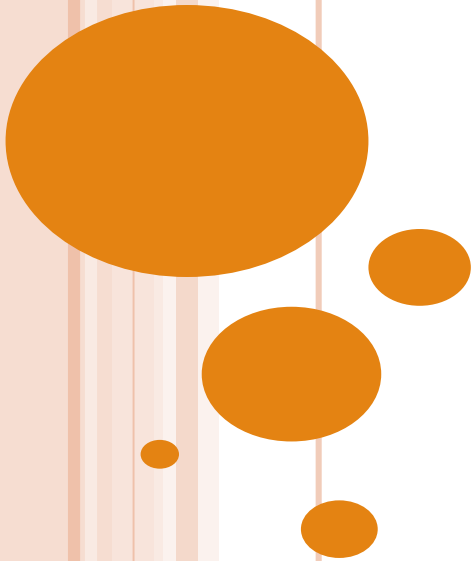
Automação de Testes - JUnit

Testes de Softwares



Roteiro

- Introdução
- Configuração do JUnit
- Anotações do Junit
  - Exemplos
- Asserções do JUnit
  - Exemplos
- Esquema de Classes
- Boas práticas com Junit
- Introdução ao TDD (Test Driven Development)
  - Exemplo

Introdução

- JUnit é um framework para realizar teste de unidade em código Java
- Verifica se cada método de uma classe funciona da forma esperada

- **Open-source**
  - http://junit.org/junit4/

Configuração do JUnit

- Baixar o arquivo **JAR**

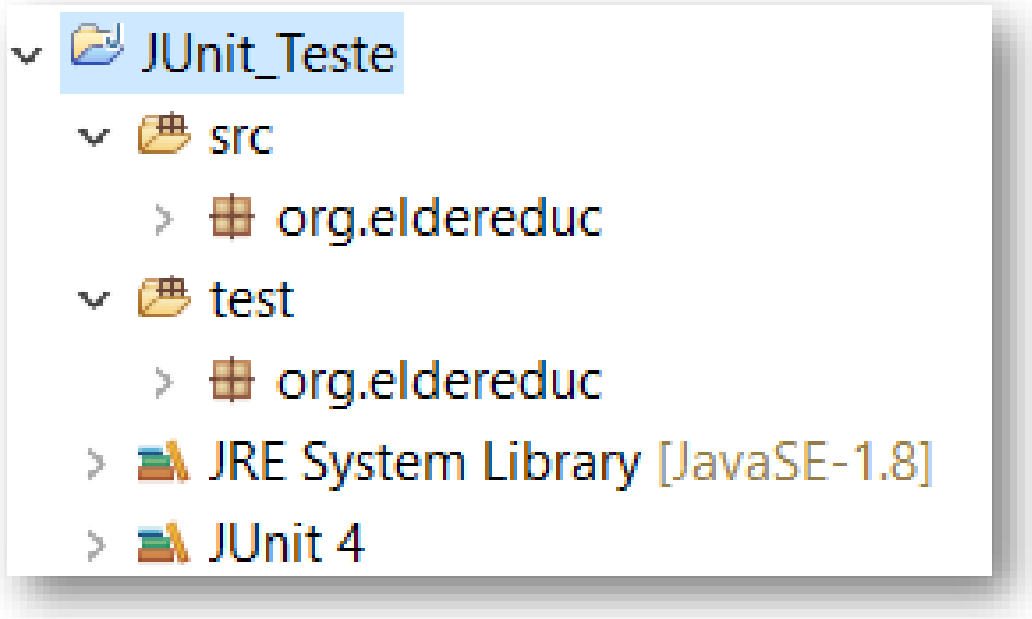
```
junit.jar
```

- Ou Dependência **Maven**

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

Configuração do JUnit

- Exemplo de esquema de Projeto com JUnit



ANOTAÇÕES DO JUNIT

- @Test
  - Identifica que o método é um teste.
- @BeforeClass
  - Executa uma única vez antes dos métodos de testes.
- @AfterClass
  - Executa uma única vez após os métodos de testes
- @Before
  - Executa antes de cada teste, usado para preparar o ambiente de teste.
- @After
  - Executa depois de cada teste, usado para liberar recursos do teste

ANOTAÇÕES DO JUNIT

- Arquivo de teste, exemplo:
  - ApredendoAnotacoes.java

```
public class ApredendoAnotacoes {
    @BeforeClass
    public static void beforeClasse(){
        System.out.println("before_classe");
    }

    @AfterClass
    public static void beforeAfter(){
        System.out.println("after_classe");
    }

    @Before
    public void before(){
        System.out.println("before");
    }

    @After
    public void after(){
        System.out.println("after");
    }

    @Test
    public void test_1() {
        System.out.println("teste 1");
    }

    @Test
    public void test_2() {
        System.out.println("teste 2");
    }
}
```

Mais anotações do JUnit

- @Test(expected=Exception.class)
  - Falha se o método testado não retornar a exceção do parâmetro.
- @Test(timeout=100)
  - Falha se o método demorar mais de 100 milissegundos.
- @Ignore
  - Ignora algum método de teste.

ANOTAÇÕES DO JUNIT

- Arquivo de teste, exemplo:
  - ApredendoAnotacoes2.java

```java
public class AprendendoAnotacoes2 {
    @Test
    public void teste1() {
        System.out.println("teste 1");
    }

    @Test(expected=ArithmeticException.class)
    public void teste2() {
        double valor = 3 / 0;
        System.out.println("teste 2");
    }

    @Test(timeout=1000)
    public void teste3() {
        try{
            Thread.sleep(1000);
        }catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("teste 3");
    }

    @Ignore
    @Test
    public void teste4() {
        System.out.println("teste 4");
    }
}
```

Asserções do JUnit

- assertEquals(long expected, long actual)
  - Afirma que dois longs são iguais.
  - Se eles não são, uma AssertionError é lançada.

- assertEquals(String message, long expected, long actual)
  - Afirma que dois longs são iguais.
  - Se eles não são, uma AssertionError é lançada com uma data mensagem.

- assertEquals(double expected, double actual, double **delta**)
  - Afirma que dois dobles ou floats são igual dentro de um delta positivo.
  - Se eles não são, uma AssertionError é lançada sem uma mensagem.

- assertEquals(String message, double expected, double actual, double **delta**)
  - Afirma que dois dobles ou floats são igual dentro de um delta positivo.
  - Se eles não são, uma AssertionError é lançada com uma data mensagem.

Asserções do JUnit

- Parâmetro **delta** do método **assertEeqauls( )**
  - Podemos encontrar algum problema relacionado a precisão nas operações quando trabalhamos com ponto flutuante, exemplo,
    - 0.1 * 0.1 = 0.01 (multiplicação normal)
    - 0.1 * 0.1 = 0.010000000000000002 (multiplicação no Java)
  - Nesse exemplo, supondo que o valor atual, 0.010000000000000002, e o valor esperado, 0.01, logo o **valor delta** é igual a diferença entre o valor atual e valor esperado,
    - 0.0000000000000002 **=** 0.010000000000000002 – 0.01
  - Nesse exemplo, o **valor delta** pode ser igual ou um pouco superior ao valor apresentado
    - 0.0000000000000002
    - 0.1 (número superior)

Asserções do JUnit

- Arquivo de teste, exemplo:
  - ApredendoAssercoes.java

```java
public class AprendendoAssercoes {
    @Test
    public void teste_1() {
        long esperado = 23;
        long atual = 23;
        assertEquals(esperado, atual);
    }
    @Test
    public void teste_2() {
        long esperado = 23;
        long atual = 24;
        assertEquals("Erro: Teste 2", esperado, atual);
    }
    @Test
    public void teste_3() {
        double esperado = 0.01;
        double atual = 0.1 * 0.1;
        double delta = 0.1;
        assertEquals(esperado, atual, delta);
    }
    @Test
    public void teste_4() {
        double esperado = 3.001;
        double atual = 0.1 * 0.1;
        assertEquals("Erro: Teste 4", esperado, atual, 0.1);
    }
}
```

Asserções do JUnit

- assertEquals(Object expected, Object actual)
  - Afirma que dois objetos são iguais.
  - Se eles não são, uma AssertionError é lançada sem mensagem.
  - Se o expected e actual são null, eles são considerados iguais.

- assertEquals(String message, Object expected, Object actual)
  - Afirma que dois objetos são iguais.
  - Se eles não são, uma AssertionError é lançada com uma dada mensagem.
  - Se o expected e actual são null, eles são considerados iguais.

Asserções do JUnit

- Arquivo de teste, exemplo:
  - ApredendoAssercoes2.java

```java
public class AprendendoAssercoes2 {

    @Test
    public void teste_1() {
        Pessoa p1 = new Pessoa("111.111.111-23", "Elder");
        Pessoa p2 = new Pessoa("111.111.111-23", "Elder");
        assertEquals(p1, p2);
    }

    @Test
    public void teste_2() {
        Pessoa p1 = new Pessoa("111.111.111-23", "Elder");
        Pessoa p2 = new Pessoa("111.111.222-23", "Elder");
        assertEquals("Erro: p1 é diferente de p2", p1, p2);
    }

    @Test
    public void teste_3() {
        String nome_1 = "Elder";
        String nome_2 = "Elder";
        assertEquals(nome_1, nome_2);
    }

    @Test
    public void teste_4() {
        String nome_1 = null;
        String nome_2 = null;
        assertEquals(nome_1, nome_2);
    }
}
```

Asserções do JUnit

- Arquivo de teste, exemplo:
  - ApredendoAssercoes2.java

```
public class AprendendoAssercoes2 {
    //...
    @Test
    public void teste_array_list() {
        List<Pessoa> l_arr_1 = new ArrayList();
        l_arr_1.add(new Pessoa("111", "Elder"));
        l_arr_1.add(new Pessoa("112", "Maria"));
        List<Pessoa> l_arr_2 = new ArrayList();
        l_arr_2.add(new Pessoa("111", "Elder"));
        l_arr_2.add(new Pessoa("112", "Maria"));
        assertEquals(l_arr_1, l_arr_2);
    }
    @Test
    public void teste_hash_set() {
        Set<Pessoa> l_set_1 = new HashSet();
        l_set_1.add(new Pessoa("111", "Elder"));
        l_set_1.add(new Pessoa("112", "Maria"));
        Set<Pessoa> l_set_2 = new HashSet();
        l_set_2.add(new Pessoa("111", "Elder"));
        l_set_2.add(new Pessoa("112", "Maria"));

        assertEquals(l_set_1, l_set_2);
    }
    @Test
    public void teste_hash_map() {
        Map<String, Pessoa> l_map_1 = new HashMap();
        l_map_1.put("111", new Pessoa("111", "Elder"));
        Map<String, Pessoa> l_map_2 = new HashMap();
        l_map_2.put("111", new Pessoa("111", "Elder"));
        assertEquals(l_map_1, l_map_2);
    }
```

Asserções do JUnit

- assertTrue(boolean condition)
  - Afirma que uma condição é verdadeira.
  - Se ela não é, uma AssertionError é lançada sem mensagem.

- assertTrue(String message, boolean condition)
  - Afirma que uma condição é verdadeira.
  - Se ela não é, uma AssertionError é lançada com um dada mensagem.

- assertFalse(boolean condition)
  - Afirma que uma condição é falsa.
  - Se ela não é, uma AssertionError é lançada sem mensagem.

- assertFalse(String message, boolean condition)
  - Afirma que uma condição é falsa.
  - Se ela não é, uma AssertionError é lançada com um dada mensagem.

Asserções do JUnit

- Arquivo de teste, exemplo:
  - ApredendoAssercoes3.java

```java
public class AprendendoAssercoes3 {

    @Test
    public void teste_1() {
        boolean valor = true;
        assertTrue(valor);
    }

    @Test
    public void teste_2() {
        boolean valor = false;
        assertTrue("Erro: esperado valor true", valor);
    }

    @Test
    public void teste_3() {
        boolean valor = false;
        assertFalse(valor);
    }

    @Test
    public void teste_4() {
        boolean valor = true;
        assertFalse("Erro: esperado valor false", valor);
    }
}
```

Asserções do JUnit

- assertNull(Object object)
  - Afirma que um objeto é null.
  - Se ele não é, uma AssertionError sem mensagem é lançada.

- assertNull(String message, Object object)
  - Afirma que um objeto é null.
  - Se ele não é, uma AssertionError com uma dada mensagem é lançada.

- assertNotNull(Object object)
  - Afirma que um objeto não é null.
  - Se ele é, uma AssertionError sem mensagem é lançada.

- assertNotNull(String message, Object object)
  - Afirma que um objeto não é null.
  - Se ele é, uma AssertionError com uma dada mensagem é lançada.

Asserções do JUnit

- Arquivo de teste, exemplo:
  - ApredendoAssercoes4.java

```java
public class AprendendoAssercoes4 {

    @Test
    public void teste_1() {
        String nome = null;
        assertNull(nome);
    }

    @Test
    public void teste_2() {
        String nome = "Elder";
        assertNull("Erro: nome não é null", nome);
    }

    @Test
    public void teste_3() {
        String nome = "Elder";
        assertNotNull(nome);
    }

    @Test
    public void teste_4() {
        String nome = null;
        assertNotNull("Erro: nome é null", nome);
    }
}
```

Asserções do JUnit

- fail( )
  - Falha um teste sem mensagem.
- fail(String message)
  - Falha um teste com uma data mensagem.

```
public class AprendendoAssercoes4 {


    @Test
    public void teste_fail_1() {
        fail();
    }

    @Test
    public void teste_fail_2() {
        fail("Erro: introduzindo uma falha");
    }

    // ...
```

Asserções do JUnit

- assertArrayEquals(int[ ] expecteds, int[ ] actuals)
  - Afirma que dois arrays int são iguais.
  - Se eles não são, uma AssertionError é lançada sem mensagem.

- assertArrayEquals(String message, int[ ] expecteds, int[ ] actuals)
  - Afirma que dois arrays int são iguais.
  - Se eles não são, uma AssertionError é lançada com uma data mensagem.

- assertArrayEquals(long[ ] expecteds, long[ ] actuals)
  - Afirma que dois arrays long são iguais.
  - Se eles não são, uma AssertionError é lançada sem mensagem.

- assertArrayEquals(String message, long[ ] expecteds, long[ ] actuals)
  - Afirma que dois arrays long são iguais.
  - Se eles não são, uma AssertionError é lançada com uma data mensagem.

- ApredendoAssercoes5.java

```java
public class AprendendoAssercoes5 {

    @Test
    public void teste_array_int_1() {
        int[] arr_int_1 = {1, 3, 4, 5};
        int[] arr_int_2 = {1, 3, 4, 5};
        assertArrayEquals(arr_int_1, arr_int_2);
    }

    @Test
    public void teste_array_int_2() {
        int[] arr_int_1 = {1, 3, 4, 5};
        int[] arr_int_2 = {1, 3, 7, 5};
        assertArrayEquals("Erro: os arrays são diferentes", arr_int_1, arr_int_2);
    }

    @Test
    public void teste_array_long_1() {
        long[] arr_long_1 = {1, 3, 4, 5};
        long[] arr_long_2 = {1, 3, 4, 5};
        assertArrayEquals(arr_long_1, arr_long_2);
    }

    @Test
    public void teste_array_long_2() {
        long[] arr_long_1 = {1, 3, 4, 5};
        long[] arr_long_2 = {1, 3, 7, 5};
        assertArrayEquals("Erro: os arrays são diferentes", arr_long_1, arr_long_2);
    }
}
```

Asserções do JUnit

- assertArrayEquals(byte[ ] expecteds, byte[ ] actuals)
  - Afirma que dois arrays bytes são iguais.
  - Se eles não são, uma AssertionError é lançada sem mensagem.

- assertArrayEquals(String message, byte[ ] expecteds, byte[ ] actuals)
  - Afirma que dois arrays bytes são iguais.
  - Se eles não são, uma AssertionError é lançada com uma data mensagem.

- assertArrayEquals(char[ ] expecteds, char[ ] actuals)
  - Afirma que dois arrays char são iguais.
  - Se eles não são, uma AssertionError é lançada sem mensagem.

- assertArrayEquals(String message, char[ ] expecteds, char[ ] actuals)
  - Afirma que dois arrays char são iguais.
  - Se eles não são, uma AssertionError é lançada com uma data mensagem.

Asserções do JUnit

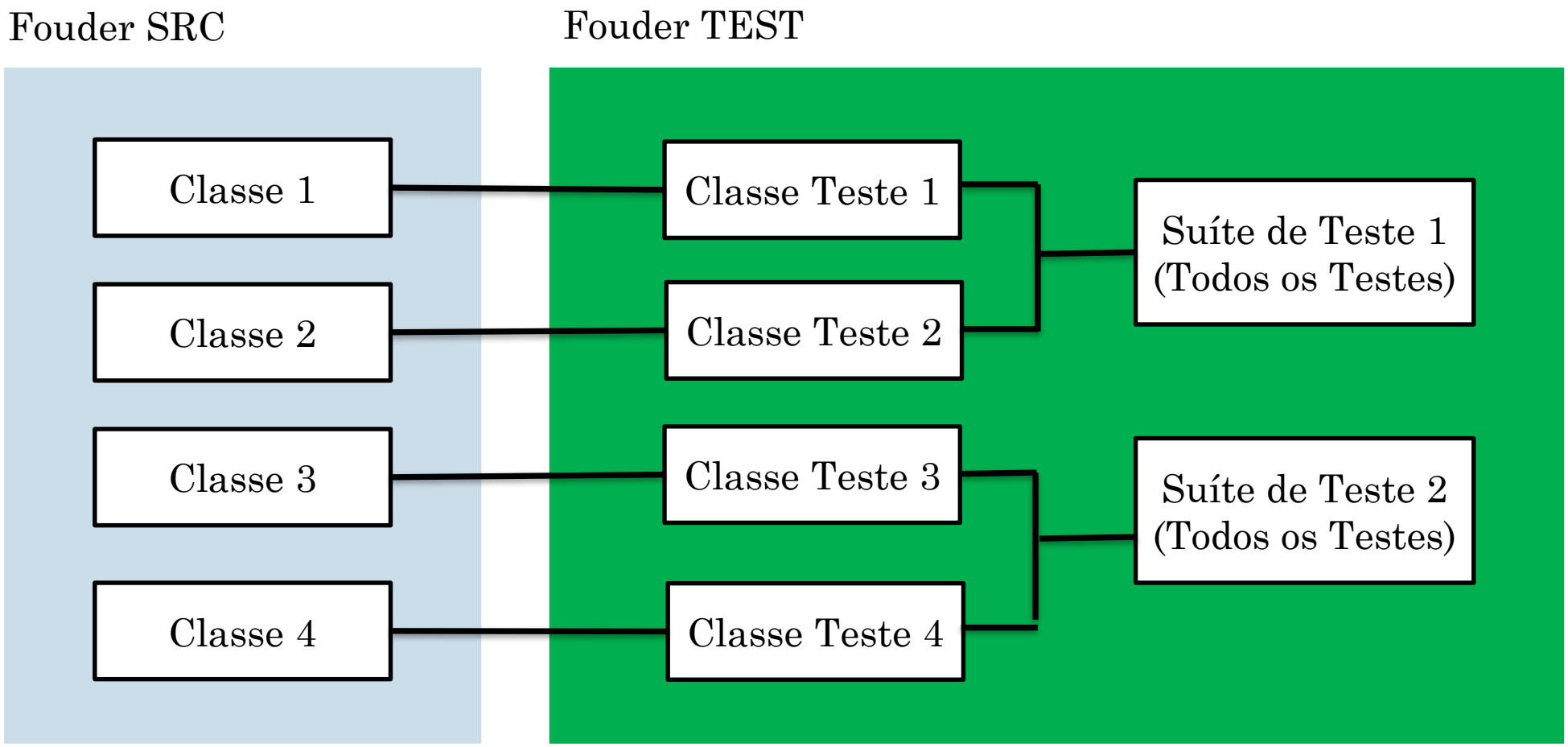
- assertArrayEquals(short[ ] expecteds, short[ ] actuals)
  - Afirma que dois arrays short são iguais.
  - Se eles não são, uma AssertionError é lançada sem mensagem.

- assertArrayEquals(String message, short[ ] expecteds, short[ ] actuals)
  - Afirma que dois arrays short são iguais.
  - Se eles não são, uma AssertionError é lançada com uma data mensagem.

- assertArrayEquals(double[ ] expecteds, double[ ] actuals, delta)
  - Afirma que dois arrays double são iguais.
  - Se eles não são, uma AssertionError é lançada sem mensagem.

- assertArrayEquals(String message, double[ ] expecteds, double[ ] actuals, delta)
  - Afirma que dois arrays double são iguais.
  - Se eles não são, uma AssertionError é lançada com uma data mensagem.

Esquema de Classes



Esquema de Classes

Suíte de Teste 1

```
@RunWith(Suite.class)
@SuiteClasses({ AprendendoAnotacoes.class, AprendendoAnotacoes2.class })
public class SuiteTesteAnotacoes {

}
```

Suíte de Teste 2

```
@RunWith(Suite.class)
@SuiteClasses({ AprendendoAssercoes.class, AprendendoAssercoes2.class,
    AprendendoAssercoes3.class, AprendendoAssercoes4.class,
    AprendendoAssercoes5.class })
public class SuiteTesteAssercoes {

}
```

Boas práticas com JUnit

- Cada classe do sistema deve ter sua própria classe testadora
- Cada método do sistema deve ter seu próprio método testador
- O nome de cada classe testadora deve seguir um padrão, por exemplo:
  - TestNomeClasseTestada
- O nome de cada método testador deve seguir também um padrão, por exemplo:
  - testNomeMetodoTestado
- Classes testadoras devem ser armazenadas em pastas separadas no projeto

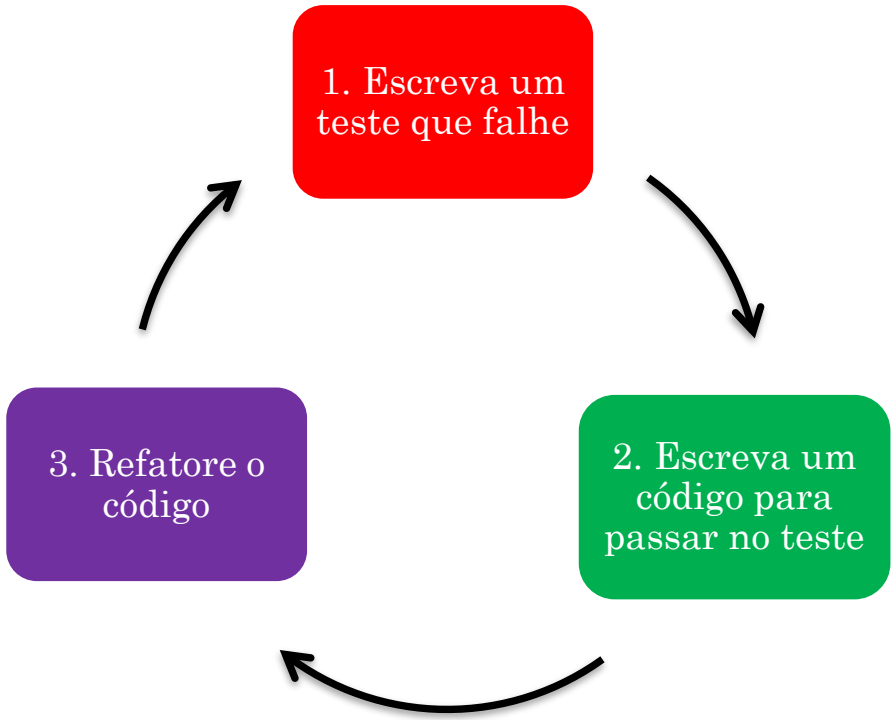
Introdução ao TDD (Test Driven Development)

- Também conhecido como Desenvolvimento Guiado por Testes
  - Tem como ideia, escrever primeiro o teste para posteriormente escrever o código

Introdução ao TDD (Test Driven Development)

- Ciclo do TDD:

1) O desenvolvedor escreve um caso de teste, chamado de teste falho
   - A funcionalidade ainda não existe, desta forma, o teste deve retornar uma falha
2) O desenvolvedor escreve um código para fazer com que o teste seja bem sucedido
   - O desenvolvedor já sabe que código implementar
3) O código escrito deve ser refatorado
   - Elimine duplicações de código
   - Caso precise



TDD - Algumas Vantagens

- O desenvolvedor já tem definido o comportamento do código antes de implementar
- Evita que bugs sejam adicionados ao código
- Tonam os bugs mais fáceis de serem corrigidos
- Mantém o desenvolvedor focado no problema a ser resolvido
- Manutenção de código garantida pela refatoração
- Deixa códigos mais simples

Exemplo Prático - Calculadora

- Crie testes de unidade para os métodos das quatro operações básicas de uma classe chamada Calc
  - Somar,
  - Subtrair,
  - Multiplicar,
  - Dividir

Toteiro de Testes			
Casos de teste	**Operação**	**Entradas Planejadas**	**Saídas Esperadas**
CT01	soma	a = 5 | b = 6	11
CT02	subtração	a = 5 | b = 6	-1
CT03	mult	a = 6 | b = 6.5	39
CT04	divisão	a = 10 | b = 2	5
CT05	divisão	a=10 | b=0	Exceção

Exemplo Prático - Calculadora

1. Criando método de teste para soma;

```
public class CalcTest {

    @Test
    public void test_soma() {
        double esperado = 11;
        double atual = Calc.somar(5, 6);
        assertEquals(esperado, atual, 0.01);
    }
```

1. Escreva um teste que falhe

Exemplo Prático - Calculadora

2. Criando método para somar;

```
public class Calc {

    public static double somar(double a, double b){
        return a + b;
    }
```

2. Escreva um código para passar no teste

Exemplo Prático - Calculadora

3. Criando método de teste para subtração;

```
public class CalcTest {

    @Test
    public void test_soma() {
        double esperado = 11;
        double atual = Calc.somar(5, 6);
        assertEquals(esperado, atual, 0.01);
    }
    @Test
    public void test_subtracao() {
        double esperado = -1;
        double atual = Calc.subtrair(5, 6);
        assertEquals(esperado, atual, 0.01);
    }
```

1. Escreva um teste que falhe

Exemplo Prático - Calculadora

4. Criando método para subtrair;

```
public class Calc {

    public static double somar(double a, double b){
        return a + b;
    }

    public static double subtrair(double a, double b) {
        return a - b;
    }
```

2. Escreva um código para passar no teste

Exemplo Prático - Calculadora

5. Criando método de teste para multiplicação;

```
public class CalcTest {

    public void test_soma() {
    public void test_subtracao() {

    @Test
    public void test_multiplicao() {
        double esperado = 39;
        double atual = Calc.multiplicar(6, 6.5);
        assertEquals(esperado, atual, 0.01);
    }
}
```

1. Escreva um teste que falhe

Exemplo Prático - Calculadora

6. Criando método para multiplicar;

```
public class Calc {

    public static double somar(double a, double b){
        return a + b;
    }

    public static double subtrair(double a, double b) {
        return a - b;
    }
    public static double multiplicar(double a, double b) {
        return a * b;
    }
}
```

2. Escreva um código para passar no teste

Exemplo Prático - Calculadora

7. Criando método de teste para divisão;

```
public class CalcTest {

    public void test_soma() {
    public void test_subtracao() {
    public void test_multiplicao() {

    @Test
    public void test_divisao() {
        double esperado = 5;
        double atual = Calc.dividir(10, 2);
        assertEquals(esperado, atual, 0.01);
    }
}
```

1. Escreva um teste que falhe

Exemplo Prático - Calculadora

8. Criando método para dividir;

```
public class Calc {

    public static double somar(double a, double b){
        return a + b;
    }

    public static double subtrair(double a, double b) {
        return a - b;
    }

    public static double multiplicar(double a, double b) {
        return a * b;
    }

    public static double dividir(double a, double b) {
        return a / b;
    }
```

2. Escreva um código para passar no teste

Exemplo Prático - Calculadora

9. Criando método de teste para divisão por zero;

```
public class CalcTest {

    public void test_soma() {
    public void test_subtracao() {
    public void test_multiplicao() {

    @Test
    public void test_divisao() {
        double esperado = 5;
        double atual = Calc.dividir(10, 2);
        assertEquals(esperado, atual, 0.01);
    }

    @Test(expected=ArithmeticException.class)
    public void test_divisao_zero() {
        double valor = Calc.dividir(10, 0);
    }
```

1. Escreva um teste que falhe

Exemplo Prático - Calculadora

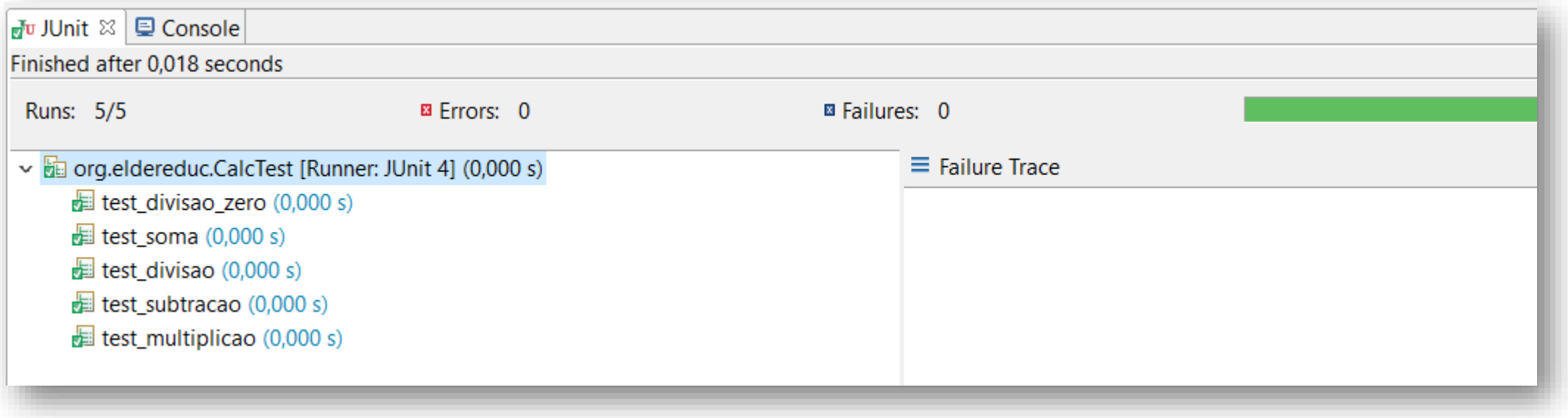
10. Refinando método para dividir por zero;

```java
public class Calc {

    public static double somar(double a, double b){
        return a + b;
    }

    public static double subtrair(double a, double b) {
        return a - b;
    }

    public static double multiplicar(double a, double b) {
        return a * b;
    }

    public static double dividir(double a, double b) {
        if(b == 0){
            throw new ArithmeticException("Erro: Divisão por ZERO");
        }
        return a / b;
    }
```

3. Refatore o código

Exemplo Prático - Calculadora

11. Final de execução;



Fim

Referências

- Introdução ao JUnit. Disponível em: <http://junit.org/junit4/>
- Class Assert JUnit. Disponível em: <http://junit.sourceforge.net/javadoc/org/junit/Assert.html#assertEquals%28java.lang.String, %20double,%20double,%20double%29 >
- Introdução ao Desenvolvimento Guiado por Teste. Disponível em: http://www.devmedia.com.br/introducao-ao-desenvolvimento-guiado-por-teste-tdd-com-junit/26559
- TDD: Fundamentos do desenvolvimento orientado por testes. Disponível em: <http://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151>