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

**Quadro de Revisão do Documento**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versão** | **Data** | **Autor** | **Descrição** |
| 1.0 | 07/06/2019 | Felipe Amorim | Criação do Documento |

**Lista de Distribuição**

|  |  |  |
| --- | --- | --- |
| **Nome** | **Responsabilidade** | **E-mail** |
| Adriane Kaori Oshiro | N/A | aoshiro@artit.com.br |
| Aldemir Francisco Sales Júnior | N/A | asales@artit.com.br |
| Daniel Venturini | N/A | dventurini@artit.com.br |
| Jean Duclos | N/A | jduclos@artit.com.br |
| Marcelo Roland Bernardino | N/A | mrbernardino@artit.com.br |
| Marcelo Sim de Oliveira | N/A | msoliveira@artit.com.br |

**Sumário**

[Objetivo do Documento 4](#_Toc11001445)

[1. Documentos de Referência 4](#_Toc11001446)

[2. Definições, Acrônimos e Abreviaturas. 4](#_Toc11001447)

[3. Conteúdos 4](#_Toc11001448)

[3.1 Tipos de chamadas de classe. 4](#_Toc11001449)

[3.2 Outputs. 5](#_Toc11001450)

[3.3 Variáveis de ambiente e de sistema. 8](#_Toc11001451)

[3.4 Diretórios. 9](#_Toc11001452)

[3.5 Arquivos. 9](#_Toc11001453)

[3.6 Conteúdo extra – If/ Else in line & validação de boleanos. 10](#_Toc11001454)

# Objetivo do Documento

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

# Documentos de Referência

N/A

# Definições, Acrônimos e Abreviaturas.

N/A.

# Conteúdos

## Tipos de chamadas de classe.

Métodos estáticos são essencialmente métodos trancados com valores definidos em tempo de compilação, ou seja, não conseguem acessar métodos dinâmicos em tempo de execução, por terem sua escrita feita em compilação. Entretanto, métodos e variáveis estáticos, aceitam novos valores em tempo de execução, quando este vem de um método não estático.

Métodos e variáveis estáticas, podem armazenar valores mesmo com diferentes chamadas de classe e instâncias

Exemplo:

Em uma classe Main, temos as seguintes variáveis em tempo de compilação (Momento de execução inicial)

static int a = 10;

int b = 5;

A classe Executora1 faz as seguintes modificações em tempo de execução

a = 20;

b = 10;

Posteriormente, a classe Executora2 faz a seguinte chamada

System.out.println(a);

System.out.println(b);

O valor imprimido em tela será:

20

5

Explicação:

Como a variável a possui o parametro static, ela armazena o valor para todas as chamadas de classe (permanece estática, imutável)

Como a variável b não possui o parametro static, em uma nova chamada, ela tem o seu valor inicial exibido (em uma nova chamada, responde de forma dinâmica)

OBS: chamadas à variáveis estaticas são diferentes de não estáticas, exemplo rápido de chamada correta para o caso acima:

Main.a = 20;

Main m = new Main();

m.b = 10;

Ou, para acessar valores não estáticos, pode-se usar *extends*. Estudar conteúdo de diagrama de sequência, caso necessário

## Outputs.

A função println já coloca uma quebra de linha no final de cada execução

Exemplo:

System.out.println(conteudo);

System.out.println(conteudo);

Imprime:

"conteudo"

"conteudo"

A função print apenas imprime o valor na tela, sem inserir quebras de linha

Exemplo:

System.out.print(conteudo);

System.out.print(conteudo);

Imprime:

"conteudo""conteudo"

"Escaped values" são funções de strings, valores reservados que executam determinadas funções para texto. A não tradução é proposital, certas coisas, se for traduzir, dificulta na hora de realizar buscas no google.

O escaped value \n realiza uma quebra de linha, ou seja, o valor será exibido uma linha a mais para baixo

O escaped value \t realiza uma tabulação, ou seja, o valor será exibido com a distância de um tab

Escaped values podem ser utilizados à vontade e a grande maioria das funções java conseguem interpreta-los

Exemplo:

System.out.println("\t\t\t\t"+conteudo);

Imprime:

"conteudo"

System.out.println("\t\t\t"+conteudo);

Imprime:

"conteudo"

System.out.println("\t\t"+conteudo);

Imprime:

"conteudo"

System.out.println("\t"+conteudo);

Imprime:

"conteudo"

A letra “f” no final da palavra "print" significa "formatted". Os especificadores de formato são como marcadores de lugares para um valor, especificando o tipo da saída dos dados que iniciam com um sinal de porcentagem (%) seguido por um caractere representando seu tipo de dado. Os valores a serem imprimidos seguem a ordem de input no "printf".

Exemplo:

System.out.printf("%s\n%s\n", arg1, arg2);

Imprime:

"arg1"

"arg2"

System.out.printf("%s\n%s\n", arg2, arg1);

Imprime:

"arg2"

"arg1"

%d representa números inteiros

%f representa números floats

%2f representa números doubles

%b representa valores booleanos

%c representa valores char

%s representa valores string

## Variáveis de ambiente e de sistema.

Capturar os elementos, caminhos, variáveis de ambiente, variáveis do sistema, entre outros, pode poupar muito trabalho na construção de um software visando portabilidade, ou seja, um projeto criado na máquina A que utiliza variáveis de captura de ambiente dinâmicos, pode ser executado na máquina B com o mínimo de retrabalho, pois o mesmo já irá capturar automaticamente as informações do sistema e diretórios necessários para executar as suas determinadas funções.

Nomes de classes são considerados à partir da pasta java executável do projeto, seja test ou main, ou seja, o valor de caminho da classe só pode ser DENTRO da pasta java executável, a pasta Java é definida nos módulos do projeto, é basicamente à partir de qual diretório a máquina irá processar arquivos, interpretando a linguagem Java.

OBS: Considerar como IDE o Intellij Community com os seguintes parâmetros na data de descrita deste documento

Versão: 2019.1.3

Build: 191.7479.19

Data de disponibilidade: May 27, 2019.

Para acessar os módulos, clicar com o botão direito na pasta raiz do projeto e acessar o menu "Open Module Settings" ou pressionar F4 e então, acessar a aba "*Sources*".

Exemplo de retorno do nome da classe sendo executada:

System.out.println(Ambiente.class.getName());

OBS: caso o método não fosse estático, a chamada de classe seria this.getClass().getName()

Exemplo de retorno do nome e o caminho da classe sendo executada:

System.out.println(Ambiente.class.getSimpleName());

OBS: caso o método não fosse estático, a chamada de classe seria this.getClass().getSimpleName()

Exemplo de variáveis do sistema:

Caminho raiz do projeto: System.getProperty("user.dir");

Sistema operacional: System.getProperty("os.name");

Arquitetura do OS: System.getProperty("os.arch");

Versão do OS: System.getProperty("os.version");

Versão do Java: System.getProperty("java.version");

Usuário de execução: System.getProperty("user.name”);

OBS: Existem outras variáveis de sistema, listei as que considero mais importantes

## Diretórios.

A função mkdir() irá criar/deletar apenas diretórios ou arquivos singulares, ou seja, em caso de subpastas ou, caso alguma pasta do caminho do arquivo/pasta almejado não existe, e esse não seja o último, o mkdir() será ignorado.

Para múltiplos diretórios, é necessário informar ao Java que ele deve criar todas as subpastas até o arquivo/pasta almejado, para isso usamos a função mkdirs(). Para remover um diretório, incluindo o conteúdo dentro dele, seja pasta ou arquivo, deve-se usar a função FileUtils.deleteDirectory("caminho");

Os casos de .delete(), .mkdir() e .mkdirs() retornam boleano, ou seja, a maneira mais eficiente de se utilizar estes, seria em um if ou em um return.

## Arquivos.

Existem muitas formas de escrever arquivos em java, eu separei a que considero mais eficiente, que está hoje sendo utilizada na automação da fábrica.

O método BufferedWriter já cria o arquivo, se o mesmo não existir e consegue adicionar valores ao arquivo, caso seja necessário.

Exemplo de BufferedWritter (Escrita de arquivo):

try (Writer writer = new BufferedWriter(new OutputStreamWriter(

new FileOutputStream(arquivo, false), StandardCharsets.UTF\_8))) {

writer.write("Conteúdo");

} catch (IOException e) {

e.printStackTrace();

}

Exemplo de BufferedReader (Leitura de arquivo):

BufferedReader br = new BufferedReader(new FileReader(arquivo));

String st;

while ((st = br.readLine()) != null) {

System.out.println(st);

}

br.close();

OBS: A função br.readLine() faz com que o java procure a próxima linha do arquivo, portanto, o valor da linha considerado em br.readLine() dentro do if, validando que a linha possui algum conteúdo (não é nulo/ !null) é necessário ser transmitido para uma string, para então ser lido e exibido caso contrário, ao dar System.out.println(br.readLine()); a próxima linha seria considerada e como o nosso log possui apenas uma linha o valor exibido seria: null

Exemplo de FileUtils.copyFile (Cópia de arquivos):

File pasta = new File(System.getProperty("user.dir")+"/src/test/resources/copia");

OBS: Para copiar o arquivo, é necessário ter a sua pasta destino criada

if (pasta.mkdir()){

System.out.println("Pasta criada!");

}

FileUtils.copyFile(arquivo, new File(pasta + "/arquivo\_copiado.log"));

OBS2: Ao realizar uma cópia, se já existir um arquivo com o mesmo nome e extensão no local almejado, o mesmo será substituído

OBS3: Não há a necessidade de instanciar objetos para utiliza-los, portanto, como o segundo parâmetro, estou instanciando o objeto file dentro do método de cópia.

## Conteúdo extra – If/ Else in line & validação de boleanos.

O if/else in line, é uma maneira diferenciada, porém, muito útil de escrever ifs e elses de forma mais compacta.

Não encontrei artigos sobre/não acredito que isso impacte no processamento, pois a tomada de decisão é a mesma e a quantidade de linhas, tirando casos de centenas, se torna dispensável para o processamento da máquina.

Para realizar o if/else in line, escrevemos a condição e em seguida, um (?) para indicar o resultado

Para cada (?) é necessário ter um (:) subsequente, sendo o (:) a condição de saída de erro (else)

Traduzindo ao pé da letra:

CONDIÇÃO ? (THEN/ENTÃO) RESULTADO : (ELSE/SE NÃO) RESULTADO;

CONDIÇÃO ? (THEN/ENTÃO) RESULTADO : (ELSE/SE NÃO) CONDIÇÃO ? (THEN/ENTÃO) RESULTADO :(ELSE/SE NÃO) RESULTADO;

Ao realizar um if, podemos negar o valor de qualquer condição ao inserir um (!) antes da mesma, contanto que a condição seja em boleanos.

Exemplo:

String a = "";

String b = "";

if (a.equals(b))

Verifica: se a é igual a b

if(!a.equals(b))

Verifica: se a é diferente de b

Portanto, para realizar validações com boleanos, podemos apenas inserir a variável, que retornará true ou false

Exemplo:

boolean a = false;

if(a)

Verifica: se a é true

if(!a)

Verifica: se a é false

OBS: Nem todas as condições permitem o uso do (!), normalmente condições com operadores lógicos não retornam boleanos

Exemplo:

==, >=, <=, >, <, !=

OBS: Para comparação com operadores lógicos, em negação, se usa (!=)