



JUnit 4.11

Framework para Automatizar a Execução de Teste Unitário

Auri Marcelo Rizzo Vincenzi¹, Márcio Eduardo Delamaro² e
José Carlos Maldonado²

¹Instituto de Informática
Universidade Federal de Goiás

²Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo

Organização

Introdução

Histórico

Pré-requisitos

JUnit Básico

Instalação

O Framework

O Exemplo

Compilando e Executando via Linha de Comando

Projeto Maven no Eclipse

Copiando a Aplicação para o Projeto

Entendendo o Código

Criando Casos de Teste

Encontrando uma Falha e Corrigindo o Defeito

Métodos Especiais

JUnit Avançado

Temporização em Casos de Teste

Teste de Exceções

Definindo um Conjunto de Teste

Executando os Teste sem o Eclipse

Ignorando Casos de Teste

Testes Parametrizados

Ferramentas Similares

Referências

Introdução

- Histórico
- Pré-requisitos

JUnit Básico

- Instalação
- O Framework

O Exemplo

- Compilando e Executando via Linha de Comando
- Projeto Maven no Eclipse
- Copiando a Aplicação para o Projeto
- Entendendo o Código
- Criando Casos de Teste
- Encontrando uma Falha e Corrigindo o Defeito
- Métodos Especiais

JUnit Avançado

- Temporização em Casos de Teste
- Teste de Exceções
- Definindo um Conjunto de Teste
- Executando os Teste sem o Eclipse
- Ignorando Casos de Teste
- Testes Parametrizados

Ferramentas Similares

Referências

Histórico

- Desenvolvido por Kent Beck e Erich Gamma, o framework de teste de unidade JUnit se tornou uma das bibliotecas Java mais utilizadas no mundo.
 - De acordo com Martin Fowler “Never in the field of software development was so much owed by so many to so few lines of code”.
 - Início do JUnit: 1994.
 - Versão 3.8: lançada em agosto de 2002.
 - Versão 3.8.1: lançada em setembro de 2002 com defeitos corrigidos.
 - Versão 4.0: lançada em fevereiro de 2006.
 - Versão 3.8.2: lançada em março de 2006.
 - ...
 - Versão 4.11: lançada em novembro de 2012.

Pré-requisitos

- ▶ Conhecimento da linguagem Java.
 - ▶ Conhecimento de teste de software.
 - ▶ Kit de desenvolvimento Java versão 1.5 ou superior. Versão 4.x do JUnit utiliza recursos de anotações disponível apenas a partir desta versão do Java.
 - ▶ Desejável: conhecimento de Eclipse e Maven.



Introdução

Histórico

Pré-requisitos

JUnit Básico

Instalação

O Framework

O Exemplo

- Compilando e Executando via Linha de Comando
- Projeto Maven no Eclipse
- Copiando a Aplicação para o Projeto
- Entendendo o Código
- Criando Casos de Teste
- Encontrando uma Falha e Corrigindo o Defeito
- Métodos Especiais

JUnit Avançado

- Temporização em Casos de Teste
- Teste de Exceções
- Definindo um Conjunto de Teste
- Executando os Teste sem o Eclipse
- Ignorando Casos de Teste
- Testes Parametrizados

Ferramentas Similares

Referências

Instalação (1)

- ▶ Página do projeto: <http://www.junit.org/>
 - ▶ Distribuído em forma compactada.
 - ▶ A versão 4.11 traz uma série de melhorias em relação a versão predecessora 3.8.2.

Instalação (2)

- ▶ Para usar a ferramenta, basta fazer o download dos arquivos jars abaixo e incluí-los no CLASSPATH do projeto.
 - ▶ junit.jar – [junit-4.11.jar](#)
 - ▶ hamcrest-core.jar – [hamcrest-core-1.3.jar](#)
- ▶ Opcionalmente, basta criar os projetos usando Maven e adicionar a dependência abaixo no arquivo pom.xml.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
```

O Framework JUnit

- ▶ Framework de código aberto para o teste de programas Java.
- ▶ Passos básicos para uso do framework:
 - ▶ Criar alguns objetos.
 - ▶ Enviar algumas mensagens a esses objetos.
 - ▶ Verificar se o resultado obtido é igual ao esperado com o uso de asserções (disponíveis no framework).

Introdução

Histórico

Pré-requisitos

JUnit Básico

Instalação

O Framework

O Exemplo

- Compilando e Executando via Linha de Comando
- Projeto Maven no Eclipse
- Copiando a Aplicação para o Projeto
- Entendendo o Código
- Criando Casos de Teste
- Encontrando uma Falha e Corrigindo o Defeito
- Métodos Especiais

JUnit Avançado

- Temporização em Casos de Teste
- Teste de Exceções
- Definindo um Conjunto de Teste
- Executando os Teste sem o Eclipse
- Ignorando Casos de Teste
- Testes Parametrizados

Ferramentas Similares

Referências

A Especificação Exemplo

- ▶ Programa Identifier: parte de um compilador que verifica se um identificador é válido em dada linguagem de programação
- ▶ Código fonte: `Identifier.java` e `IdentifierMain.java` (disponíveis no material do treinamento)
- ▶ Classe principal: `IdentifierMain`
- ▶ Forma de execução:
 - ▶ Identificador deve ser fornecido como parâmetro na invocação do programa.

Compilando o Identifier

```
cd src  
src$ javac Identifier.java IdentifierMain.java
```

Executando o Identifier

Para invocar o programa, basta executar o comando abaixo, dentro do diretório onde o programa foi instalado:

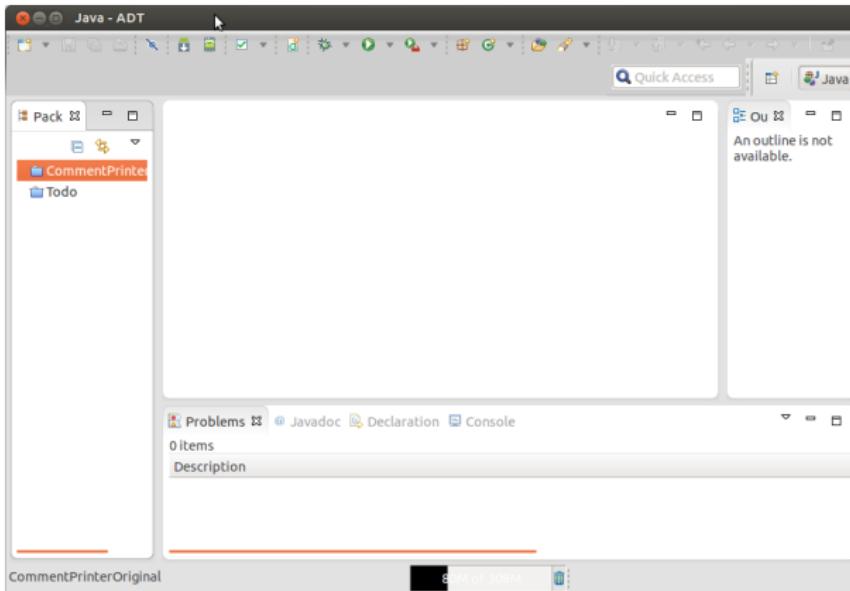
```
1 cd src
2
3 src$ java IdentifierMain
4 Uso: IdentifierMain <string>
5
6 src$ java IdentifierMain "abc12"
7 Valido
8
9 src$ java IdentifierMain "cont*1"
10 Invalido
11
12 src$ java IdentifierMain "1soma"
13 Invalido
14
15 src$ java IdentifierMain "a123456"
16 Invalido
```

Invocando o Eclipse

- ▶ Durante o treinamento, os exemplos serão executados, preferencialmente, no Eclipse.
 - ▶ A versão utilizada é a que acompanha o <http://developer.android.com/sdk/index.html>, já configurada com a maioria dos plugins necessários para o treinamento, incluindo o JUnit.
 - ▶ Plug-ins e ferramentas adicionais serão instalados quando necessário.
 - ▶ Para invocar o Eclipse, basta executar:

<DIRETORIO_INSTALACAO>/adt-bundle-linux-x86_64-20131030/eclipse/eclipse

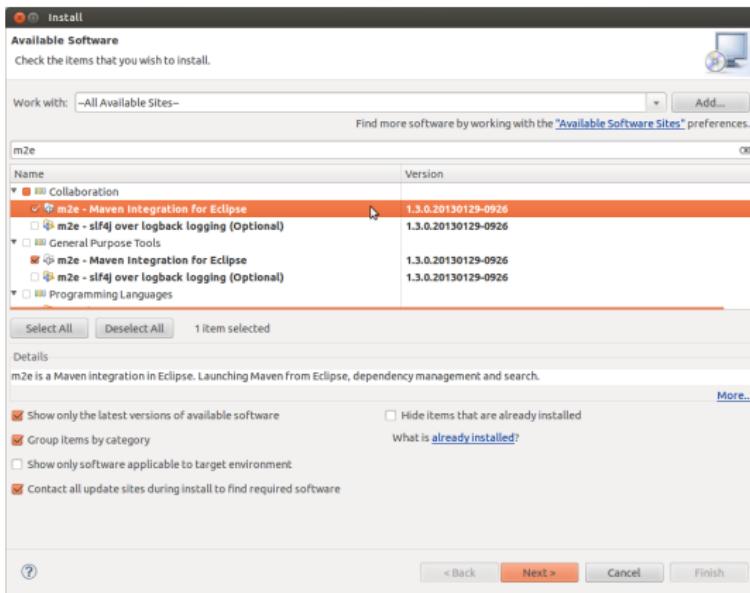
Tela Inicial do Eclipse



Instalando o Plugin m2eclipse

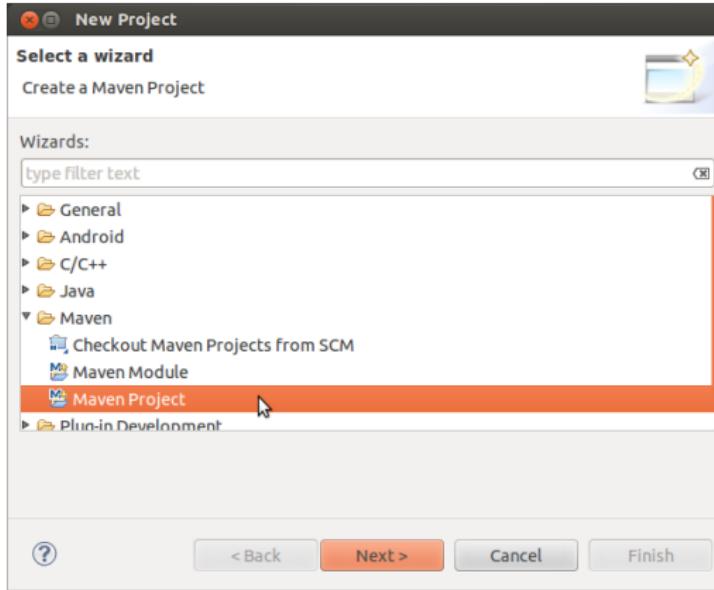
- ▶ Aberto o Eclipse, entre no menu **Help->Install New Software...**
 - ▶ No campo **Work with** selecione – **All Available Sites** –
 - ▶ No filtro preencha com **m2e** (abreviatura para Maven To Eclipse plugin)
 - ▶ Após aparecerem as opções, selecione para instalação o **m2e - Maven Integration for Eclipse**
 - ▶ Prossiga com a instalação normal. Ao final do processo, reinicialize o Eclipse para concluir a instalação.

Tela Instalação de Plugins



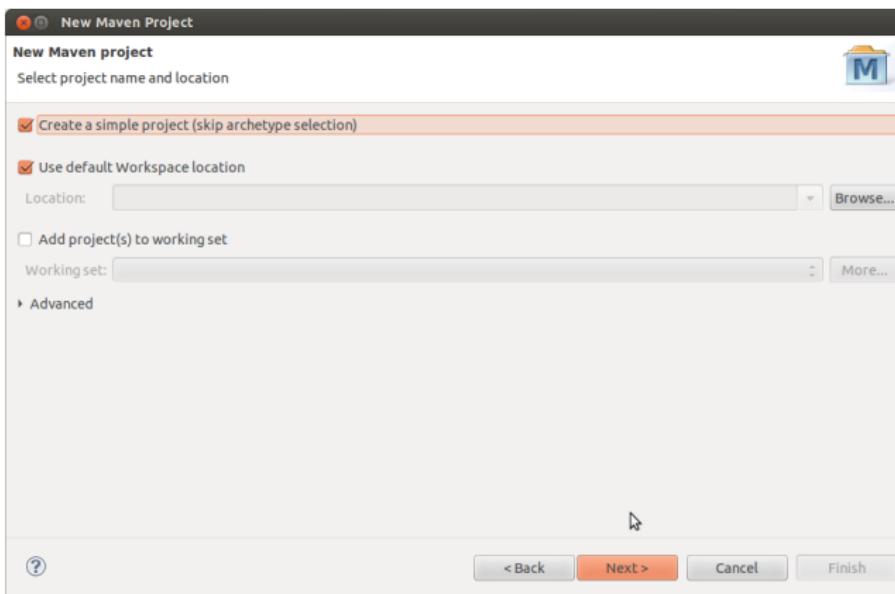
Criando um Projeto Maven (1)

- ▶ Aberto o Eclipse, entre no menu **File->New->Project...**
 - ▶ Escolha **Maven** e **Maven Project**



Criando um Projeto Maven (2)

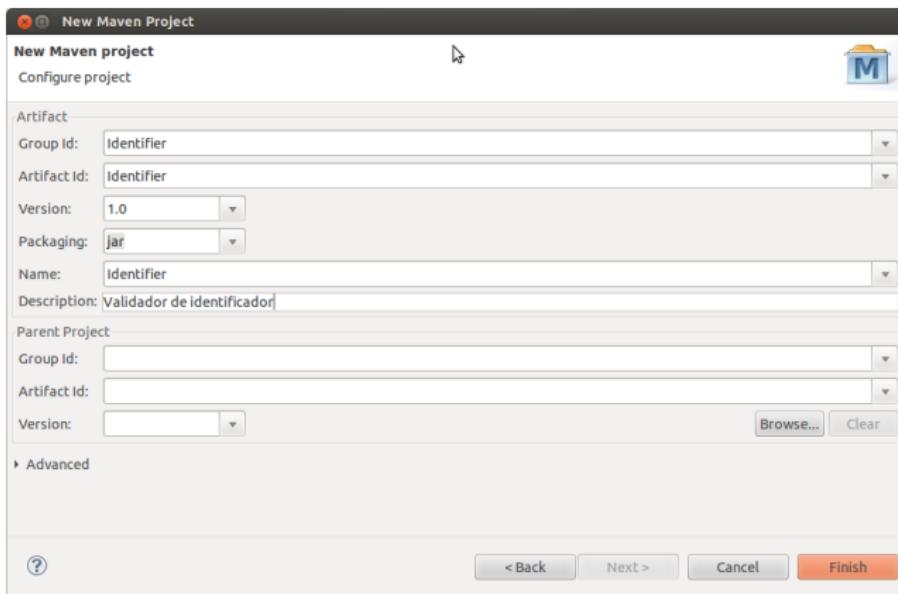
- Em seguida, assinale a opção **Create simple project (skip archetype selection)**



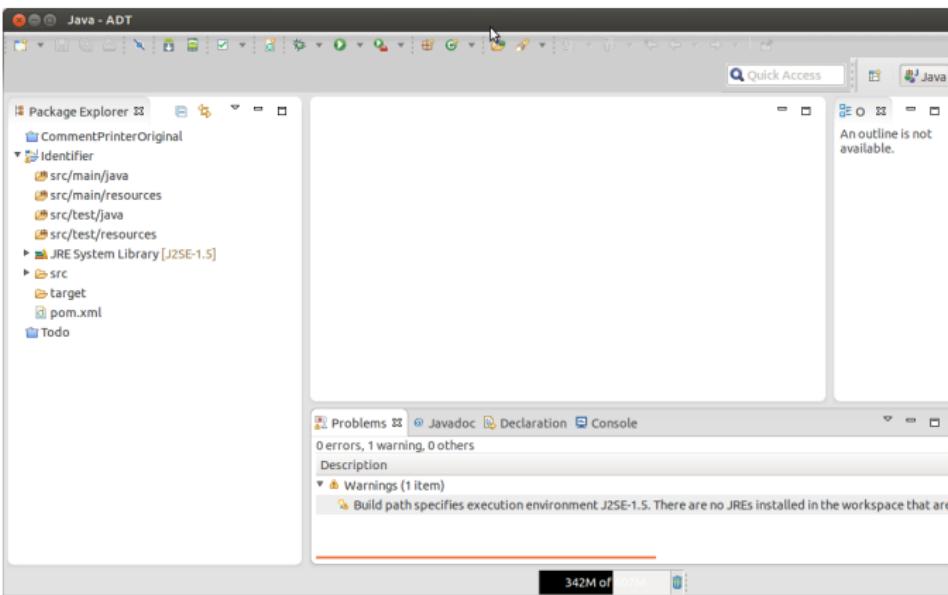
Criando um Projeto Maven (3)

- ▶ Em seguida, preencha os campos para o projeto Identifier, conforme tela a seguir e finalmente clique em **Finalizar**.

Criando um Projeto Maven (4)



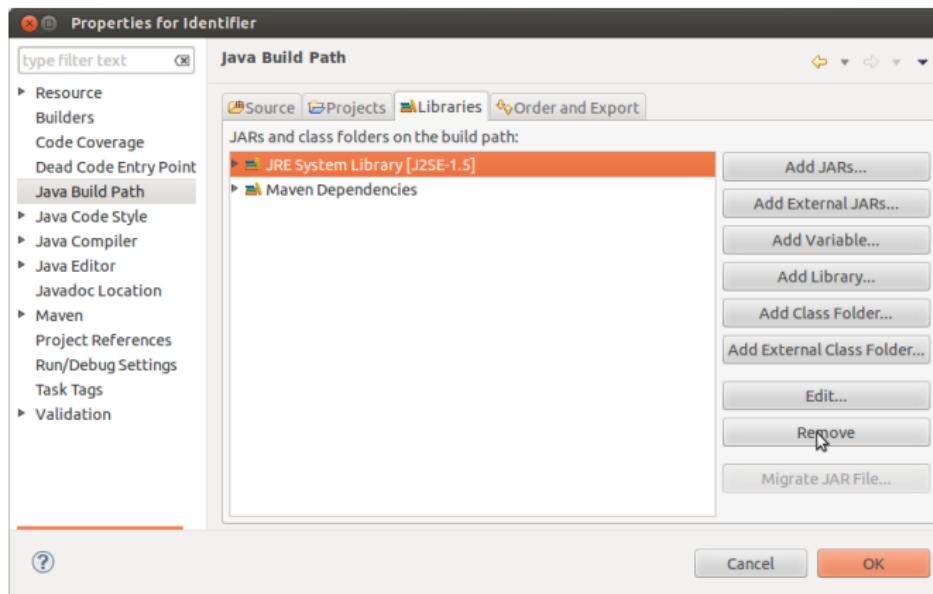
Criando um Projeto Maven (5)



Criando um Projeto Maven (6)

- ▶ Observe na tela anterior que o Eclipse está acusando um warning nesse projeto.
- ▶ Ambiente de execução do J2SE-1.5 definido no projeto, mas essa versão do Java não está disponível.
- ▶ Para alterar isso, clique com o direito do mouse sobre o nome do projeto Identifier e escolha a última opção do menu suspenso **Properties...**

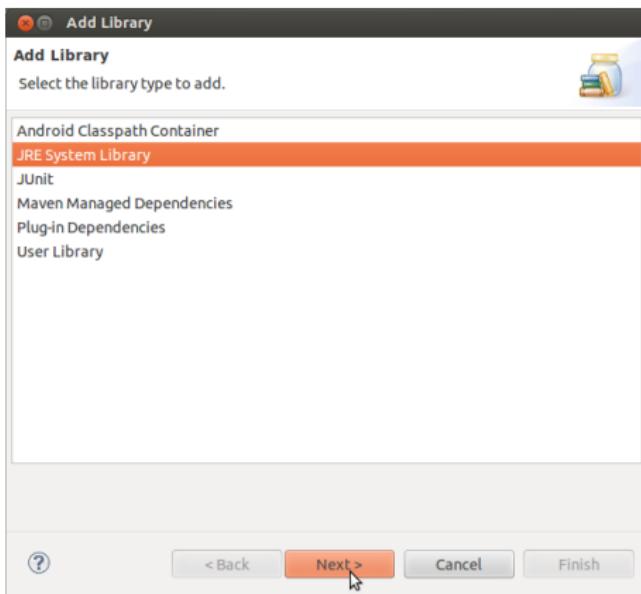
Criando um Projeto Maven (7)



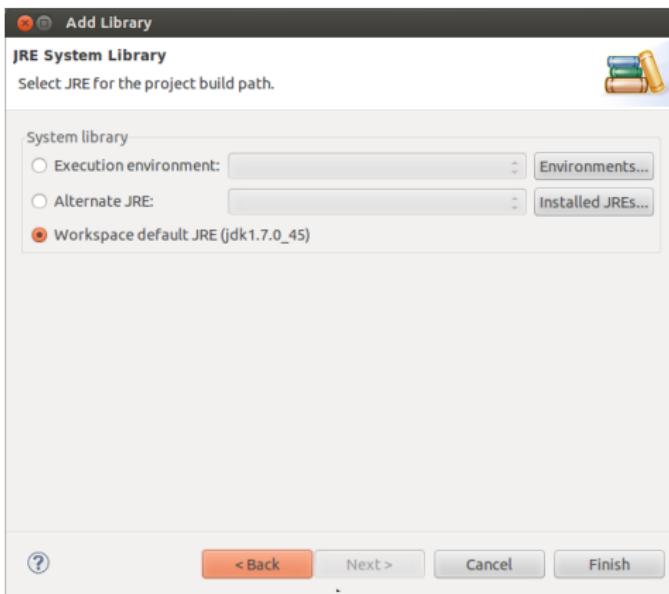
Criando um Projeto Maven (8)

- ▶ Na tela que abrir, selecione **JRE System Library [J2SE-1.5]** e clique em remover.
- ▶ Em seguida clique no botão **Add Library...** e selecione **JRE System Library** e clique em **Next**

Criando um Projeto Maven (9)



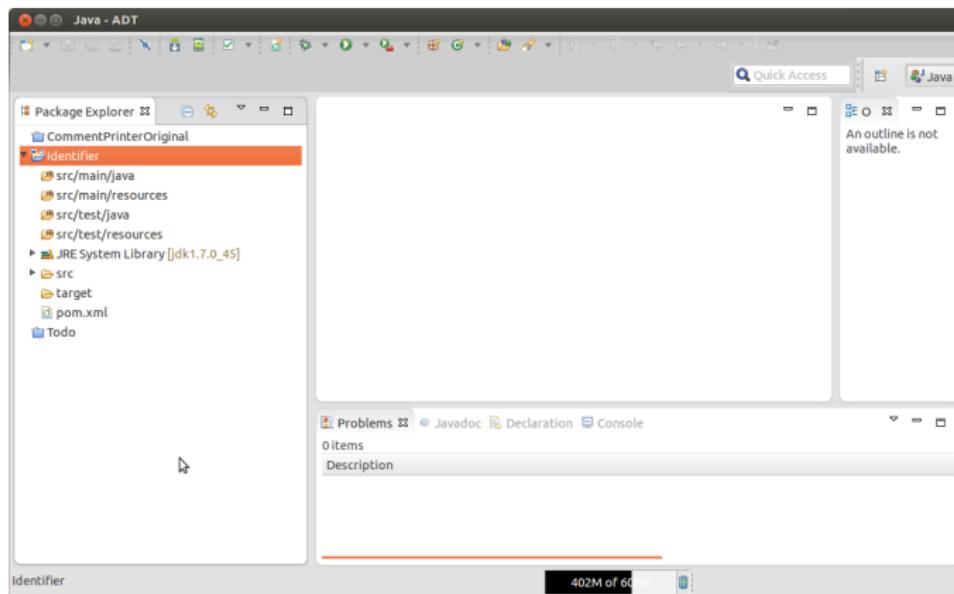
Criando um Projeto Maven (10)



Criando um Projeto Maven (11)

- ▶ Escolha a opção padrão e clique em **Finish**
- ▶ Finalmente, aparecerá o projeto criado sem warnings associado.

Criando um Projeto Maven (12)



Copiando a Aplicação para o Projeto

- ▶ Copie os arquivos Identifier.java e IdentifierMain.java para dentro da pasta src/main/java.
- ▶ Após copiar, clique com o botão direito sobre o nome do projeto Identifier e escolha a opção **Refresh** do menu suspenso.
- ▶ Com isso, o Eclipse irá compilar o projeto e, ao final, a aplicação está pronta para ser executada de dentro do Eclipse.

O Método Principal (1)

- ▶ O método principal do programa Identifier é o método:

```
public boolean validateIdentifier(String s)
```

- ▶ Recebe um string s que corresponde a um dado identificador
- ▶ Retorna true se for um identificador válido em **Silly Pascal**, ou false caso contrário.

“Um identificador válido deve começar com uma letra e conter apenas letras ou dígitos. Além disso, deve ter no mínimo um e no máximo seis caracteres de comprimento.”

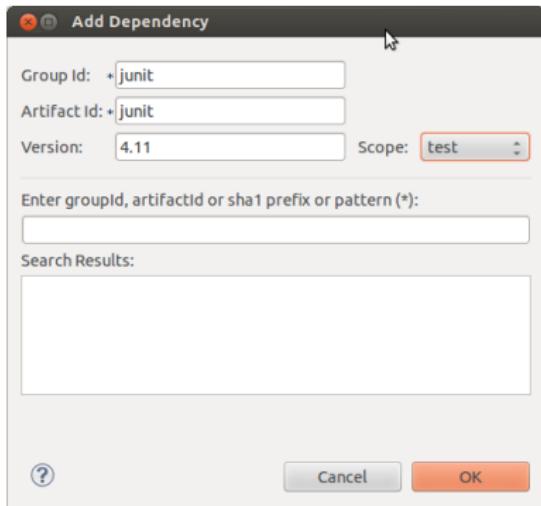
Copiando a Aplicação para o Projeto

O Método Principal (2)

```
2  public boolean validateIdentifier(String s) {
3      char achar;
4      boolean valid_id = false;
5      achar = s.charAt(0);
6      valid_id = valid_s(achar);
7      if (s.length() > 1) {
8          achar = s.charAt(1);
9          int i = 1;
10         while (i < s.length() - 1) {
11             achar = s.charAt(i);
12             if (!valid_f(achar))
13                 valid_id = false;
14             i++;
15         }
16     }
17
18     if (valid_id && (s.length() >= 1) && (s.length() < 6))
19         return true;
20     else
21         return false;
22 }
```

Incluindo Dependência do JUnit no Maven (1)

- ▶ Antes de iniciar a criação da classe de teste é possível incluir a dependência da biblioteca do JUnit no projeto Maven.
 1. Clique o botão direito do mouse sobre projeto Identifier.
 2. Escolha a opção **Maven->Add Dependency**. Preencha a tela conforme abaixo:



Incluindo Dependência do JUnit no Maven (2)

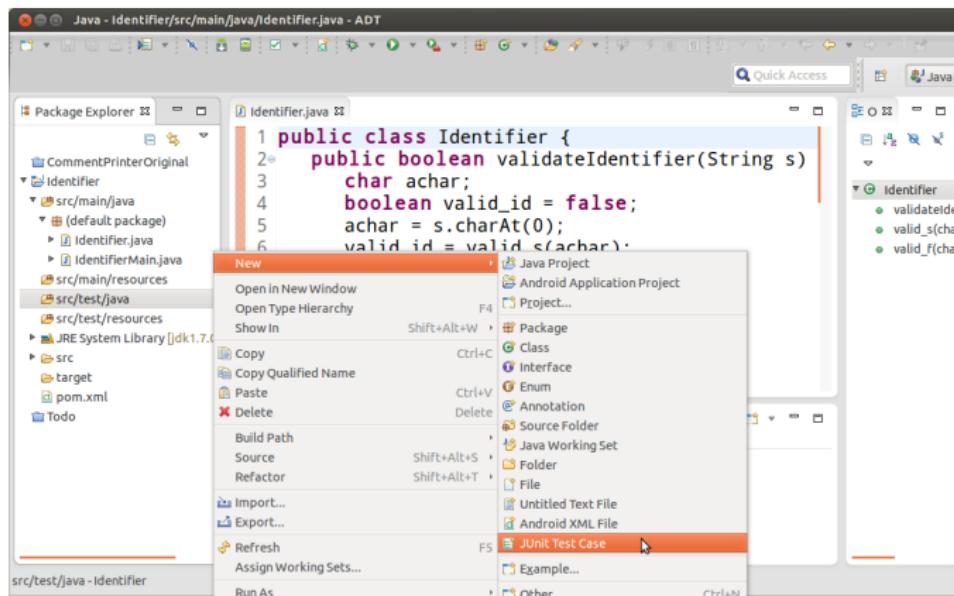
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>Identifier</groupId>
  <artifactId>Identifier</artifactId>
  <version>1.0</version>
  <name>Identifier</name>
  <description>Validador de identificador</description>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Criando o Template do JUnit (1)

- ▶ Clique com o botão direito sobre a pasta `src/test/java` no projeto Identifier.
- ▶ Escolha a opção **New->JUnit Test Case**.

Copiando a Aplicação para o Projeto

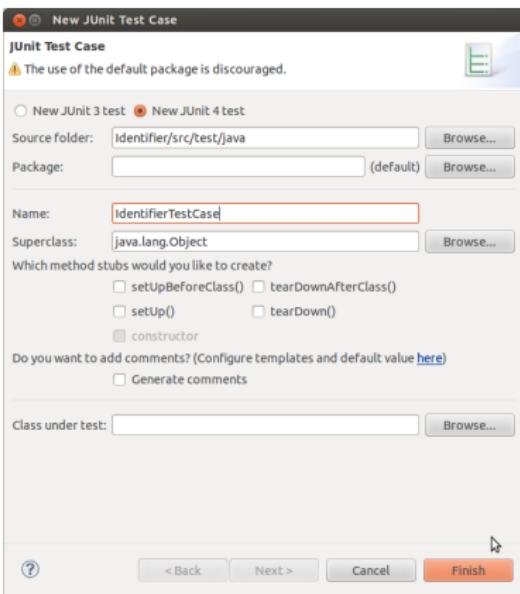
Criando o Template do JUnit (2)



Criando o Template do JUnit (3)

- ▶ Na janela que abrirá, preencha apenas o nome da classe de teste a ser criada com `IdentifierTestCase` (sem a extensão) e clique **Finish**
- ▶ Veja tela a seguir para detalhes.

Criando o Template do JUnit (4)



Criando o Template do JUnit (5)

- ▶ O código criado é bastante simples e não dá ideia do poder que existe por traz dele.

```
1 import static org.junit.Assert.*;  
2  
3 import org.junit.Test;  
4  
5  
6 public class IdentifierTestCase {  
7     @Test  
8     public void test() {  
9         fail("Not yet implemented");  
10    }  
11 }  
12  
13 }
```

Entendendo o Código (1)

- ▶ Uma classe de teste do JUnit pode conter vários casos de teste.
 - ▶ Um caso de teste é um método dentro dessa classe, precedido da anotação @Test.
 - ▶ É uma boa prática de escrita de casos de teste unitário, separar cada caso de teste em um método de teste.
 - ▶ Da mesma forma, deve-se agrupar em uma classe de teste, os casos de teste referentes a determinada classe e/ou método em teste, visando facilitar a localização dos testes.

Entendendo o Código (2)

- ▶ A primeira parte do código inclui as classes exigidas pelo framework até o momento.
- ▶ Classes de asserções do pacote `org.junit.Assert` que implementam os métodos que permitem a comparação entre o resultado obtido e o esperado.
- ▶ Classe `org.junit.Test` que implementa a anotação `@Teste`.

```
1 import static org.junit.Assert.*;  
2  
3 import org.junit.Test;
```

Entendendo o Código (3)

- ▶ Em seguida vem a definição do nome da classe de teste IdentifierTestCase que contém, até o momento, um único método de teste, denominado test().
- ▶ Na classe de teste, podem existir tantos métodos quanto necessário mas apenas aqueles precedidos da anotação @Teste é que são considerados pelo framework como sendo casos de teste.

```
6 public class IdentifierTestCase {  
7  
8     @Test  
9     public void test() {  
10         fail("Not yet implemented");  
11     }  
12 }  
13 }
```

Entendendo o Código (4)

- ▶ Um método de teste sempre deve:
 - ▶ ter modificador de acesso public
 - ▶ ter retorno do tipo void
 - ▶ não possuir parâmetros

```
8  @Test
9  public void test() {
10     fail("Not yet implemented");
11 }
```

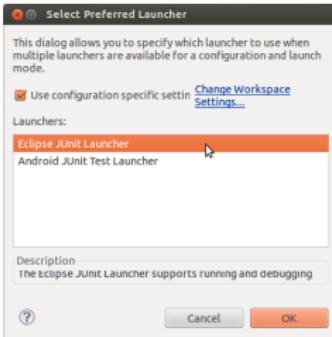
Entendendo o Código (5)

- ▶ No corpo de um método de teste basicamente o que se faz é:
 1. criar um objeto da classe que contém o método a ser testado (se o método a ser testado for um método de instância)
 2. invocar o método em teste com os parâmetros desejados e armazenar o valor de retorno
 3. utilizar uma das asserções disponíveis no framework para comparar o resultado obtido com o resultado esperado.
- ▶ No template criado, nenhum dos passos acima é realizado.
- ▶ O método contém apenas uma chamada ao método `fail`, disponível no pacote `org.junit.Assert`, que, se executado, acusa que o teste falhou.

```
8      @Test  
9      public void test() {  
10         fail("Not yet implemented");  
11     }
```

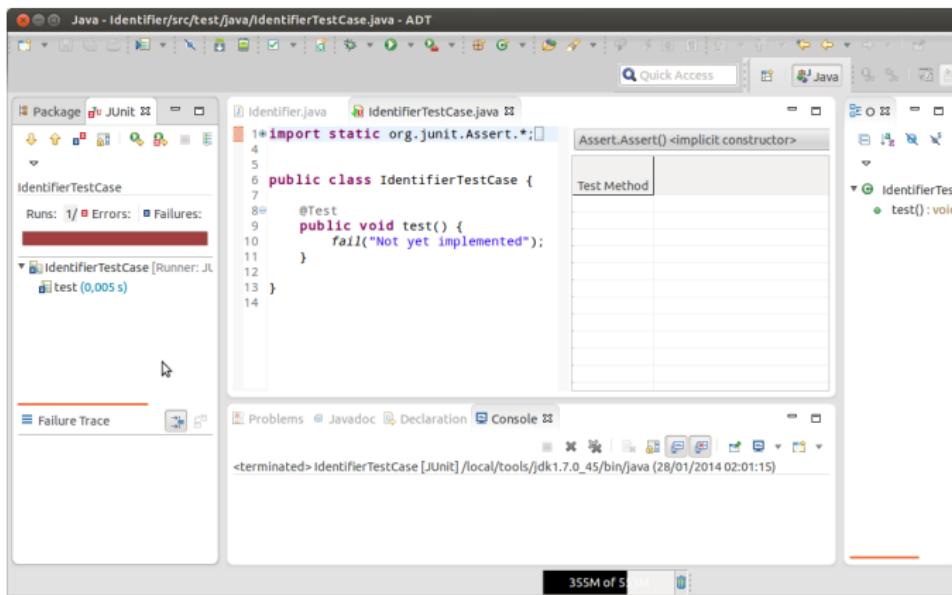
Entendendo o Código (6)

- ▶ Para executar o teste, basta clicar com o botão direito sobre o nome do arquivo de casos de teste e escolher a opção **Run As->JUnit Test**
- ▶ Na primeira execução, é solicitado ao usuário escolher qual executor de teste é para ser usado. Escolha o **Eclipse JUnit Launcher** e clique em **Ok**.



Entendendo o Código

Entendendo o Código (7)



Criando Casos de Teste (1)

- ▶ Para a criação de um caso de teste é necessário:
 - ▶ Identificar o método a ser testado: `validateIdentifier`
 - ▶ Compreender a especificação do método: o que recebe de entrada e qual a saída produzida em função da entrada escolhida.
 - ▶ Entrada: cadeia de caracteres (`String`)
 - ▶ Saída: `true` ou `false`
 - ▶ Comparar a saída produzida (obtida) com aquela que deveria ser gerada conforme a especificação.

Criando Casos de Teste (2)

- ▶ Primeiro teste completo.

```
7  @Test
8  public void validate01() {
9      Identifier id = new Identifier();
10     boolean obtido;
11     obtido = id.validateIdentifier("a1");
12     assertEquals(true, obtido);
13 }
```

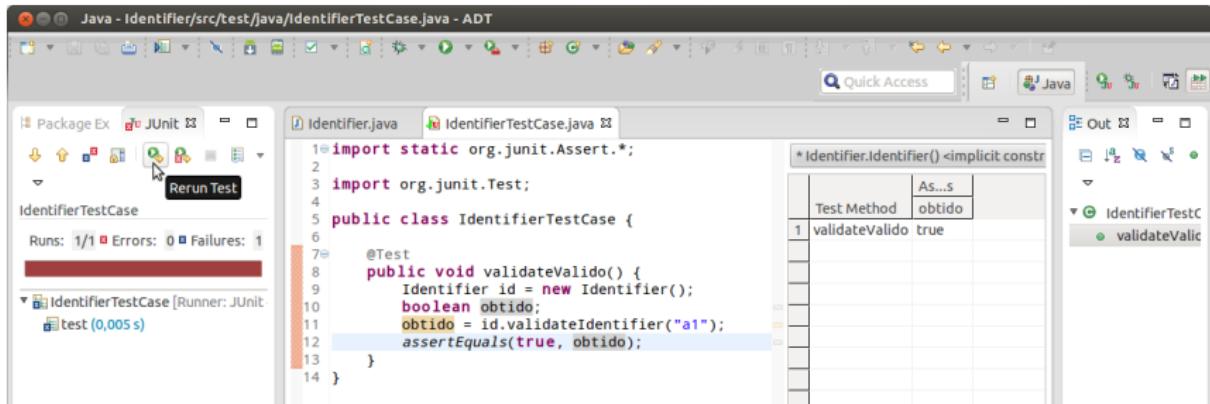
Criando Casos de Teste (3)

- ▶ Código completo da classe de teste:
IdentifierTestCase.java, conforme JUnit Versão 4.11.

```
1 import static org.junit.Assert.*;
2
3 import org.junit.Test;
4
5 public class IdentifierTestCase {
6
7     @Test
8     public void validate01() {
9         Identifier id = new Identifier();
10        boolean obtido;
11        obtido = id.validateIdentifier("a1");
12        assertEquals(true, obtido);
13    }
14}
```

Criando Casos de Teste (4)

- ▶ Novamente, para executar o teste criado, basta clicar com o botão direito sobre o nome do arquivo de casos de teste e escolher a opção **Run As->JUnit Test**; ou
 - ▶ Se a janela do JUnit continua aberta, basta clicar no ícone para reexecutar o conjunto de teste, conforme abaixo.



Criando Casos de Teste (4)

- ▶ Resultado final: teste aprovado.
- ▶ Ou seja, para a entrada "a1" que representa, conforme a especificação um identificador válido, o método validateIdentifier retornou true.
- ▶ O método assertEquals compara o resultado esperado true com o obtido, armazenado na variável obtido que, nesse caso, também continha o valor true.

The screenshot shows an IDE interface with several windows:

- Java - Identifier/src/test/java/IdentifierTestCase.java - ADT**: The active window contains Java test code:

```
import static org.junit.Assert.*;  
import org.junit.Test;  
  
public class IdentifierTestCase {  
    @Test  
    public void validateValido() {  
        Identifier id = new Identifier();  
        boolean obtido;  
        obtido = id.validateIdentifier("a1");  
        assertEquals(true, obtido);  
    }  
}
```
- Identifier.java**: Shows the implementation of the Identifier class.
- IdentifierTestSuite [Runner: JUnit]**: Shows the test suite configuration.
- Out**: Shows the test results table:

Test Method	As...s
validateValido	true

Criando Casos de Teste (5)

- ▶ Isso é tudo o que se precisa saber no momento para se criar e executar de forma automática teste com o JUnit.
- ▶ Obviamente existe muitos outros recursos que serão explorados durante o treinamento.



Vamos praticar?!

Encontrando uma Falha (1)

- ▶ Melhorar o conjunto de teste incluindo novos casos de teste

Conjunto de teste inicial

```
T0 = { ("a1",Valido), ("",Invalido),  
        ("A1b2C3d",Invalido), ("2B3",Invalido),  
        ("Z#12",Invalido) }
```

Encontrando uma Falha (2)

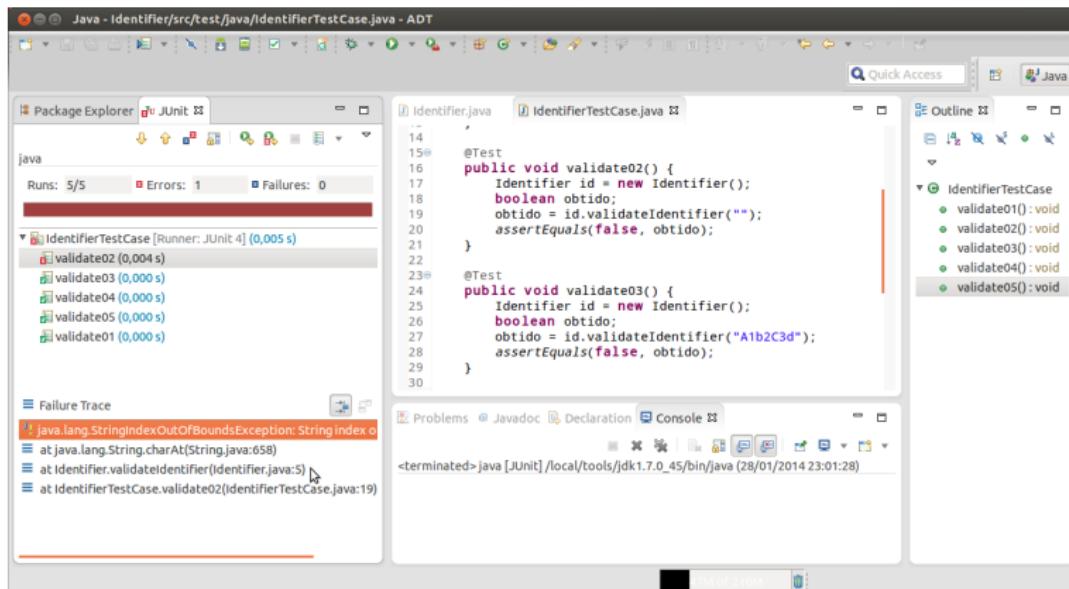
```
15     @Test
16     public void validate02() {
17         Identifier id = new Identifier();
18         boolean obtido;
19         obtido = id.validateIdentifier("");
20         assertEquals(false, obtido);
21     }
22
23     @Test
24     public void validate03() {
25         Identifier id = new Identifier();
26         boolean obtido;
27         obtido = id.validateIdentifier("A1b2C3d");
28         assertEquals(false, obtido);
29     }
```

Encontrando uma Falha (3)

```
31     @Test
32     public void validate04() {
33         Identifier id = new Identifier();
34         boolean obtido;
35         obtido = id.validateIdentifier("2B3");
36         assertEquals(false, obtido);
37     }
38
39     @Test
40     public void validate05() {
41         Identifier id = new Identifier();
42         boolean obtido;
43         obtido = id.validateIdentifier("Z#12");
44         assertEquals(false, obtido);
45     }
```

Encontrando uma Falha (4)

Executando os novos testes...



Encontrando uma Falha e Corrigindo o Defeito

Corrigindo o Defeito (1)

```
2  public boolean validateIdentifier(String s) {
3      char achar;
4      boolean valid_id = false;
5      achar = s.charAt(0);
6      valid_id = valid_s(achar);
7      if (s.length() > 1) {
8          achar = s.charAt(1);
9          int i = 1;
10         while (i < s.length() - 1) {
11             achar = s.charAt(i);
12             if (!valid_f(achar))
13                 valid_id = false;
14             i++;
15         }
16     }
17
18     if (valid_id && (s.length() >= 1) && (s.length() < 6))
19         return true;
20     else
21         return false;
22 }
```

Corrigindo o Defeito (2)

```
2     public boolean validateIdentifier(String s) {
3         char achar;
4         boolean valid_id = false;
5         if (s.length() > 0) {
6             achar = s.charAt(0);
7             valid_id = valid_s(achar);
8             if (s.length() > 1) {
9                 achar = s.charAt(1);
10                int i = 1;
11                while (i < s.length() - 1) {
12                    achar = s.charAt(i);
13                    if (!valid_f(achar))
14                        valid_id = false;
15                    i++;
16                }
17            }
18        }
19        if (valid_id && (s.length() >= 1) && (s.length() < 6))
20            return true;
21        else
22            return false;
23    }
```

Encontrando uma Falha e Corrigindo o Defeito

Corrigindo o Defeito (3)

Reexecutando os testes...

The screenshot shows the Eclipse IDE interface with the following components:

- Package Explorer:** Shows the project structure with `Identifier` and `IdentifierTest`.
- Identifier.java:** Contains the `Identifier` class with methods `validateIdentifier` and `valid_s`. The `validateIdentifier` method has several bugs, including a missing closing brace at line 25.
- IdentifierTestCase.java:** Contains five test cases: `validate01`, `validate02`, `validate03`, `validate04`, and `validate05`. `validate02` is currently selected.
- Outline View:** Shows the class `Identifier` and its methods: `validateIdentifier()`, `valid_s(char)`, `valid_id`, and `valid_f(char)`.
- Problems View:** Shows various warnings and errors related to the code.

Definindo Assertivas (1)

- ▶ JUnit oferece diversos tipos de assertivas.
- ▶ Para uma lista completa, consulte a [classe Assert na API do JUnit](#)
- ▶ Algumas mais comuns são:
 - ▶ `assertEquals(<esperado>, <obtido>)`
 - ▶ `assertNotSame(<esperado>, <obtido>)`
 - ▶ `assertSame(<esperado>, <obtido>)`
 - ▶ `assertTrue(<expressão lógica>)`
 - ▶ `assertFalse(<expressão lógica>)`
 - ▶ `assertNull(Object)`
 - ▶ `assertNotNull(Object)`
 - ▶ `fail(String)`

Definindo Assertivas (2)

- ▶ Oráculo: comparar o resultado obtido com o esperado, viabiliza a execução automática dos casos de teste.
- ▶ Para utilizar:
 - ▶ `import org.junit.Assert;`
 - ▶ Uso: `Assert.assertEquals(a,b);`
 - ▶ `import static org.junit.Assert.*;`
 - ▶ Uso: `assertEquals(a,b);`

Definindo Assertivas (cont.)

```
7  @Test
8  public void validate01() {
9      Identifier id = new Identifier();
10     boolean obtido;
11     obtido = id.validateIdentifier("a1");
12     assertEquals(true, obtido);
13 }
14
15 @Test
16 public void validate02() {
17     Identifier id = new Identifier();
18     boolean obtido;
19     obtido = id.validateIdentifier("");
20     assertEquals(false, obtido);
21 }
22
23 @Test
24 public void validate03() {
25     Identifier id = new Identifier();
26     boolean obtido;
27     obtido = id.validateIdentifier("A1b2C3d");
28     assertEquals(false, obtido);
29 }
```

Anotações Especiais do Framework (1)

- ▶ Existem situações em que, antes da execução de cada caso de teste, algumas ações precisam ser executadas.
- ▶ Em outras, após a execução de cada caso de teste é que se deve executar alguma ação.
- ▶ Duas anotações especiais são definidas:
 - ▶ `@Before` – faz com que o método anotado seja executado **antes** de cada **caso de teste**. Utilizado para colocar o programa num estado conhecido.
 - ▶ `@After` – faz com que o método anotado seja sempre executado **após** cada **caso de teste**. Utilizado, em geral, para liberar recursos utilizado pelo caso de teste.
- ▶ Para a utilização dessas anotações é necessário importar as classes `org.junit.Before` e `org.junit.After`, respectivamente.

Anotações Especiais do Framework (2)

Observe a classe de teste abaixo na qual o objeto id é inicializado em cada método de teste.

```
7  @Test
8  public void validate01() {
9      Identifier id = new Identifier();
10     boolean obtido;
11     obtido = id.validateIdentifier("a1");
12     assertEquals(true, obtido);
13 }
14
15 @Test
16 public void validate02() {
17     Identifier id = new Identifier();
18     boolean obtido;
19     obtido = id.validateIdentifier("");
20     assertEquals(false, obtido);
21 }
22
23 @Test
24 public void validate03() {
25     Identifier id = new Identifier();
26     boolean obtido;
27     obtido = id.validateIdentifier("A1b2C3d");
28     assertEquals(false, obtido);
29 }
```

Anotações Especiais (3)

- ▶ Essa inicialização pode ser feita dentro de um método anotado com `@Before`

```
1 import static org.junit.Assert.*;
2
3 import org.junit.Before;
4 import org.junit.Test;
5
6 public class IdentifierTestCase {
7
8     private Identifier id;
9
10    @Before
11    public void inicializa() {
12        id = new Identifier();
13    }
14
15    @Test
16    public void validate01() {
17        boolean obtido;
18        obtido = id.validateIdentifier("a1");
19        assertEquals(true, obtido);
20    }
}
```

Anotações Especiais (4)

```
22  @Test
23  public void validate02() {
24      boolean obtido;
25      obtido = id.validateIdentifier("");
26      assertEquals(false, obtido);
27  }
28
29  @Test
30  public void validate03() {
31      boolean obtido;
32      obtido = id.validateIdentifier("A1b2C3d");
33      assertEquals(false, obtido);
34  }
35
36  @Test
37  public void validate04() {
38      boolean obtido;
39      obtido = id.validateIdentifier("2B3");
40      assertEquals(false, obtido);
41  }
42
43  @Test
44  public void validate05() {
45      boolean obtido;
46      obtido = id.validateIdentifier("Z#12");
47      assertEquals(false, obtido);
48 }
```

Anotações Especiais do Framework (5)

- ▶ Uma novidade introduzida na versão 4.x é a possibilidade de executar algum método antes e após o início do conjunto de teste como um todo.
- ▶ Duas anotações especiais são definidas para isso:
 - ▶ `@BeforeClass` – faz com que o método anotado seja executado **antes** de cada **classe de teste**. Útil para alocar recursos computacionalmente “caros” uma única vez.
 - ▶ `@AfterClass` – faz com que o método anotado seja sempre executado **após** cada **classe de teste**. Útil para liberar recursos computacionalmente “caros” uma única vez.
- ▶ Para a utilização dessas anotações é necessário importar as classes `org.junit.BeforeClass` e `org.junit.AfterClass`, respectivamente.

Anotações Especiais do Framework (6)

Síntese das anotações especiais para inicializar e encerrar conjunto de teste e caso de teste.

@BeforeClass e @AfterClass	@Before e @After
Somente um método por classe pode ser anotado.	Múltiplos métodos podem ser anotados. A ordem de execução é indefinida. Métodos sobrescritos não são executados.
Nomes dos métodos são irrelevantes.	Nomes dos métodos são irrelevantes.
Executa uma única vez por classe de teste.	Executa antes/após cada método de teste.
Método @BeforeClass da superclasse é executado antes do da subclasse. Método @AfterClass da superclasse é executado após o da subclasse.	Métodos @Before da superclasse são executados antes dos da subclasse. Métodos @After da superclasse são executados após os da subclasse.
Método deve ser público public e estático static.	Método deve ser público public e não-estático.
É garantida a execução do método @AfterClass mesmo se um método @BeforeClass lançar uma exceção.	É garantida as execuções dos métodos @After mesmo se os métodos @Before ou @Test lancarem exceções.

Adaptada de Gonçalves (2006).



Introdução

Histórico

Pré-requisitos

JUnit Básico

Instalação

O Framework

O Exemplo

- Compilando e Executando via Linha de Comando
- Projeto Maven no Eclipse
- Copiando a Aplicação para o Projeto
- Entendendo o Código
- Criando Casos de Teste
- Encontrando uma Falha e Corrigindo o Defeito
- Métodos Especiais

JUnit Avançado

- Temporização em Casos de Teste
- Teste de Exceções
- Definindo um Conjunto de Teste
- Executando os Teste sem o Eclipse
- Ignorando Casos de Teste
- Testes Parametrizados

Ferramentas Similares

Referências

Caso de Teste Temporizado (1)

- ▶ Não presente nas versões anteriores à versão 4.x
- ▶ Casos de testes devem executar rapidamente.
- ▶ Impede que um caso de teste execute indefinidamente em caso de looping.
- ▶ Útil no teste de conexões com servidores ou banco de dados.
 - ▶ Caso o servidor esteja lento ou não respondendo, os casos de testes podem continuar a execução.

Caso de Teste Temporizado (2)

Considere a existência de um defeito na linha 15, conforme ilustrado abaixo, no método validateIdentifier.

```
2  public boolean validateIdentifier(String s) {
3      char achar;
4      boolean valid_id = false;
5      if (s.length() > 0) {
6          achar = s.charAt(0);
7          valid_id = valid_s(achar);
8          if (s.length() > 1) {
9              achar = s.charAt(1);
10             int i = 1;
11             while (i < s.length() - 1) {
12                 achar = s.charAt(i);
13                 if (!valid_f(achar))
14                     valid_id = false;
15                 //i++; //Defeito
16             }
17         }
18     }
19     if (valid_id && (s.length() >= 1) && (s.length() < 6))
20         return true;
21     else
22         return false;
23 }
```

Exercício: Compile e tente executar com os casos de testes atuais.

Caso de Teste Temporizado (3)

A execução do teste 3 está em looping...

The screenshot shows the Eclipse IDE interface with the following details:

- Java - Identifier/src/main/java/Identifier.java - ADT**: The current file is `Identifier.java`. The code defines a class `Identifier` with a method `validateIdentifier` that loops through the characters of a string `s` to check if it's a valid identifier. A bug is present where the loop continues even if the identifier is valid.
- IdentifierTestCase [Runner: JUnit 4]**: The test case contains five methods: `validate02`, `validate03` (selected), `validate04`, `validate05`, and `validate01`. The status bar indicates "Runs: 2/5", "Errors: 0", and "Failures: 0".
- Outline View**: Shows the structure of the `Identifier` class, including its methods: `validateIdentifier`, `valid_s(char)`, and `valid_f(char)`.
- Failure Trace**: No failure trace is shown.
- Problems View**: No problems are listed.

Caso de Teste Temporizado (4)

Considere que a classe de teste fosse alterada como ilustrado abaixo.

```
1 import static org.junit.Assert.*;
2
3 import org.junit.Before;
4 import org.junit.Test;
5
6 public class IdentifierTestCase {
7
8     private Identifier id;
9     public static final int LIMIT=200;
10
11     ...
12
13     @Test(timeout=LIMIT)
14     public void validate03() {
15         boolean obtido;
16         obtido = id.validateIdentifier("A1b2C3d");
17         assertEquals(false, obtido);
18     }
19
20     @Test(timeout=LIMIT)
21     public void validate04() {
22         boolean obtido;
23         obtido = id.validateIdentifier("2B3");
24         assertEquals(false, obtido);
25     }
}
```

Caso de Teste Temporizado (5)

Após o timeout a execução termina acusando problemas na execução dos testes...

```
Java - Identifier - src/test/java/IdentifierTestCase.java - ADT
-----
1 import static org.junit.Assert.*;
2
3 import org.junit.Before;
4 import org.junit.Test;
5
6 public class IdentifierTestCase {
7
8     private Identifier id;
9     public static final int LIMIT=200;
10
11     @Before
12     public void inicializa() {
13         id = new Identifier();
14     }
15
16     @Test(timeout=LIMIT)
17     public void validate01() {
18         boolean obtido;
19         obtido = id.validateIdentifier("a1");
20         assertEquals(true, obtido);
21     }
22
23     @Test(timeout=LIMIT)
24     public void validate02() {
25         boolean obtido;
```

Caso de Teste Temporizado (6)

Após o correção do defeito, os testes voltam a executar normalmente...

The screenshot shows the Eclipse IDE interface. The Java Editor on the right displays the `Identifier.java` file with the following code:

```
1 public class Identifier {
2     public boolean validateIdentifier(String s) {
3         char achar;
4         boolean valid_id = false;
5         if (s.length() > 0) {
6             achar = s.charAt(0);
7             valid_id = valid_s(achar);
8             if (s.length() > 1) {
9                 achar = s.charAt(1);
10                int i = 1;
11                while (i < s.length() - 1) {
12                    achar = s.charAt(i);
13                    if (!valid_f(achar))
14                        valid_id = false;
15                    i++;
16                }
17            }
18            if (valid_id && (s.length() >= 1) && (s.length() < 6))
19                return true;
20            else
21                return false;
22        }
23    }
24
25    public boolean valid_s(char ch) {
```

The JUnit View on the left shows the results of the test run:

- Runs: 5/5
- Errors: 0
- Failures: 0

The results are listed as:

- validate01 (0,001 s)
- validate02 (0,000 s)
- validate03 (0,000 s)
- validate04 (0,000 s)
- validate05 (0,001 s)

The status bar at the bottom indicates "71M".

Testando Exceções (1)

Embora o exemplo em questão não lance nenhum tipo de exceção, a criação de um caso de teste para avaliar se uma exceção correta está sendo lançada é ilustrado abaixo:

```
51  @Test(expected = IndexOutOfBoundsException.class)
52  public void excecaoString() {
53      String str = new String("Exemplo JUnit");
54      str.substring(30);
55  }
```

Testando Exceções (2)

Resultado do teste de exceção: aprovado.

Java - Identifier/src/test/java/IdentifierTestCase.java - ADT

Package Explorer JUnit

Finished after 0,013 seconds

Runs: 6/6 Errors: 0 Failures: 0

IdentifierTestCase [Runner: JUnit 4] (0,002 s)

- validate01 (0,000 s)
- validate02 (0,000 s)
- validate03 (0,000 s)
- validate04 (0,000 s)
- validate05 (0,001 s)
- excecaoString (0,001 s)

Failure Trace

Identifier.java

```
43     @Test(timeout = LIMIT)
44     public void validate05() {
45         boolean obtido;
46         obtido = id.validateIdentifier("Z#12");
47         assertEquals(false, obtido);
48     }
49
50     @Test(expected = IndexOutOfBoundsException.class)
51     public void excecaoString() {
52         String str = new String("Exemplo JUnit");
53         str.substring(30);
54     }
55
56 }
```

IdentifierTestCase

- id : Identifier
- LIMIT : int
- inicializa() : void
- validate01() : void
- validate02() : void
- validate03() : void
- validate04() : void
- validate05() : void
- excecaoString() : void

Problems Javadoc Declaration Console

Writable Smart Insert 67:1 114M of 16G

Definindo um Conjunto de Teste (1)

- ▶ Um conjunto de teste serve para agrupar várias classes de teste e executá-las em sequência.
- ▶ Na versão 4.x isso é feito por meio de anotações:
 - ▶ `@RunWith` – define qual a classe responsável pela execução dos testes. Quando não especificada, o JUnit utiliza a classe padrão `org.junit.internal.runners.TestClassRunner`.
 - ▶ `@Suite` – define o nome das classes de teste que irão compor o conjunto de teste.
- ▶ Para utilizar essas anotações é necessário importar `org.junit.runner.RunWith` e `org.junit.runners.Suite`, respectivamente.

Definindo um Conjunto de Teste (2)

Exemplo de criação de um conjunto de teste (`TodosTestes.java`) incluindo uma única classe:

```
1 import org.junit.runner.RunWith;
2 import org.junit.runners.Suite;
3
4 @RunWith(Suite.class)
5 @Suite.SuiteClasses({
6     IdentifierTestCase.class,
7     // Incluir outras classes de teste aqui.
8 })
9
10 public class TodosTestes {
11 }
```

Definindo um Conjunto de Teste

Definindo um Conjunto de Teste (3)

A execução de TodosTestes invoca a classe de teste IdentifierTestCase...

The screenshot shows the Eclipse IDE interface with the following details:

- Java - Identifier/src/test/java/TodosTestes.java - ADT**: The active workspace.
- Package Explorer**: Shows a green progress bar at the top with the text "Finished after 0,013 seconds". Below it, it says "Runs: 6/6 Errors: 0 Failures: 0". A tree view shows the test suite `TodosTestes [Runner: JUnit 4] (0,001 s)` and its test cases: `IdentifierTestCase (0,001 s)`, `validate01 (0,000 s)`, `validate02 (0,000 s)`, `validate03 (0,000 s)`, `validate04 (0,001 s)`, `validate05 (0,000 s)`, and `excecaoString (0,000 s)`.
- Editor Area**: Displays the Java code for `TodosTestes.java`. The code includes imports for `org.junit.runner.RunWith` and `org.junit.runners.Suite`, and annotations `@RunWith(Suite.class)` and `@Suite.SuiteClasses({ IdentifierTestCase.class })`. It also defines a public class `TodosTestes` with a single brace on line 10.
- Outline View**: Shows the outline of the current file, including the `TodosTestes` class.
- Problems, Javadoc, Declaration, Console**: Standard Eclipse toolbars at the bottom.

Executando os Teste sem o Eclipse

- ▶ Embora a execução dentro do IDE seja uma facilidade para o desenvolvedor, em algumas ocasiões é melhor a execução fora do ambiente interativo, por exemplo, para a execução via scripts
- ▶ O JUnit oferece mais de uma opção para a execução dos testes fora do IDE, dentre elas:
 - ▶ Via Maven;
 - ▶ Via executor do JUnit – `org.junit.runner.JUnitCore`;
 - ▶ Via executor personalizado – `MyTestRunner`, por exemplo.

Execução via Maven

- ▶ Executar o comando `mvn test`, a partir do diretório no qual o arquivo `pom.xml` se encontra.

```
Identifier$ mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] Building Identifier
[INFO]   task-segment: [test]
[INFO]
[INFO]
[INFO] ...
[INFO] -----
[INFO] T E S T S
[INFO] -----
Running IdentifierTestCase
Tests run: 6, Failures: 0, Errors: 0, Skipped: 1, Time elapsed: 0.049 sec
Running ParameterizedTestCase
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.006 sec

Results :

Tests run: 9, Failures: 0, Errors: 0, Skipped: 1

[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 1 second
[INFO] Finished at: Thu Jan 30 20:47:20 BRST 2014
[INFO] Final Memory: 12M/148M
```

Execução via JUnitCore

- ▶ A partir do diretório no qual o arquivo pom.xml se encontra, execute:

```
Identifier$ java -cp target/Identifier-1.0.jar:target/test-classes:\n> ../../ferramentas/junit4.11/junit-4.11.jar:\n> ../../ferramentas/junit4.11/hamcrest-core-1.3.jar \\\n> org.junit.runner.JUnitCore TodosTestes\nJUnit version 4.11\n.\n.\n.\nTime: 0,011\n\nOK (8 tests)
```

- ▶ O JUnitCore recebe de parâmetro o nome da(s) classe(s) de teste que se deseja executar
- ▶ Observe, nesse caso, a necessidade de indicar para a máquina virtual Java a localização dos diretórios onde se encontram as classes da aplicação e de teste, além das bibliotecas do JUnit (junit-4.11.jar e hamcrest-core-1.3.jar).

Execução via MyTestRunner (1)

- É possível ainda construir um executor personalizado:

```
1 import org.junit.runner.JUnitCore;
2 import org.junit.runner.Result;
3 import org.junit.runner.notification.Failure;
4
5 public class MyTestRunner {
6     public static void main(String[] args) {
7         Result result = JUnitCore.runClasses(TodosTestes.class);
8
9         System.out.println(result.getRunCount() + " tests run on "
10                    + result.getRunTime() + "ms");
11         System.out.println("Ignored tests: " + result.getIgnoreCount());
12         System.out.println("Failures detected: " + result.getFailures().size());
13         for (Failure failure : result.getFailures()) {
14             System.out.println(failure.toString());
15         }
16     }
17 }
```

Execução via MyTestRunner (2)

- ▶ A partir do diretório no qual o arquivo pom.xml se encontra, execute:

```
Identifier$ java -cp target/Identifier-1.0.jar:target/test-classes:\n..\..\ferramentas/junit4.11/junit-4.11.jar:\n..\..\ferramentas/junit4.11/hamcrest-core-1.3.jar \\\nMyTestRunner\n\n8 tests run on 9ms\nIgnored tests: 1\nFailures detected: 0
```

- ▶ Novamente, nesse caso, a necessidade de indicar para a máquina virtual Java a localização dos diretórios onde se encontram as classes da aplicação e de teste, além das bibliotecas do JUnit (junit-4.11.jar e hamcrest-core-1.3.jar).

Ignorando Casos de Teste (1)

- ▶ Existem situações que requerem que algum caso de teste deixe de ser executado:
 - ▶ Caso de teste incompleto.
 - ▶ Caso de teste toma muito tempo da execução.
 - ▶ O recurso exigido pelo caso de teste ainda não se encontra disponível.
 - ▶ O caso de teste está acusando uma falha que ainda não será corrigida.

Ignorando Casos de Teste (2)

- ▶ Remover a anotação `@Test` do método de teste.
- ▶ Comentar o método de teste.
- ▶ Excluir o método de teste da classe de teste.

Embora funcionem, o executor de casos de teste deixará de reportar a existência de tais testes.

Ignorando Casos de Teste (3)

- ▶ Na versão 4.x é possível simplesmente ignorar um caso de teste.
- ▶ Uso da anotação `@Ignore` ou `@Ignore("Mensagem")`:

```
54     @Ignore
55     @Test(expected = IndexOutOfBoundsException.class)
56     public void excecaoString() {
57         String str = new String("Exemplo JUnit");
58         str.substring(30);
59     }
```

Ignorando Casos de Teste (4)

Saída produzida:

The screenshot shows the Eclipse IDE interface with the following details:

- Package Explorer:** Shows the project structure with "IdentifierTestC" selected.
- JUnit View:** Displays the test results:
 - Finished after 0,035 seconds
 - Runs: 6/6 (1 ignored)
 - Errors: 0
 - Failures: 0
- Code Editor:** Shows the Java code for `IdentifierTestCase.java`. The code includes several test methods and one ignore annotation.

```
44
45    @Test(timeout = LIMIT)
46    public void validate05() {
47        boolean obtido;
48        obtido = id.validateIdentifier("Z#12");
49        assertEquals(false, obtido);
50    }
51
52    @Ignore
53    @Test(expected = IndexOutOfBoundsException.class)
54    public void excecaoString() {
55        String str = new String("Exemplo JUNIT");
56        str.substring(30);
57    }
58
59
60
61
62
63
64
65
66
67
68}
```
- Outline View:** Shows the class structure with methods: `id:Identifier`, `inicializa():void`, `validate01():void`, `validate02():void`, `validate03():void`, `validate04():void`, `validate05():void`, and `excecaoString():void`.
- Bottom Status Bar:** Shows "94M of 1".

Testes Parametrizados (1)

- ▶ Considere os casos de teste abaixo:

```

17     @Test(timeout = LIMIT)
18     public void validate01() {
19         boolean obtido;
20         obtido = id.validateIdentifier("a1");
21         assertEquals(true, obtido);
22     }
23
24     @Test(timeout = LIMIT)
25     public void validate02() {
26         boolean obtido;
27         obtido = id.validateIdentifier("");
28         assertEquals(false, obtido);
29     }

```

```

31     @Test(timeout = LIMIT)
32     public void validate03() {
33         boolean obtido;
34         obtido =
35             id.validateIdentifier("A1b2C3d");
36         assertEquals(false, obtido);
37     }
38
39     @Test(timeout = LIMIT)
40     public void validate04() {
41         boolean obtido;
42         obtido = id.validateIdentifier("2B3");
43         assertEquals(false, obtido);
44     }
45
46     @Test(timeout = LIMIT)
47     public void validate05() {
48         boolean obtido;
49         obtido =
50             id.validateIdentifier("Z#12");
51         assertEquals(false, obtido);
52     }

```

- ▶ O que eles possuem em comum?

Testes Parametrizados (2)

- ▶ Todos testam o mesmo método `validateIdentifier`.
- ▶ Possuem uma estrutura bastante semelhante.
 - ▶ Variam o parâmetro fornecido ao `validateIdentifier` e o valor esperado.
- ▶ Em situações como essa é possível definir um conjunto de teste parametrizado.
- ▶ Basicamente isso significa:
 - ▶ Criar uma coleção com valores de parâmetros e saídas esperadas.
 - ▶ Definir a estrutura do método a ser testado em função dos valores da coleção.
- ▶ Duas anotações especiais são utilizadas:
`@RunWith(Parameterized.class)` e `@Parameters`.

Testes Parametrizados (3)

```

1 import static org.junit.Assert.*;
2 import org.junit.Before;
3 import org.junit.Test;
4 import org.junit.runner.RunWith;
5 import org.junit.runners.Parameterized;
6 import org.junit.runners.Parameterized.Parameters;
7
8 import java.util.Arrays;
9 import java.util.Collection;
10
11 @RunWith(Parameterized.class)
12 public class ParameterizedTestCase {
13
14     private Identifier id = new Identifier();
15
16     private String param;
17     private boolean result;
18
19     @Parameters
20     public static Collection<Object[]> data() {
21         return Arrays.asList(new Object[][] {
22             { "Abcd5", true },
23             { "&123", false },
24             { "Inv@lido", false }
25         });
26     }

```

```

29     String param, boolean result) {
30         this.param = param;
31         this.result = result;
32     }
33
34     @Before
35     public void inicializa() {
36         id = new Identifier();
37     }
38
39     @Test(timeout = 200)
40     public void validate() {
41         boolean value =
42             id.validateIdentifier(param);
43         assertEquals(result, value);
44     }
45

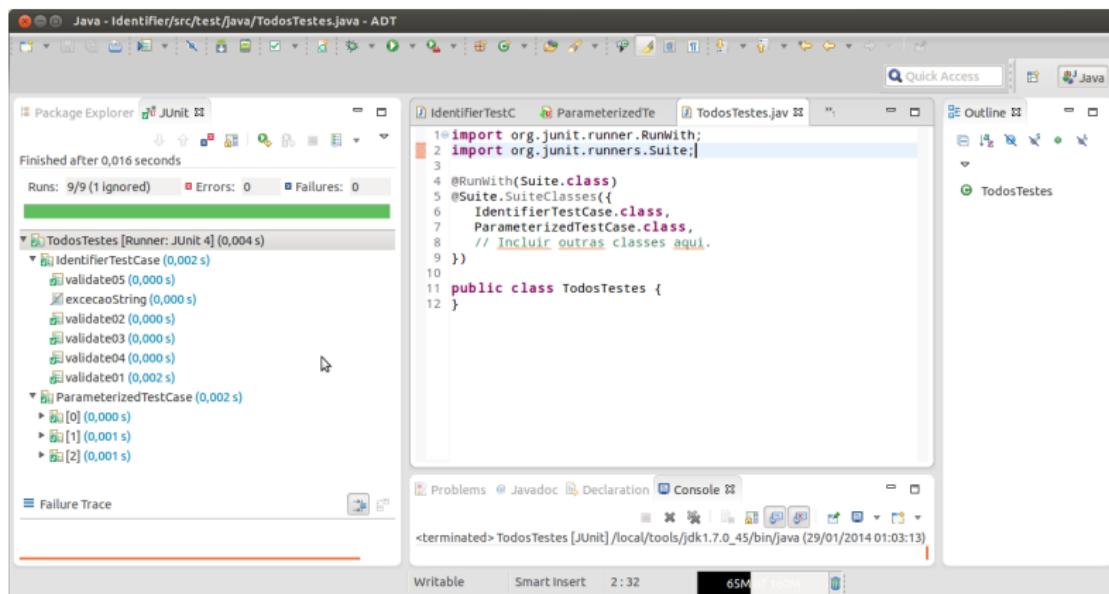
```

Atualizando TodosTestes (1)

```
1 import org.junit.runner.RunWith;
2 import org.junit.runners.Suite;
3
4 @RunWith(Suite.class)
5 @Suite.SuiteClasses({
6     IdentifierTestCase.class,
7     ParameterizedTestCase.class,
8     // Incluir outras classes de teste aqui.
9 })
10
11 public class TodosTestes {
12 }
```

Atualizando TodosTestes (2)

A execução de TodosTestes invoca as classes de teste IdentifierTestCase e ParameterizedTestCase...



Outros Recursos para Pesquisar

- ▶ Para mais informações sobre as novidades do JUnit consulte o [Sumário de Mudanças da versão 4.11](#)
 - ▶ Algumas delas não abordadas nesse treinamento são:
 - ▶ Definindo a ordem de execução dos testes: `@FixMethodOrder` ([mais informação aqui e aqui](#))
 - ▶ Melhorando a identificação de teste parametrizados: `@Parameters` ([mais informação aqui e aqui](#))
 - ▶ Declaração de suposições sobre as condições para o teste ter significado: `Assume` ([mais informação aqui](#))
 - ▶ Definição de regras: `@Rule` e `@RuleClass` ([mais informação aqui](#))



Introdução

Histórico

Pré-requisitos

JUnit Básico

Instalação

O Framework

O Exemplo

- Compilando e Executando via Linha de Comando
- Projeto Maven no Eclipse
- Copiando a Aplicação para o Projeto
- Entendendo o Código
- Criando Casos de Teste
- Encontrando uma Falha e Corrigindo o Defeito
- Métodos Especiais

JUnit Avançado

- Temporização em Casos de Teste
- Teste de Exceções
- Definindo um Conjunto de Teste
- Executando os Teste sem o Eclipse
- Ignorando Casos de Teste
- Testes Parametrizados

Ferramentas Similares

Referências

Ferramentas Similares

- ▶ JUnit não é a única ferramenta de apoio a documentação e execução automática de casos de teste.
- ▶ Considerando as linguagens de programação mais populares, existem outras ferramentas similares, tais como:
 - ▶ TestNG (<http://testng.org/>) para Java.
 - ▶ DUnit (<http://dunit.sourceforge.net/>) para Delphi.
 - ▶ cUnit (<http://sourceforge.net/projects/cut/>) para C.
 - ▶ Jeté (<http://jete.sourceforge.net/>) teste de integração para Java.
 - ▶ Dentre outras...
 - ▶ Uma extensa lista pode ser encontrada em
<http://www.testingfaqs.org/> e
<http://www.opensourcetesting.org/>.



Introdução

Histórico

Pré-requisitos

JUnit Básico

Instalação

O Framework

O Exemplo

- Compilando e Executando via Linha de Comando
- Projeto Maven no Eclipse
- Copiando a Aplicação para o Projeto
- Entendendo o Código
- Criando Casos de Teste
- Encontrando uma Falha e Corrigindo o Defeito
- Métodos Especiais

JUnit Avançado

- Temporização em Casos de Teste
- Teste de Exceções
- Definindo um Conjunto de Teste
- Executando os Teste sem o Eclipse
- Ignorando Casos de Teste
- Testes Parametrizados

Ferramentas Similares

Referências

Referências

- ▶ JUnit Homepage: package, documentation and extensions –
<http://www.junit.org/>
 - ▶ JUnit Start Guide – <http://www.diasparsoftware.com/articles/JUnit/jUnitStarterGuide.html>
 - ▶ Harold, E. "An early look at JUnit 4". Artigo On-line, Setembro, 2005. Disponível em:
<http://www.ibm.com/developerworks/java/library/j-junit4.html>. Acesso em: 22/11/2006.
 - ▶ Gonçaves, A. "Get Acquainted with the New Advanced Features of JUnit 4". Artigo On-line, Julho, 2006. Disponível em:
<http://www.devx.com/Java/Article/31983>. Acesso em: 11/22/2006.
 - ▶ Open Source Java Developer Testing Tools – Coletânea de endereços para várias ferramentas de teste de código livre (não somente para programas Java) –
<http://www.opensourcetesting.org/>
 - ▶ Software Testing FAQs – Coletânea de endereços para várias ferramentas de teste – <http://www.testingfaqs.org/>
 - ▶ Open Source Testing Tools in Java –
<http://java-source.net/open-source/testing-tools/>