Wilkerson de Lucena Andrade

wilkerson.andrade[at]gmail.com

Introdução

- JUnit é um framework open source usado para o desenvolvimento e execução de testes escritos em Java
- Desenvolvido por Eric Gamma e Kent Beck
- A maioria das IDEs incorporam o JUnit dentro de seu ambiente de desenvolvimento
  - JBuilder
  - JDeveloper
  - Netbeans
  - Eclipse

Como usar o JUnit?

- Depende da metodologia de testes que está sendo usada:
  - Código existente
  - Desenvolvimento guiado por testes (TDD)
- Onde obter o JUnit?
  - **www.junit.org**
- Como instalar?
  - Incluir o arquivo **junit.jar** no classpath

Testando código com JUnit

- Crie uma classe de teste para cada classe a ser testada

```
Public class MyClassTest {

...

}
```

Testando código com JUnit

- Para cada método **`xxx(args)`** a ser testado defina um método **`@Test public void xxx()`**:
  - MyClass:

```
public String setObject(Object o) {
...
}
```

  - MyClassTest:

```
@Test public void setObject() {...}
```

O que colocar em um teste?

- Cada método criado na sua classe de teste pode ser um caso de teste
  - Escreva o código para verificar o correto funcionamento da unidade de código dentro deste método
- Use asserções do JUnit para verificar os resultados do teste e para causar falhas se o resultado não for o esperado

O que colocar em um teste?

Método	Descrição
assertTrue	Verifica se uma condição é verdade
assertFalse	Verifica se uma condição é falsa
assertEquals	Verifica se dois objetos são iguais
assertNotNull	Verifica se um objeto não é null
assertNull	Verifica se um objeto é null
assertSame	Verifica se dois objetos apontam para um mesmo objeto
assertNotSame	Verifica se dois objetos não apontam para um mesmo objeto
fail	Faz com que um teste falhe

Como executar um teste?

- Para executar digite:
  - java -classpath .;dir/junit-4.4.jar **org.junit.runner.JUnitCore** [classes de teste]

Como funciona?

- Para cada método de teste **`public void xxx()`**, a ferramenta executa:
  - O método anotado com **`@Before`**
  - O próprio método **`xxx()`**
  - O método anotado com **`@After`**
- Um teste pode **terminar**, **falhar** ou causar uma **exceção**

Anotações

Anotação	Descrição
@BeforeClass	Métodos invocados antes da execução da suíte de teste
@AfterClass	Métodos invocados após a execução da suíte de teste
@Before	Métodos que são executados antes de todos os testes
@After	Métodos que são executados depois de todos os testes
@Test	Métodos reais de teste
@Ignore	Testes que ainda não foram implementados podem ser desabilitados temporariamente

Fixture

- São os dados utilizados por vários testes

```
public class CollectionNames {
  protected Collection<String> stringCollection;
  @Before public void setUp() throws Exception {
    stringCollection = new ArrayList<String>();
    stringCollection.add("Maria");
  }

  @Test public void testLength(){
    assertEquals(1, stringCollection.size());
  }

  @Test public void testToString(){
    assertEquals("[Maria]", stringCollection.toString());
  }
...
```

Teste de situações de falha

```
@Test(expected=ProductException.class)
public void testInvalidCode() {
    Product product = new Product(-2);
}

public void testInvalidCode() {
   try {
     Product product = new Product(-2);
     fail("Should have caused Exception!");
   } catch (Exception e) {
     assertNotNull(e.getMessage());
}
```

TestSuite

- Representa uma composição de testes
- Boa prática: crie uma classe **`AllTests`** em cada pacote de testes

```
@RunWith(Suite.class)
@SuiteClasses({CelsiusTemperatureTest.class,
                    FahrenheitTemperatureTest.class})
public class AllTests {
}
```

TestSuite

- Boa prática: crie uma classe para a execução de todos os testes da sua aplicação
  - Inclua nesta classe as suites de teste de cada pacote

```
@RunWith(Suite.class)
@SuiteClasses({tempconverter.app.AllTests.class,
                    tempconverter.scales.AllTests.class})
public class AllTests {
}
```