Testes com JUnit

Kaíque Matheus

29/01/2021

Padrões

- Por padrão e boas práticas de programação em testes de software e de prática utilizar a seguinte estrutura de nome nas classes de teste

- Utilizar o nome da classe que vamos testar com o acréscimo da palavra teste no final. Utilizando também a notação Camelo (primeira letra da palavra em maiúscula).

  Ex: CalculadoraTest.java
  GerenciadorClientesTest.java

- Já os métodos que vamos criar utiliza-se a seguinte notação. Iniciamos com a palavra test e descrevemos o método ou ação que será testada. Utilizando também a notação Camelo (primeira letra da palavra em maiúscula)

  Ex: testPesquisaCliente

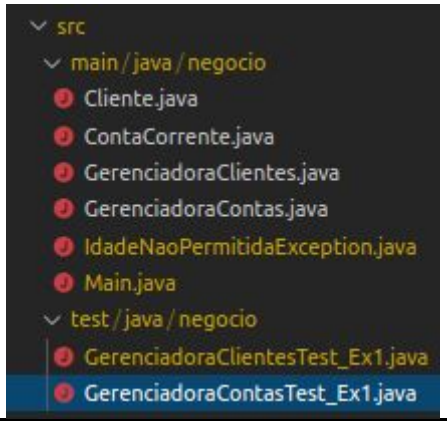
- Conforme vamos codificando os testes é de suma importância ir criando blocos de código documentados. Criando o corpo do código sempre com um comentário bem explicado do funcionamento, o que está acontecendo nesta parte, etc…

- É importante codificar testes unitários com a mesma boa prática de programação que se codifica o sistema

Esquema de pastas e classes.

Podemos seguir a seguinte estrutura para facilitar o compreendimento e melhorar a organização dos nossos testes. O projeto de teste vai seguir a mesma estrutura de pastas do projeto que iremos testar. Vamos manter um padrão de testes onde criamos a classe de testes com todos os respectivos testes desta classe com o diretório de pastas similar a classe e método testada.

ex:
vamos testar a classe man/java/negocio/GerenciadoraContas.java
então vamos criar o teste em test/java/negocio/GerenciadoraContasTest.java



Estrutura de um Bom Teste

Vamos definir uma estrutura básica com 3 partes: Cenário, Execução e Verificação. Além desses 3 blocos é de suma importância deixar bem comentado tudo que for pertinente. Em um teste automático é importante que deixemos bem claro as 3 partes do teste, pois estamos criando um código e o código nem sempre é fácil de entender.

- Montagem do cenário
- Execução
- Verificação

```
@Test
Run Test | Debug Test
public void testNomeDoMetodo(){

    /* ############ Cenário ############ */

    /* ############ Execução ############ */

    /* ############ Verificação ############ */

}
```

```
@Test
Run Test | Debug Test
public void testPesquisaCliente2(){

    /* ############ Cenário ############ */

    //Criamos alguns clientes
    Cliente cli1 = new Cliente(1, "Kaíque", 23, "kaique@gmail.com", 123, true);
    Cliente cli2 = new Cliente(2, "Matheus", 23, "matheus@gmail.com", 321, true);

    // Inserimos os clientes em uma lista de clientes do banco
    List<Cliente> clientesDoBanco = new ArrayList<>();
    clientesDoBanco.add(cli1);
    clientesDoBanco.add(cli2);

    // Instanciando a classe gerenciadora clientes
    GerenciadoraClientes gerClientes = new GerenciadoraClientes(clientesDoBanco);

    /* ############ Execução ############ */

    // Invocando o método pesquisa cliente passando o id de um dos clientes existentes
    Cliente cliente = gerClientes.pesquisaCliente(1);


    /* ############ Verificação ############ */

    // Verificamos se o pesquisa cliente retornou o cliente correto.
    assertEquals(1, cliente.getId());
    assertEquals("kaique@gmail.com", cliente.getEmail());
```

Comentar bem o código deixa ele muito mais legível e de fácil manutenção. Ainda mais se pensarmos que a manutenção de um projeto de testes pode não ser feita por nós mesmos. Quando pegamos o código de outro programador o gral de dificuldade para compreender o raciocínio de outra pessoa é elevado.
Uma boa prática também e comentar os cenários descrevendo bem seu código

```java
package negocio;

import static org.junit.jupiter.api.Assertions.assertEquals;
import java.util.ArrayList;
import java.util.List;
import org.junit.jupiter.api.Test;

/**
 * Classe de teste criada para garantir o funcionamento das principais operações
 * sobre contas, realizadas pela classe {@link GerenciadoraContas}.
 *
 * @author Kaíque Matheus
 * @date 29/01/2021
 */
Run Test | Debug Test | ✓
public class GerenciadoraContasTest_Ex1 {

    /**
     * Teste básico da transferência de um valor da conta de um cliente para outro,
     * estando ambos os clientes ativos e havendo saldo suficiente para tal transferência
     * ocorrer com sucesso.
     *
     * @author Kaíque Matheus
     * @date 29/01/2021
     */
    @Test
    Run Test | Debug Test | ✓
    public void testTransfereValor(){

    /* ############ Cenário ############ */

    // Criando algumas contas
    ContaCorrente conta01 = new ContaCorrente(1, 200, true);
    ContaCorrente conta02 = new ContaCorrente(2, 0, true);

    // Criando uma lista de contas e adicionando contas a ela
    List<ContaCorrente> contasBanco = new ArrayList<>();
    contasBanco.add(conta01);
    contasBanco.add(conta02);

    GerenciadoraContas gerContas = new GerenciadoraContas(contasBanco);

    /* ############ Execução ############ */

    // Vamos chamar o método transfereValor e tentar transferir 100 reais da conta 1 para a conta 2
    gerContas.transfereValor(1, 100, 2);

    /* ############ Verificação ############ */

    // Vamos verificar se o valores das contas estão corretas.
    assertEquals(100.0, conta01.getSaldo());
```

Testes com JUnit

Kaíque Matheus

29/01/2021

Documentação

JUnit 5 https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions
JUnit 4 https://junit.org/junit4/javadoc/latest/

Testes com JUnit

Kaíque Matheus

29/01/2021