Testes De Software

Professor Gilmar Luiz de Borba

Prática.

Objetivos:

- Praticar a partir de um exemplo a criação de testes com MOCKS.
- Entender o processo de criação de um teste usando MOCKS.
- Saber quando usar MOCKS.

1 – O que são MOCKS?

Segundo DevMedia (www.devmedia.com), O termo "Mock Objects" é utilizado para descrever casos especiais de objetos que imitam objetos reais para teste. Características principais:

1. São criados através de frameworks;
2. São usados nas principais linguagens;
3. São uma forma de implementar objetos de testes;
4. São bastante difundidos na comunidade de métodos ágeis (XP);
5. São usados para simular objetos durante o processo do TDD;
6. São usados no aprofundamento da realização de testes tornando muito útil para os desenvolvedores de software.

Há várias ferramentas de auxílio para criação de Mocks

Para plataforma JAVA:

JMockit, Mockito, EasyMock, JMock, MockCreator, MockLib e HibernateMock.

Para plataforma .NET

NMockLib, Rhino Mocks, NMock e NMock 2 TypeMock.

Na presente atividade usaremos o MOCKITO por ser um dos mais populares e mais usados no mercado.

2 – Como habilitar a biblioteca MOCKITO?

2.1 - Baixar o MOCKITO (arquivo: mockito-all-1.9.5.jar)

https://code.google.com/p/mockito/downloads/detail?name=mockito-all-1.9.5.jar&can=2&q=

2.2 - Colocar o MOCKITO em: LIB: C:\Program Files\Java\jdk1.8.0_60\lib

... ou dentro da própria pasta do projeto.

2.3 – Importar o MOCKITO no ambiente Eclipse:

1. Selecionar o projeto
2. Botão direito do mouse
3. Escolher a opção Build Path
4. Escolher Add External Archives
5. Escolher o arquivo: mockito-all-1.9.5.jar

3 – Sobre o Projeto

O projeto consiste em uma interface chamada "***ICalculadora***" que expõe um único método chamado "adicionar". Esse método tem como parâmetros dois inteiros ("x" e "y").

O projeto possui outra classe chamada "***ChamaServico***" que possui o método "adicionardoisNumeros(x,y)" que empacota o método adicionar da interface.

```
public int adicionaDoisNumeros(int x, int y) {
      -- método empacotado
      return calc.adicionar(x, y);
}
```

A classe "*ChamaServico*" tem como objetivo criar um tipo ICalculadora, que invoca o método "adicionar" a partir do método, "*adicionarDoisNumeros*".

O projeto consistirá em criar uma classe de teste chamada "*ChamaServicoTeste*" (JUnitTestCase), que invocará o método (adicionarDoisNumeros), porém haverá um erro do tipo *NullPointerException* pois não existe a implementação na classe concreta do método "*adicionar()*". Como resolver o problema?

4 – Criação do projeto

Criar um novo projeto: **PrjMockCalculadora**
Criar um Package para o projeto: pkgCalculadora

5 –Criar a Interface "ICalculadora".

```
package pkg_Calculadora;
public interface ICalculadora {
    public int adicionar(int x, int y);
}
```

c:\prof_gilmar_borba\imagens\auditoria_testes\

6 –Criar a classe "ChamaServico" com um método de nome *"adicionarDoisNumeros"* com os métodos *getters* e *setters*.

```
package pkg_Calculadora;
public class ChamaServico {
    ICalculadora calc;

    public int adicionaDoisNumeros(int x, int y) {
        return calc.adicionar(x, y);
    }

    public ICalculadora getCalc() {
        return calc;
    }

    public void setCalc(ICalculadora calc) {
        this.calc = calc;
    }

}
```

c:\prof_gilmar_borba\imagens\auditoria_testes\

7 –Criar a classe "ChamaServicoTeste" (JunitTest CASE). Siga os passos:

1 - Somente habilite o método SETUP
2 - A classe base deve ser: ChamaServico.
3 – Na classe ChamaServicoTeste, criar o objeto do tipo ChamaServico (SETUP - *@Before*)
4 – Implementar o método assertEquals (pode sobrescrever o existente - *@Test*).
3 - Veja o código completo:

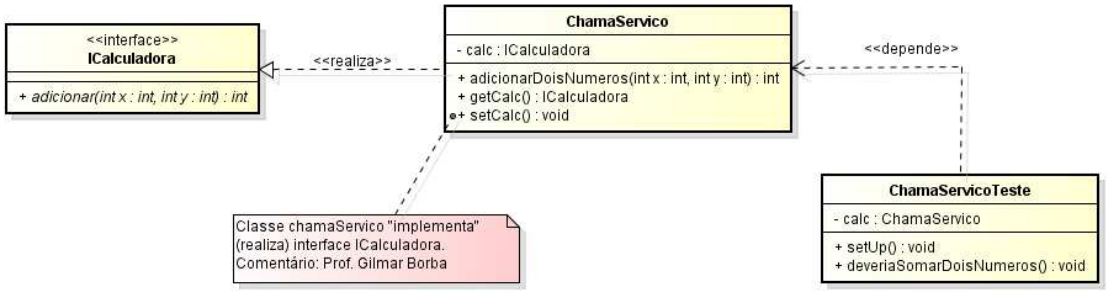
```
package pkgCalculadora;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class ChamaServicoTeste {

    ChamaServico calc;
    @Before
    public void setUp() throws Exception {
        calc = new ChamaServico();
    }

    @Test
    public void deveriaSomarDoisNumeros() {
        assertEquals(10, calc.adicionaDoisNumeros(4, 6),0);
    }

}
```

c:\prof_gilmar_borba\imagens\auditoria_testes\

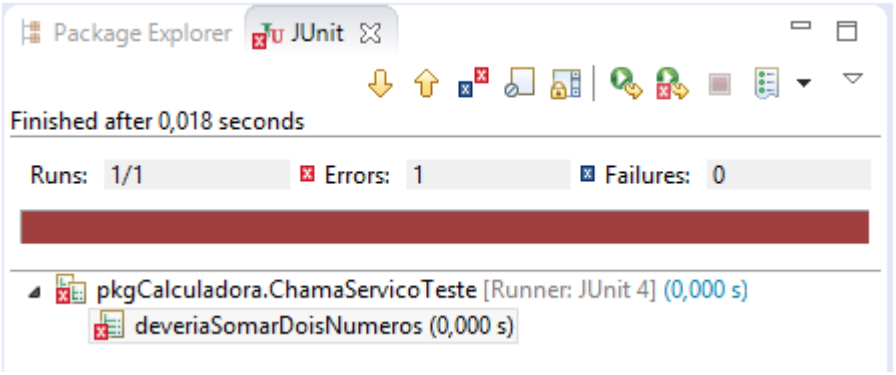
Testes De Software

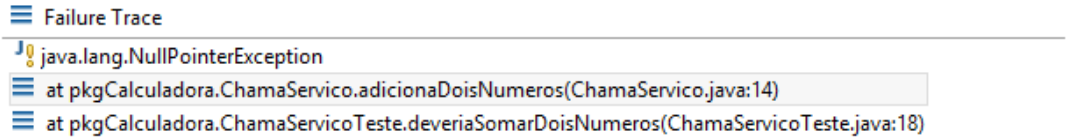
Professor Gilmar Luiz de Borba

CLASSES DO PROJETO



8 –Executar o teste, veja o resultado.





c:\prof_gilmar_borba\imagens\auditoria_testes\

9 –Habilitar/adicionar a biblioteca mockito ao projeto.

(9.1) Baixar o MOCKITO (arquivo: mockito-all-1.9.5.jar) em:
https://code.google.com/p/mockito/downloads/detail?name=mockito-all-1.9.5.jar&can=2&q=

(9.2) Colocar o MOCKITO me LIB: C:\Program Files\Java\jdk1.8.0_60\lib

(9.3) Importar o MOCKITO no ambiente Eclipse:

Selecionar o projeto
Botão direito do mouse
Escolher a opção Build Path
Escolher Add External Archives
Escolher o arquivo: mockito-all-1.9.5.jar

Testes De Software

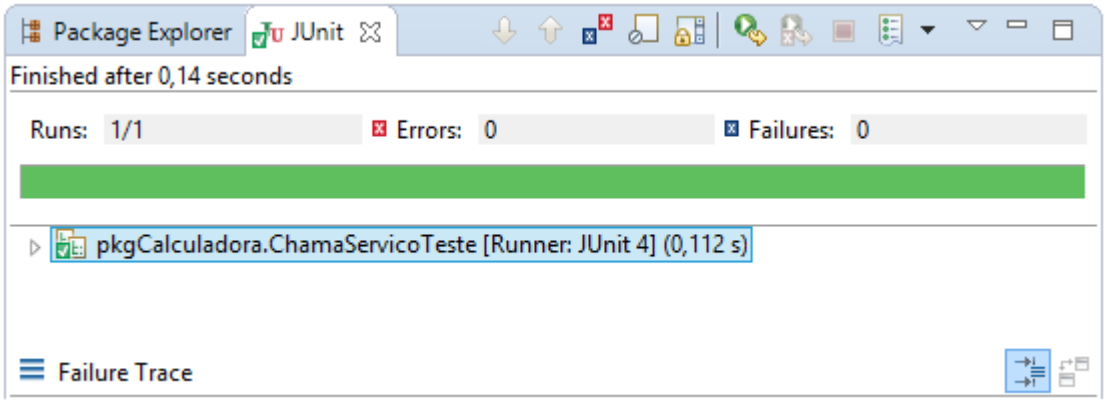
Professor Gilmar Luiz de Borba

10 –Criar o MOCK para permitir o teste anterior. Executar novamente o teste, veja o que acontece.

```
package pkgCalculadora;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Test;
import org.mockito.Mockito;

public class ChamaServicoTeste {

    ChamaServico calc;
    @Before
    public void setUp() throws Exception {
        ICalculadora ical = Mockito.mock(ICalculadora.class);
        Mockito.when(ical.adicionar(4, 6)).thenReturn(10);
        calc = new ChamaServico();
        calc.setCalc(ical);
    }

    @Test
    public void deveriaSomarDoisNumeros() {
        assertEquals(10, calc.adicionaDoisNumeros(4, 6),0);
    }

}
```

c:\prof_gilmar_borba\imagens\auditoria_testes\

Resultado:



c:\prof_gilmar_borba\imagens\auditoria_testes\

Testes De Software

Professor Gilmar Luiz de Borba

QUESTÕES (PRÁTICA):

(01) O são *MOCKS*?

(02) Quais são as características principais dos MOCKS?

(03) Pr que foi usado o MOCKITO em nossa atividade?

(04) Na prática, quando devemos “mockar” nossos testes?

(05) Em nossa atividade, por que o teste falhou na primeira vez?

(06) Após implementar o teste com MOCK ele servirá para qualquer valor informado?