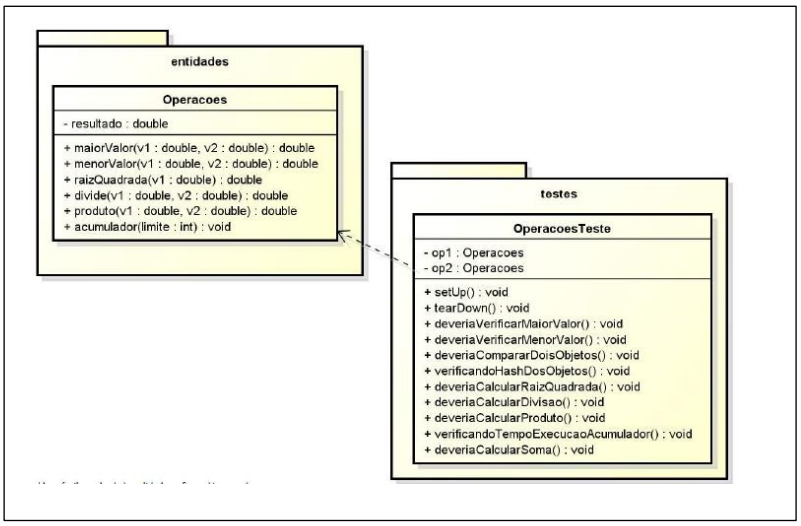
Prática

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()



A CLASSE “Operacoes” SERÁ A CLASSE A SER TESTADA

Anotações e descrição
@Test A anotação do teste diz ao JUnit que o método deve ser executado como um caso de teste.
@Before Vários testes precisam de objetos semelhantes criados antes que eles possam ser executados. Anotar um método @Before faz com que esse método seja executado antes de cada método de teste.

@After Os métodos que tiverem a anotação @After faz com que esse método seja executado após o método de teste.
@BeforeClass Anotar um método como @BeforeClass faz com que ele seja executado uma vez antes de qualquer um dos métodos de teste da classe.
@AfterClass Isso executará o método depois que todos os testes terminarem. Isso pode ser usado para realizar, por exemplo, destruição de objetos.
@Ignore A anotação Ignore é usada para ignorar o teste, logo não será executado.

ANOTAÇÕES

Para determinar se um método é de teste utilizamos logo acima da classe de teste a anotação: * @Test
Exemplo:

```
@Test
public void testName() {
}
```

Para ignorar um método de teste utilizamos a anotação:

- @Ignore

Exemplo:

```
@Ignore
@Test
public void testName() {
}
```

Para determinar que um método vai ser executado antes dos demais métodos da classe de teste utilizamos a anotação:

- @BeforeClass.

Essa funcionalidade serve para que possamos antes de uma classe de teste por exemplo iniciar a conexão com o banco de dados, inicializar variaveis entre outras possibilidades. Exemplo:

```
@BeforeClass
public void setup() {
    dataBase = new DataBase();
   dataBaseConnection.open();
   dataBase.populate();
   list_name = new ArrayList<>();
   index = 0;
}
```

Para determinar que um método vai ser executado depois dos demais métodos da classe de teste utilizamos a anotação:

- @AfterClass.

Essa funcionalidade serve para que possamos depois de uma classe de teste por exemplo fechar a conexão com o banco de dados, ajudar o garbage collection a limpar os dados ociosos entre outras possibilidades.

Exemplo:

```
@AfterClass
public void close() {
    dataBase.drop();
    dataBaseConnection.close();
    dataBase = null;
    list_name = null;
    index = null;
}
```

Para determinar que um método vai ser executado antes de cada caso de teste utilizamos a anotação:

- @Before.

Essa funcionalidade serve por exemplo para que antes de um método possamos inicializar variaveis. Exemplo:

```
@Before
public void initialize() {
    list_name = new Calendar.getAllNames();
    index = 55;
}
```

Para determinar que um método vai ser executado depois de cada caso de teste utilizamos a anotação:

- @After.

Essa funcionalidade serve por exemplo para que antes de um método possamos finalizar variaveis. Exemplo:

```
@After
public void finalize() {
    list_name = calendar.getAllNames();
    index = current_index;
}
```

Exemplos de Afirmações Equals

Afirmações.assertEquals() verifica se um valor esperado e calculado são iguais. No caso, ambos não são iguais, retornará AssertError.

Se ambos são nulos, eles são considerados iguais.

Quando você chama assertEquals (object1, object2), ele usa o método igual a esse tipo de Objeto.

Por exemplo:

Se objeto1 e objeto2 forem do tipo String, então o método de string é chamado para determinar a relação de igualdade.

Afirmações de ponto flutuante

Quando você quiser comparar pontos flutuantes como double e float, então você deve sempre usar o parâmetro extra para evitar arredondar enquanto compara pontos flutuantes. O nome dele é delta.

public static void assertEquals(double expected, double actual, double **delta**)

Exemplos de Afirmações assertSame

Afirma que dois objetos se referem ao mesmo objeto. Se eles não são os mesmos, ocorrerá um erro. **Parâmetros:**

- *esperado* - o objeto esperado
- *real* - o objeto a comparar com o esperado

Orientações:

(A) Crie um novo projeto JAVA: TesteUnitarioOperacoes

(B) Crie um pacote de nome "entidade", (ou com um nome a seu critério)

(C) Crie a classe Operacoes (no pacote entidades).

(D) Copie e cole o código fornecido.

(E) Criar a Classe de Teste (OperacoesTeste no pacote testes), 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*

(F) 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:
Which 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]

(G) Escolha os métodos que deverão ser testados:

Selecionar todos os Métodos *(da classe Operacoes)*

Acione o botão [FINISH]

(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() {


}
```

- 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, se necessário, lance throws para 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, se necessário, lance throws para 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 ... :

Resumo de algumas assertivas:

(1) AssertEquals, AssertTrue e AssertNull