Apostila de Exercícios Groovy – F1 e F2

Desenvolvedor Groovy

Exercício 1

```
package aula

class Exercicio1 {
  static void main (args) {
    // Exer1
    println "Ola mundo groovy"

    // Exer2
    // interpolação de string:
    // cancatena automaticamente - $variavel ou $objeto.atributo
    String nome = "Fernando"
    int idade = 36

    String frase = "O $nome tem $idade anos."
    println frase
  }
}
```

Exercício 1.2

```
class Exercicios {

  @Test
  void exercicio1ponto2() {
    int a = 10
    println a.class // tem atributos e métodos.
    println a.toString()
    println 12l.class.name

    // muda 2 coisas do java:
    // 1 - coloca "g" no literal para virar BigInteger.
    println 11g.class.name

    // 2 - literal flutuante é considerado BigDecimal por padrão.
    BigDecimal valor = 200.50
    println valor
  }
```

Exercício 2

```
package classes

class Cliente {
  String nome
  Date data
  Integer somar(Integer v1, Integer v2) {
    v1 + v2
  }
}

  @Test
  void exercicio2() {
    Cliente c = new Cliente()
    c.setNome "Fer" // sem parenteses.
    c.setData new Date()
    println c.somar (10, 10)
    println c.getNome() + " "+ c.getData()
  }
```

Exercício 3

```
@Test
void exercicio3() {
  // veja que não fizemos construtor na classe Cliente
  Cliente c = new Cliente()
  println c.getNome() + " - " + c.getData()

  c = new Cliente(nome: "fernando")
  println c.getNome() + " - " + c.getData()

  c = new Cliente(data: new Date())
  println c.getNome() + " - " + c.getData()

  c = new Cliente(nome: "fernando", data: new Date())
  println c.getNome() + " - " + c.getData()
}
```

Exercício 4

```
@Test
void exercicio4() {
  Cliente c = new Cliente(nome: "fernando", data: new Date())
  println c["nome"]
  c["nome"] = "marcão"
  println c["nome"]
}
```

Exercício 5

```
package classes;

// 3 motivos para fazer essa classe em java:
// 1. Gerar uma classe em java e ver se eles gostam de voltar a digitar o que o  groovy faz.
// 2. Provar que funciona o mix de java e grovvy.
// 3. Usar o recurso de Direct field access operator.
public class Produto {

  private String nome;
  private double valor;

  public Produto(String nome, double valor) {
    this.nome = nome;
    this.valor = valor;
  }
  public String getNome() {
    System.out.println("getNome");
    return nome;
  }
  public void setNome(String nome) {
    System.out.println("setNome");
    this.nome = nome;
  }
  public double getValor() {
    System.out.println("getValor");
    return valor;
  }
  public void setValor(double valor) {
    System.out.println("setValor");
    this.valor = valor;
  }
}
```

```
@Test
void exercicio5() {
  Produto p = new Produto("CD", 12.00)
  println p.nome + "-" + p.valor
  p.nome = "CD Calcinha Preta"
  p.valor = 15.00
  println p.nome + "-" + p.valor
}
```

Exercício 6

```
 @Test
 void exercicio6() {
   // import classes.Cliente as Xu
   Xu c = new Xu(nome: "fernando", data: new Date())
   println  c.getNome() + " - " + c.getData()

   //import static javax.swing.JFrame.EXIT_ON_CLOSE as ex
   println ex
 }
```

Exercício 7

```
package classes

class Venda {
   double vender(double valor, int taxa = 10) {
      double rs = valor * taxa / 100
      rs // só para lembrar do return como ultima linha.
   }
}
```

```
 @Test
 void exercicio7() {
   Venda v = new Venda()
   println  v.vender(100)
   println  v.vender(100, 15)
 }
```

Exercício 8

```
package classes

class Somar {
  double somar(double[] valores) {
    double rs = 0;
    for (double v: valores) {
      rs += v;
    }
    rs
  }
}

 @Test
 void exercicio8() {
    Somar soma = new Somar()
    println soma.somar(10)
    println soma.somar(10, 10)
    println soma.somar(10, 10, 10)
    // depurar dentro da classe somar.
 }
```

Exercício 9

```
@Test
void exercicio9() {
   Cliente c = null
   c?.nome = "Fernando"
   println c?.getNome()
   // em java geraria null point exception

   Cliente c2 = new Cliente()
   c2?.nome = "Fernando"
   println c2?.getNome()
}
```

Exercício 10

```
@Test
void exercicio10() {
  List<String> colecao = new ArrayList<>()
  colecao.add("fernando")
  colecao.add(null)
  colecao.add("anny")
  colecao.add("rita")
  println colecao
  colecao = colecao*.toUpperCase()
  println colecao
  colecao = colecao*.replace("A", "@")
  println colecao
}
```

Exercício 11

```
@Test
void exercicio11() {
  // Construtor lança 2 exception, veja javadoc.
  URL url = new URL("http://www.google.com.br")
  println url
  // depois faça dar errado para ver o lancamento da exception
  //URL errado = new URL("hbla blalbal")
}
```

Exercício 12

```
@Test
void exercicio12() {
  String nome = null
  if (nome) {
    println true
  } else{
    println false
  }
  nome = "Fer"
  if (nome) {
    println true
  } else{
    println false
  }
  int valor = 0
  if (valor) {
    println true
  } else{
    println false
  }
  valor = 1
  if (valor) {
    println true
  } else{
    println false
  }
  List<String> colecao = new ArrayList(0)
  if (colecao) {
    println true
  } else{
    println false
  }
  colecao.add("fer")
  if (colecao) {
    println true
  } else{
    println false
  }
  // nenhum desses funcionam em java.
}
```

Exercício 13

```
package classes

class Nota {
  Integer itens
  Double valor

  Nota plus(Nota nota) {
    println "plus"
    Nota novo = new Nota();
    novo.itens = this.itens + nota.itens
    novo.valor = this.valor + nota.valor
    novo
  }

  Nota next() {
    println "next"
    this.itens += 1
    this.valor *= 2
    this
  }
}
```

```
@Test
void exercicio13() {
  Nota n1 = new Nota(itens: 2, valor: 20)
  Nota n2 = new Nota(itens: 2, valor: 20)
  Nota n3 = n1+ n2
  println n3.valor + " - "+ n3.itens
  n1++
  println n1.valor + " - "+ n1.itens
}
```

Exercício 13.2

```
@Test
void exercicio13ponto2() {
  BigDecimal vl = 10
  println vl
  vl = vl + 1
  println vl
  vl++
  println vl

  int a = 2
  vl = vl + a;
  println vl
  println vl - 5

  Date data = new Date()
  println  data
  data++
  println data
  data++
  println data + 10
  // consulte o documentação do groovy para saber detalhes das sobrecargas.
  // se caso não existir, vc pode implementar.
}
```

Exercício 13.3

```
package classes

trait Animal {
  String nome
  abstract void pular()
  void falar() {
    println "Animal "+ nome + " falando.."
  }
}
```

```
package classes

trait Lutador {
  String arma
  abstract void correr()
  void lutar() {
    println "Lutador "+ arma + " luitando..."
  }
}
```

```groovy
package classes

// O plugin de eclipse de groovy não sabe tratar trait, ele vai pensar que é uma interface
// se vc mandar o eclipse implementar ele vai gerar todas as implementações....
class Pessoa implements Animal, Lutador {
  @Override
  void pular() {
    println "pessoa " + nome + " pulando"
  }

  @Override
  void correr() {
    println "pessoa " + nome + " correndo"
  }
}
```

Exercício 14

```groovy
package classes

import groovy.transform.ToString

@ToString // Exemplo 1
//@ToString(excludes=["idade"])  // Exemplo 2
//@ToString(includeNames=true, excludes="idade, salario") // Exemplo 3
class Funcionario {
  String nome
  Integer idade
  Double salario
}

 @Test
 void exercicio14() {
   Funcionario f = new Funcionario(nome: "Fer", idade: 10, salario: 1500.59)
   println f
 }
```

Exercício 15

```
package classes

import groovy.transform.EqualsAndHashCode

@EqualsAndHashCode
class Funcionario2 {
  String nome
  Integer idade
  Double salario
}

 @Test
 void exercicio15() {
   Funcionario2 f1 = new Funcionario2(nome: "Fer", idade: 10, salario: 10)
   Funcionario2 f2 = new Funcionario2(nome: "Fer", idade: 10, salario: 10)
   // 1) execute na 1 sem a anotação para dar false.
   println f1.equals(f2)

   // 2) Acrescente a notação e depois executa denovo para dar true.
   println f1.equals(f2)
 }
```

Exercício 16

```
package classes

import groovy.transform.Immutable

@Immutable
class Pedido {
  String cliente
  Integer numero
}

 @Test
 void exercicio16() {
   //1) Parte
   Pedido p = new Pedido(cliente:"fernando", numero: 10)
   println p.cliente + "-" + p.numero
   println p

   //2) Parte - tenta alterar alguns atributo e vera erro de propriedade readOnly.
   //p.cliente = "fer"
 }
```

Exercício 17

```
package classes

@Singleton
class Conexao {
  Double valor
}
```

```
@Test
void exercicio17() {
  //1) Parte
  Conexao.instance.valor = 10
  println Conexao.instance.valor
  Conexao con =  Conexao.instance
  con.valor = 11
  println Conexao.instance.valor

  //2) Parte - tente criar uma instancia e vera o erro q não pode instanciar uma classe singleton.
  //Conexao x = new Conexao()
}
```

Exercício 18

```
package classes

import groovy.transform.builder.Builder

@Builder
class Comida {
  String fruta
  String bebida
  String doce
}
```

```
@Test
void exercicio18() {
  Comida comida = Comida.builder().fruta("maca").bebida("coca cola").doce("casadinho").build()
  println comida.fruta
  println comida.bebida
  println comida.doce
}
```

Exercício 19

```
@Test
void exercicio19() {
  def objeto = "texto"
  println objeto.getClass()

  objeto = 10
  println objeto.getClass()

  objeto = 10.00
  println objeto.getClass()

  objeto = new Nota(itens: 2, valor: 20)
  println objeto.getClass()
  println objeto.valor

  objeto = new Pedido(cliente:"fernando", numero: 10)
  println objeto.getClass()
  println objeto.cliente
}
```

Exercício 20

```
package classes

class Teste {
  def metodo(valor) {
    valor + 1
  }
}
```

```groovy
@Test
void exercicio20() {
  def teste = new Teste()
  def v1 = teste.metodo("fer")
  println v1.getClass()
  println v1

  v1 = teste.metodo(5)
  println v1.getClass()
  println v1

  v1 = teste.metodo(new BigDecimal(5))
  println v1.getClass()
  println v1

  def data = new Date()
  println data
  v1 = teste.metodo(data)
  println v1.getClass()
  println v1
}
```

Exercício 21

```
@Test
void exercicio21() {
  def colecao = new ArrayList<String>()
  colecao.add("fer")
  colecao.add("anny")
  // executando tipo string
  for (item in colecao) {
    println item
  }

  colecao = new ArrayList<Integer>()
  colecao.add(1)
  colecao.add(2)
  // executando tipo integer
  for (item in colecao) {
    println item
  }

  colecao = "fernando esta aqui parado"
  // executando tipo carater
  for (item in colecao) {
    println item
  }

  colecao = 10
  // executando tipo inteiro unico
  for (item in colecao) {
    println item
  }
}
```

Exercício 22

```
@Test
void exercicio22() {
  def metodo = {int v1, int v2-> v1 + v2}
  println metodo(1, 2)
  println metodo(2, 3)
}
```

Exercício 23

```
@Test
void exercicio23() {
  def imprimir = {String v->
    String temp = v.trim().replace("a", "@")
    temp = temp.toUpperCase()
    return temp // return é opcional
  }
  println imprimir("  fernando  ")
  println imprimir("marta")
}
```

Exercício 24

```
@Test
void exercicio24() {
  def funcao = {v1, v2-> v1 + v2}
  def  r1 = funcao(5, 5)
  println  r1.class
  println r1
  def  r2 = funcao("fer", "nando")
  println r2.class
  println r2
  def  r3 = funcao(10.50, 5.50)
  println r3.class
  println r3
}
```

Exercício 25

```
package classes

class Relatorio {
  void emitir(cliente, cabecalho) {
    def rel = cabecalho(cliente)
    println "Relatório de vendas: " + rel + " ****"
  }
}
```

```
@Test
void exercicio25() {
  def limpeza = {texto-> texto.trim().replace("a", "@").replace(" ", "").capitalize() }
  def rel = new Relatorio()
  rel.emitir("   f e r n a n d o  ", limpeza)
  rel.emitir("   J a n a  ", limpeza)
}
```

Exercício 26

```
@Test
void exercicio26() {
  def rel = new Relatorio()
  rel.emitir("FERNANDO") { v-> v +" DA SILVA" }
  rel.emitir("FERNANDO") { a-> a.replace("N", "#") }
}
```

Exercício 27

```
@Test
void exercicio27() {
  def rel = new Relatorio()
  rel.emitir("FERNANDO") { it +" DA SILVA" }
  rel.emitir("FERNANDO") { it.reverse() }
}
```

Exercício 28

```
package classes

public interface Cantor {
  void cantar()
}
```

```
package classes

class Palco {
  void show(Cantor c) {
    c.cantar()
  }
}
```

```groovy
@Test
// implementação de interface funcionais - 1 método.
void exercicio28() {
  Palco palco = new Palco()
  Cantor cantor = null

  def imp = {println "vou cantar"}
  cantor = imp
  cantor.cantar()
  palco.show(cantor)

  cantor = {println "agora vou chorar"}
  cantor.cantar()

  palco.show({println "clousure cantanto como se fosse cantor"})
}
```

Exercício 29

```groovy
package classes

public interface Torcida {
  void pular();
  void gritar(String texto)
}
```

```groovy
@Test
// implementação de interface funcionais - vários metodos.
void exercicio29() {
  Torcida t = null
  def corintiano = [
    pular: {println "curintia pulando"},
    gritar: {p -> println "curintia - "+ p}
  ] as Torcida
  // quando uma interface com varios metodos, use o operado "as"
  //para converter a clousure na interface polimorfica

  t = corintiano
  t.pular()
  t.gritar("vai ae")

  def porcada = [
    pular: {println "porco eooo"},
    gritar: {p -> println "poorrcooooooooooo - "+ p}
  ] as Torcida

  t = porcada
  t.pular()
  t.gritar("verdão")
}
```

Exercício 30

```groovy
package classes

import java.awt.FlowLayout

import javax.swing.JButton
import javax.swing.JFrame
import javax.swing.JOptionPane

class Tela extends JFrame {
  public Tela() {
    setTitle("Tela")
    setSize(200, 200)
    def botao = new JButton("Clique Aqui")
    getContentPane().setLayout(new FlowLayout())
    add(botao)
    // preenchendo uma interface via clousure = pattern strategy
    botao.addActionListener({e-> JOptionPane.showMessageDialog(null, "Foi via closure")})
  }
}
```

```
//exercicio30
static main(arg) {
  def tela = new Tela()
  tela.setVisible(true)
}
```

Exercício 31.1

```
package classes

class Fatura {
}

@Test
void exercicio31ponto1() {
  def fat = new Fatura()
  // 1) tente executar a 1 vez, não vai dar pq não existe.
  //fat.vender(10.00)

  // 2) Adicionando método dinâmico no objeto.
  fat.metaClass.vender = {valor -> println " venda no valor="+ valor}

  fat.vender(10.00)
  fat.vender(1052.98)
}
```

Exercício 31.2

```
@Test
void exercicio31ponto2() {
  def f1 = new Fatura()
  // 1) tente executar a 1 vez, não vai dar pq não existe.
  //f1.faturar(10)

  // 2) Adicionando método dinâmico na classe.
  Fatura.metaClass.faturar = {valor -> println " faturar no valor="+ valor}

  def f2 = new Fatura()
  f2.faturar(10)
  def f3 = new Fatura()
  f3.faturar(22)
}
```

Exercício 32.1

```
@Test
void exercicio32ponto1() {
  def fat = new Fatura()
  // 1) tente executar a 1 vez, não vai dar pq não existe.
  //fat.cliente = "Fernando"

  // 2) Adicionando atributo dinâmico no objeto.
  fat.metaClass.cliente = "Fernando"
  println fat.cliente
  fat.cliente = "outra pessoa"
  println fat.cliente
}
```

Exercício 32.2

```
@Test
void exercicio32ponto2() {
  def f1 = new Fatura()
  // 1) tente executar a 1 vez, não vai dar pq não existe.
  //f1.cliente = "teste"

  // 2) Adicionando atributo dinâmico na classe.
  Fatura.metaClass.cliente = ""

  def fat = new Fatura()
  fat.cliente = "Luana"
  println fat.cliente
}
```

Exercício 32.3

```
@Test
void exercicio32ponto3() {
  // metodo
  Fatura.metaClass.static.impressao = {println "metodo estatico ok"}
  Fatura.impressao()
  def f = new Fatura()
  f.impressao()
}
```

Exercício 32.4

```
package classes

class Viajar {
  void viajar(String destino, BigDecimal valor) {
    valor += 0.50 // taxa
    println "Viagem até $destino custa $valor"
  }
}
```

```
@Test
void exercicio32ponto4() {
  Viajar v = new Viajar()
  v.viajar("Curitiva", 10)
  def novoMetodo = {String lugar, BigDecimal valor ->
    valor += 3.50
    println "Novo preço $lugar será $valor"
  }
  v.metaClass.viajar = novoMetodo
  v.viajar("Curitiva", 10)
}
```

Exercício 32.5

```
@Test
void exercicioExpando32ponto5() {
  //1) Cria um objeto e adicione atributos e comportamentos.
  Expando cliente = new Expando()
  cliente.nome = "Fernando"
  cliente.idade = 37
  cliente.impressao = {println "nome $nome idade é $idade"}
  cliente.impressao()

  // 2) Crie um objeto e já no próprio construtor crie os campos
  Expando livro = new Expando(autor: "Jonas", paginas: 100)
  println livro.autor
  println livro.paginas
}
```

Exercício 33

```
@Test
void exercicio33() {
  def v1 = new BigDecimal("10.50")
  def v2 = 10.50
  println v2.getClass().name
  println v1 == v2

  def v3 = v1 + v2
  println v3.getClass()
  println v3

  v3 = v1 - v2
  println v3

  v3 = v1 * v2
  println v3
}
```

Exercício 34

```
@Test
void exercicio34() {
  // comparação
  def v1 = "Fernando "
  def v2 = "Fernando "
  println v1 == v2

  // sobrecarag de operador -
  def v3 = v1 - "nando"
  println v3

  // novos métodos e closures
  String texto = "fernando"
  println texto.capitalize()
  println texto.findIndexOf ({it == "o"})

  // multiline
  String textoGrande = '''
  Meu texto grande
  É muto grande
  e não precisa ficar usado + toda hora
  '''
  // string el, q evita dry "+ var +"
  def nome = "fernando"
  def idade = 35
  def salario = 1555.90

  def sql = "insert into cliente (nome, idade, salario) values(${nome}, ${idade}, ${salario})"
  println sql
  println textoGrande
}
```

Exercício 35

```
@Test
void exercicio35() {
  def data1 = new Date()
  // setar valores usando constantes do import static java.util.Calendar.*
  data1[YEAR] = 2010
  data1[MONTH] = 1 // 0 JANEIRO, 1 FEV
  data1[DATE] = 14
  println data1

  def data2 = new Date()
  data2[YEAR] = 2015
  data2[MONTH] = DECEMBER
  data2[DATE] = 1
  println data2

  // comparação entre datas com > sobrecarga
  if (data2 >= data1) {
    println "maior"
  }

  // operadores sobrecarregados
  Date data = new Date()
  println  data
  data +=  1
  println data
  data -= 2
  println data
  data++
  println data
  data--
  println data

  // formatação na mesma classe
  println data.format("dd/MM/yyyy hh:mm:ss")
}
```

Exercício 36

```
@Test
void exercicio36() {
  // numero são objetos e eles tem closures da GDK.
  10.times{ println it }
  1.upto(10) { println it }
  5.downto(1) {println  it }
}
```

Exercício 37

```
@Test
void exercicio37() {
  // criar e escrever
  def x = new FileWriter("D:/1.txt").withWriter{ e -> e.write("fernando franzini")}
  def a = new File("D:/2.txt")
  a.write("outro")

  def b = new File("D:/3.txt")
  b.text = "Linha 1"
  5.times{ b << "\r\nnova linha usando sobrecarga de operador" }

  // ler
  def c = new File("D:/3.txt")
  println c.text

  // lendo todas as linhas e usando spread para colocar tudo maisc.
  println c.readLines()*.toUpperCase()

  new File("D:/3.txt").eachLine{ linha-> println linha}

  // Esse não precisa passar na aula.
  a.delete()
  b.delete()
  c.delete()
  new File("D:/1.txt").delete()

  // lendo todos os diretorios
  new File("C:/").eachFile{ println it.name }
}
```

Exercício 38

```
static main(args) {
  Thread.start{10.times{ println "rodando na outra thread" }}
}
```

Exercício 39

- Montar o hsql e gerar os scripts

```
@Test
void exercicio39() {
  Sql con = Sql.newInstance("jdbc:hsqldb:file:D:/hsqldb/base;shutdown=true", "sa", "1234")
  con.eachRow("select * from cliente"){p -> println p.id + " - " + p.nome + "-" + p.email}

  def lista = con.rows("select * from cliente")
  lista.each{cli -> println cli.email}

  con.executeInsert("insert into cliente(nome,email)values('groovy xuxu','groovy@groovy.com')")
  println "pulando..."
  con.eachRow("select * from cliente"){p -> println p.nome}

  DataSet tabela = con.dataSet("cliente")
  tabela.add(nome: "g2", email:"g@g.com") // gerando o SQL para vc..(ORM)
  println "pulando..."
  con.eachRow("select * from cliente"){p -> println p.nome}
}
```

Exercício 40

```
@Test
void exercicio40() {
  //ArrayList<String> lista = new ArrayList()

  // Nova sintaxe otimizada []
  def lista1 = [1, 2, 3, 4]
  println lista1.getClass().name
  def lista2 = [
    "Fer",
    "Anny",
    "Lucas",
    "Cida"
  ]
  println lista2.getClass().name

  def lista3 = new ArrayList<BigDecimal>()
  lista3.add(1.50)

  // List tem operador sobrecarregado para adicionar
  lista3 << 10.50
  lista3 << 20.50

  // closures de conveniencias gerais

  // iteração rapida
  lista1.each {println it}
  lista2.each {println it}
  lista3.each {println it}

  // iteração com soma
  def total = 0
  lista3.each {total += it}
  println total

  // criação de um lista de clientes
  // Fazer o toString no Cliente
  def clientes = []
  clientes << new Cliente(nome: "Xico", data: new Date())
  clientes << new Cliente(nome: "Luana", data: new Date())
  clientes << new Cliente(nome: "Fernando", data: new Date())
  clientes << new Cliente(nome: "Anny", data: new Date())
  clientes << new Cliente(nome: "Luciano", data: new Date())
```

```
    // Facilidade para procurar um elemento.
    def achou = clientes.find {c -> c.nome.contains("Lu")}
    println achou

    // Facilidade para procurar varios elementos.
    def encontrandos = clientes.findAll {c -> c.nome.contains("Lu")}
    encontrandos.each {c -> println c}

    // Facilidade ordenar com regras customizadas
    clientes.sort {c1, c2 -> c1.nome.compareTo(c2.nome)}
    println clientes

    // conversão de list para set
    def set = clientes as Set
    println set.getClass().name
    set.each{c -> println c}

    // conversão de set para list
    def lista4 = set as List

    // lista e set imutaveis
    def listaImutavel = lista4.asImmutable()
    def setImmutavel = set.asImmutable()

    // lista e set sincronizadas
    def listaSyn = lista4.asSynchronized()
    def setSyn = set.asSynchronized()

    // transformações entre coleções
    def func = clientes.collect {c -> new Funcionario(nome: c.nome)}
    println func
}
```

Exercício 40.2

```groovy
@Test
void exercicio40ponto2() {
  def mapa = [:]
  println mapa.getClass().name

  // Adicionando um chave /valor
  mapa["pai"] = "Fernando"
  mapa["mae"] = "Anny"
  println mapa

  // Acessando uma chave
  println mapa["pai"]
  println mapa["filha"]

  // Removendo
  mapa.remove("pai")
  println mapa

  // Criação via mapa com construtor
  def pessoas = ["jonas": 10, "pedro":11, "rebeca": 12]
  println pessoas

  // iterando nos valores
  pessoas.values().each { println it}
  // iterando nas chaves
  pessoas.keySet().each { println it}
}
```

Exercício 41

```
@Test
void exercicio41() {
  def writer = new StringWriter()
  def html = new MarkupBuilder(writer)
  html.html {
    head {title "Minha Pagina" }
    body(id: "main") {
      h1 "Titulo 1"
      p "linha de texto"
      p { strong "outro texto"}
      a href: "pagina.html", "Clique aqui."
    }
  }
  println writer
}
```

Exercício 42

```
static main(args) {
  SwingBuilder bld = new SwingBuilder()
  JFrame tela = bld.frame(title: "Tela", size: [300, 120], defaultCloseOperation: JFrame.EXIT_ON_CLOSE,
  locationRelativeTo: null){
    panel( constraints: BorderLayout.CENTER) {
      label(text: "Nome:")
      textField(id: "nome", columns: 20)
      label(text: "E-mail:")
      textField(id: "email", text: "", columns: 20)
    }
    panel( constraints: BorderLayout.SOUTH) {
      button(text: "Gravar", actionPerformed: {
        JOptionPane.showMessageDialog(null,
            "gravou nome:" + nome.text +" - email:" + email.text)
        nome.text = ""
        email.text =""
      })
      button(text: "Fechar", actionPerformed: { System.exit(0)})
    }
  }
  tela.setVisible(true)
}
```

Exercício 43

```
import groovy.sql.Sql

// exercício 1
def valor = 10
def resultado = 0;
valor.times() {
  println it
  resultado += it
}
println resultado

// exercício 2
Sql con = Sql.newInstance("jdbc:hsqldb:file:D:/Java/hsqldb-2.3.4/bases/aula;shutdown=true", "sa", "1234")
con.eachRow("select * from cliente") { c-> println c.nome}
```