Qualidade e Testes De Software

Professor Gilmar Luiz de Borba

Prática

Testes Unitários.

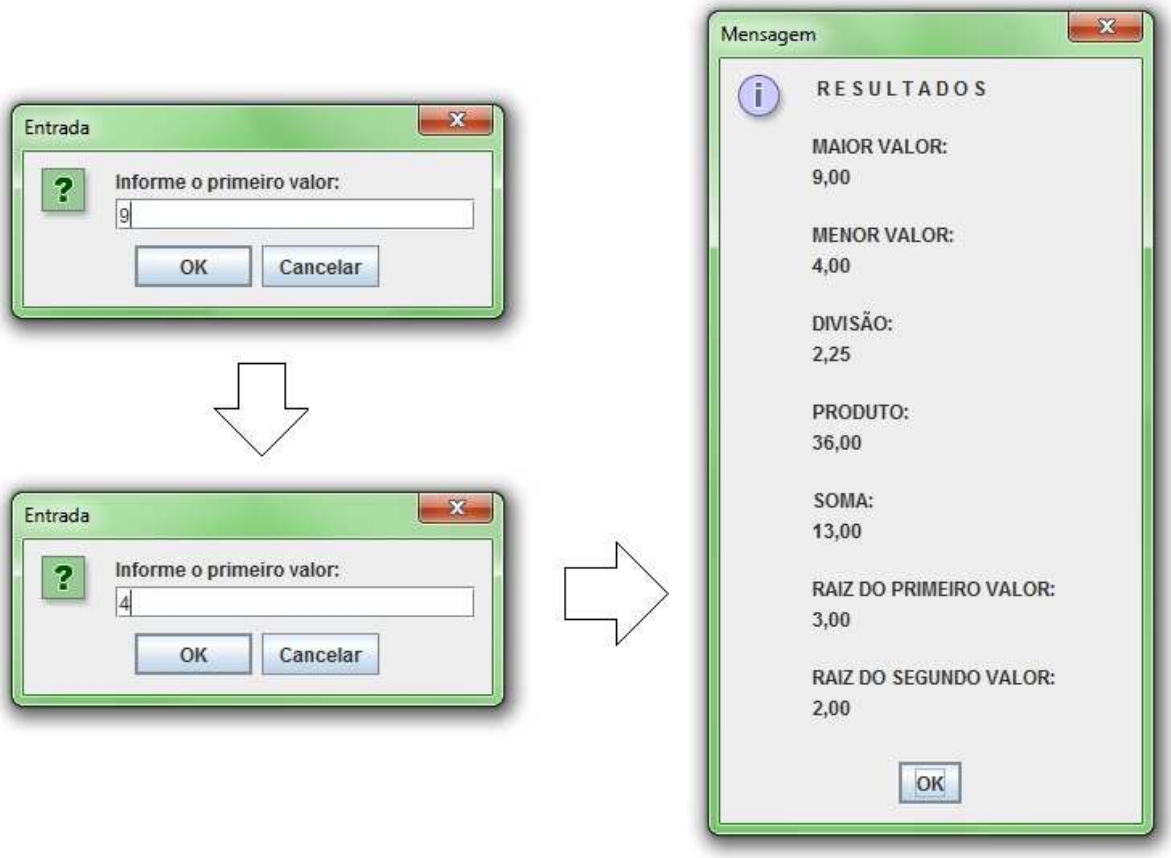
Objetivos:

- Praticar a partir de um exemplo simples os ***Testes Unitários*** para métodos específicos de uma classe.
- Entender o processo de criação de ***Testes Unitários*** *automatizados*.
- Conhecer as diretivas: @test, @after, @before.
- Conhecer e trabalhar com os métodos: assertEquals(), assertTrue(), assertFalse(), @assertNull(),@assertNotNull(), @assertSame(), @assertNotSame(), setUp() e tearDown().

Considerações sobre a atividade:

O programa tem por objetivo mostrar algumas operações matemáticas básicas a partir de dois valores digitados pelo usuário.

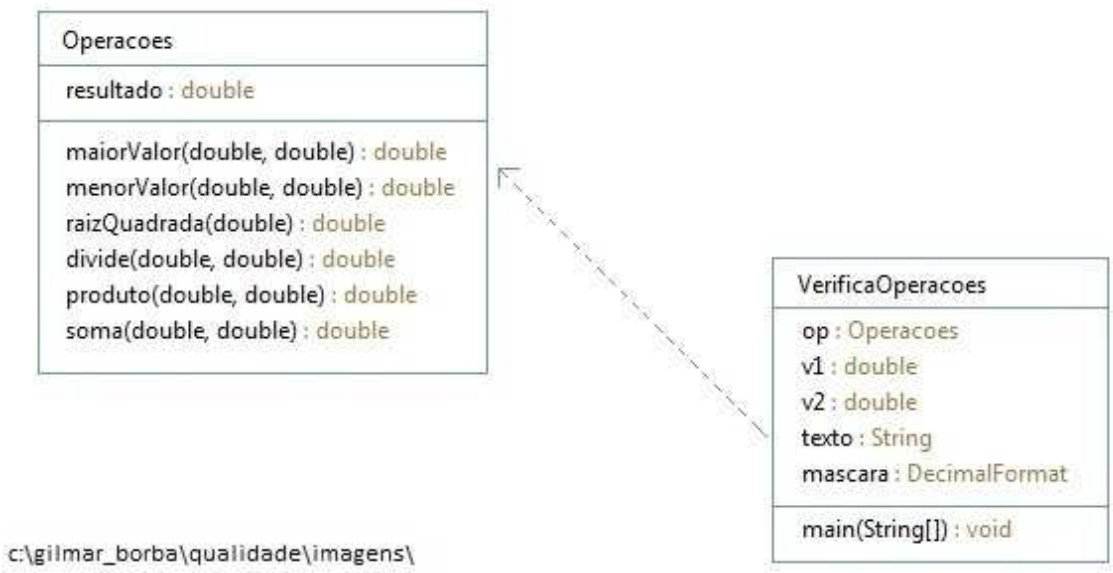
Veja possíveis interfaces (IHM) do programa:



Qualidade e Testes De Software

Professor Gilmar Luiz de Borba

Diagrama de classes:



A CLASSE ***OPERACOES*** SERÁ A CLASSE A SER TESTADA

Orientações:

(A) Crie um novo projeto JAVA: **OperacoesTesteUnitario**

(B) Crie um pacote (com um nome a seu critério)

(C) Crie a classe **Operacoes** e os métodos conforme o diagrama de classes acima.

Veja a implementação da classe na próxima página.

```java
package pkgOperacoesTestesUnitarios;
public class Operacoes {
	private double resultado = 0;
	// Maior valor
	public double maiorValor(double v1, double v2) {
		if (v1 > v2)
			return v1;
		else
			return v2;
	}
	// Menor valor
	public double menorValor(double v1, double v2) {
		if (v1 < v2)
			return v1;
		else
			return v2;
	}
	// Raiz quadrada
	public double raizQuadrada(double v1) {
		if (v1 >=0)
			resultado = Math.sqrt(v1);
		else
			throw new IllegalArgumentException(
			"O Valor não pode ser negativo");
		return resultado;
	}
	// Divisão
	public double divide(double v1, double v2) {
		try {
			resultado = v1 / v2;
			return resultado;
		}catch (Exception e) {
			System.out.println(
			"Um erro ocorreu"+e.getMessage());
		}
		System.out.println(
		"O(s) valor(es) não pode(m) se nulos (Zeros!)");
		throw new IllegalArgumentException(
		"O(s) Valor(es) não pode(m)"+
		"ser nulo(s)");
	}
	// Produto - multiplicacao
	public double produto(double v1, double v2) {
		resultado = v1 * v2;
		return resultado;
	}

	// Soma
	public double soma(double v1, double v2) {
		resultado = v1 + v2;
		return resultado;
	}
}
```

(D) Criar a Classe de Teste, siga os passos:

1 - Selecionar o pacote
2 – Acionar o botão direito do mouse (menu)
3 – Selecionar a opção ***New***
4 – Selecionar a opção ***Junit Test Case***

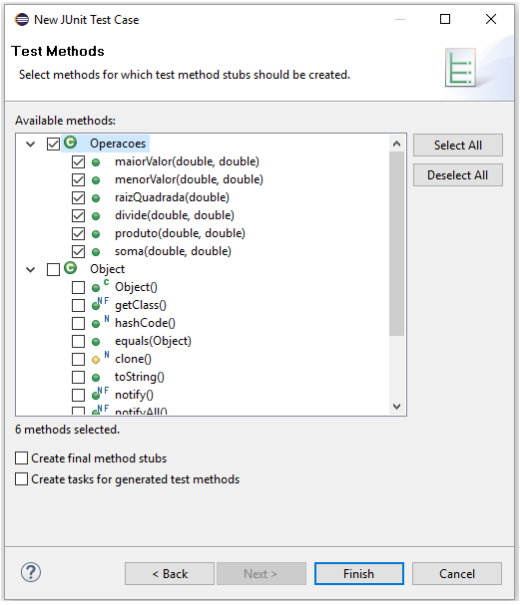
(E) Uma vez na caixa de diálogo **New JUnit Test Case** inserir as seguintes informações:

1 – Name: OperacoesTESTE (nome da classe de teste)
2 – Deixar selecionado somente o método setUP() na opção:
Wich method stubs would you like to create?
3 – Na opção "Class Under test", acionar o botão [Browse] informe Operacoes, para localizar e definir a classe Operacoes do pacote do nosso projeto.
4 – Após localizar a classe "Operações" do nosso pacote acione o botão [OK]
5 - Ao retornar a tela anterior ***(New JUnit Test Case)*** Acione o botão [NEXT]

(F) Escolha os métodos que deverão ser testados:

Selecionar todos os Métodos *(da classe Operacoes)*

*Acione o botão [**FINISH**]*



(G) Na primeira vez que for criada uma classe de teste será questionado sobre o caminho (build Path) de procura do JUnit 4 . Deixe a opção: ***"Perform the folowing action"*** selecionada e aciona o botão [OK]

(H) Nesse momento foi gerada a classe de teste

```
1  package pkgOperacoesTestesUnitarios;
2
3  import static org.junit.Assert.*;
7
8  public class OperacoesTeste {
9
10     @Before
11     public void setUp() throws Exception {
12     }
13
14     @Test
15     public void testMaiorValor() {
16         fail("Not yet implemented");
17     }
18
19     @Test
20     public void testMenorValor() {
21         fail("Not yet implemented");
22     }
23
24     @Test
25     public void testRaizQuadrada() {
26         fail("Not yet implemented");
27     }
28
29     @Test
30     public void testDivide() {
31         fail("Not yet implemented");
32     }
33
34     @Test
35     public void testProduto() {
36         fail("Not yet implemented");
37     }
38
39     @Test
40     public void testSoma() {
41         fail("Not yet implemented");
42     }
43
44 }
```

c:\professor_Gilmar_Borba\QualidadeTestes\Imagens

(I) Criar os tipos "***op1*** *e* ***op2***" da classe Operação, no corpo da classe de teste, antes do método setUP().

(II) Em seguida criar o objeto "***op1*** *e* ***op2***" dentro do método setUp(). Veja o código:

```
1  package pkgOperacoesTestesUnitarios;
2
3  import static org.junit.Assert.*;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8  public class OperacoesTeste {
9
10     Operacoes op1, op2;
11     @Before
12     public void setUp() throws Exception {
13         op1 = new Operacoes();
14         op2 = new Operacoes();
15     }
```

c:\professor_Gilmar_Borba\QualidadeTestes\Imagens

(J) Caso de teste 1, 2,3 (maiorValor()):

- Alterar o nome do primeiro método para: *deveriaTestarMaiorValor()*
- Implementar o seguinte método *assertEquals():*

Texto	Valor esperado	Parâmetro 1	Parâmetro 2
CASO 1:	16	16	9
CASO 2:	17	16	9
CASO 3:	15	16	9

Obs: para os casos 2 e 3, os testes devem passar também, use o Delta para isso.

(K) Caso de teste 4, 5 (menorValor()):

- Alterar o nome do primeiro método para: *deveriaTestarMenorValor()*
- Implementar os seguintes métodos *assertTrue()* e *assertFalse():*

Texto	Valor esperado	Parâmetro 1	Parâmetro 2	Método
CASO 4:	16	16	9	*assertFalse()*
CASO 5:	9	16	9	*assertTrue()*

(L) Caso de teste 6, 7 (comparar objetos o1 e o2):

- **Criar o método**: *deveriaCompararDoisObjetos()*

```
@Test
public void deveriaCompararDoisObjetos() {


}
```

c:\professor_Gilmar_Borba\QualidadeTestes\Imagens

- Implementar os seguintes métodos *assertSame()* e *assertNotSame():*

Texto	Objeto 1	Objeto 2
CASO 6:	Passar o objeto adequado	Passar o objeto adequado
CASO 7:	Passar o objeto adequado	Passar o objeto adequado

(M) Caso de teste 8 (Raiz Quadrada):

- Alterar o nome do primeiro método para: *deveriaTestarRaizQuadrada()*
- Implementar o seguinte método: *assertEquals():*

Texto	Valor esperado	Parâmetro 1	Delta
CASO 8:	9.380	88	Use o delta adequado

Obs: para o caso 8, o teste deve passar, use o Delta para isso.

(N) Casos de teste 9, 10 e 11 (Divisão):

- Alterar o nome do primeiro método para: *deveriaTestarDivisao()*

- Implementar o seguinte método: *assertEquals():*

Texto	Valor esperado	Parâmetro 1	Parâmetro 2	Resultado real esperado
CASO 9:	**25**	**50**	**2**	**25**
CASO 10:	**0**	**50**	**0**	**Infinity**
CASO 11:	**0**	**0**	**0**	**NaN**

Obs: para os casos 9, 10, 11, os testes devem passar, use a exceção esperada AssertionError e lance (throws) a exceção no carimbo do método.

(O) Caso de teste 12 (Produto):

- Alterar o nome do primeiro método para: *deveriaTestarProduto()*
- Implementar o seguinte método: *assertEquals():*

Texto	Valor esperado	Parâmetro 1	Parâmetro 2
CASO 12:	1.000.000.000	1.000.000	1.000.000

Obs: para os casos 12, o teste deve passar, use a exceção esperada AssertionError e lance (throws) a exceção no carimbo do método.

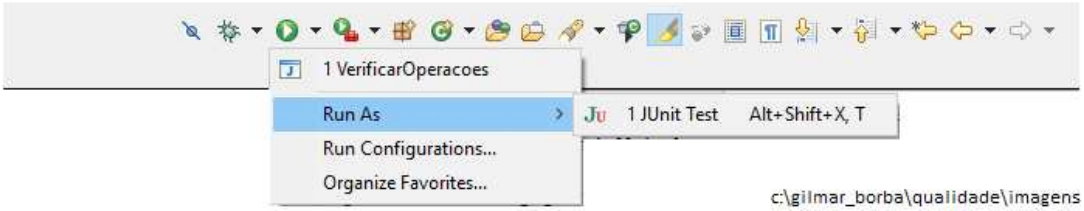
(P) Casos de teste 13, 14 e 15 (Soma):

- Alterar o nome do primeiro método para: *deveriaTestarSoma()*
- Implementar o seguinte método: *assertEquals():*

Texto	Valor esperado	Parâmetro 1	Parâmetro 2	Método de teste
CASO 13:	20	10	10	assertEquals()
CASO 14:	21	10	10	assertFalse()
CASO 15:	20	10	10	assertTrue()

Obs: para os casos 13, 14, 15, os testes devem passar.

Para executar o teste acione Run As + JUnit Test ... :



Verificar os resultados

QUESTÕES

(01) Qual é o objetivo da assertiva: assertSame() e assertNotSame()?

(02) O que é uma asserção no contexto do teste de software?

(03) Explicar o código: *@Test(expected = AssertionError.class)*