

DITIG – TESTE UNITÁRIO COM JUNIT

MARCELO DA SILVA



O QUE É TESTE UNITÁRIO?

Os testes unitários visam verificar um código na sua menor fração.

Os testes unitários são integrados ao código através de classes de testes.

São verificados classes e métodos.

Evita a criação de métodos “main” para testar um código.

Teremos um pedaço de código verificando outro pedaço de código.

PROJETO I - EXEMPLO

- Endereço Github:
 - <https://github.com/marcelosprogrammer/ditig-testeunitario.git>



Separação
em
Camadas

Interfaces
em java

Enums

Model,
DAO,
Services

CONCEITOS
PRÉVIOS

PROJETO EXEMPLO

▼ ditig-testejunit [ditig-testeunitario main]
▼ src/main/java
 > (default package)
 > br.com.marcelos.dao
 > br.com.marcelos.dao.impl
 > br.com.marcelos.enums
 > br.com.marcelos.interfaces
 > br.com.marcelos.model
 > br.com.marcelos.services
 src/main/resources
 src/test/java
 src/test/resources
 > JRE System Library [JavaSE-11]
 > Maven Dependencies
 > src
 target
 pom.xml
 README.md

```
1 package br.com.marcelos.interfaces;  
2  
3 import java.util.List;  
4  
5 import br.com.marcelos.model.Funcionario;  
6  
7 public interface IFuncionarioServices {  
8  
9     public void salvar (Funcionario funcionario) throws Exception;  
10    public void atualizar (Funcionario funcionario) throws Exception;  
11    public void excluir (Funcionario funcionario) throws Exception;  
12    public Funcionario carregar (Integer id) throws Exception;  
13    public Funcionario buscarPorNome(String nome) throws Exception;  
14    public List<Funcionario> listar() throws Exception;  
15  
16  
17  
18 }  
19
```

```
package br.com.marcelos.model;

public class Funcionario {

    private int id;
    private String nome;
    private String cpf;
    private String telefone;
    private Enum sexo;
    private String funcao;
    private Double salario;

    public Funcionario() {

    }

}
```

MODEL EXEMPLO

INTERFACE DAO

```
package br.com.marcelos.dao;

import java.util.List;

import br.com.marcelos.model.Funcionario;

public interface FuncionarioDAO {

    public void salvar (Funcionario funcionario) throws Exception;
    public void atualizar (Funcionario funcionario) throws Exception;
    public void excluir (Funcionario funcionario) throws Exception;
    public Funcionario carregar (Integer id) throws Exception;
    public Funcionario buscarPorNome(String nome) throws Exception;
    public List<Funcionario> listar() throws Exception;

}
}
```

```
package br.com.marcelos.dao.impl;

import java.util.ArrayList;
import java.util.List;

import br.com.marcelos.dao.FuncionarioDAO;
import br.com.marcelos.model.Funcionario;

public class FuncionarioDaoImpl implements FuncionarioDAO {

    List<Funcionario> lista = new ArrayList<Funcionario>();

    @Override
    public void salvar(Funcionario funcionario) throws Exception {
        // TODO Auto-generated method stub
        lista.add(funcionario);
    }

    public void atualizar(Funcionario funcionario) throws Exception {}

    public void excluir(Funcionario funcionario) throws Exception {}

    public Funcionario carregar(Integer id) throws Exception {}

    @Override
    public Funcionario buscarPorNome(String nome) throws Exception {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public List<Funcionario> listar() throws Exception {
        // TODO Auto-generated method stub
        return lista;
    }

}
```

IMPLEMENTAÇÃO DO DAO

ENUM DO EXEMPLO

```
package br.com.marcelos.enums;

public enum Sexo {

    MASCULINO("M"), FEMININO("F");

    private String sexo;

    Sexo(String sexo) {
        this.sexo = sexo;
    }

    public String getSexo() {
        return sexo;
    }

}
```

```
package br.com.marcelos.interfaces;

import java.util.List;

import br.com.marcelos.model.Funcionario;

public interface IFuncionarioServices {

    public void salvar (Funcionario funcionario) throws Exception;
    public void atualizar (Funcionario funcionario) throws Exception;
    public void excluir (Funcionario funcionario) throws Exception;
    public Funcionario carregar (Integer id) throws Exception;
    public Funcionario buscarPorNome(String nome) throws Exception;
    public List<Funcionario> listar() throws Exception;

}

```

INTERFACE DE SERVIÇOS

```
package br.com.marcelos.services;

import java.util.List;

public class FuncionarioServices implements IFuncionarioServices{

    FuncionarioDaoImpl funcionarioImpl = new FuncionarioDaoImpl();

    @Override
    public void salvar(Funcionario funcionario) throws Exception {
        funcionarioImpl.salvar(funcionario);
    }

    public void atualizar(Funcionario funcionario) throws Exception {

    }

    public void excluir(Funcionario funcionario) throws Exception {

    }

    public Funcionario carregar(Integer id) throws Exception {

    }

    public Funcionario buscarPorNome(String nome) throws Exception {

    }

    @Override
    public List<Funcionario> listar() throws Exception {
        return funcionarioImpl.listar();
    }

}
```

IMPLEMENTANDO SERVICES

```

import br.com.marcelos.model.Funcionario;
import br.com.marcelos.services.FuncionarioServices;
import br.com.marcelos.enums.Sexo;

public class Principal {

    public static void main(String[] args) throws Exception {

        FuncionarioServices funcionarioServices = new FuncionarioServices();

        //Preenchimento do Objeto Funcionario
        Funcionario funcionario= new Funcionario();
        funcionario.setId(1);
        funcionario.setFuncao("Desenvolvedor");
        funcionario.setNome("Joana Dark");
        funcionario.setSalario(20000.00);
        funcionario.setSexo(Sexo.FEMININO);
        funcionario.setTelefone("047 87853-9879");

        funcionarioServices.salvar(funcionario);

        Funcionario funcionarioB= new Funcionario();
        funcionarioB.setId(2);
        funcionarioB.setFuncao("Analista de Sistemas");
        funcionarioB.setNome("Godan Five");
        funcionarioB.setSalario(40000.00);
        funcionarioB.setSexo(Sexo.MASCULINO);
        funcionarioB.setTelefone("048 98769-0985");

        funcionarioServices.salvar(funcionarioB);

        for (Funcionario unidadeFuncionario : funcionarioServices.listar()) {
            System.out.println(unidadeFuncionario.toString());
        }
    }
}

```

UMA CLASSE PRINCIPAL PARA TESTAR

USANDO JUNIT PARA TESTAR

```
package br.com.marcelos.testes;

import static org.junit.Assert.*;

import org.junit.Test;

public class FuncionarioTest {

    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```


The screenshot shows an IDE with a Java test class and its execution results. The left pane displays the test results, and the right pane shows the source code of the test class.

Test Results (Left Pane):

- Finished after 0,058 seconds
- Runs: 2/2
- Errors: 0
- Failures: 1
- br.com.marcelos.testes.FuncionarioTest [Runner: JUnit 4] (0,000 s)
 - testarSalvarSalarioComValorNegativo (0,000 s)
 - test (0,000 s)

Failure Trace (Bottom Left):

```
java.lang.AssertionError: O valor do Salario é menor ou igual a ZERO
at br.com.marcelos.testes.FuncionarioTest.testarSalvarSalarioComValorNegativo(FuncionarioTest.java:27)
```

Source Code (Right Pane):

```
1 package br.com.marcelos.testes;
2
3 import static org.junit.Assert.*;
4 import org.junit.Assert;
5 import org.junit.Test;
6 import br.com.marcelos.enums.Sexo;
7 import br.com.marcelos.model.Funcionario;
8 import br.com.marcelos.services.FuncionarioServices;
9
10 public class FuncionarioTest {
11
12     @Test
13     public void test() {
14         //fail("Not yet implemented");
15     }
16
17     @Test
18     public void testarSalvarSalarioComValorNegativo() {
19         FuncionarioServices funcionarioServices = new FuncionarioServices();
20         Funcionario funcionario = new Funcionario();
21         funcionario.setId(1);
22         funcionario.setFuncao("Desenvolvedor");
23         funcionario.setNome("Joana Dark");
24         funcionario.setSalario(-20000.00);
25         funcionario.setSexo(Sexo.FEMININO);
26         funcionario.setTelefone("047 87853-9879");
27         Assert.assertTrue("O valor do Salario é menor ou igual a ZERO ",funcionario.getSalario().doubleValue() > 0);
28     }
29 }
30
```

A CLASSE FUNCIONARIOTEST

CONVENÇÕES NA ESCRITA DE TESTES

- Nomes de classes tem a palavra Test em inglês. Ex: FuncionarioTest
- As classes de Tests ficam em pacotes separados do código de produção.

ASSERTS

- Um conjunto de métodos de asserção úteis para escrever testes.
- Somente asserções com falha são registradas.

MÉTODOS ASSERT

- `assertTrue(condição)`
- `assertFalse(condição)`
- `assertNull(objeto)`
- `assertNotNull(objeto)`
- `assertSame(esperado, valor atual)`, retornará true se forem iguais
- `assertNotSame(esperado, valor atual)`

MÉTODOS ASSERT

- `assertEquals(esperado, atual)`
- `assertEquals(esperado[i], atual[i])`
- `assertArrayEquals(esperado[i], atual[i])`
- Fail (mensagem)

A CLASSE TESTSUITE

```
1 import org.junit.runner.JUnitCore;
2 import org.junit.runner.Result;
3 import org.junit.runner.notification.Failure;
4
5 import br.com.marcelos.testes.FuncionarioTest;
6 import junit.framework.*;
7 public class TestSuite {
8
9     public static void main(String[] a) {
10
11         Result result = JUnitCore.runClasses(FuncionarioTest.class);
12         System.out.println("Total de Testes " + result.getRunCount());
13         System.out.println("Total de testes falhos " + result.getFailureCount());
14
15         for(Failure failure : result.getFailures())
16         {
17             System.out.println(failure.getMessage());
18         }
19         System.out.println("O TESTE RETORNOU: "+result.wasSuccessful());
20     }
21 }
22
23
```


VISÃO FINAL DO PROJETO

```

└─ > ditig-testejunit [ditig-testeunitario main]
  └─ > src/main/java
    └─ > (default package)
      > > Principal.java
      > > TestSuite.java
    > > br.com.marcelos.dao
    > > br.com.marcelos.dao.impl
    > > br.com.marcelos.enums
    └─ > br.com.marcelos.interfaces
      > > IFuncionarioServices.java
      > > ISalario.java
    > > br.com.marcelos.model
    └─ > br.com.marcelos.services
      > > FuncionarioServices.java
    └─ > br.com.marcelos.testes
      > > FuncionarioTest.java
  src/main/resources
  src/test/java
  src/test/resources
  > JRE System Library [JavaSE-11]
  > Maven Dependencies
  └─ > src
    > > main
    > > test
  target
  pom.xml
  README.md

```


TESTSUITE - B

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

import br.com.marcelos.testes.FuncionarioTest;

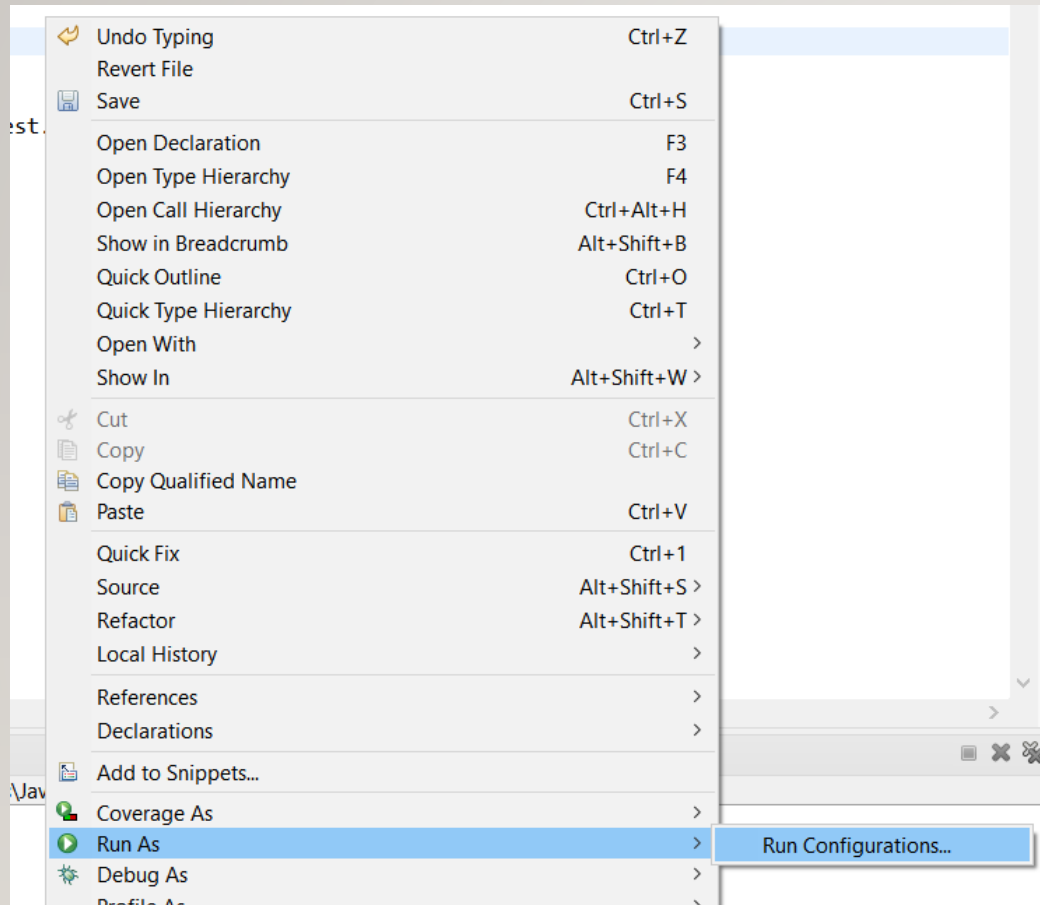
public class TestSuiteB {

    @RunWith(Suite.class)
    @Suite.SuiteClasses({ FuncionarioTest.class })

    public class TestSuite {

    }

}
```



RUN CONFIGURATIONS

Run Configurations

Create, manage, and run configurations

Create a configuration that will launch a JUnit test.

Configure launch settings from this dialog:

- Press the 'New Configuration' button to create a configuration of the selected type.
- Press the 'New Prototype' button to create a launch configuration prototype of the selected type.
- Press the 'Export' button to export the selected configurations.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.

or view an existing configuration by selecting it.

Select launch configuration(s) and then select 'Link Prototype' menu item to link a prototype.

Select launch configuration(s) and then select 'Unlink Prototype' menu item to unlink a prototype.

Select launch configuration(s) and then select 'Reset with Prototype Values' menu item to reset with prototype values.

Reset launch perspective settings from the [Perspectives](#) preference page.

Filter matched 25 of 26 items

Run Close

RUN CONFIGURATIONS

Run Configurations

Create, manage, and run configurations

Create a configuration that will launch a JUnit test.

The screenshot shows the Eclipse IDE's 'Run Configurations' dialog. On the left, a tree view lists various run configurations, with 'JUnit' expanded to show 'New_configuration' selected. The main area is titled 'Name: New_configuration' and contains several tabs: 'Test' (selected), 'Arguments', 'JRE', 'Dependencies', 'Source', 'Environment', and 'Common'. The 'Test' tab is active, showing fields for 'Project' (ditig-testejunit), 'Test class' (empty), and 'Test method' (empty). Below these is a radio button for 'Run all tests in the selected project, package or source folder:' which is selected, with a corresponding 'Search' button. The 'Include and exclude tags' section has a 'Configure...' button. The 'Test runner' is set to 'JUnit 4'. At the bottom, there is a checkbox for 'Keep JUnit running after a test run when debugging' (unchecked) and buttons for 'Show Command Line', 'Revert', and 'Run'.

type filter text

- HTTP Preview
- J2EE Preview
- Java Applet
- Java Application
 - Principal
 - TestSuite
- JUnit
 - FuncionarioTest
 - New_configuration
 - JUnit Plug-in Test
- Launch Group
- Launch NPM
- Maven Build
- Node.js application
- OSGi Framework
- Task Context Test

Filter matched 26 of 27 items

Name: New_configuration

Test Arguments JRE Dependencies Source Environment Common

Project: ditig-testejunit Browse

Test class: Search

Test method: Search

☒ Run all tests in the selected project, package or source folder: ditig-testejunit Search

Include and exclude tags: Configure...

Test runner: JUnit 4

☐ Keep JUnit running after a test run when debugging

Show Command Line Revert Run

RUN CONFIGURATIONS