|  |
| --- |
|  |
| |  | | --- | |  |     **Material do curso de automação – básico** |
| Aula 1 |
|  |
|  |
|  |

**Quadro de Revisão do Documento**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versão** | **Data** | **Autor** | **Descrição** |
| 1.0 | 25/06/2019 | Felipe Amorim | Criação do Documento |

**Lista de Distribuição**

|  |  |  |
| --- | --- | --- |
| **Nome** | **Responsabilidade** | **E-mail** |
| Paulo Luis | N/A | pluis@artit.com.br |
| Roberto Junior Tempesta | N/A | rjtempesta@artit.com.br |
| Murilo Antunes Alves | N/A | maalves@artit.com.br |
| Renato Oliveira | N/A | rvoliveira@artit.com.br |
| Denis Augusto L. de Castro | N/A | dlcastro@artit.com.br |
| Cesar Augusto Cordeiro | N/A | ccordeiro@artit.com.br |
| Estevan Luis Gregori | N/A | egregori@artit.com.br |
| Rubens Gonçalves | N/A | rngoncalves@artit.com.br |

**Sumário**

[Objetivo do Documento 4](#_Toc15253129)

[1. Documentos de Referência 4](#_Toc15253130)

[2. Definições, Acrônimos e Abreviaturas. 4](#_Toc15253131)

[3. Conteúdos 4](#_Toc15253132)

[3.1 Revisão. 4](#_Toc15253133)

[3.2 JUnit. 8](#_Toc15253134)

[3.3 Variáveis de ambiente e de sistema. 11](#_Toc15253135)

# Objetivo do Documento

Transmitir o material passado durante a primeira aula do curso de automação de testes – básico.

# Documentos de Referência

N/A

# Definições, Acrônimos e Abreviaturas.

N/A.

# Conteúdos

## Revisão.

Sabendo que é um pré-requisito algum curso de introdução à programação ou ao Java, toma-se como princípio que este tópico tem como objetivo apenas relembrar alguns itens importantes na hora de estabelecer um projeto Java.

**Comentários:**

//Exemplo de comentário

/\*

Exemplo de comentário para multiplas linhas

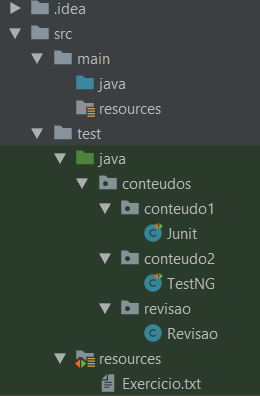
\*/

//TODO exemplo de comentário em highlight

**Pacotes Java:**

Pacotes (packages) Java são todas as pastas dentro dos diretórios delimitados pelo compilador como fonte do Java. Pacotes devem sempre começar com letras minúsculas.

Exemplo:



-não é um pacote  
-não é um pacote  
-não é um pacote  
-diretório delimitador  
-não é um pacote  
-não é um pacote  
-diretório delimitador  
-pacote Java  
-pacote Java

**Classes:**

Classes e interfaces são os arquivos Java que contém fonte para ser compilados

Classes devem sempre começar com letras maiúscula

Classes e interfaces devem sempre estar dentro dos diretórios Java

CamelCase para classes e variáveis, é a escrita mais indicada

Exemplo de CamelCase: ClasseDoPrimeiroDia

**Static:**

Métodos ou variáveis estáticas (static) são itens que pertencem à classe e não à sua instância

Métodos ou variáveis estáticas só conseguem 'conversar' com métodos e variáveis estáticas

Exemplo:

Para uma chamada de classe A, define-se a variável int a = 0 e posteriormente, atribui-se o valor a = 10;

Se a variável for estática, em uma segunda chamada à classe A, ainda teremos o valor 10.

Se a variável não for estática, em uma segunda chamada à classe, seu valor volta a ser 0 (valor de compilação)

OBS: Simplificando, métodos e variáveis estáticos mantém valores até que algum método os apague, já os não estáticos têm seus valores zerados à cada chamada.

**Tipos de variáveis**:

int - inteiros; numeros sem casas decimais de até 4 bytes

long - numeros sem casa decimais de até 8 bytes

float - numeros contendo casas decimais de até 4 bytes

double - numeros contendo casas decimais de até 8 bytes

String - textos

char - caracteres individuais

boolean - verdadeiro ou falso/true ou false

Object - container generalizado, armazena qualquer dado (gasta mais memória)

**Métodos:**

Um método é construído da seguinte forma

**1** - Seu nível de acesso (exemplo: public, private, protected)

OBS: O mesmo pode ser ignorado, e então, o nível de acesso será restrito apenas para classes do mesmo pacote

**2** - Seu retorno (void, String, int, boolean, etc)

OBS: Qualquer variável ou objeto pode ser retornado de um método

OBS2: Salvo nulo, qualquer retorno declarado torna obrigatório o uso da função 'return'

**3** - Seu nome sem espaços ou caracteres especiais (exemplo: meuPrimeiroMetodo)

OBS: Métodos, assim como pacotes, devem iniciar apenas com letras minusculas, letras maiúsculas são apenas para classes

OBS: Assim como classes, métodos devem ser escritos em CamelCase

**4** - Seus parâmetros de entrada (exemplo: int posicaoX, int posicaoY)

OBS: Qualquer variável ou objeto pode ser um parâmetro de entrada de um método

Exemplo de método construído:

public int metodoSoma(int a, int b){

return a+b;

}

Para chamar um método, basta digitar o seu nome (considerando instanciação de classes e hierarquia) e inserir seus argumentos.

Exemplo:

metodoSoma(2,3);

Exemplo considerando uma chamada de outra classe:

new Revisao().metodoSoma(2,3);

Exemplo considerando uma chamada de outra classe2:

Revisao revisao = new Revisao();

revisao.metodoSoma(2,3);

Exemplo considerando o método como estático:

Revisao.metodoSoma(2,3);

Exemplo considerando o método como estático2:

import static conteudos.revisao.Revisao.metodoSoma;

metodoSoma(2,3);

Exemplo considerando hierarquia de classes:

public class Exemplo extends Revisao

metodoSoma(2,3);

**Maven:**

Maven é um controlador de artefatos/dependências (Outro exemplo é o gradle)

Através do arquivo POM.xml, localizado sempre na raiz do projeto, é possível inserir uma série de configurações do projeto

Exemplo:

Como estamos utilizando os plugins Junit e TestNG, colocamos a dependência no maven na tag <dependencies> e os arquivos .jar são automaticamente baixados para o nosso %appdata% e inseridos no projeto

## JUnit.

Junit é um plugin gratuito e foi inicialmente feito para atender à demanda de testes unitários.

**Tags**:

**@Test** - Executa um determinado bloco de código, dispensando artefatos .jar e compiladores via --Java

**@Before** - É executado antes de todos os @Test

**@BeforeClass** - É executado uma vez, antes do primeiro @Before, ou seja, quando a classe compila

**@After** - É executado depois de todos os @Test

**@AfterClass** - É executado uma vez, depois do ultimo @After, ou seja, assim que os testes terminam

OBS: As tags podem ser inseridas em qualquer ordem, não impacta na sua execução.

**Vantagens**:

Possui integração nativa com muitas ferramentas ótimas como mockito, cucumber, prettyreport, etc.

Possui chamadas de massa de dados fantásticas, muito fáceis de usar

**Desvantagens:**

Possui limitações de uso

Exemplo: poucos parâmetros de entrada nos @Before e @After

Exemplo2: Os métodos @BeforeClass e @AfterClass devem ser estáticos, já o @Before e @After não podem, o que as vezes dificulta a usabilidade

Segue como exemplo a implementação do assertEquals, serve para comparar variáveis e exibir a comparação caso ocorra falha.

@BeforeClass  
public static void beforeClass(){  
 System.*out*.println("Inicio da classe");  
}  
  
@Before  
public void before(){  
 System.*out*.println("Antes do teste");  
}  
  
@After  
public void after(){  
 System.*out*.println("Depois do teste");  
}  
  
@AfterClass  
public static void afterClass(){  
 System.*out*.println("Fim da classe");  
}  
  
@Test  
public void primeiroTeste(){  
 System.*out*.println("Teste de soma");  
 System.*out*.println(*metodoSoma*(3,4));  
}  
  
@Test  
public void segundoTeste(){  
 System.*out*.println("Teste de multiplicação");  
 System.*out*.println(metodoMultiplicacao(3,4));  
}  
  
@Test  
public void testeDeIgualdadeInt(){  
 System.*out*.println("Teste de igualdade inteiros");  
 *assertEquals*(*metodoSoma*(2,2), metodoMultiplicacao(2,2));  
}  
  
@Test  
public void testeDeIgualdadeString(){  
 System.*out*.println("Teste de igualdade string");  
 *assertEquals*("Paulo", "Paulo");  
}  
  
@Test  
public void testeDeIgualdadeStringComErroForcado(){  
 System.*out*.println("Teste de igualdade string com erro");  
 *assertEquals*("Paulo", "paulo");  
}  
  
@Test  
public void testeDeIgualdadeComErroForcado(){  
 System.*out*.println("Teste de igualdade int com erro");  
 *assertEquals*(*metodoSoma*(3,2), metodoMultiplicacao(2,2));  
}

## Variáveis de ambiente e de sistema.

O TestNG foi feito a partir do Junit, tentando aprimorar as suas 'falhas' e também inserir outras funcionalidades

Assim como no Junit, o TestNG possui as mesmas tags, sendo elas

**Tags:**

**@Test** - Executa um determinado bloco de código, dispensando artefatos .jar e compiladores via --Java

**@BeforeMethod** - É executado antes de todos os @Test

**@BeforeClass** - É executado uma vez, antes do primeiro @Before, ou seja, quando a classe compila

**@AfterMethod** - É executado depois de todos os @Test

**@AfterClass** - É executado uma vez, depois do ultimo @After, ou seja, assim que os testes terminam

OBS: As tags podem ser inseridas em qualquer ordem, não impacta na sua execução

**Vantagens:**

Correções de alguns problemas, como por exemplo a implementação forçada de static para @BeforeClass e @AfterClass

Uma variedade enorme de parâmetros de entrada, listeners, construtores e configurações de qualidade

**Desvantagens:**

Pouca integração com massa de testes

**Segue como exemplo:**

1. A implementação do Assert.assertEquals, serve para comparar variáveis e exibir a comparação caso ocorra falha
2. A implementação do SoftAssert, que não impede a execução até que se chame o método assertAll();.
3. A utilização do parâmetro 'priority' que ordena quais testes devem ser executados primeiro
4. A utilização do parâmetro 'dependsOnMethods' que não executa o teste, caso algum método declarado tenha falhado
5. A utilização do parâmetro 'description' que insere uma descrição para o teste (útil para relatórios e acompanhamento)

@BeforeMethod  
public void beforeTest(ITestResult iTestResult){  
 System.*out*.println("Antes do teste - "+iTestResult.getMethod().getMethodName());  
}  
  
@BeforeClass  
public void beforeClass(){  
 System.*out*.println("Inicio da classe");  
}  
  
@AfterMethod  
public void afterMethod(ITestResult iTestResult){  
 String status = iTestResult.getStatus() == 1 ? "Sucesso" : "Falha";  
 System.*out*.println("Depois do teste - "+iTestResult.getMethod().getMethodName()+" - "+status);  
}  
  
@AfterClass  
public void afterClass(){  
 System.*out*.println("Fim da classe");  
}  
  
@Test(priority = 3, description = "Exemplo de descrição de teste")  
public void primeiroTeste(){  
 System.*out*.println("primeiro teste");  
}  
  
@Test(priority = 2)  
public void segundoTeste(){  
 System.*out*.println("Teste de igualdade string com erro");  
 Assert.*assertEquals*("Paulo", "paulo");  
 new SoftAssert().assertEquals("Paulo", "paulo");  
}  
  
@Test(priority = 1)  
public void terceiroTeste(){  
 System.*out*.println("Teste de igualdade string com erro antes do primeiro assert");  
 SoftAssert softAssert = new SoftAssert();  
 softAssert.assertEquals("Paulo", "paulo");  
 System.*out*.println("Teste de igualdade string com erro depois do primeiro assert");  
 softAssert.assertEquals("Marcelo", "paulo");  
 System.*out*.println("Teste de igualdade string com erro depois do segundo assert");  
 softAssert.assertAll();  
}  
  
@Test(priority = 4, description = "Exemplo de descrição de teste", dependsOnMethods = {"primeiroTeste", "segundoTeste"})  
public void quartoTeste(){  
 System.*out*.println("quarto teste");  
}