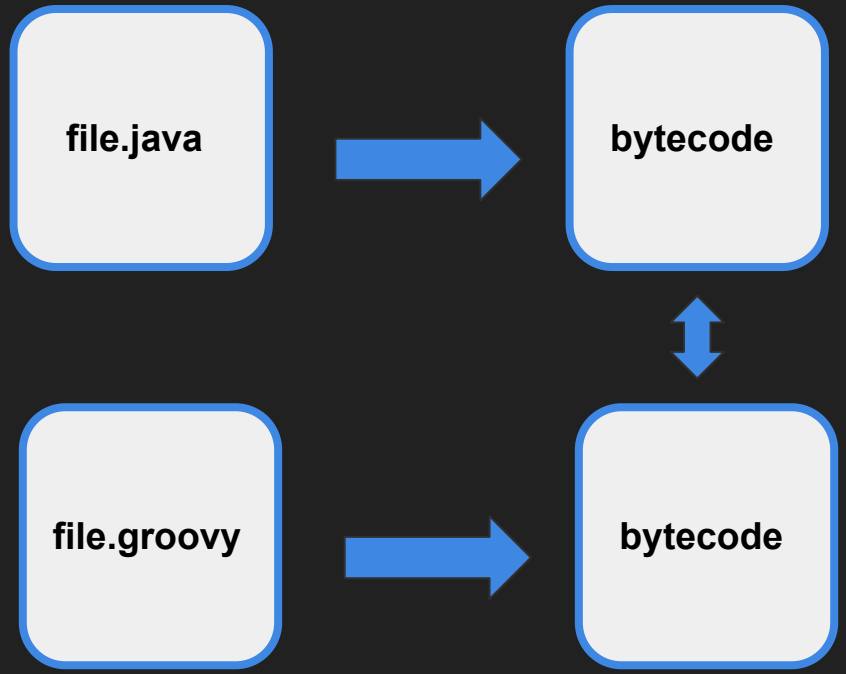
Groovy

Disciplina: Linguagem de Programação
Componentes: Gabriela Bonelli, Iagoh Lima e Jonas Fiorini

Introdução

- James Strachan foi o primeira a falar sobre o desenvolvimento do Groovy em 2003;
- Após a padronização do Java Community Process (JCP), foi lançada uma versão 1.0 em 2007;
- O Groovy é uma Linguagem Orientada a Objeto dinâmica desenvolvida com base em Java, mas com recursos semelhantes ao Python, Ruby, Perl e Smalltalk.

Tradução



Amarrações

- **Identificadores:**
  - Os identificadores começam com letras, cifrão ou um sublinhado. Eles não podem começar com números.
    - **Exemplos de identificadores Válidos:** def name, def tipo2, def $teste;
    - **Exemplos de identificadores Inválidos:** def 1errei, def 21continuo_errando;

Amarrações

- **Palavras reservadas:**

as	assert	break	case
catch	class	const	continue
def	default	do	else
enum	extends	false	finally
for	goto	if	implements
import	in	instanceof	interface
new	null	package	return
super	switch	this	throw
throws	trait	true	try
while			

Amarrações

- **O binding funciona da mesma forma como ocorre em Java:**

```
int a = 0
float b
String mantra = "Java is Groovy, Groovy is Java"

int[] vet = [1,2,3]

def umaClosure = { ->
    assert str == "Oi! Sou uma Closure!"
}
```

Amarrações

- **Escopo:**
  - Possui escopo estático;
  - Assim como em Java, só permite ocultamento entre atributo e variável local;

Amarrações

- **Escopo:**

```
class Teste {
    static x=10
    static imprime() {println("Não consigo imprimir ->" + y)}
    /*"...variable 'y' was found in static scope but doesn't refer to a local variable..."*/
    static void main(String[] args){
        println("Hello World!" + x)
        def y=1000
        imprime()
    }
}
```

Valores e tipos de Dados

- **Tipagem:**
  - Diferente de java, Groovy é considerada por alguns autores como sendo “Opcionalmente” tipada.

```
int i = 1
float j = 0.54
def x
```

  - Tipagem dinâmica:
    - Duck typing: “Comporta-se como um pato, então é um pato”

Valores e tipos de Dados

- **Tipos existentes em Groovy:**
  - String
  - Integral
  - Decimal
  - Booleano
  - Listas
  - Arrays
  - Maps

Valores e tipos de Dados

- **String:**
  - Groovy permite instanciar objetos java.lang.string;
  - Aceita concatenação de string pelo operador +;
  - Aceita interpolação pela inserção do placeholder ${};
  - Diferente de Java, Groovy não tem caractere explícito.

Valores e tipos de Dados

- **String - EXEMPLOS:**

```
// Equivalência da string utilizando o operador + para concatenar
assert 'ab' == 'a' + 'b'

def name = ‘Iagoh’

def frase = “Quem está apresentando é: ${name}”


// Formas de se declarar um caractere
/* 1ª Forma: apropriada para quando se quer manter o caractere em uma variavel*/
char c1 = 'A'
assert c1 instanceof Character
```

Valores e tipos de Dados

- **String - EXEMPLOS:**

```
/* 2ª Forma: apropriada para quando quer passar um valor de caractere como argumento de uma chamada de método.*/
def c2 = 'B' as char
assert c2 instanceof Character

/* 3ª Forma: apropriada para quando quer passar um valor de caractere como argumento de uma chamada de método.*/
def c3 = (char)'C'
assert c3 instanceof Character
```

Valores e tipos de Dados

- **Integral:**
  - Os tipos literais integrais são os mesmos que java: byte, char, short, int, long, java.lang.BigInteger;
  - Se utilizar a palavra chave *def,* o tipo da integral irá variar: vai adaptar-se à capacidade do tipo que pode conter esse valor;

Valores e tipos de Dados

- **Integral - EXEMPLOS:**

```
// Tipos primitivos
byte  b = 1
char  c = 2
short s = 3
int     i = 4
long   l = 5

// Precisao infinita
BigInteger bi =  6
```

Valores e tipos de Dados

- **Integral - EXEMPLOS (utilizando def):**

```
def a = 1
assert a instanceof Integer

// Integer.MAX_VALUE
def b = 2147483647
assert b instanceof Integer

// Integer.MAX_VALUE + 1
def c = 2147483648
assert c instanceof Long
```

Valores e tipos de Dados

- **Decimal:**
  - Os tipos decimais são os mesmos que java: float, double, *java.lang.BigDecimal*;
  - Os decimais podem utilizar expoentes. Basta acrescentar e ou E seguido de um sinal desejado e um número inteiro representando o expoente.

Valores e tipos de Dados

- **Decimal - EXEMPLOS:**

```
// Tipos Primitivos
float  f = 1.234
double d = 2.345

// Precisao infinita
BigDecimal bd =  3.456
```

Valores e tipos de Dados

- **Decimal - EXEMPLOS (Expoente):**

```
assert 1e3  ==  1_000.0
assert 2E4  == 20_000.0
assert 3e+1 ==     30.0
assert 4E-2 ==      0.04
assert 5e-1 ==      0.5
```

Valores e tipos de Dados

- **Booleano:**
  - Podem ser armazenados em variáveis;
  - ***Verdadeiro*** e ***Falso*** são os únicos valores booleanos primitivos;

Valores e tipos de Dados

- **Booleano - EXEMPLOS:**

```
def myBooleanVariable = true
boolean untypedBooleanVar = false
booleanField = true
```

Valores e tipos de Dados

- **Lista:**
  - Implementada da mesma forma que ocorre em Java, fazendo uso do pacote *java.util.List*;
  - Suporta qualquer tipo de dados, incluindo o tipo genérico *Object*.

Valores e tipos de Dados

- **Lista - EXEMPLOS:**

```
def desenhos = ['Dragon Ball Z', 'Johnny Bravo', 'Laboratório de Dexter']

def desenhosComDefault = ['Johnny Test', 'Batman', 'Scooby Doo'].withDefault
{ 'Smurfs' }
```

Valores e tipos de Dados

- **Array:**
  - Implementado da mesma forma que ocorre em Java, fazendo uso do pacote *java.util.Arrays* para manipulação;
  - Pode ser visto como lista e também suporta todo tipo de dados, incluindo o tipo genérico *Object*.

Valores e tipos de Dados

- **Array - EXEMPLOS:**

```
def carros = new String[3]

def carros = ["Palio", "Gol", "Ferrari"] as String[]

def numeros = [32, 44, 12, 9, 100, 180]
```

Valores e tipos de Dados

- **Map:**
  - Implementado da mesma forma que ocorre em Java, fazendo uso do pacote *java.util.Map* para manipulação;
  - Possui conjuntos de dados com chave e valor;
  - Suporta todo tipo de dados, incluindo o tipo genérico *Object*.

Valores e tipos de Dados

- **Map - EXEMPLOS:**

```
HashMap<String, Integer> contaPalavras = [   "bola": 10,
                                             "dado": 5,
                                             "xbox": 7 ]

def aluno = ["nome":"Jonas", "curso":"EngComp", idade:22]

def map = [:]
```

Valores e tipos de Dados

- **Groovy permite múltipla declaração**
  - def(a,b,c) = [12,20,’foo’]
  - def(int a , String b) = [12,’foo’]

Valores e tipos de Dados

	byte	char	short	int	long	BigInteger	float	double	BigDecimal
byte	int	int	int	int	long	BigInteger	double	double	BigDecimal
char		int	int	int	long	BigInteger	double	double	BigDecimal
short			int	int	long	BigInteger	double	double	BigDecimal
int				int	long	BigInteger	double	double	BigDecimal
long					long	BigInteger	double	double	BigDecimal
BigInteger						BigInteger	double	double	BigDecimal
float							double	double	double
double								double	double
BigDecimal									BigDecimal

Resultado de operações matemáticas entre tipos

Expressões e comandos

- **Operadores aritméticos Binários :**

```
assert  1  + 2 == 3
assert  4  - 3 == 1
assert  3  * 5 == 15
assert  3  / 2 == 1.5
assert 10  % 3 == 1
assert  2 ** 3 == 8
```

Expressões e comandos

- **Operadores Unarios :**

```
def a = 2
def b = a++ * 3
assert a == 3 && b == 6
```

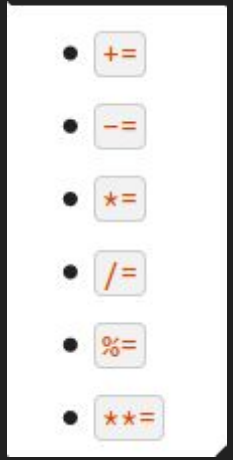
pós-fixo

```
def e = 1
def f = ++e + 3
assert e == 2 && f == 5
```

prefixo

Expressões e comandos

- **Operadores aritméticos e atribuição:**

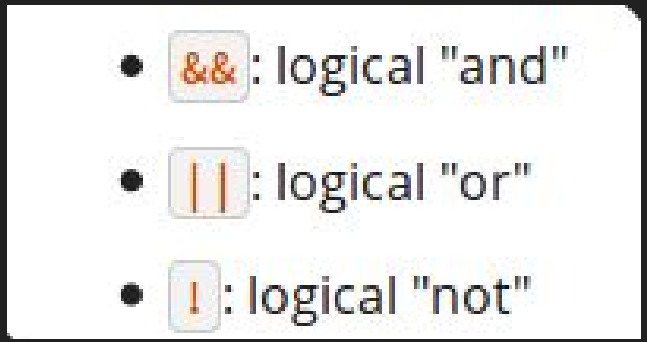


- **Operadores relacionais :**

Operator	Purpose
==	equal
!=	different
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

Expressões e comandos

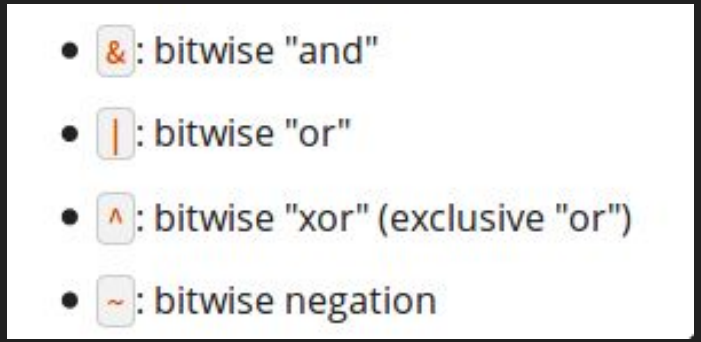
- **Operadores Lógicos:**



- **Precedência :** Not -> And -> Or
- **Linguagem tem curto circuito**

Expressões e comandos

- **Operadores Binários :**



  - aplicado em *byte* ou *int* e retorna *int*;
  - Vale ressaltar que a representação interna de tipos primitivos segue a Java Language Specification.

Expressões e comandos

- **Operadores de identificação:**
  - **is** -> Compara por referência
  - **==** -> igual a função equals()
- **Operador de Coerção:**
  - as

```
Integer x = 123
String s = x as String
```

- **Operadores de Ternário :**

```
result = string ? 'Found' : 'Not found'
```

Expressões e comandos

- **Operador Elvis:**
  - `displayName = user.name ?: 'Anonymous'`
- **Outros Operadores:**
  - *
  - interable
  - Range
- **Groovy aceita SobreEscrita e SobreCarga**

Expressões e comandos

- **Ordem de Precedência dos operadores**

Level	Operator(s)	Name(s)
1	`new` `()`	object creation, explicit parentheses
	`()` `{}` `[]`	method call, closure, literal list/map
	`.` `.&` `.@`	member access, method closure, field/attribute access
	`?.` `*` `*.` `*:`	safe dereferencing, spread, spread-dot, spread-map
	`~` `!` `(type)`	bitwise negate/pattern, not, typecast
	`[]` `++` `--`	list/map/array index, post inc/decrement
2	`**`	power

Expressões e comandos

3	`++` `--` `+` `-`	pre inc/decrement, unary plus, unary minus
4	`*` `/` `%`	multiply, div, remainder
5	`+` `-`	addition, subtraction
6	`<<` `>>` `>>>` `..` `..<`	left/right (unsigned) shift, inclusive/exclusive range
7	`<` `<=` `>` `>=` `in` `instanceof` `as`	less/greater than/or equal, in, instanceof, type coercion
8	`==` `!=` `<=>`	equals, not equals, compare to
	`=~` `==~`	regex find, regex match
9	`&`	binary/bitwise and

Expressões e comandos

10	`^`	binary/bitwise xor
11	`|`	binary/bitwise or
12	`&&`	logical and
13	`||`	logical or
14	`? :`	ternary conditional
	`?:`	elvis operator
15	`=` `**=` `*=` `/=` `%=` `+=` `-=` `<<=` `>>=` `>>>=` `&=` `^=` `|=`	various assignments

Expressões e comandos

- **Comando Condicional:**
  - if/else
  - Switch/case
- **Comando Iterativo**
  - For
  - For iterativo
  - while
- **Desvio incondicional**
- **Contém Label**

Expressões e comandos

- **Existe funções do tipo void()**

```
class NestedClosures {
    void run() {
        def nestedClosures = {
            def cl = { this }
            cl()
        }
        assert nestedClosures() == this
    }
}
```

Expressões e comandos

- **Para acessar métodos e propriedades usa-se “.”**

```
trait Counting {
    int x
    void inc() {
        x++
    }
}
class Counter implements Counting {}
def c = new Counter()
c.inc()
```

Modularização

- **Tipos abstratos:**

```groovy
abstract class Abstract {
    String name
    abstract def abstractMethod()
    def concreteMethod() {
        println 'concrete'
    }
}
```

```groovy
trait FlyingAbility {
        String fly() { "I'm flying!" }
}
```

```groovy
interface Greeter {
    protected void greet(String name)
}

class SystemGreeter implements Greeter {
    void greet(String name) {
        println "Hello $name"
    }
}
def greeter = new SystemGreeter()
assert greeter instanceof Greeter
```

Modularização

- **Abstração do processo - Parâmetro:**
  - Aceita valores default;
  - Oferece varargs;
  - Como utiliza-se apenas objetos em groovy, tem-se passagem bidirecional do objeto ou unidirecional da referência;
  - Usa o modo normal no momento da passagem.

Modularização

- **Passagem de parâmetro**

```
class PersonWOConstructor {
    String name
    Integer age
}

def person5 = new PersonWOConstructor(name: 'Marie')
def person6 = new PersonWOConstructor(age: 1)
def person7 = new PersonWOConstructor(name: 'Marie', age:
2)
```

Modularização

- **Passagem de parâmetro**

```groovy
def concat1 = { String... args -> args.join('') }
assert concat1('abc','def') == 'abcdef'

def concat2 = { String[] args -> args.join('') }
assert concat2('abc', 'def') == 'abcdef'

def multiConcat = { int n, String... args ->
    args.join('')*n
}
assert multiConcat(2, 'abc','def') == 'abcdefabcdef'
```

Polimorfismo

- **Coerção:**
  - Define um operador de coerção chamado *as*:

```
Integer x = 123
String s = x as String
```

  - As regras de coerção diferem uma da outra, depende do destino e fonte. A coerção pode falhar caso não encontre uma regra definida. As regras personalizadas de coerção podem ser implementadas graças ao método *asType.*

Polimorfismo

- **Sobrecarga:**
  - Similar ao Java, porém, Groovy aceita sobrecarga de operadores realizado pelo programador;
  - Permite a sobrecarga de operadores apenas redefinindo o método do operador;

Polimorfismo

- **Sobrecarga - EXEMPLO:**

```
class Bucket { //Bucket implementa um método especial chamado plus.
Apenas com isso, a classe Bucket pode ser utilizado como o operador
+.
    int size

    Bucket(int size) { this.size = size }

    Bucket plus(Bucket other) {
        return new Bucket(this.size + other.size)
    }
}
```

Polimorfismo

- **Sobrecarga - EXEMPLO:**

```
def b1 = new Bucket(4)
def b2 = new Bucket(11)
assert (b1 + b2).size == 15
```

Polimorfismo

- **Paramétrico - Tipo Genérico:**
  - Mesmo funcionamento do Tipo Genérico de Java

```
public class ListType<T> {
   private T localt;

   public T get() {
      return this.localt;
   }

   public void set(T plocal) {
      this.localt = plocal;
   }
}
```

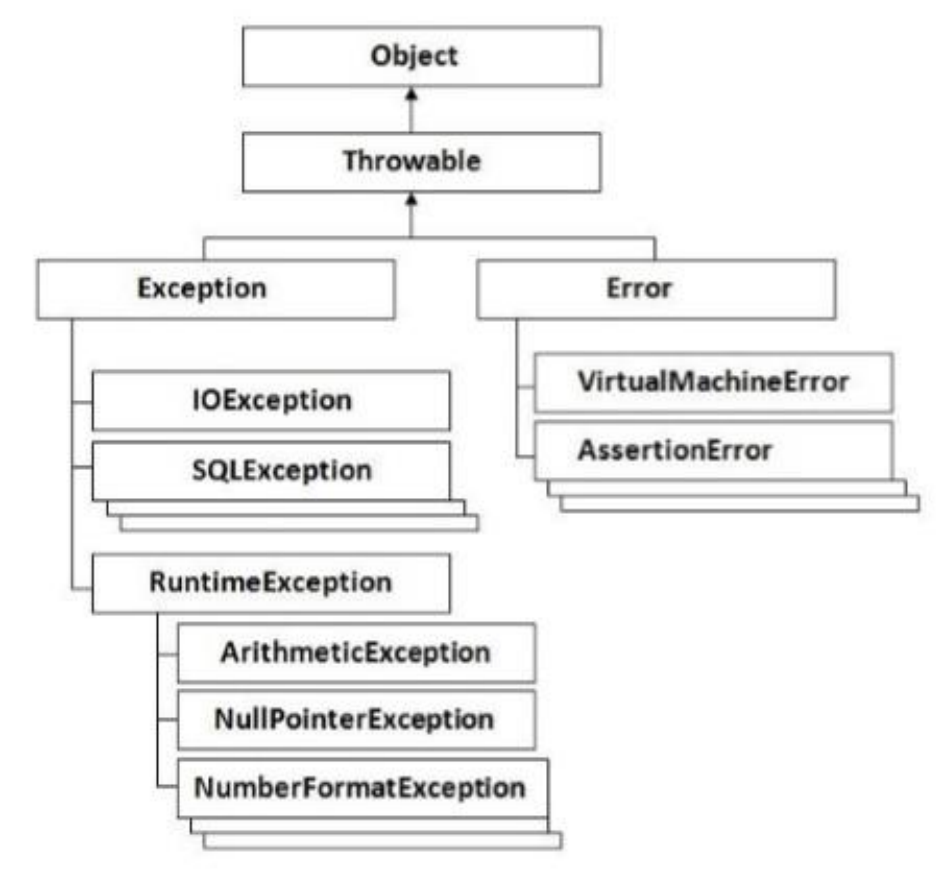
Polimorfismo

- **Inclusão - Herança:**
  - Classes têm capacidade de herdar o comportamento de outras classes;
  - As classes que provê os comportamentos herdados, são chamadas de super-classes;
  - Para herdar de outras classes, usa-se *extends;*
  - Não existe herança múltipla em Groovy, assim como não existem em Java. Mas é possível simular utilizando *interfaces.*

Exceções

- **O tratamento de exceções do Groovy é feito da mesma forma que em Java, mas deixa opcional o uso do throws na assinatura dos métodos, tanto para exceções checadas (RuntimeException e Error) como não checadas;**
- **O programador fica livre para definir como e onde essas exceções serão tratadas.**

Exceções


Object
Throwable
Exception
Error
IOException
SQLException
RuntimeException
ArithmeticException
NullPointerException
NumberFormatException
VirtualMachineError
AssertionError

Exceções

```
class Exemplo {
  static void main(String[] args) {
    try {
      def vet = new int[3];
      vet[5] = 5;
    } catch(Exception ex) {
      println("Peguei a exceção!");
    }

    println("Segue o jogo...");
  }
}
```

Concorrência

- **Assim como em Java, pode ser implementada manualmente com o uso de *threads* e classes *syncronized* ou com o uso de bibliotecas, como *GPars* e *java.util.concurrent*.**

Concorrência

```
void sendEmails(List<Email> emails) {
    def threads = []
    def sendEm = emails.each{ email ->
        def th = new Thread({
            Random rand = new Random()
            def wait = (long)(rand.nextDouble() * 1000)
            println "in closure"
            this.sleep wait
            sendEmail(email)
        })
        println "putting thread in list"
        threads << th
    }
    threads.each { it.run() }
    threads.each { it.join() }
}
```

Concorrência

```
def sendEmails(emails) {

  GParsPool.withPool {
    emails.eachParallel { email ->
      def wait = (long) new Random().nextDouble() * 1000
      println "in closure"
      this.sleep wait
      sendEmail(email)
    }
  }
}
```

Avaliação do Groovy

Critérios Gerais	C	C++	Java	Groovy
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Não
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim

Avaliação do Groovy

Critérios Gerais	C	C++	Java	Groovy
Método de projeto	Estruturado	Estruturado e OO	OO	OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

Avaliação do Groovy

Critérios específicos	C	C++	Java	Groovy
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos	Sim	Sim	Sim	Não
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Passagem de parâmetros	lista de variáveis e por valor	Lista variável, default, por valor e referência	Lista variável, por valor e por cópia de referência	Lista variável, por valor e por cópia de referência

Avaliação do Groovy

Critérios específicos	C	C++	Java	Groovy
Encapsulamento de proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Não	Parcial	Sim	Sim
Verificação de tipos	Estática	Estática/Dinâmica	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos	Todos
Exceções	Não	Parcial	Sim	Sim

Referências

- Groovy Language Documentation. <http://docs.groovy-lang.org/next/html/documentation/#_integral_literals>. Acesso em: 20 nov. 2017.
- Object Partners. Optional Typing in Groovy. ago. 2013. Disponível em: <https://objectpartners.com/2013/08/19/optional-typing-in-groovy/>. Acesso em: 18 nov. 2017.
- Singh, J. Groovy the concept of optional typing. set. 2016. Disponível em: <http://javabeginnerstutorial.com/groovy/groovy-the-concept-of-optional-typing/>. Acesso em: 18 nov. 2017.
- Groovy exception handling. Disponível em: <https://www.tutorialspoint.com/groovy/groovy_exception_handling.htm>. Acesso em: 25 nov. 2017.
- High Lighter. Disponível em: <http://markup.su/highlighter/>. Acesso em: 25 nov. 2017.