

# Teste de Software



Prof. Marcos Rodrigo momo, M.Sc.  
[marcos.momo@ifsc.edu.br](mailto:marcos.momo@ifsc.edu.br)

Gaspar, maio 2021.

# Roteiro aula 8



- Gravação
- Calendário
- Teste manual e teste automático
- Teste estáticos e teste dinâmico
- Tipos de testes
- Automatizando testes com JUnit
- Atividades



# Cronograma das aulas módulo 2



**Observação: não termos mais aula no sábado  
5 encontros**

MÓDULO 2: 24/05 – 26/06 (5 semanas)						
2ª feira	3ª feira	4ª feira	5ª feira	6ª feira	sábado	
						07:20:00
						08:15:00
					prog internet II	09:10:00
					prog internet II	10:25:00
						11:20:00
						13:30:00
teste de software	prog.internet II	prog.internet II				18:30:00
teste de software	prog.internet II	prog.internet II	TCC2			19:25:00
teste de software	prog.internet II	prog.internet II	TCC2			20:40:00
teste de software	prog.internet II	prog.internet II				21:35:00

# Calendário ADS 4



## ADS 4

UCs	CH	Docente	Módulo 1				Módulo 2				Módulo 3				Módulo 4					finalização e conselhos de classe
1 Teste de Software	40	MARCOS RODRIGO MOMO/	6	6	6	6	4		4	4	4									
2 Análise de Sistemas II	80	ROGÉRIO ANTONIO SCHMITT/	8	8	8	8	10	8	10	10	10									
3 Práticas em Desenvolvimento de Sistemas I	80	TAMER CAVALCANTE/										10	10	10	10	10	10	10	10	
4 Sistemas Operacionais	40	ANDREU CARMINATI/														10	10	10	10	
5 Metodologia de Pesquisa	40	LEONIDAS de MELLO Jr./	10				0	0	0	0	0		10	10	10	0	0	0	0	
6 Programação para Internet II	80	MARCOS RODRIGO MOMO/		10	10	10	10	10	10	10	10									
7 Gerência de Projetos	40	THIAGO PAES/										8	4	4	4	4	4	4	4	
<b>TOTAL</b>	<b>400</b>	<b>TOTAL</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>18</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>18</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>24</b>	<b>4</b>	



# Testes manuais X Testes automáticos



- Os testes automáticos, podem ser melhor utilizados principalmente, nos testes de unidade e teste de integração
- Nas outras partes do sistema, podem significar um custo muito excessivo, nesse caso realizamos os testes manuais

# JUnit - O que é



- JUnit é um Framework *open-source* utilizado para facilitar o desenvolvimento de códigos em Java verificando se os resultados gerados pelos métodos são os esperados
- Caso não sejam, o JUnit exibe os possíveis erros que estão ocorrendo nos métodos
- Essa verificação é chamada de teste unitário ou teste de unidade

# JUnit - Para que serve



- Atualmente, buscando cada vez mais melhorias nos softwares, os desenvolvedores fazem uma bateria de testes nos seus códigos
- Um desses testes
  - Por exemplo: teste de unidade
    - Testa a menor parte do código buscando garantir uma maior qualidade do produto no processo de desenvolvimento
- No caso da linguagem Java esse teste é feito através do JUnit em cada método separadamente

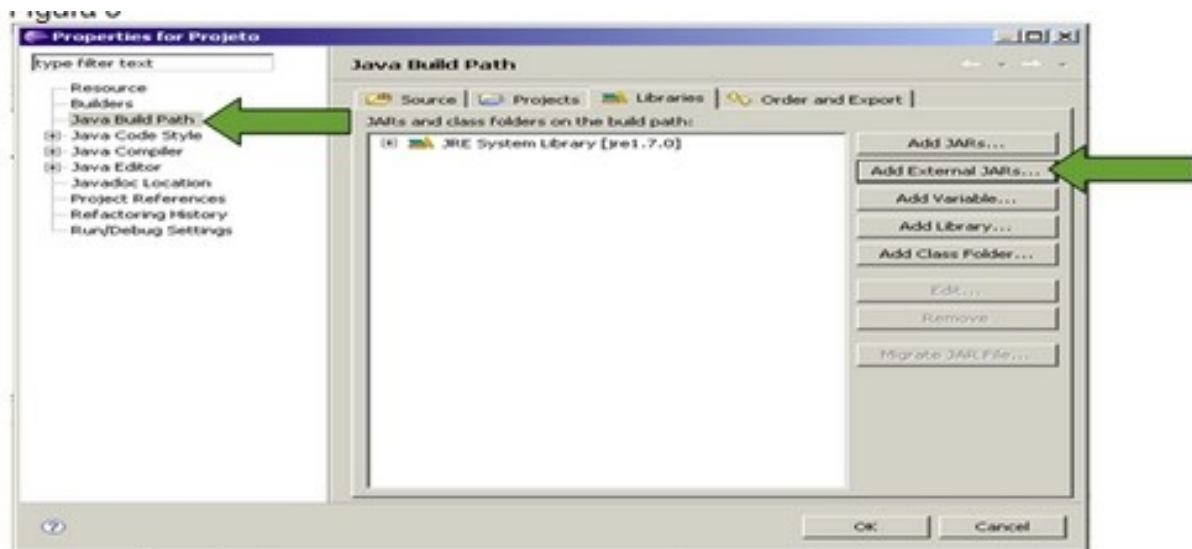
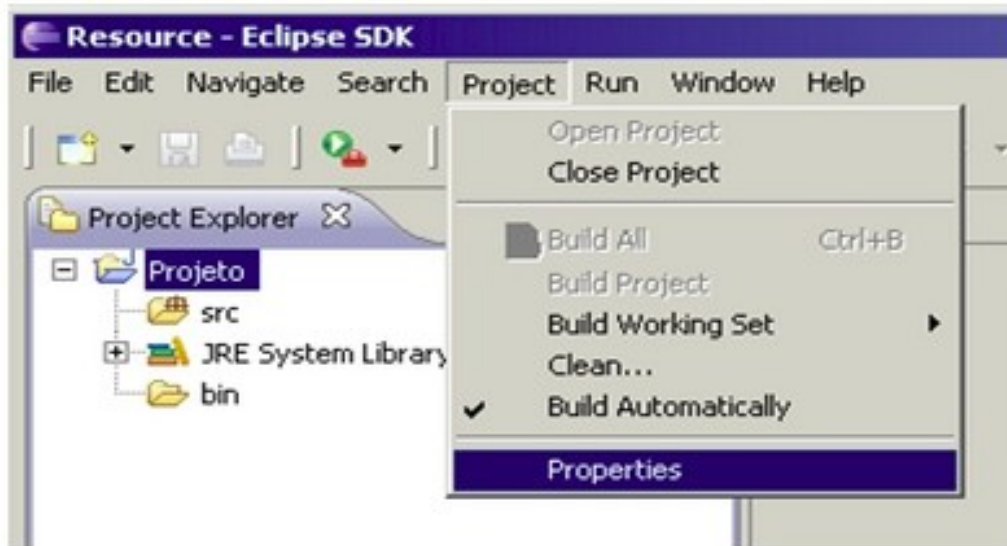
# JUnit - Instalação



- Fazer download das bibliotecas (SIGAA)
  - Junit
  - Hamcrest-core
- Importar para do projeto



# JUnit - Instalação






# Atividades práticas

## Configuração do ambiente



- Nessas atividades práticas utilizaremos:
  - IDE Eclipse
  - Junit

---

 SistemaBancario	3 itens
 hamcrest-core-1.3.jar	45,0 kB
 junit-4.13.jar	381,8 kB

```
package negocio;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;
import java.util.ArrayList;
import java.util.List;
import org.junit.Test;

public class GerenciadoraClientesTest_Ex1 {

    @Test

    public void testPesquisaCliente() {

        // criando alguns clientes

        Cliente cliente01 = new Cliente(1, "Gustavo Farias", 31, "gugafarias@gmail.com", 1, true);
        Cliente cliente02 = new Cliente(2, "Felipe Augusto", 34, "felipeaugusto@gmail.com", 2, true);

        // inserindo os clientes criados na lista de clientes do banco

        List<Cliente> clientesDoBanco = new ArrayList<>();

        clientesDoBanco.add(cliente01);
        clientesDoBanco.add(cliente02);

        GerenciadoraClientes gerClientes = new GerenciadoraClientes(clientesDoBanco);

        Cliente cliente = gerClientes.pesquisaCliente(1);

        assertThat(cliente.getId(), is(2));

        assertThat(cliente.getEmail(), is("gugafarias@gmail.com"));

    }
}
```

# Atividade prática com Junit

## Sistema Bancário

Teste 1: Implementar um teste para a classe GerenciadoraClientes para o método pesquisaCliente

# Teste de unidade



- O teste de unidade é importante na construção de métodos, pois permite ao programador testá-los durante a construção do sistema garantindo com isso a implementação de métodos livres de erros lógicos, que ocorrem com muita frequência
- Por possibilitar os testes antes da conclusão do sistema, o programador pode testar seus métodos separadamente assim que eles estejam prontos
- Isso evita que o programador fique percorrendo o código inteiro para descobrir os erros lógicos que apareciam quando o programa já estava pronto

# Partes de um teste



```
public class GerenciadoraClientesTest_Ex3 {  
    private GerenciadoraClientes gerClientes;  
  
    @Test  
    public void testPesquisaCliente() {  
  
        /* ===== Montagem do cenário ===== */  
  
        // criando alguns clientes  
        Cliente cliente01 = new Cliente(1, "Gustavo Farias", 31, "gugafarias@gmail.com", 1, true);  
        Cliente cliente02 = new Cliente(2, "Felipe Augusto", 34, "felipeaugusto@gmail.com", 1, true);  
  
        // inserindo os clientes criados na lista de clientes do banco  
        List<Cliente> clientesDoBanco = new ArrayList<>();  
        clientesDoBanco.add(cliente01);  
        clientesDoBanco.add(cliente02);  
  
        gerClientes = new GerenciadoraClientes(clientesDoBanco);  
  
        /* ===== Execução ===== */  
        Cliente cliente = gerClientes.pesquisaCliente(1);  
  
        /* ===== Verificações ===== */  
        assertThatassertThat(cliente.getId(), is(1));  
    }  
}
```

# Partes de um teste



- Otimizando testes com cenários parecidos com duas anotações do Junit:
  - @Before - setUp()
    - Configurar, preparar, em outras palavras montar o cenário
    - Esse método executa antes de cada método de teste
  - @After - tearDown()
    - Executa depois de cada método de teste
    - O Junit executa o tearDown() automaticamente após cada método de teste ser realizado
    - O resultado da execução é a “limpeza” de todas as variáveis utilizadas no cenário pelo método de teste, ou seja, vai liberar todos os recursos utilizados no método de teste
    - É como se voltasse na configuração para a criação de um novo cenário de testes

# Exemplo de uso anotações @Before e @After



```
package negocio;

import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;

import java.util.ArrayList;
import java.util.List;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

/**
 * Classe de teste criada para garantir o funcionamento das principais operações
 * sobre clientes, realizadas pela classe {@link GerenciadoraClientes}.
 * @author Gustavo Farias
 * @date 21/01/2035
 */
public class GerenciadoraClientesTest_Ex7 {
    private GerenciadoraClientes gerClientes;
    private int idCliente01 = 1;
    private int idCliente02 = 2;
```



# Exemplo de uso anotações @Before e @After



```
@Before
public void setUp() {
    /* ===== Montagem do cenário ===== */

    // criando alguns clientes
    Cliente cliente01 = new Cliente(idCLiente01, "Gustavo Farias", 31, "gugafarias@gmail.com", 1, true);
    Cliente cliente02 = new Cliente(idCLiente02, "Felipe Augusto", 34, "felipeaugusto@gmail.com", 1, true);

    // inserindo os clientes criados na lista de clientes do banco
    List<Cliente> clientesDoBanco = new ArrayList<>();
    clientesDoBanco.add(cliente01);
    clientesDoBanco.add(cliente02);

    gerClientes = new GerenciadoraClientes(clientesDoBanco);

    System.out.println("Before foi executado");
}

@After
public void tearDown() {
    gerClientes.limpa();

    System.out.println("After foi executado");
}
```



@Test

```
public void testPesquisaCliente() {
```

```
    /* ===== Execu000o ===== */
```

```
    Cliente cliente = gerClientes.pesquisaCliente(idCLiente01);
```

```
    /* ===== Verifica00es ===== */
```

```
    assertThatassertThat(cliente.getId(), is(idCLiente01));
```

```
    System.out.println("TesPesquisarCliente executado");
```

```
}
```

```
/**
```

```
 * Teste básico da remo00o de um cliente a partir do seu ID.
```

```
 *
```

```
 * @author Gustavo Farias
```

```
 * @date 21/01/2035
```

```
 */
```

@Test

```
public void testRemoveCliente() {
```

```
    /* ===== Execu000o ===== */
```

```
    boolean clienteRemovido = gerClientes.removeCliente(idCLiente02);
```

```
    /* ===== Verifica00es ===== */
```

```
    assertThatassertThat(clienteRemovido, is(true));
```

```
    assertThatassertThat(gerClientes.getClientesDoBanco().size(), is(1));
```

```
    assertNull(gerClientes.pesquisaCliente(idCLiente02));
```

```
    System.out.println("TesRemoverCliente executado");
```

```
}
```

```
}
```



# Saídas anotações @Before e @After



```
Before foi executado  
TesPesquisarCliente executado  
After foi executado  
Before foi executado  
TesRemoverCliente executado  
After foi executado
```

# Suíte de testes



- Uma suíte de testes é uma coleção de casos de teste destinados a testar um sistema para verificar a determinados comportamentos.
- As suítes de teste são usualmente divididas de acordo com as funcionalidades do sistema ou com o tipo de teste executado

# Classe de suíte de testes



- Suíte de testes é um conjunto de testes de software unitário desenvolvidos para realizar testar um módulo do sistema.
- Possibilita realizar todos os testes (as classes de testes) a partir de uma única classe de teste
- Criar uma classe java com duas anotações
  - `@RunWith(Suite.class)`
  - `@SuiteClasses`
- Passar como parâmetro as classes de testes



# Por exemplo: Classe: todosOsTestes.java



```
GerenciadoraClientesTest_Ex1.java  Cliente.java  *todosOsTestes.java ✖
1  package negocio;
2
3  import org.junit.runner.RunWith;
4  import org.junit.runners.Suite;
5  import org.junit.runners.Suite.SuiteClasses;
6
7  @RunWith(Suite.class)
8  @SuiteClasses({
9      GerenciadoraClientesTest_Ex1.class,
10     GerenciadoraClientesTest_Ex2.class,
11     GerenciadoraClientesTest_Ex3.class,
12     GerenciadoraClientesTest_Ex4.class,
13     GerenciadoraClientesTest_Ex8.class,
14     GerenciadoraClientesTest_Ex10.class,
15     GerenciadoraContasTest_Ex3.class,
16     GerenciadoraContasTest_Ex4.class,
17     GerenciadoraContasTest_Ex6.class,
18     GerenciadoraContasTest_Ex11.class,
19 })
20
21 public class todosOsTestes {
22
23 }
24
```

Problems Javadoc Declaration Progress ✖

# Atividades Práticas de testes



- Utilizando o projeto de Sistema Bancário, disponibilizado na aula 8, realizar testes unitários automáticos para:
  - Testar todos os métodos da classe GerenciadoraClientes
  - Testar todos os métodos da classe GerenciadoraContas
- Para todas as classes, usar as boas práticas de programação de testes:
  - Organização em pacotes das classes de testes
  - Nomeação de classes e métodos de teste
  - Fazer a documentação necessária para explicar cada teste
- Postar no SIGAA o projeto contendo todas as classes de testes