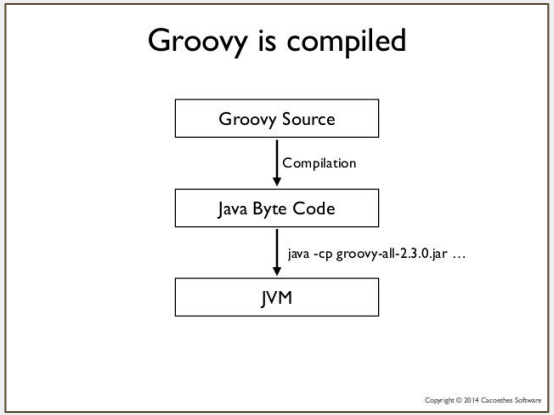
Linguagens de Programação
Groovy

Integrantes:
- Bruno Ramiro
- Danilo Felicio
- Matheus Bongiovani
- Matusalem Mansur
- Vitor Araujo

Introdução

>Groovy é uma linguagem de programação baseado em Java, influenciado pelas funcionalidades de linguagens como Python, Ruby, Perl e Smalltalk.

>Possui implementação híbrida utilizando a JVM.



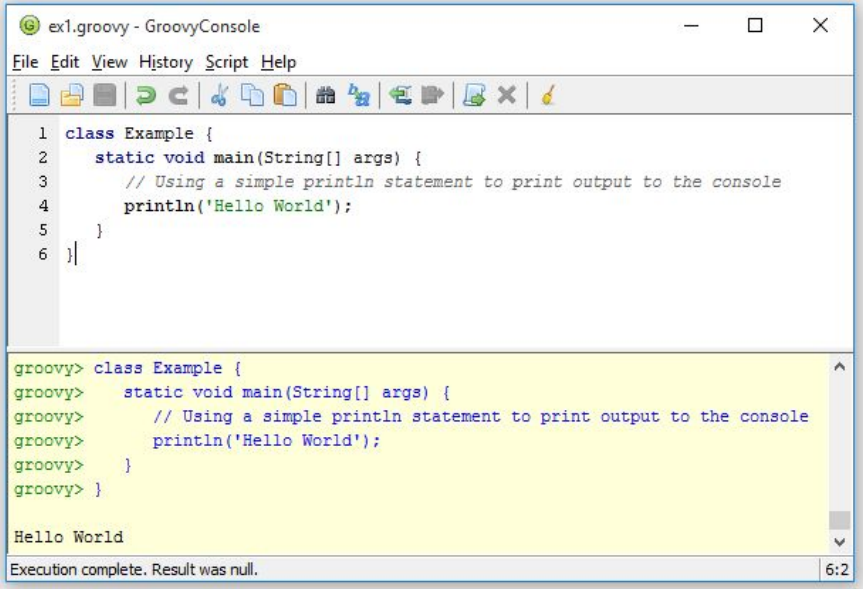
>O Groovy também pode executar código Java, e até mesmo importar suas bibliotecas

>Possui um Groovy Console para rodar scripts

>Podemos baixar o Groovy em: http://groovy-lang.org/download.html
Versão atual estável: 2.5.3

>Também precisaremos do Java: https://www.java.com/pt_BR/download/

Hello World



Groovy Console

```groovy
//ex1.groovy
class Example {
  static void main(String[] args) {
     // Using a simple println
statement to print output to the
console
     println('Hello World');
  }
}
```

Vale a pena comentar

- No Groovy ; (ponto e vírgula) é obrigatório somente se usar mais de uma instrução por linha.
- Todos os atributos de uma classe são, por padrão, private.
- Todas as classes são, por padrão, pública.
- Getters and Setters criado automaticamente e dinamicamente, porém deve ser declarado como 'final' para não gerar o setter. `final String constante = "inalterável"`
- Groovy define vários pacotes bases com importação automática, logo não é obrigatório dar um import java.math, java.lang.string ou java.util.*

Colinha “rsrs”

1.1.2. Keywords

The following list represents all the keywords of the Groovy language:

Table 1. Keywords

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			

http://docs.groovy-lang.org/docs/groovy-2.5.3/html/documentation/

Groovy-logo

Paradigm	Object-oriented, imperative, scripting
Designed by	James Strachan
Developer	Guillaume Laforge (PMC Chair) Jochen Theodorou (Tech Lead) Paul King Cedric Champeau
First appeared	2003; 15 years ago
Stable release	2.5.3 / October 10, 2018; 20 days ago[1]
Preview release	3.0.0 Alpha 3 / June 23, 2018; 4 months ago[2]
Typing discipline	Dynamic, static, strong, duck
Platform	Java SE
License	Apache 2.0
Filename extensions	.groovy
Website	groovy-lang.org
Influenced by	
Java, Python, Ruby, Perl, Smalltalk, Objective-C	
Influenced	
Kotlin	

Amarrações

Identificadores: começam com uma letra, um cifrão $, ou um underline.

>Não podem começar com números, nem ter caracteres especiais no meio (exceto $ e _).

>Exceções testadas: não pode ser apenas '$' (tem q ser $alguma_coisa).

>Mas pode ser apenas: _

>Começar com um identificador com $, impede de usar o valor dela ${identificador} "String interpolation"

>println''' permite quebra de linhas e só termina

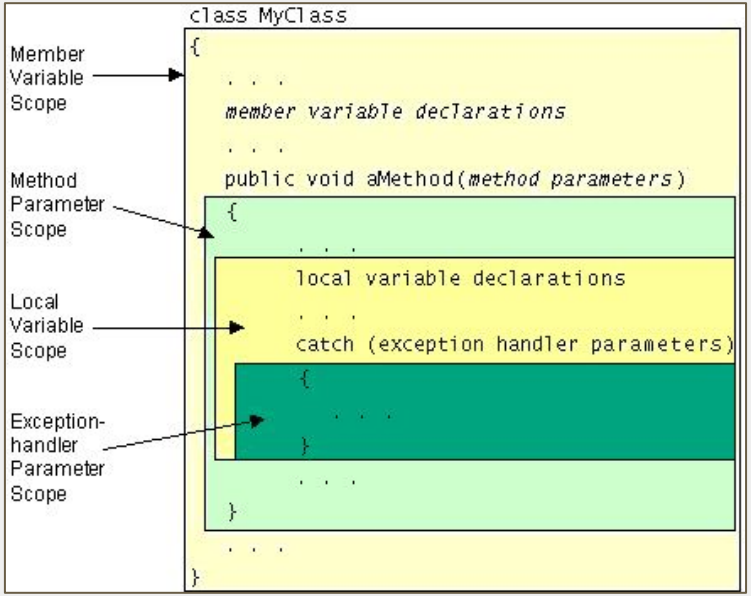
quanto tiver terminado 'kkkkk ''')

```
//exemplos de identificadores
def nl=0
int _=1; //Funciona!
def $=2; //Não Funciona!
float $_=3; // Funciona!
println("O valor de nl é: $nl") // o valor de nl é: 0
println("O valor de _ é: $_") //o valor de _ é: 1
println'''Já se o identificador começa com $
não dá pra usar \${identificador}  ''');
println("Só se for do jeito normal \$_= "+$_)
//Só se for do jeito normal $_= 3.0

def vet = [0..10,11,12,13]
println("O valor de vet é: $vet e vet[2] é
"+vet[2])
/*O valor de vet é: [0..10, 11, 12, 13] e
vet[2] é 12*/
```

Escopo e D&D

- Assim como em Java (e a maioria das LPs), Groovy possui escopo dinâmico
- Possui pacotes, classes, funções e blocos iguais a Java
- Definições e Declarações das amarrações são iguais a Java

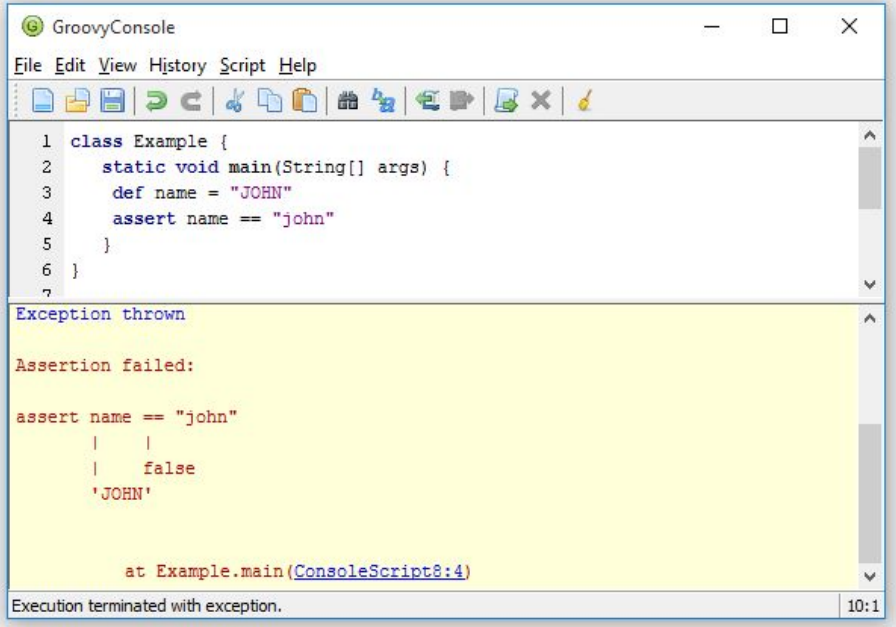


Valores e Tipos de Dados

- Groovy tem as mesmas “Built-in Data Types” do que Java
    - byte, short, int, long, float, double, char, Boolean e String em `java.lang`
- E é claro as classes importadas, como `java.lang.BigDecimal` ou `java.util.List`
- A definição dos tipos de variáveis no Groovy podem ser dinâmicas ou estáticas
- Como ‘true’ e ‘false’ são palavras reservadas em Groovy, podemos criar um dado do tipo Booleano usando tipagem dinâmica. `def verdadeiro = false`
- Infelizmente possui a palavra ‘pointer’ na sua documentação... **Method pointer operator (`.&`)** que é usado para guardar a referência de um método em uma variável.

Exemplos

Tipo assert




```groovy
class Example {
  static void main(String[] args) {
   def str = "Matheus"
   def CAPS = str.&toUpperCase
   def maiusculo = CAPS()
   assert maiusculo == str.toUpperCase()
   assert maiusculo instanceof String
   Boolean igual = (maiusculo ==
str.toUpperCase())
   println(igual)
   println(str)
   println(maiusculo)
   }
}
//true
//Matheus
//MATHEUS
```

Variáveis e Constantes

- A definição dos tipos de variáveis no Groovy pode acontecer dinamicamente ou de modo estático.

Entretanto há uma diferença semântica entre as duas definições:

- Se ela é declarada como no primeiro exemplo, ela é uma variável local.
- No caso do segundo exemplo, ele entra no script binding. A ligação é visível a partir dos métodos.

```
int i = 10
println(i.class)

long l = 10000000000000000000
println(l.class)

d = 10.0
println(d.class)

s = "olá, estamos estudando Groovy!"
println(s.class)
```

```
=> class java.lang.Integer
=> class java.lang.Long
=> class java.math.BigDecimal
=> class java.lang.String
```

Expressões e Comandos

- Todos os operadores de Java são suportados em Groovy;

Operator	Purpose
+	addition
-	subtraction
*	multiplication
/	division
%	remainder
**	power

Expressões e Comandos

- Os operadores unários de incremento (++) e decremento (--) estão disponíveis:

```
def a = 2
def b = a++ * 3

assert a == 3 && b == 6

def c = 3
def d = c-- * 2

assert c == 2 && d == 6

def e = 1
def f = ++e + 3

assert e == 2 && f == 5

def g = 4
def h = --g + 1

assert g == 3 && h == 4
```

‘a’ será incrementado depois da expressão ter sido avaliada e atribuída;

‘c’ será decrementada depois da expressão ter sido avaliada e atribuída;

‘e’ será incrementada antes da expressão ser avaliada e atribuída;

‘g’ será decrementada antes da expressão ser avaliada e atribuída.

Expressões e Comandos

- Operadores aritméticos de atribuição:

  - `+=`
  - `-=`
  - `*=`
  - `/=`
  - `%=`
  - `**=`

```
def a = 4
a += 3

assert a == 7

def b = 5
b -= 3

assert b == 2

def c = 5
c *= 3

assert c == 15

def d = 10
d /= 2

assert d == 5

def e = 10
e %= 3

assert e == 1

def f = 3
f **= 2

assert f == 9
```

Expressões e Comandos

- Operadores relacionais:

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

```
assert 1 + 2 == 3
assert 3 != 4

assert -2 < 3
assert 2 <= 2
assert 3 <= 4

assert 5 > 1
assert 5 >= -2
```

Expressões e Comandos

- Operadores Lógicos

  - `&&` : logical "and"
  - `||` : logical "or"
  - `!` : logical "not"

```
assert !false
assert true && true
assert true || false
```

O operador lógico '!' tem prioridade sobre o '&&' , logo a expressão

```
assert (!false && false) == false
```

é verdadeira.

E o operador '&&' tem prioridade sobre o operador '||'

```
assert true || true && false
```

Expressões e Comandos

- Operadores binários

- `&` : bitwise "and"
- `|` : bitwise "or"
- `^` : bitwise "xor" (exclusive "or")
- `~` : bitwise negation

Aplicado em *byte* ou *int* e retorna *int*;

Operador Ternário: Você pode escrever isso:

```
result = (string!=null && string.length()>0) ? 'Found' : 'Not found'
```

Sendo equivalente a isso:

```
if (string!=null && string.length()>0) {
    result = 'Found'
} else {
    result = 'Not found'
}
```

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
3	`++` `--` `+` `-`	pre inc/decrement, unary plus, unary minus

Expressões e Comandos

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

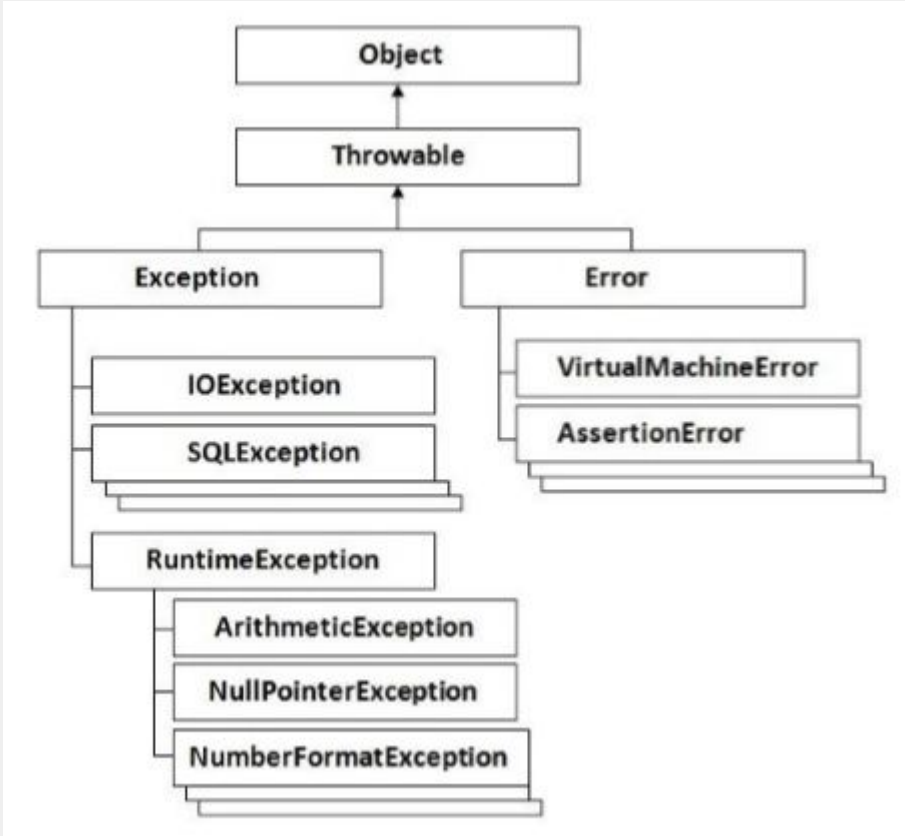
Exceções

- **Erros geralmente causam a falha de um programa e o seu término, já exceções podem e devem ser tratadas no programa.**

- **O tratamento de exceções é feito da mesma forma que em Java**
  - Bem como a propagação de exceções

- **Exceções checadas**
  - Extends Throwable
  - Tempo de compilação

- **Exceções não checadas**
  - Extends RuntimeException, Error
  - Tempo de execução



Hierarquia de exceções

Exceções

- **Capturando Exceções**

```
try {
  //Código Protegido
} catch(NomeDaExceção e1) {
  //Bloco de captura
}
```

- **Múltiplas capturas**

```
try {
  //Código protegido
} catch(ExceçãoTipo1 e1) {
  //Bloco de captura
} catch(ExceçãoTipo2 e2) {
  //Bloco de captura
}
```

```
class Exemplo {
  static void main(String[] args) {
    try {
      def arr = new int[3];
      arr[5] = 5;
    }catch(ArrayIndexOutOfBoundsException ex) {
      println("Capturando a exceção 'Array out of
Bounds'");
    }catch(Exception ex) {
      println("Capturando a Exceção");
    }
    println("Continuando o código após a
exceção");
  }
}
//Capturando a exceção 'Array out of Bounds'
//Continuando o código após a exceção
```

Exceções

- **Bloco Finally**

```
try {
  //Código Protegido
} catch(ExceçãoTipo1 e1) {
  //Bloco de captura
} catch(ExceçãoTipo2 e2) {
  //Bloco de captura
} catch(ExceçãoTipo3 e3) {
  //Bloco de captura
} finally {
  //O bloco finally sempre executa
}
```

```
class Exemplo {
  static void main(String[] args) {
    try {
      def arr = new int[3];
      arr[5] = 5;
    } catch(ArrayIndexOutOfBoundsException ex) {
      println("Capturando a exceção 'Array out of
Bounds'");
    }catch(Exception ex) {
      println("Capturando a Exceção");
    } finally {
      println("Execução do bloco finally");
    }

    println("Continuando o código após a exceção");
  }
}
//Capturando a exceção 'Array out of Bounds'
//Execução do bloco finally
//Continuando o código após a exceção
```

Exceções

- **Alguns métodos**
  - public String getMessage()
    - Retorna uma mensagem detalhada sobre a exceção.
  - public Throwable getCause()
    - Retorna a causa da exceção.
  - public String toString()
    - Retorna o nome da classe concatenado com o getMessage().
  - public void printStackTrace()
    - Retorna o resultado de toString() junto com o Stack Trace de System.err
  - public StackTraceElement [] getStackTrace()
    - Retorna um vetor contendo todos os elementos da Stack Trace

Modularização

- **Classe Normal**
  - Concretas
  - Podem ser instanciadas sem restrições de outras classes
  - Pública, por padrão
- **Classe Aninhada**
- **Classe Abstrata**
- **Interfaces**
- **Traits**

```
class Pessoa {

  String nome
  Integer idade
  def aumentaIdade(Integer anos)
{
      this.idade += anos
  }
}

def p = new Pessoa()
```

Modularização

- **Classe Normal**
- **Classe Aninhada**
  - Definidas dentro de outra classe
  - Classe externa acessa classe interna normalmente
  - Classe interna acessa dados da classe externa mesmo que sejam privados
  - Outras classes, com exceção da classe externa, não têm acesso à classe interna
- **Classe Abstrata**
- **Interfaces**
- **Traits**

```
class Externa {
   private String strPrivada

   def chamaMetodoInterno() {
      new Interna().metodoA()
   }

   class Interna {
      def metodoA() {
         println "${strPrivada}."
      }
   }
}
```

Modularização

- **Classe Normal**
- **Classe Aninhada**
  - Aumentam encapsulamento, escondendo a classe interna de outras classes.
  - Melhoram a organização, agrupando as classes usadas por apenas uma classe.
  - Aumenta a manutenibilidade do código, já que as classes internas estão mais perto da classe que a utiliza.
- **Classe Abstrata**
- **Interfaces**
- **Traits**

```
class Externa {
   private String strPrivada

   def chamaMetodoInterno() {
      new Interna().metodoA()
   }

   class Interna {
      def metodoA() {
         println "${strPrivada}."
      }
   }
}
```

Modularização

- **Classe Normal**
- **Classe Aninhada**
- **Classe Abstrata**
  - Representam conceitos genéricos de classes
  - Não pode ser instanciada
  - Incluem campos/propriedades e métodos abstratos ou concretos
  - Métodos abstratos devem ser implementados pelas subclasses.
- **Interfaces**
- **Traits**

```
abstract class Abstrata {
    String nome

    abstract def metodoAbstrato()

    def metodoConcreto() {
        println 'concreto'
    }
}
```

Modularização

- **Classe Normal**
- **Classe Aninhada**
- **Classe Abstrata**
- **Interfaces**
  - Interface define um “contrato” em que a classe precisa seguir
  - Define apenas uma lista de métodos (assinatura) que precisam ser implementados
  - Métodos precisam ser sempre públicos
  - Diferença para classes abstratas:
    - Classes abstratas podem conter propriedades e/ou métodos concretos, enquanto interfaces contém apenas as assinaturas dos métodos
- **Traits**

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

class SystemGreeter implements Greeter {
   void greet(String name) {
      println "Hello $name"
   }
}

def greeter = new SystemGreeter()
assert greeter instanceof Greeter
```

Modularização

- **Classe Normal**
- **Classe Aninhada**
- **Classe Abstrata**
- **Interfaces**
- **Traits**
  - **Construção estrutural da linguagem que permite:**
    - Composição de comportamentos
    - Implementação, em tempo de execução, de interfaces
    - Sobreposição de comportamento
  - **Podem ser vistos como Interfaces que carregam consigo estados e implementações default.**

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

class Bird implements FlyingAbility {}
def b = new Bird()
assert b.fly() == "I'm flying!"
```

Modularização

- **Correspondência entre parâmetros**
  - **Posicional**
  - **Pode ser por valores default**
  - O número de parâmetros pode variar
  - Passagem de parâmetros

```
def fun(x = 10, y = 0, w = 2){
  println(x + y + w)
}

fun() //Imprime 12
fun("oi",198) //Imprime oi1982
```

Modularização

- **Correspondência entre parâmetros**
  - Posicional
  - Pode ser por valores default
  - **O número de parâmetros pode variar**
  - Passagem de parâmetros

```
def metodo1(int... args){
   for (int i = 0; i<args.length; i++){
      println(args[i])
   }
}

metodo1(1,3,5,6,0,7)

def metodo2(Object[] args){
   for (int i = 0; i<args.length; i++){
      println(args[i])
   }
}

metodo2("Olá mundo", true, 12345)
```

Modularização

- **Correspondência entre parâmetros**
  - Posicional
  - Pode ser por valores default
  - O número de parâmetros pode variar
  - **Passagem de parâmetros**
    - Passagem bidirecional de objetos e unidirecional de referências
    - Momento da passagem normal(eager)

```
def inicializa(int x, int[] vetor){
   x = 0
   for(int i = 0; i<vetor.length;i++){
      vetor[i] = 0
   }
   vetor = [1,2,3,4,5]
   println(vetor)
}

int[] vetor = new int[6]
vetor[5] = 1
vetor[2] = 3
vetor[4] = 10
int x = 100

inicializa(x, vetor)

println(vetor)                    //[1, 2, 3, 4, 5]
println(x)                        //[0, 0, 0, 0, 0, 0]
                                  //100
```

Polimorfismo

Conceito de polimorfismo : Em LP’s, refere-se a possibilidade de se criar código capaz de operar sobre valores e tipos diferentes.

Quanto mais polimorfismo, maior a possibilidade de se criar código reutilizável.

- **Sistema de tipos :** Descreve de modo adequado os dados, aumentando a legibilidade e a redigibilidade, evitando que programas executem operações incoerentes.

Verificação de tipos

- A definição dos tipos de variáveis no Groovy pode acontecer dinamicamente ou de modo estático, ou seja, definir o tipo das variáveis no Groovy é opcional;

- LP’s dinamicamente tipadas perdem eficiência computacional, devido a inclusão de verificação de tipos em tempo de execução -> erros só podem ser verificados em tempo de execução;

- Menor confiabilidade;

- A partir do Groovy 2.0, se tornou possível ativar um recurso adicional para fazer tipagem estática. Antes da classe adiciona-se a seguinte anotação: @groovy.transform.TypeChecked (Também é necessária uma biblioteca)

Verificação de tipos - Exemplo

```groovy
int i = 10
println(i.class)

long l = 10000000000000000000
println(l.class)

d = 10.0
println(d.class)

s = "olá, estamos estudando Groovy!"
println(s.class)
```

```
=> class java.lang.Integer
=> class java.lang.Long
=> class java.math.BigDecimal
=> class java.lang.String
```

Equivalência de tipos

- Existem muitos tipos de conversões explícitas, como atribuições de char para int, de int para char e etc.

- Em Groovy, não são consideradas equivalentes classes diferentes implementadas pelo programador com os mesmos atributos.

  Exemplo : Uma instância da classe “automóvel” tem os atributos placa e cor, e uma instância da classe “carro” também possui os atributos placa e cor. Não é possível passar um “carro” como parâmetro para um método que pede um automóvel como parâmetro.

Coerção

- O operador de coerção (as) é uma variante de conversão. A coerção converte objetos de um tipo para outro sem que eles sejam compatíveis para atribuição.

  Exemplo :

```
Integer x = 123
String s = (String) x
```

  Utilizando o operador de coerção:

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

Coerção(2)

- Quando um objeto é coagido para outro, a menos que o tipo de destino seja o mesmo que o tipo de origem, a coerção retornará um novo objeto. As regras de coerção diferem dependendo dos tipos de origem e destino, e a coerção pode falhar se nenhuma regra de conversão for encontrada. Regras de conversão personalizadas podem ser implementadas graças ao método asType :

Coerção - Exemplo

```
class Identifiable {
    String name
}
class User {
    Long id
    String name
    def asType(Class target) {                                          1
        if (target == Identifiable) {
            return new Identifiable(name: name)
        }
        throw new ClassCastException("User cannot be coerced into $target")
    }
}
def u = new User(name: 'Xavier')                                        2
def p = u as Identifiable                                               3
assert p instanceof Identifiable                                        4
assert !(p instanceof User)                                             5
```

Sobrecarga

- Similar a linguagem JAVA, com a exceção de que em Groovy, operadores podem ser sobrecarregados pelo programador;

- Permite a sobrecarga dos operadores apenas redefinindo o método do operador;

- Aumenta a legibilidade e a redigibilidade, porém aumenta a complexidade da LP.

Sobrecarga - Exemplo

```
class Bucket {
    int size

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

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

1

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

1

Sobrecarga - Tabela : operadores x métodos

Operator	Method
a + b	a.plus(b)
a - b	a.minus(b)
a * b	a.multiply(b)
a ** b	a.power(b)
a / b	a.div(b)
a % b	a.mod(b)
a | b	a.or(b)
a & b	a.and(b)
a ^ b	a.xor(b)

Operator	Method
a++ or ++a	a.next()
a-- or --a	a.previous()
a[b]	a.getAt(b)
a[b] = c	a.putAt(b, c)
a << b	a.leftShift(b)
a >> b	a.rightShift(b)
switch(a) { case(b) : }	b.isCase(a)
~a	a.bitwiseNegate()
-a	a.negative()

Operator	Method
a == b	a.equals(b) or a.compareTo(b) == 0 **
a != b	! a.equals(b)
a <=> b	a.compareTo(b)
a > b	a.compareTo(b) > 0
a >= b	a.compareTo(b) >= 0
a < b	a.compareTo(b) < 0
a <= b	a.compareTo(b) <= 0

Paramétrico

- Tipos genéricos (A partir do Groovy 1.5);
- Mesmo funcionamento de tipos genéricos em JAVA;

Exemplo :

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

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

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

Inclusão - Herança

- Igual em JAVA;
- Não existe herança múltipla em Groovy, assim como não existe em JAVA, porém podemos simular utilizando interfaces;
- Para herdar de outras classes, usa-se Extends;
- Amarração tardia;
- Permite ampliação e estreitamento;
- Possui classes abstratas e interfaces;
- Aceita metaclasses.

Inclusão - Herança - Exemplo

```java
public class Produto {
  protected String nome;
  protected double preco;

  public Produto(String nome, double preco) {
     this.nome = nome;
     this.preco = preco;
  }

  public boolean ehCaro() {
     return (preco > 1000);
  }

  /* Métodos de acesso ... */
}
```

```java
public class Livro extends Produto {
  private String autor;
  private int paginas;

  public Livro(String nome, double preco,
                String autor, int paginas) {
     super(nome, preco);
     this.autor = autor;
     this.paginas = paginas;
  }

  public boolean ehGrande() {
     return (paginas > 200);
  }
}
```

```java
public class Loja {
  public static void main(String[] args) {
     Produto p = new Produto("HD externo", 500);
     System.out.println(l.ehCaro());

     Livro l;
     l = new Livro("Linguagem de Programação",
          74.90, "Flávio Varejão", 334);

     System.out.println(l.ehCaro());
     System.out.println(l.ehGrande());
  }
}
```

Meta-classes

- **Metaprogramação :** Adicionar comportamento às classes em tempo de execução;
  - Adicionar métodos às classes;
  - Adicionar propriedades;

- Todas as classes escritas em Groovy herdam de object e implementam a interface Groovy Object;

- **Meta Classes :**
  - As classes definem o comportamento dos objetos;
  - As meta Classes definem o comportamento de certas classes ou de suas instâncias.

Concorrência

A concorrência em Groovy é implementado manualmente com o uso threads , classes syncronized e também com o uso de bibliotecas como Gpars e java.util.concurrent

Implementação de Threads usando a classe Thread

```
public class MinhaThread extends Thread {
        private String nome
        private int tempo
        public MinhaThread(String nome, int tempo){
                this.nome = nome
                this.tempo = tempo
                start()
        }
        public void run(){
                try {
                        for (int i=0; i<6; i++){
                                System.out.println(nome + " contador " + i)
                                Thread.sleep(tempo)
                        }
                } catch (InterruptedException e) {
                        e.printStackTrace()
                }

                println(nome + " terminou a execução")
        }
}
```

```
public class Teste {
        public static void main(String[] args) {

                MinhaThread thread = new MinhaThread("Thread #1", 600)
                //thread.start()

                MinhaThread thread2 = new MinhaThread("Thread #2", 900)

                MinhaThread thread3 = new MinhaThread("Thread #3", 500)
        }
}
```

Saída

```
Thread #1 contador 0
Thread #2 contador 0
Thread #3 contador 0
Thread #3 contador 1
Thread #1 contador 1
Thread #2 contador 1
Thread #3 contador 2
Thread #1 contador 2
Thread #3 contador 3
Thread #1 contador 3
Thread #2 contador 2
Thread #3 contador 4
Thread #1 contador 4
Thread #3 contador 5
Thread #2 contador 3
Thread #1 contador 5
Thread #3 terminou a execução
Thread #2 contador 4
Thread #1 terminou a execução
Thread #2 contador 5
Thread #2 terminou a execução
```

```java
public class Teste {
        public static void main(String[] args) {

                MinhaThread thread = new MinhaThread("Thread #1", 600)
                //thread.start()

                MinhaThread thread2 = new MinhaThread("Thread #2", 900)

                MinhaThread thread3 = new MinhaThread("Thread #3", 500)
        }
}
```

Implementação de Threads usando a interface Runnable

```
public class MinhaThread implements Runnable {
        private String nome
        private int tempo
        public MinhaThread(String nome, int tempo){
                this.nome = nome
                this.tempo = tempo
                Thread t = new Thread(this)
                t.start()
        }
        @Override
        public void run() {
                try {
                        for (int i=0; i<6; i++){
                                println(nome + " contador " + i);
                                Thread.sleep(tempo)
                        }
                } catch (InterruptedException e) {
                        e.printStackTrace()
                }
                println(nome + " terminou a execução")
        }
}
```

```
public class Teste {
        public static void main(String[] args) {

                MinhaThread  thread1 = new MinhaThread("#1", 900)
                        //Thread t1 = new Thread(thread1)
                        //t1.start()
                 MinhaThread thread2 = new MinhaThread("#2", 650)

                 MinhaThread  thread3 = new MinhaThread("#3", 1100)

        }
}
```

Saída

```
#1 contador 0
#2 contador 0
#3 contador 0
#2 contador 1
#1 contador 1
#3 contador 1
#2 contador 2
#1 contador 2
#2 contador 3
#3 contador 2
#2 contador 4
#1 contador 3
#2 contador 5
#3 contador 3
#1 contador 4
#2 terminou a execução
#3 contador 4
#1 contador 5
#1 terminou a execução
#3 contador 5
#3 terminou a execução
```

```
public class Teste {
      public static void main(String[] args) {

         MinhaThread  thread1 = new MinhaThread("#1", 900)
                  //Thread t1 = new Thread(thread1)
                  //t1.start()
          MinhaThread thread2 = new MinhaThread("#2", 650)

          MinhaThread  thread3 = new MinhaThread("#3", 1100)

        }}
```

Synchronized

```
public class Calculadora {
        private int soma

        public synchronized int somaArray(int[] array){

                soma = 0

                for (int i=0; i<array.length; i++) {

                        soma += array[i]

                          println("Executando a soma " +
                         Thread.currentThread().getName() +
                        " somando o valor " + array[i] + " com total de " + soma)

                        try {
                                Thread.sleep(100)
                        } catch (InterruptedException e) {
                                e.printStackTrace()
                        }

                }

                return soma
        }
}
```

```
public class MinhaThreadSoma implements Runnable {
        private String nome
        private int[] nums
        private static Calculadora calc = new Calculadora()

        public MinhaThreadSoma(String nome, int[] nums){
                this.nome = nome
                this.nums = nums
                new Thread(this, nome).start();
                //Thread t = new Thread(this, nome)
                //t.start()
        }
        @Override
        public void run() {

                println(this.nome + " iniciada")

                int soma = calc.somaArray(this.nums)

               println("Resultado da soma para thread " +
this.nome + " é: " + soma)

                println(this.nome + " terminada")

        }

}
```

Synchronized

```
public class Teste {
	public static void main(String[] args) {

		int[] array = {1, 2, 3, 4, 5};
		MinhaThreadSoma t1 = new MinhaThreadSoma("#1", array);
		MinhaThreadSoma t2 = new MinhaThreadSoma("#2", array);

		//15
	}
}
```

Saída sem Synchnized

#2 iniciada
#1 iniciada
Executando a soma #1 somando o valor 1 com total de 1
Executando a soma #2 somando o valor 1 com total de 1
Executando a soma #1 somando o valor 2 com total de 3
Executando a soma #2 somando o valor 2 com total de 5
Executando a soma #1 somando o valor 3 com total de 8
Executando a soma #2 somando o valor 3 com total de 11
Executando a soma #2 somando o valor 4 com total de 15
Executando a soma #1 somando o valor 4 com total de 19
Executando a soma #2 somando o valor 5 com total de 24
Executando a soma #1 somando o valor 5 com total de 29
Resultado da soma para thread #2 é: 29
#2 terminada
Resultado da soma para thread #1 é: 29
#1 terminada

Saída com Synchnized

#2 iniciada
Executando a soma #2 somando o valor 1 com total de 1
#1 iniciada
Executando a soma #2 somando o valor 2 com total de 3
Executando a soma #2 somando o valor 3 com total de 6
Executando a soma #2 somando o valor 4 com total de 10
Executando a soma #2 somando o valor 5 com total de 15
Executando a soma #1 somando o valor 1 com total de 1
Resultado da soma para thread #2 é: 15
#2 terminada
Executando a soma #1 somando o valor 2 com total de 3
Executando a soma #1 somando o valor 3 com total de 6
Executando a soma #1 somando o valor 4 com total de 10
Executando a soma #1 somando o valor 5 com total de 15
Resultado da soma para thread #1 é: 15
#1 terminada

Avaliação do Groovy

Critérios Gerais	C	C++	Java	Groovy
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Facilidade de 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 Orientado a Objeto	Orientado a Objeto	Orientado a Objeto,funcional
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende do Projeto	Depende do Projeto	Depende do Projeto	Depende do Projeto

Avaliação do Groovy

Critérios Específico	C	C++	Java	Groovy
Escopo	Sim	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim	Sim
Tipos Primitivos e Compostos	Sim	Sim	Sim	Não possui tipos primitivos
Gerenciamento de Memória	Programador	Programador	Sistema	Sistema
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	por valor e por cópia de referência

Avaliação do Groovy

Critérios Específico	C	C++	Java	Groovy
Encapsulamento e 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
Concorrência	Não	Não	Sim	Sim

Referências

http://groovy-lang.org

https://en.wikipedia.org/wiki/Apache_Groovy

https://pt.wikipedia.org/wiki/Groovy

https://www.tutorialspoint.com/groovy/groovy_object_oriented.htm

https://www.devmedia.com.br/linguagem-de-programacao-groovy-introducao

http://docs.groovy-lang.org/docs/groovy-2.5.3/html/documentation