DITIG – TESTE UNITÁRIO COM JUNIT

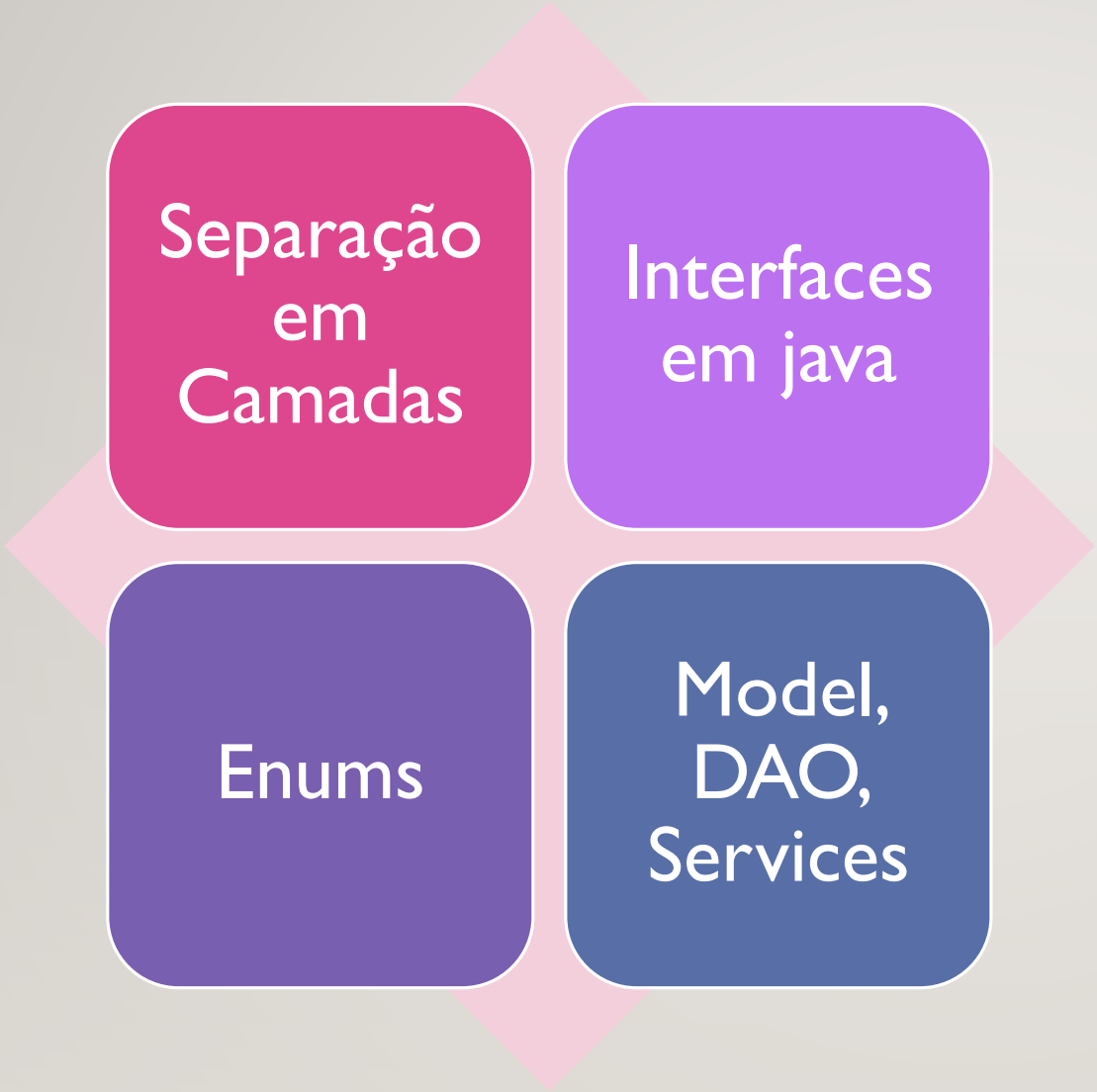
MARCELO DA SILVA

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.

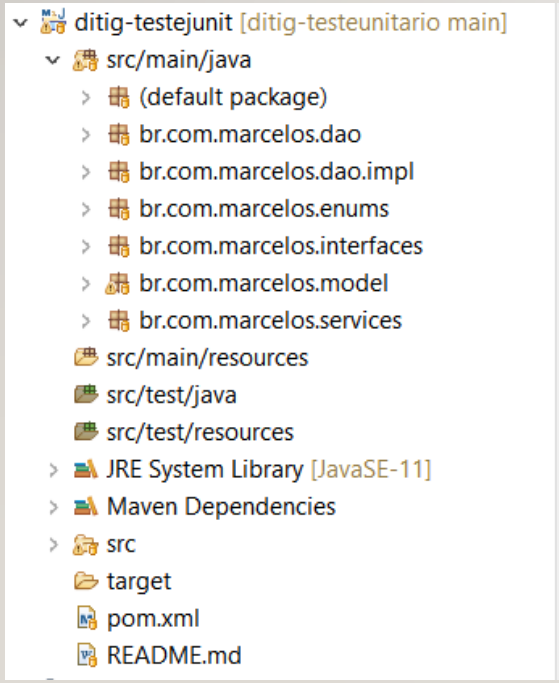
PROJETO 1 - EXEMPLO

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



CONCEITOS PRÉVIOS

PROJETO EXEMPLO




```java
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;


}
```

```java
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

```java
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;



}
```

```java
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

```java
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;
    }

}
```

```java
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

```java
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();
    }


}
```

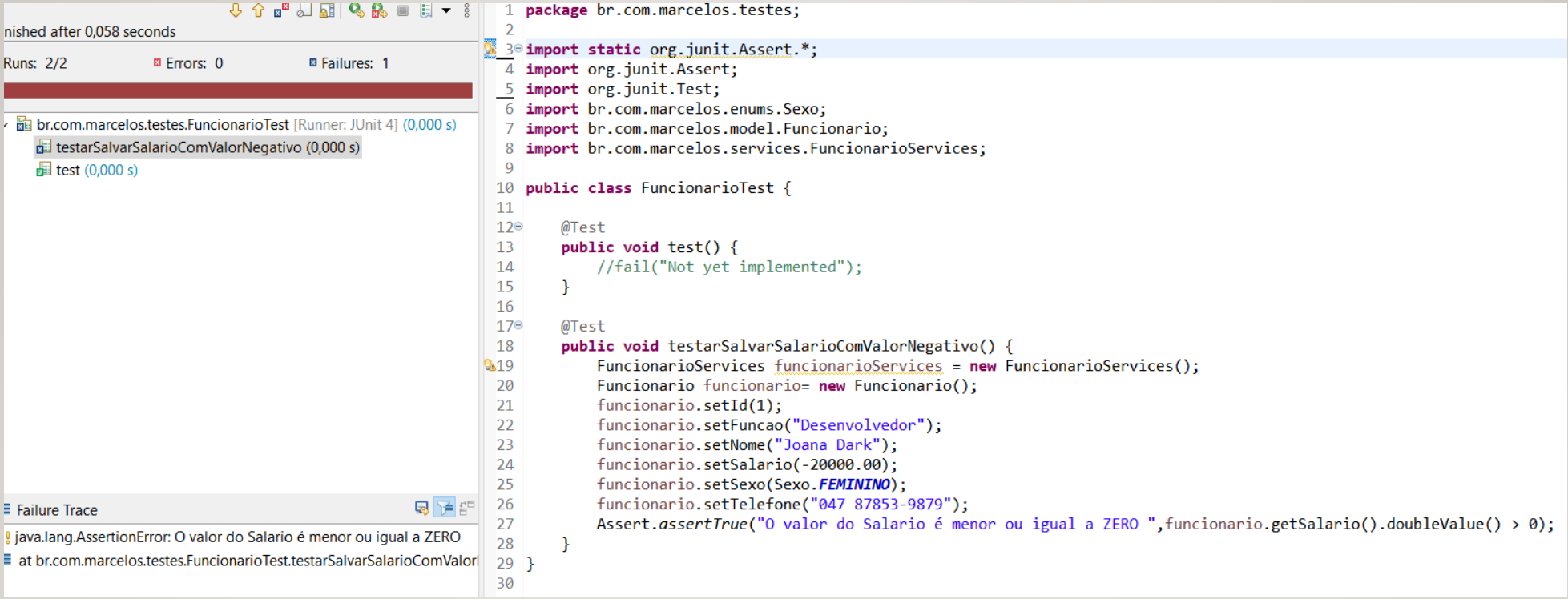
IMPLEMENTANDO SERVICES

```java
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

```java
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");
    }

}
```



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)

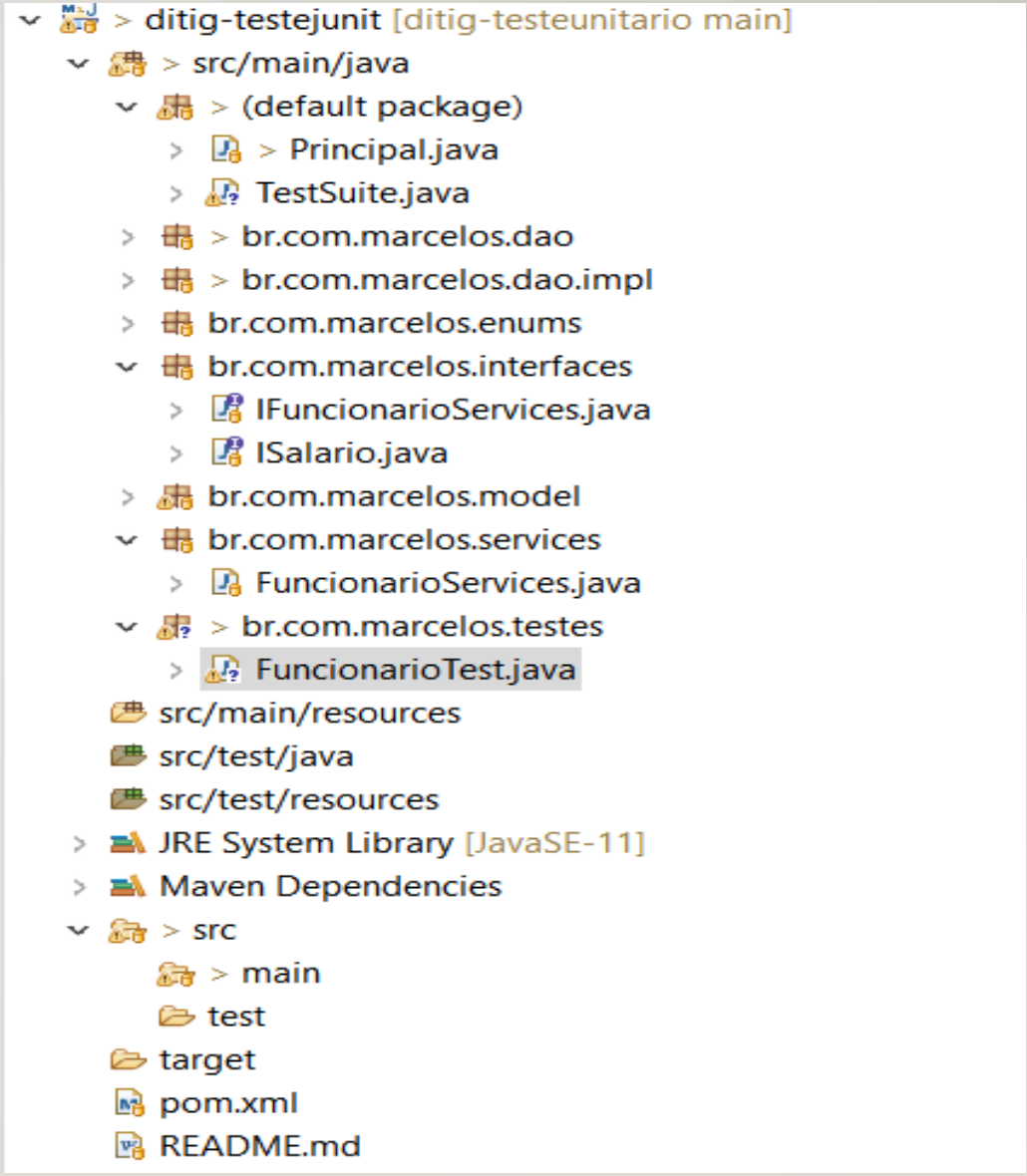
MÉTODOS ASSERT

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

A CLASSE TESTSUITE

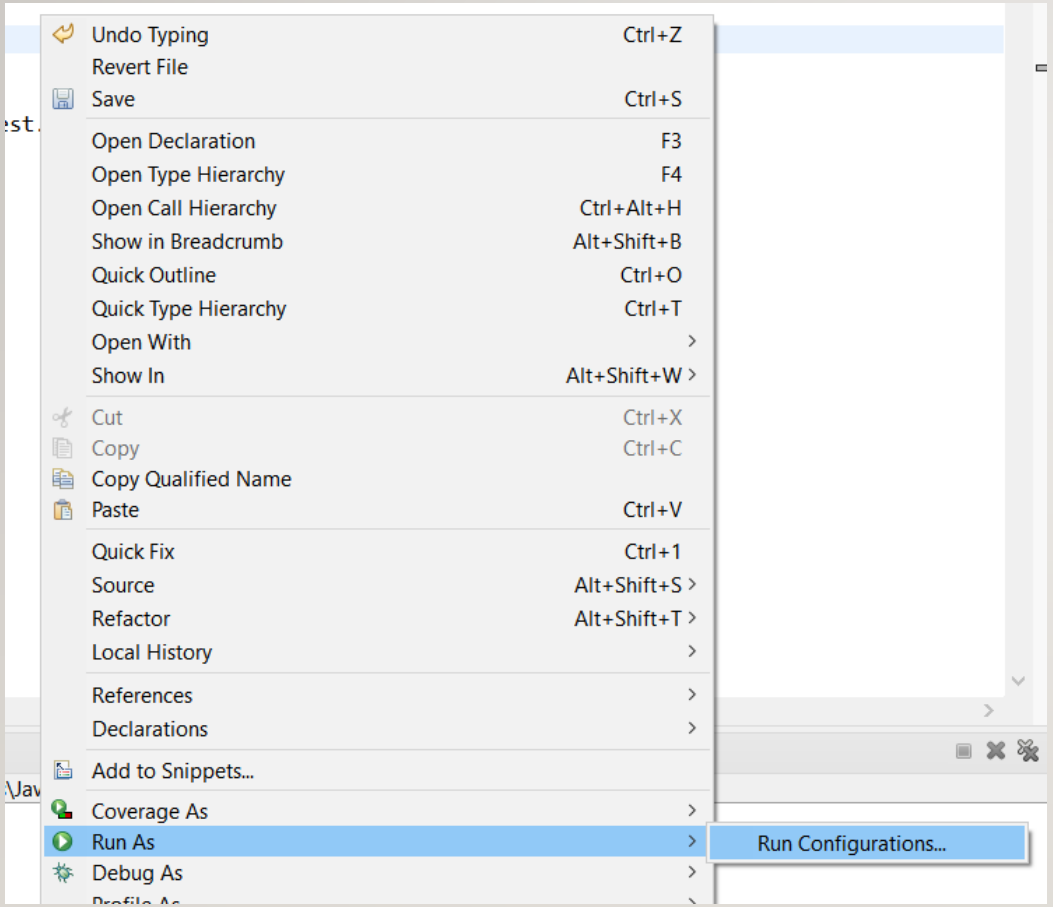
```java
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

import br.com.marcelos.testes.FuncionarioTest;
import junit.framework.*;
public class TestSuite {

    public static void main(String[] a) {

        Result result = JUnitCore.runClasses(FuncionarioTest.class);
        System.out.println("Total de Testes " + result.getRunCount());
        System.out.println("Total de testes falhos " + result.getFailureCount());

        for(Failure failure : result.getFailures())
        {
            System.out.println(failure.getMessage());
        }
        System.out.println("O TESTE RETORNOU: "+result.wasSuccessful());
       }

}
```

VISÃO FINAL DO PROJETO

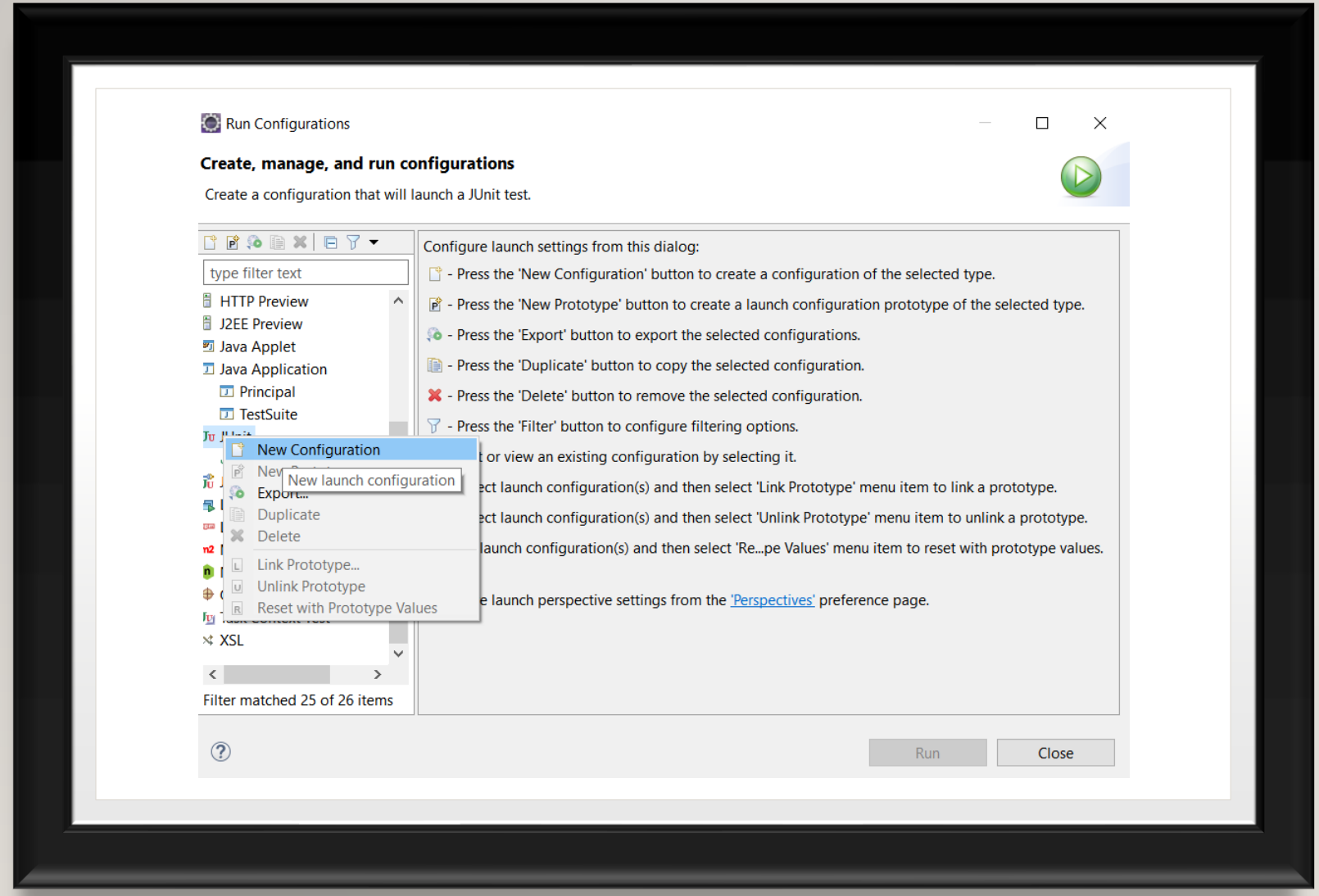


TESTSUITE - B

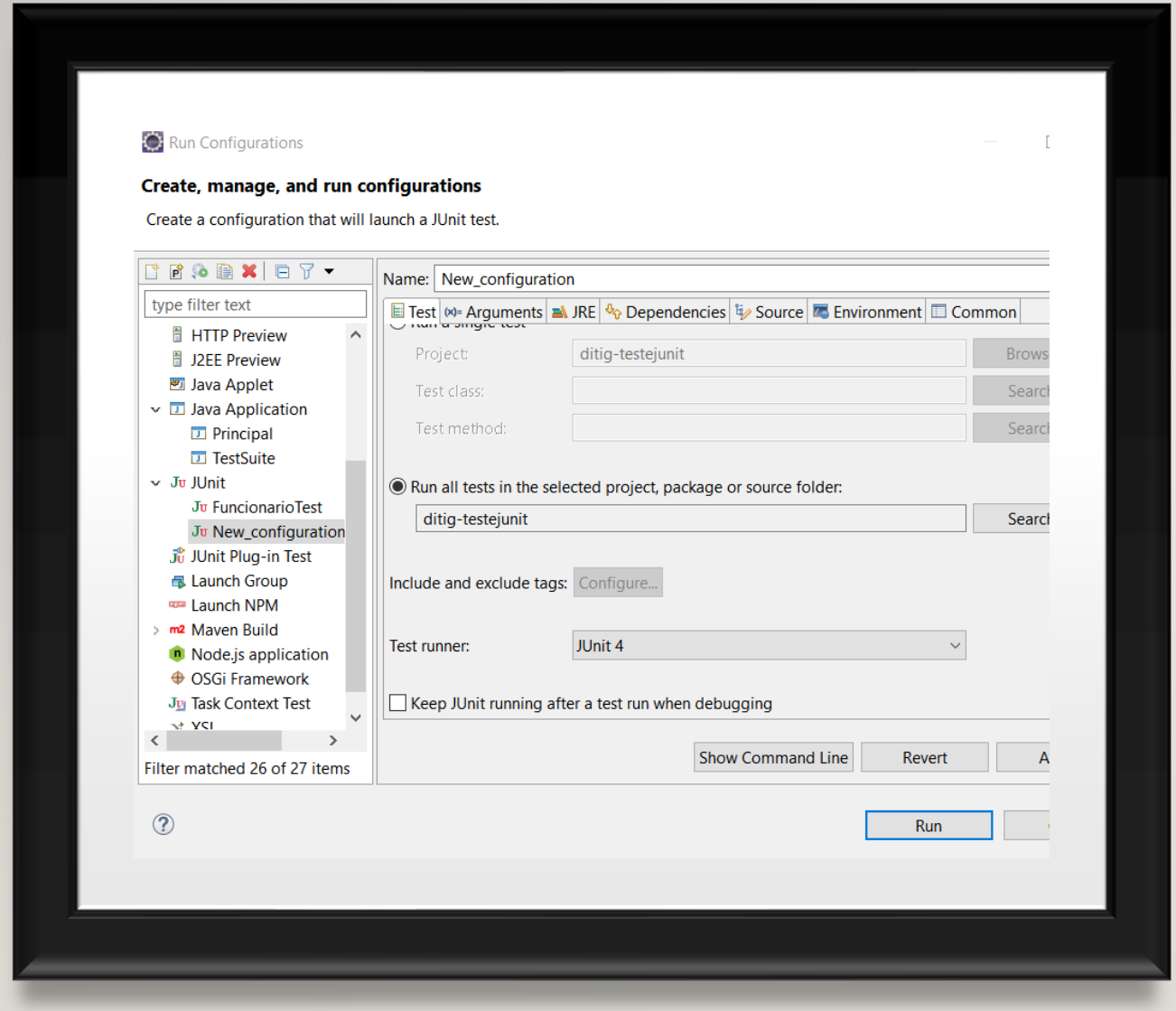
```java
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

import br.com.marcelos.testes.FuncionarioTest;

public class TestSuiteB {


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

    public class TestSuite {

    }

}
```



RUN CONFIGURATIONS



RUN CONFIGURATIONS



RUN CONFIGURATIONS