**Declarações**

*def name = “Groovy”*Define uma variável dinamicamente tipada

*int age = 10*  
Define uma variável estaticamente tipada

*from = “codehaus”*  
Define uma variável dinamicamente tipada

**Imprimir**

*println “My name is”*  
Imprime um texto na tela

*println “My name is ${name}”*  
Imprime um texto na tela e uma variável, independente do tipo dessa variável

**Closures**

São blocos de códigos quem não precisam estar associados a um método e também não precisam estar contidos dentro de uma classe.

Declaração de Closures

*def square = {number -> number \* number}*

* Toda closures precisar ser declarada dentro de chaves
* Começar por um parâmetro
* Após os parâmetros deve conter o separador **->**

E tudo que vem após o separador é o código que deve ser executado

* Todo código que estiver após o = estará associado a variável square

Imprimir Closure

*println “Groovy way to execute a code block”  
println square(2)*

* Armazena o resultado da closure dentro de uma variável para ser utilizado em algo:  
  ***def*** *variavel =* *square(2)*
* Exibe o resultado da closure na tela, porém não é armazenada:   
  *println square(2)*

**Collections**

ARRAY *names = ["Paris", "New York", "Tokio", "Las Vegas", "Rio de Janeiro", "London"]*  
Coleção do tipo array, onde cada elemento é uma string  
  
SORT

*names = ["Paris", "New York", "Tokio", "Las Vegas", "Rio de Janeiro", "London"].sort()*No momento da impressão sistema irá exibir em ordem alfabética

EACH e IT *names.each{   
 println “The city is ${it}”  
}*

* Coleção each é parecido com for
* No elemento *it* fica implícito o que estou chamando, ou seja, se não existir um elemento criado para substituir o *it* por padrão o elemento será sempre *it*, então não é obrigatório passar ele como parâmetro, apenas informá-lo no println
* Passa-se a criar outros elementos quando for usar um each dentro de outro ou quando tiver alguma situação que precise retornar mais de um *it*, por questão de boas práticas nomeamos o elemento, como pode ser visto abaixo:

*names.****each*** *{  
 element ->  
 println “The city is ${element}”  
 state.each{  
 println “The State is ${it}”  
 }  
}*

* element irá receber a informação da lista e irá ser passado como parâmetro no momento da impressão, depois irá para o próximo da lista

INTERVALO DE POSIÇÕES

*(0..10).each{  
 println “The city is ${it}”*

* Podemos definir um array com posições de intervalo para impressão

***def*** *numbers = [\*4..16].reverse()*

* Aqui podemos ver um array criado a partir da posição 4 até a posição 16
* A funcionalidade reverse() irá ordenar de forma inversa esse array começando pela ultima posição definida

INCREMENTANDO ARRAY

*def numbers = [\*4..16].reverse()  
numbers << 14*

* << Adiciona algo no final do Array

***def*** *numbers = [\*4..16].reverse()  
numbers << 14  
numbers.pop()*

* .pop() remove a ultima posição inserida no array

MAP

*technologies = [name: 'Groovy', sentiment: 'Cool!']  
println technologies*

* Map é uma coleção estilo lista, porém é representado por uma chave e outros valores associados a essa chave
* No exemplo acima definimos uma MAP chamada *technologies* que contém a chave *name* e o valor *Groovy* e outra chave *sentiment* de valor *Cool*

Formas de imprimir valores de um MAP:

*println "technologies.name = ${technologies.name}"  
println "technologies['name'] = ${technologies['name']}"  
println "technologies.getAt('name') = " + technologies.getAt('name')  
println "technologies.get('name') = " + technologies.get('name')*

**Input and Output**

IMPORTANDO ARQUIVO TXT

*new File("C:\\Users\\Dev07\\Desktop\\Groovy Thalison\\arquivos\\groovy.txt").eachLine{  
 line ->  
 if(line = ‘Thalison’)  
 println "Found a good language at this line: ${line}"  
 println(line)  
}*

* eachLine - Pega o arquivo texto e percorre linha a linha
* O método eachLine toma uma Closure como parâmetro e para cada linha a mesma será evocada
* new File – cria um objeto da classe File
* if – Condição
* line – Variável onde será armazenada os valores das linhas de forma temporária

COMO NAVEGAR DENTRO DE UM DIRETÓRIO ESPECÍFICO

*dirPath = "C:\\Users\\Dev07\\Desktop\\Groovy Thalison\\arquivos"  
depth = 0  
println "\nListing directory contents of ${dirPath}:"  
new File(dirPath).eachFile{  
 file ->  
 if(! file.file) {  
 listDirectory(file, depth)  
 }* ***else*** *{  
 println "File: ${file.name}"  
 }  
}*

* dirPath – Variável que irá receber o diretório onde queremos navegar
* eachFile – Irá percorrer toda estrutura do diretório, esse método também toma uma Closure como parâmetro e para cada diretório percorrido será evocada
* if e else – Condição

CRIANDO ARQUIVO TXT

*new File ("C:\\Users\\Dev07\\Desktop\\Groovy Thalison\\arquivos\\testando.txt") << """\  
 Groovy is an amazing language!  
 Let's code together!  
 Are you ready?  
"""  
println "\nListing directory contents from system command line process:"*

EXECUÇÃO DE PROCESSO POR MEIO DO CMD

*Process p = “cmd /C dir”.execute()  
println '${p.text}'*

* execute() – Irá executar o comando que foi passado
* Process p – Cria um processo chamado P
* “${p.text} – Irá exibir o resultado do comando executado

*p = "cmd /C dir".execute()  
p.****in****.eachLine {  
 line ->  
 println ">>> ${line}"  
}*

* eachLine – Percorre linha a linha na saída gerada, também toma uma Closure como parâmetro
* in – Input

**XML Handling**

*import groovy.xml.MarkupBuilder*

* *Aqui é importado uma classe do Groovy que chama-se MarkupBuilder*
* *Com essa classe podemos gerar em código XML*
* *O MarkupBuilder nos permite fazer uma sequência de chamadas de alguns métodos, onde método representa o nó e esse método recebe uma MAP como parâmetro*
* *Cada MAP representa os atributos do nó em questão*

*def out = new File('C:\\Users\\Dev07\\Desktop\\Groovy Thalison\\arquivos\\devmediapublications.xml')*

* *Define uma variável out que armazena o arquivo que representa o arquivo xml*

*def writer = new FileWriter(out)*

* *Define uma variável*
* *Basicamente depois que é criado o arquivo, ele será associado a uma instancia de classe FileWriter que nos permite gravar o conteúdo gerado pelo nosso código dentro do arquivo xml*

*def xml = new MarkupBuilder(writer)*

* *Uma vez que o arquivo e o FileWriter estão instanciados ao arquivo, writer é associado ao MarkupBuilder passando como parâmetro*

*xml.publications(){  
 publication(title:'Introduction to JavaServer Faces', year:2006){  
 author(){  
 author('Rogerio Pereira Araujo')  
 }  
 }  
 publication(title:'JavaServer Faces Validators', year:2007){  
 authors(){  
 author('Rogerio Pereira Araujo')  
 }  
 }  
 publication(title:'Introduction to Groovy', year:2007){  
 authors(){  
 author('Rogerio Pereira Araujo')  
 }  
 }  
}*

* *xml.publications representa o MarkupBuilder*
* *O MarkupBuilder nos permite que seja possível fazer a chamadas a vários métodos*
* *Cada método pode receber uma MPA e uma Closure como parâmetro*
* *O método principal seria publications que representa o nó principal*
* *Dentro desse método estamos passsando uma Closure que representa mais 3 métodos que são os publication*
* *E cada um desses 3 métodos possui dois atributos que são title e year, representado pela entrada depois dos :*
* *Cada um desses métodos também possui uma Closure como parâmetro*
* *Essa Closure chama outro método chamada authors*
* *Esse método não possui atributos, porém possui nós filhos que seria o nó author que possui um único argumento*

Leitura de um XML

*def xml = new XmlSlurper().parse(new File('C:\\Users\\Dev07\\Desktop\\Groovy Thalison\\arquivos\\devmediapublications.xml'))*

* Define uma variável chamada XML
* Vai receber a instancia de um objeto chamada Gpath que é gerado a partir do momento que informamos new XmlSluper()
* .parse é um método que recebe um parâmetro, que no caso seria o arquivo devmediapublication.xml que contém o código XML
* Uma vez que esse arquiv o é aberto e é instanciado pelo XmlSlurper, ou seja, quando finaliza o processo de .parse e é constatado que o xml está bem formado, está correto, esse método .parse retorna uma instancia do Gpath que será associadaa variável XML
* Feito isso, podemos acessar todos os nós dentro do conteúdo xml através de expressões Gpath

*def publications = xml.publication*

* Esse é um método de como acessar um conteúdo xml através de expressões Gpath
* Aqui definimos uma variável chamada publications
* Essa variável está recebendo uma lista de nós publication que está dentro do coteúdo XML
* Uma vez que esses nós são armazenados na variável publications, então essa variável se torna uma coleção de nós publications

*publications.each{*

*println "Publication: ${it.@title}"  
println "Year: ${it.@year}"  
 it.authos.each{  
 println " Author: ${it.text()}"  
 }  
 println""  
}*

Utilizando Switch Case

*def calculaImc = { p, a ->  
 return (p / a \* a)  
}*

*def peso = 103.8  
def altura = 1.81  
def imc = calculaImc(peso, altura)*

*switch (imc){  
 case {it < 18.5}:  
 imprimir "Abaixo do peso ideal"  
 break  
 case {it >= 18.5 || it <= 25}:  
 imprimir "Peso normal"  
 break  
 case {it >= 25.1 || it <= 30}:  
 imprimir "Excesso de peso"  
 break  
 case {it >= 30.1 || it <= 35}:  
 imprimir "Obesidade (grau I)"  
 break  
 case {it >= 35.1 || it <= 40}:  
 imprimir "Obesidade (grau II)"  
 break  
 case {it > 40}:  
 imprimir "Obesidade (grau III)"  
 break  
}*

A função e a forma de passar os parâmetros também poderiam ser utilizadas da seguinte forma:

*def calculaImc = { p, a ->  
 return (p / (a \* a))  
}  
def imc = calculaImc(80, 1.80)*

Na função acima *p* de peso e *a* de altura foram passados como parâmetros dentro da função *calculaImc*, na linha 4 então foi criado a variável *imc* para receber o return na chamada da função *calculaImc* na qual contém os parâmetros *(80, 1.80)* entre parênteses.

Utilizando IF

*meses = ["Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho", "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro"]*

*def contador = 1*

*meses.each{ mesnome->  
 imprimir ""  
 if(contador == 1 || contador == 2){  
 if(mesnome == 'Janeiro' && contador == 1){  
 imprimir "Mês: ${contador} = ${mesnome} - Verão"  
 }  
 if(mesnome == 'Fevereiro' && contador == 2)  
 imprimir "Mês: ${contador} = ${mesnome} - Verão"  
 }  
 if(contador == 3 || contador == 4 || contador == 5 || contador == 6){  
 if(mesnome == 'Março' && contador == 3){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Outono"  
 }  
 if(mesnome == 'Abril' && contador == 4){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Outono"  
 }  
 if(mesnome == 'Maio' && contador == 5){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Outono"  
 }   
 if(mesnome == 'Junho' && contador == 6){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Outono"  
 }   
 }   
 if(contador == 7 || contador == 8 || contador == 9){   
 if(mesnome == 'Julho' && contador == 7){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Inverno"  
 }  
 if(mesnome == 'Agosto' && contador == 8){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Inverno"  
 }  
 if(mesnome == 'Setembro' && contador == 9){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Inverno"  
 }  
 }  
 if(contador == 10 || contador == 11 || contador == 12){   
 if(mesnome == 'Outubro' && contador == 10){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Inverno"  
 }  
 if(mesnome == 'Novembro' && contador == 11){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Inverno"  
 }  
 if(mesnome == 'Dezembro' && contador == 12){  
 imprimir "Mês: ${contador} = ${mesnome} - Estação: Inverno"  
 }  
 }  
contador = contador + 1  
}*

Utilizando Funções

*def provaRealFuncao = { nome, i, p, a ->  
 def imcTeste = (p / (a\*a))  
 def numFormatado = Numeros.trunca(imcTeste, 1)  
 return "o ${nome.toUpperCase()} que possui a idade $i tem o imc: $numFormatado"  
}  
def aux = provaRealFuncao("Thalisson", 26, 60, 1.80)  
imprimir aux*

* Aqui podemos notar que não é necessário colocar a variável entre chaves ${} para imprimir a mesma, também pode-se utilizar apenas o $
* Esse método entre chaves ${} é mais utilizando quando é necessário passar algum tratamento como foi o caso *${nome.toUpperCase()}* que deixa a variável toda maiúscula

*def tabuada = { valorTabuada ->  
 def res = []  
 (1..10).each{ indice ->  
 x = (valorTabuada \* indice)  
 res << ("$valorTabuada x $indice = " + x)  
 }  
 res.each{  
 imprimir it  
 }  
}  
tabuada(8)*

* Nessa função podemos observar o uso de listas
* Inicialmente declaramos uma lista vazia
* Posteriormente definimos um intervalo utilizando each que tem como parâmetro índice
* Dentro desse each definimos a variável x que ira receber o resultado do calculo entre valorTabuada \* índice
* Em seguida guardamos esse resultado na lista vazia que criamos no inicio
* Por fim acionamos o res.each onde irá imprimir toda a lista, do início ao fim

*def tabuada = { valorTabuada ->  
 def res = []  
 (1..10).each{ indice ->  
 res << ("$valorTabuada x $indice = " + (valorTabuada \* indice))  
 }  
 res.each{  
 imprimir it  
 }  
}  
tabuada(53)*

* Nessa função a diferença é que não foi utilizada uma variável X para armazenar o valor do cálculo, esse foi apresentado direto no imprimir