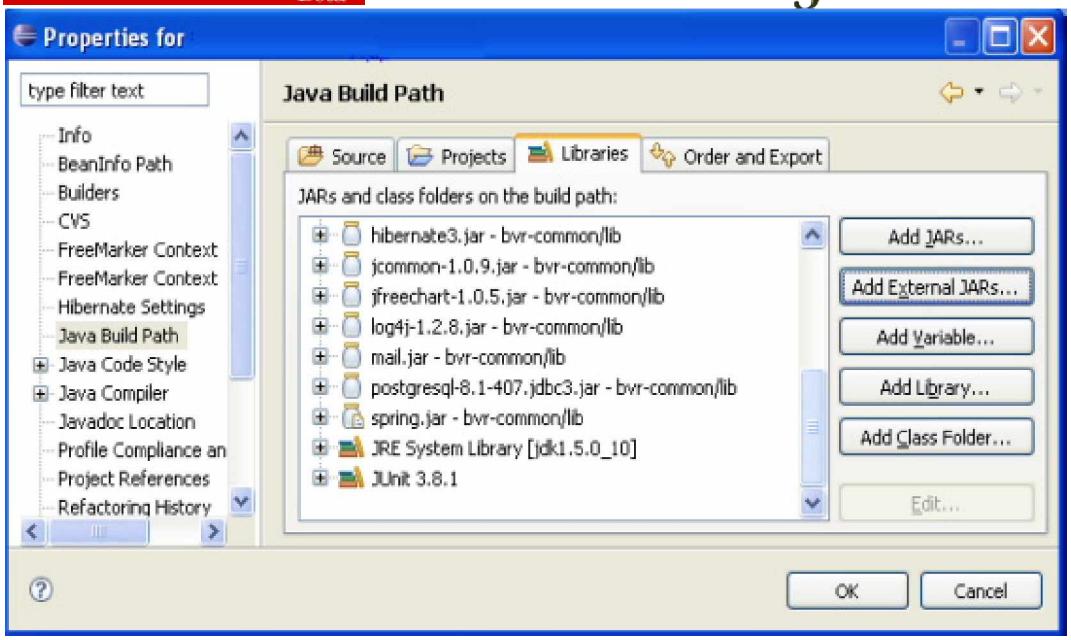

Figura 3.1 – Incluir .jar no path.

4. Arquitetura do JUnit

Para uma melhor compreensão de como o JUnit funciona é importante que entenda como suas classes estão organizadas dentro da API do framework, conforme a Figura 4.1 – Classes do JUnit.

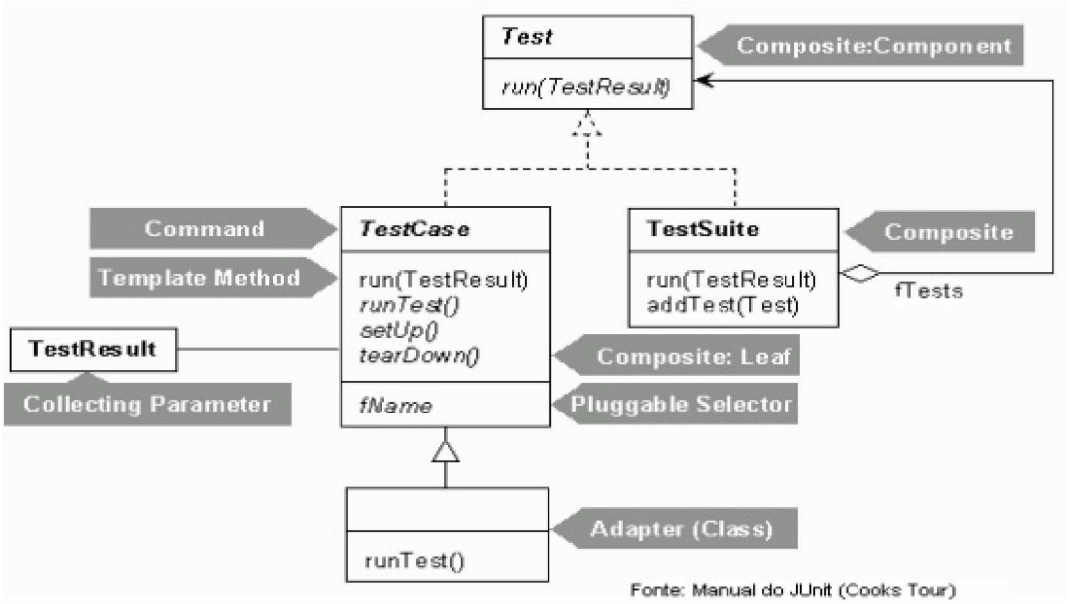

Figura 4.1 – Classes do JUnit.

Classe TestCase:

- command – O padrão (pattern) permite encapsular um pedido (de teste) como objeto e fornece um método run().
- run() – Cria um contexto (método setUp); em seguida executa o código usando um contexto e verifica o resultado (método runTest); e por fim, limpa o contexto (método tearDown).
- setUp() – Método chamado antes de cada método, pode ser utilizado para abri uma conexão de banco de dados.
- tearDown() – Método chamado depois de cada método de teste, usado para desfazer o que setUp() fez, por exemplo fechar uma conexão de banco de dados.
- runTest() – Método responsável por controlar a execução de um teste particular.

Classe TestSuite: Com esta classe, o desenvolvedor executa um único teste com vários métodos de testes e registra os resultados num TestResult.

- composite – O padrão (pattern) permite tratar objetos individuais e composições de objetos de forma uniforme. Este padrão requer os seguintes participantes:
- addTest() – Método responsável em adicionar um novo teste a rotina.

5. Utilizando o JUnit

Nesta seção apresenta o passo a passo do uso do JUnit e para exemplificar usaremos o IDE Eclipse 3.0.

Exemplo: Envio de E-mail

Crie uma classe EmailTeste que extende de TestCase, conforme a Figura 5.1 – Junit Test Case.

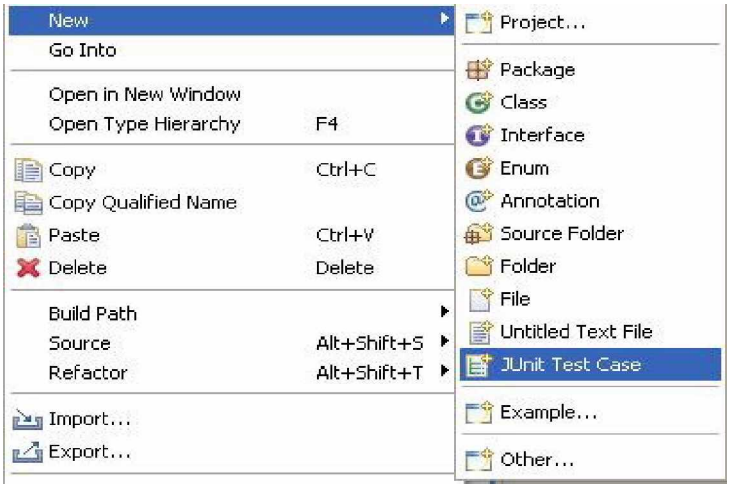

Figura 5.1 – JUnit Test Case.

A seguir é apresentado o código-fonte da classe EmailTeste (Listagem 5.1 – Classe EmailTeste), responsável por validar/testar o método de envio de e-mail da classe Email, disponibilizado posteriormente o código-fonte da classe Email (Listagem 5.2 – Classe de Envio de E-mail).

```
import junit.framework.Test;
import junit.framework.TestCase;

/**
 * Classe EmailTeste - Responsável em realizar teste
unitários na classe E-mail.
 *
 * @author Aristides Vicente (aristidesvicente@gmail.com)
 * @since 17/01/2007
*/

public class EmailTeste extends TestCase implements Test{
      public EmailTeste() {
            super();
      }

      protected void setUp(){
      System.out.println("Iniciando...");
      }

     /**
      * Método testEnvia - Método responsável em realizar
testes no Envio de E-mail.
      *
      * @since 17/01/2007
      */
      public void testEnvia() {
              assertEquals("E-mail não enviado ", true,
Email.envia("Teste JUnit - Mail"));
      }

      protected void tearDown(){
        System.out.println("Finalizando...");
      }
}
```

Listagem 5.1 – Classe EmailTeste.

Maiores informações sobre o método assertEquals e outros métodos, podem ser obtidos através http://junit.sourceforge.net/javadoc/junit/framework/Assert.html .

```
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.NoSuchProviderException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;

/**
* Classe Email – Responsável em enviar e-mail.
```

```
*
* @author Aristides Vicente (aristidesvicente@gmail.com)
* @since 15/01/2007
*/

public class Email {

      public Email() {
            super();
      }

      public static boolean envia(String mensagem){
        Properties props = new Properties();
        props.setProperty("mail.transport.protocol", "smtp");
        props.setProperty("mail.host", "HOST");
        props.setProperty("mail.user", "USER");

        Session mailSession = Session.getDefaultInstance(props,
null);
        Transport transport;
        boolean retorno = false;

           try {
            transport = mailSession.getTransport();
            MimeMessage msg = new MimeMessage(mailSession);

            msg.setSubject("SUBJECT");
            msg.setContent(mensagem, "text/plain");
            msg.addRecipient(Message.RecipientType.TO, new
InternetAddress("aristidesvicente@gmail.com"));
            transport.connect();

transport.sendMessage(msg,msg.getRecipients(Message.RecipientType
.TO));
            transport.close();
            retorno = true;

          } catch (NoSuchProviderException e) {
              e.printStackTrace();
          } catch (MessagingException e) {
                e.printStackTrace();
          }
        return retorno;
      }
}
```

Listagem 5.2 – Classe de envio de E-mail.

Agora que os testes foram criados e explicados, vamos executá-los. Para isso basta clicar com o botão direito do mouse na classe EmailTeste e procurar pelo menu Run As -> JUnit Test, conforme é apresentado na Figura 5.2 – Executando JUnit.

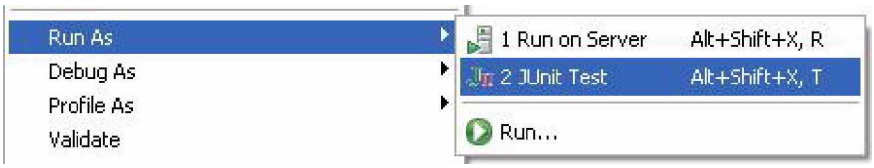

Figura 5.2 – Executando JUnit.

6. Conclusão

A utilização do framework JUnit para testes unitários em Java é muito importante quando se quer desenvolver softwares mais estáveis, e que a prática é indicada e muitas vezes requerida pelo próprio cliente, seja ele privado ou público. Existe o framework J2MEUnit, baseado no JUnit, utilizado para testes unitários em projetos desenvolvidos com J2ME, recomendo www.takenami.com.br/2007/01/28/testes-unitarios-com-j2meunit .

7. Referências

[01] – Site oficial do JUnit: http://www.junit.org

[02] – Wikipédia: http://pt.wikipedia.org/wiki/JUnit

[03] – Manual JUnit: http://junit.sourceforge.net/doc/cookstour/cookstour.htm

Sobre o autor

Aristides Vicente de Paula Neto possui graduação em Tecnologia de Análise e Desenvolvimento de Sistemas (CTDS), pela União dos Institutos Brasileiros de Tecnologia (2006). Atualmente é colaborador do fórum JavaFree e estudante do curso de especialização em Gestão de Tecnologia da Informação pelo Centro de Informática (CIn) da UFPE.

Desde 11/2006, atua como Engenheiro de Software da BVR Negócio e consultoria trabalhando com desenvolvimento de software para Web utilizando as seguintes tecnologias, frameworks, banco de dados e ferramentas: Java, Javascript, Java Server Faces, Jasper Report, Hibernate, Spring, JUnit, Postgresql e Concurrent Version System (CVS).

Outras informações: http://lattes.cnpq.br/4118113904681933