
UNIVERSIDADE ESTADUAL DA PARAÍBA
CENTRO DE CIÊNCIAS E TECNOLOGIA
CIÊNCIA DA COMPUTAÇÃO

JOÃO VICTOR
RODRIGO LOIOLA DE FARIAS

# Projeto da Unidade I

*UNO*

CAMPINA GRANDE
11 de Abril

# 1. Considerações Iniciais

## 1.1. Classificação

Groovy é uma linguagem de programação de alto nível, dinâmica e orientada a objetos que roda na Máquina Virtual Java (JVM). Ela é classificada como uma linguagem de programação multiparadigma, pois suporta programação procedural, orientada a objetos e funcional. Além disso, Groovy é uma linguagem de terceira geração.

## 1.2. Histórico

Groovy teve seu início no início dos anos 2000, quando James Strachan começou a trabalhar na linguagem como parte de um projeto open-source. A intenção por trás de Groovy era criar uma linguagem dinâmica e expressiva que pudesse ser executada na JVM (Java Virtual Machine). Nesse sentido, Groovy foi fortemente influenciado pela linguagem Java, aproveitando sua sintaxe familiar e vasta biblioteca de classes. No entanto, Groovy também foi influenciado por outras linguagens dinâmicas e expressivas, como Python e Ruby, incorporando características como closures, metaprogramação e tipagem dinâmica.

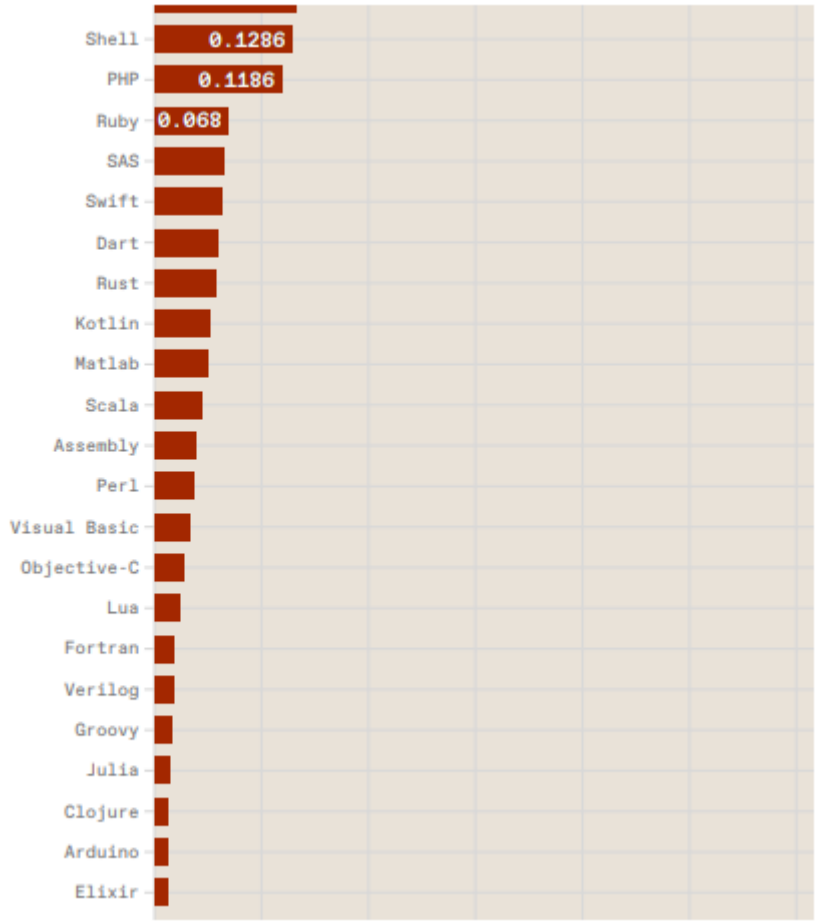
A primeira versão pública do Groovy, a versão 1.0, foi lançada em 2007. Essa versão trouxe muitos recursos inovadores, incluindo suporte a closures, sintaxe mais concisa e dinamismo. Desde então, Groovy tem passado por várias iterações e atualizações, continuando a aprimorar seus recursos, desempenho e funcionalidades. Ao longo dos anos, Groovy ganhou popularidade em uma variedade de contextos, desde scripts simples até aplicativos web complexos e sistemas de grande escala.

Uma das principais razões para o sucesso de Groovy é sua estreita integração com as ferramentas e frameworks populares da JVM, como Spring, Grails, Gradle e Spock. Essa integração facilita o uso de Groovy em diversos projetos e permite que os desenvolvedores aproveitem ao máximo o ecossistema da JVM. Em resumo, Groovy é uma linguagem dinâmica e expressiva que combina influências da linguagem Java com características de linguagens dinâmicas como Python e Ruby. Ao longo dos anos, tornou-se uma escolha popular na comunidade Java, especialmente para o desenvolvimento ágil de aplicativos web e corporativos.

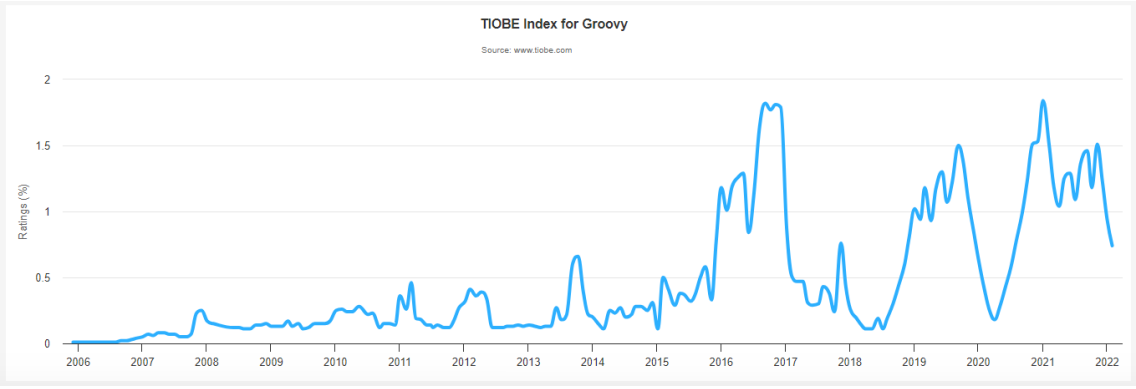
## 1.3. Popularidade

Groovy ganhou popularidade principalmente dentro da comunidade Java devido à sua integração perfeita com o ecossistema Java e sua capacidade de escrever código mais conciso e legível. No entanto, sua popularidade geral pode não ser tão alta quanto outras linguagens devido ao domínio contínuo do Java no desenvolvimento de software empresarial.

Pode ser conferido sua popularidade nos links a seguir:
**IEEE Spectrum**




**TIOBE**




### 1.4. Uso de Compilador ou Interpretador

Groovy é principalmente uma linguagem interpretada, mas também pode ser compilada para bytecode Java. Isso significa que o código Groovy é executado na JVM, assim como o código Java. A interpretação permite um

rápido ciclo de desenvolvimento, enquanto a compilação oferece melhor desempenho em produção.

## 1.5. PROXIMO TOPICO

No desenvolvimento de aplicações web, ela é comumente usada tanto no front-end quanto no back-end. Frameworks como Grails, Ratpack e Micronaut são exemplos de como Groovy é empregado para facilitar o desenvolvimento rápido e eficiente nesse domínio.

Além disso, Groovy é amplamente adotado em automação de build e gerenciamento de dependências. O Gradle, um sistema de automação de build moderno, é escrito em Groovy, o que contribui para sua popularidade na comunidade de desenvolvimento de software.

Em testes automatizados, Groovy se destaca pela sua sintaxe concisa e expressiva. O framework de teste Spock, desenvolvido em Groovy, é bastante utilizado para testes de aceitação, integração e unidade em uma variedade de projetos.

Para scripting e automação de tarefas, Groovy é uma escolha sólida devido à sua simplicidade e expressividade. É comumente utilizado em ambientes de desenvolvimento, administração de sistemas e integração de sistemas para automatizar processos e tarefas rotineiras.

Apesar de menos comum, Groovy também é aplicado no desenvolvimento de jogos e aplicativos para dispositivos móveis. Sua integração com frameworks como o LibGDX oferece uma opção viável para desenvolvedores interessados nesse domínio.

Grandes empresas como Netflix, Google, LinkedIn, IBM e Alibaba estão entre as que utilizam Groovy em seus projetos. A Netflix, por exemplo, emprega Groovy em várias áreas de seu sistema, desde automação de build até partes de sua infraestrutura de streaming. O Google também faz uso da linguagem em algumas de suas aplicações e ferramentas internas, como o plugin de análise estática para Gradle. No LinkedIn, Groovy é utilizado em automação de build, testes automatizados e desenvolvimento de ferramentas internas. Além disso, a IBM utiliza Groovy em diversas soluções de middleware e desenvolvimento de aplicativos, enquanto a Alibaba adota a linguagem em sua infraestrutura e ferramentas de automação. Essas empresas reconhecem o potencial e a eficiência de Groovy em uma variedade de cenários de desenvolvimento de software.

Pode ser encontrados algumas outras empresas que trabalham com eles no seu próprio site:


They all use Apache Groovy!
NETFLIX
amadeus
BEST BUY
CARFAX
carriots
cisco
Mutual of Omaha
Credit Suisse
AIRBUS GROUP
ENERGY TRANSFER
Europäisches Patentamt
European Patent Office
Office européen des brevets
FannieMae
Google
HYPOPORT
IBM
JPMorganChase
LinkedIn
MasterCard
MTV
MUSIC TELEVISION
COMMERZBANK
NATIONAL CANCER INSTITUTE
Nestlé
nielsen
ORACLE
Paterson
Roche
sas
sky
SONY
SmartThings
TARGET
THALES
UBS
vodafone
voyages sncf.com
Walmart
WELLS FARGO

## 2. Conceitos

### 2.1. Nomes

- **Palavras-chaves e Palavras Reservadas**

Groovy possui palavras-chave e palavras reservadas que têm significados especiais na sintaxe da linguagem e não podem ser usadas como identificadores (nomes de variáveis, métodos, classes, etc.). Algumas das palavras-chave e palavras reservadas em Groovy incluem:

**Palavras-chave:** São palavras com significados especiais na sintaxe da linguagem. São elas:

- **def**: É usado para definir variáveis. Em Groovy, você pode declarar variáveis sem especificar seu tipo usando def.
- **return**: Usado para retornar valores de métodos ou closures.
- **If e else**: Usado para controle de fluxo condicional.
- **for e while**: Usado para loops.
- **switch:** Usado para seleção de múltiplas opções.
- **try, catch e finally:** Usado para tratamento de exceções.
- **import:** Usado para importar classes ou pacotes.
- **class, interface e enum:** Usado para definir classes, interfaces e enumerações, respectivamente.
- **package:** Usado para definir um pacote.
- **assert:** Usado para testar condições de assertividade.
- **new:** Usado para instanciar objetos.
- **throw:** Usado para lançar exceções explicitamente.
- **extends, implements:** Usado para herança de classes e implementação de interfaces.
- **static:** Usado para declarar membros de classe estáticos.
- **final:** Usado para declarar constantes ou impedir a sobrescrita de métodos ou a extensão de classes.
- **true, false e null:** Valores literais de verdadeiro, falso e nulo, respectivamente.

```groovy
class Pessoa {
  def nome = null
  def idade = null

  Pessoa(String nome, int idade) {
    this.nome = nome
    this.idade = idade
  }

  String apresentar() {
    "Olá, meu nome é $nome e minha idade é $idade anos."
  }
}

def pessoa = new Pessoa("Klaudio", 20)
println(pessoa.apresentar()) //Output: Olá, meu nome é Klaudio e minha idade é 20 anos.
```

**Palavras Reservadas:** São palavras que têm significados especiais em contextos específicos e não podem ser usadas como identificadores. Por exemplo, em Groovy, ***null***, ***true*** e ***false*** são palavras reservadas que representam os valores nulos e booleanos verdadeiro e falso, respectivamente.

Além disso, Groovy também permite o uso de palavras reservadas do Java, já que é totalmente interoperável com o Java. Isso significa que as palavras reservadas do Java, como ***public***, ***private***, ***static***, ***final***, etc. Também são consideradas palavras reservadas em Groovy.

```groovy
//Exemplo Palavras Reservadas

def lista = [1, 2, 3, 4, 5]
def mapa = [nome: "João", idade: 30]
def texto = "Olá, mundo!"
def resultado = null
def condicao = true

for (numero in lista) {
    println("Número: ${numero}")
}

switch (texto) {
    case "Olá":
        resultado = "Saudação"
        break
    case "Mundo":
        resultado = "Planeta"
        break
    default:
        resultado = "Outro"
}

println(resultado)
```

## 2.2. Vinculações (Bindings)

- **Vinculações de tipos(Estáticas ou Dinâmicas)**

Groovy é uma linguagem dinamicamente tipada, o que significa que as variáveis não têm um tipo de dados estático associado a elas em tempo de compilação. A vinculação de tipos é implícita e determinada dinamicamente em tempo de execução, permitindo uma maior flexibilidade no desenvolvimento. Isso significa que você não precisa declarar explicitamente o tipo de uma variável ao defini-la.

```groovy
//Exemplo de vinculação de tipos

def x = 10 // 'x' é do tipo Integer
def y = "Olá" // 'y' é do tipo String

println(x.getClass()) // Saída: class java.lang.Integer
println(y.getClass()) // Saída: class java.lang.String
```

- **Vinculações de Armazenamento e Tempo de Vida**

Em Groovy, as variáveis são armazenadas na memória heap, que é a área de memória compartilhada entre todas as threads do programa. O tempo de vida das variáveis depende do escopo em que são declaradas. Variáveis locais dentro de métodos existem apenas durante a execução desse método e são removidas da memória assim que o método é concluído. Variáveis de instância e variáveis estáticas existem enquanto o objeto ou a classe que as contém existir na memória.

```groovy
class Exemplo {
    String nome // Variável de instância

    static String sobrenome // Variável estática

    void metodo() {
        def idade = 30 // Variável local
        println("Idade: ${idade}")
    }
}

def objeto = new Exemplo()
objeto.nome = "João"
Exemplo.sobrenome = "Silva"
objeto.metodo()
```

## 2.3. Escopos

- **Variáveis globais ou locais**

Groovy suporta variáveis globais e locais, assim como outras linguagens de programação. As variáveis globais são aquelas que podem ser acessadas de qualquer lugar do programa, enquanto as variáveis locais são acessíveis apenas dentro do bloco em que foram declaradas.

```groovy
// Exemplo variável Global
def variavelGlobal = 10

void metodo1() {
  println("Variável global dentro do método 1: $variavelGlobal")
}

void metodo2() {
  println("Variável global dentro do método 2: $variavelGlobal")
}

metodo1()
metodo2()

// Exemplo variável Local
void metodo() {
  def variavelLocal = 20
  println("Variável local dentro do método: $variavelLocal")
}

metodo()
```

- **Escopo estático ou dinâmico**

Em Groovy, o escopo é dinâmico por padrão, o que significa que as variáveis são resolvidas em tempo de execução. No entanto, é possível criar escopos estáticos usando closures (fechamentos), onde as variáveis são resolvidas em tempo de compilação.

```groovy
// Escopo estático
class Exemplo {
  static int variavelEstatica = 30

  void metodo1() {
    println("Variável estática dentro do método: $variavelEstatica")
  }
}

def objeto = new Exemplo()
objeto.metodo1()

//Escopo dinâmico
def metodo2() {
  def variavelLocal = 40
  println("Variável local dentro do método: $variavelLocal")
}

metodo2()
```

## 2.4. Tipos de dados

- **Tipos de dados suportados pela linguagem**

Groovy suporta uma variedade de tipos de dados, incluindo Tipos primitivos (boolean, byte, short, int, long, float, double), Tipos de objetos (String, List, Map, Set, Date, File) e tipos especiais (Closure, Range, Pattern).

**Tipos primitivos e tipos compostos:** Em Groovy são aqueles que armazenam valores simples e são passados por valor. Eles são ***boolean***, ***byte***, ***short***, ***int***, ***long***, ***float*** e ***double***. Os tipos compostos, por outro lado, são aqueles que armazenam referências a objetos e são passados por referência. Eles incluem ***String***, ***List***, ***Map***, ***Set***, etc.

```groovy
// Tipos primitivos
def booleano = true
def inteiro = 10
def flutuante = 3.14

// Tipos compostos
def texto = "Exemplo de texto"
```

**Tipos numéricos:** Os tipos numéricos em Groovy são byte, short, int, long, float e double. Eles diferem em sua capacidade de armazenar números e em sua precisão.

- **byte:** 8 bits, armazena números inteiros de -128 a 127.
- **short:** 16 bits, armazena números inteiros de -32,768 a 32,767.
- **int:** 32 bits, armazena números inteiros de -2,147,483,648 a 2,147,483,647.
- **long:** 64 bits, armazena números inteiros de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.
- **float:** 32 bits, armazena números de ponto flutuante com precisão simples.
- **double:** 64 bits, armazena números de ponto flutuante com precisão dupla.

```
def inteiro = 10
def flutuante = 3.14
def longo = 10000000000L // O sufixo 'L' indica que é do tipo long
```

**Tratamento das strings:** Em Groovy, ***String*** é um tipo de objeto, ou seja, é um tipo composto e o tamanho de uma ***string*** em Groovy é dinâmico e não há limite prédeterminado para o tamanho de uma ***string***.

```
def nome = "João"
def sobrenome = "Silva"
def nomeCompleto = nome + " " + sobrenome
println("Nome completo: ${nomeCompleto}")
```

## 2.5. Checagem de Tipos

Groovy é uma linguagem de tipagem dinâmica, o que significa que os tipos das variáveis são determinados em tempo de execução e não precisam ser declarados explicitamente. Isso proporciona flexibilidade e agilidade no desenvolvimento, mas também pode resultar em erros de tipo em tempo de execução se os tipos não forem tratados corretamente.

```
def x = 10 // Tipo de x é determinado em tempo de execução
x = "Hello" // Agora x é uma String
```

Groovy é considerada uma linguagem fracamente tipada, o que significa que as conversões entre tipos de dados são realizadas implicitamente sempre que possível, sem a necessidade de conversões explícitas por parte do programador. Isso pode levar a uma maior flexibilidade, mas também pode resultar em comportamentos inesperados se não forem tratados com cuidado.

```groovy
def x = 10 // x é do tipo Integer
def y = "20" // y é do tipo String

def resultado = x + y // Aqui ocorre uma concatenação de String
println(resultado) // Saída: 1020
```

## 2.6. Completude de Tipo

Em Groovy, os valores são tratados como objetos de primeira classe. Isso significa que os valores podem ser passados como argumentos para funções, atribuídos a variáveis, armazenados em estruturas de dados e retornados como resultados de funções, assim como os objetos tradicionais.

```groovy
def funcao = { x, y -> x + y }
def resultado = funcao(3, 5)
println(resultado) // Saída: 8
```

O Princípio da Completude de Tipo afirma que todos os valores têm um tipo estático conhecido em tempo de compilação. Em Groovy, devido à sua natureza dinâmica, os tipos de dados não são conhecidos em tempo de compilação, mas sim em tempo de execução. Portanto, Groovy viola o Princípio da Completude de Tipo.

```groovy
def x = 10 // Tipo de x é determinado em tempo de execução
x = "Hello" // Agora x é uma String
```

## 2.7. Expressões

**Literais:** Em Groovy, literais são representações diretas de valores de dados. Isso pode incluir números inteiros, números decimais, strings, booleanos, listas, mapas e muito mais.

```groovy
def inteiro = 42
def decimal = 3.14
def texto = "Olá, mundo!"
def booleano = true
def lista = [1, 2, 3, 4, 5]
def mapa = [nome: "João", idade: 30]
```

**Agregados:** Os agregados em Groovy referem-se a coleções de dados, como listas, conjuntos e mapas. Eles permitem que você agrupe e manipule dados de forma eficiente.

```groovy
def lista = [1, 2, 3, 4, 5]
def conjunto = [1, 2, 3, 4, 5] as Set
def mapa = [nome: "João", idade: 30
```

**Chamadas de Função:** Em Groovy, as chamadas de função são usadas para executar código encapsulado em funções ou métodos. Você pode chamar funções definidas pelo usuário, bem como funções integradas à linguagem ou funções de bibliotecas externas.

```groovy
def soma = { x, y -> x + y }
def resultado = soma(3, 5)
println(resultado) // Saída: 8
```

**Expressões Condicionais:** As expressões condicionais em Groovy são usadas para tomar decisões com base em condições lógicas. Isso inclui instruções if, else-if e else, bem como o operador ternário.

```groovy
def idade = 20
def status = (idade >= 18) ? "Maior de idade" : "Menor de idade"
println(status) // Saída: "Maior de idade
```

**Expressões Iterativas:** As expressões iterativas em Groovy são usadas para repetir blocos de código várias vezes. Isso inclui loops for, while e do-while.

```groovy
for (i in 1..5) {
    println(i)
}
```

**Acessos a Constantes e Variáveis:** Em Groovy, você pode acessar constantes e variáveis de várias formas, dependendo de seu escopo. Variáveis locais podem ser acessadas diretamente pelo nome, enquanto variáveis de instância e variáveis estáticas são acessadas usando o operador de ponto.

```groovy
class Exemplo {
    static final PI = 3.14
    def raio = 5

    def calcularArea() {
        return PI * raio * raio
    }
}

def exemplo = new Exemplo()
println(exemplo.calcularArea()) // Saída: 78.5
```

## 2.8. Sistemas de tipos

**Sobrecarga:** Groovy suporta sobrecarga de métodos, o que significa que é possível definir múltiplas versões de um método com o mesmo nome, mas com parâmetros diferentes. O método a ser invocado é determinado em tempo de execução com base nos tipos dos argumentos passados.

```groovy
class Exemplo {
    def somar(int a, int b) {
        return a + b
    }

    def somar(double a, double b) {
        return a + b
    }
}

def exemplo = new Exemplo()
println(exemplo.somar(2, 3)) // Saída: 5 (int)
println(exemplo.somar(2.5, 3.5)) // Saída: 6.0 (double)
```

**Polimorfismo Paramétrico:** Groovy suporta polimorfismo paramétrico, que é a capacidade de escrever código genérico que pode ser parametrizado com tipos específicos. Isso é comumente alcançado usando coleções genéricas, como List<T> e Map<K, V>.

```groovy
def listaInteiros = [1, 2, 3, 4, 5] // Lista de inteiros
def listaStrings = ["a", "b", "c"] // Lista de strings
```

**Polimorfismo por Inclusão:** O polimorfismo por inclusão em Groovy é semelhante ao encontrado em outras linguagens orientadas a objetos, onde uma classe derivada pode ser tratada como uma instância de sua classe base.

```groovy
class Animal {
    def fazerSom() {
        println("...")
    }
}

class Cachorro extends Animal {
    @Override
    def fazerSom() {
        println("Au Au!")
    }
}

class Gato extends Animal {
    @Override
    def fazerSom() {
        println("Miau!")
    }
}

Animal cachorro = new Cachorro()
Animal gato = new Gato()

cachorro.fazerSom() // Saída: Au Au!
gato.fazerSom() // Saída: Miau!
```

**Coerção:** Groovy suporta coerção, que é a capacidade de converter automaticamente um tipo de dado em outro, quando apropriado. Isso pode incluir a conversão de tipos numéricos, strings, etc.

```groovy
def inteiro = 10
def flutuante = 3.14

def resultado = inteiro + flutuante
println(resultado) // Saída: 13.14 (flutuante)
```

## 2.9. Armazenamento

### Variáveis:

```groovy
// Variáveis Temporárias
def x = 10

class Exemplo {
    // Variáveis Persistentes
    def y = 20
    static def z = 30
}
```

- **Variáveis Temporárias:** São variáveis que existem apenas durante a execução de um determinado escopo, como variáveis locais dentro de um método.

- **Variáveis Persistentes:** São variáveis que persistem além do escopo em que foram definidas, como variáveis de instância e variáveis estáticas em uma classe.
- **Storables:** Groovy não possui um conceito direto de "storables" como em outras linguagens. Variáveis em Groovy podem ser armazenadas em diferentes escopos, mas não existe um termo específico para variáveis que podem ser armazenadas em algum tipo de repositório persistente.
- **Variáveis Compostas:** São variáveis que armazenam mais de um valor, como listas, mapas, conjuntos, etc.
- Em Groovy, você pode atualizar variáveis tanto de forma total quanto seletiva, dependendo da necessidade. Você pode atribuir um novo valor à variável para uma atualização total ou modificar apenas partes da variável para uma atualização seletiva.

```groovy
// Atualização total
def x = 10
x = 20 // Atualização total de x para 20

// Atualização seletiva
def lista = [1, 2, 3]
lista[1] = 5 // Atualização seletiva do segundo elemento da lista para 5
```

## 2.10. Arrays

Em Groovy, os arrays são dinâmicos e flexíveis. Eles podem crescer ou diminuir de tamanho dinamicamente conforme necessário, o que oferece mais flexibilidade em comparação com arrays estáticos.

```groovy
def array = [1, 2, 3] // Array dinâmico
array << 4 // Adicionando um elemento ao array
```

Groovy segue a semântica de referência para objetos e a semântica de cópia para tipos primitivos. Isso significa que ao passar um objeto para uma função, uma referência ao objeto é passada (alterações dentro da função afetam o objeto original), enquanto que para tipos primitivos uma cópia do valor é passada (alterações dentro da função não afetam a variável original).

```groovy
def lista1 = [1, 2, 3]
def lista2 = lista1 // Referência à mesma lista

def x = 10
def y = x // Cópia do valor de x

lista2 << 4 // Modificando lista2
println(lista1) // Saída: [1, 2, 3, 4] - lista1 também é alterada
```

- **Comandos**

**Atribuição Simples:** Atribuição de valor a uma variável.

**Atribuição Múltipla:** Atribuição de múltiplos valores a múltiplas variáveis em uma única instrução.

**Atribuição Simultânea:** Atribuição de valores a múltiplas variáveis de forma simultânea.

```
// Atribuição Simples
def x = 10

// Atribuição Múltipla
def (y, z) = [20, 30]

// Atribuição Simultânea
def (a, b) = [40, 50], (c, d) = [60, 70]
```

**Chamada de Procedimento:** Uma chamada de procedimento é uma chamada de função ou método que realiza uma tarefa específica quando invocada.

```
def somar(a, b) {
    return a + b
}

def resultado = somar(3, 5)
println(resultado) // Saída: 8
```

**Comando Sequencial:** O comando sequencial executa uma sequência de comandos em ordem.

```
def x = 10
def y = 20
println(x)
println(y)
```

**Comando Colateral:** Um comando colateral é um comando que realiza uma ação secundária além de sua função principal.

```
def lista = [1, 2, 3, 4, 5]
println(lista) // Saída: [1, 2, 3, 4, 5]
```

**Comando Condicional:** O comando condicional executa um bloco de código se uma condição for verdadeira.

```groovy
def idade = 18
if (idade >= 18) {
    println("Maior de idade")
} else {
    println("Menor de idade")
}
```

**Comando Iterativo:** O comando iterativo executa um bloco de código repetidamente enquanto uma condição for verdadeira.

```groovy
def contador = 0
while (contador < 5) {
    println(contador)
    contador++
}
```

**Expressão de Comando:** Uma expressão de comando é uma expressão que causa um efeito colateral, como modificar o estado de uma variável.

```groovy
def x = 10
println(x) // Saída: 10
```

**Comando Bloco e Expressão Bloco:** Em Groovy, um bloco de comandos é um conjunto de comandos agrupados em chaves **{ }** e uma expressão bloco é uma expressão que retorna um valor, como uma função.

```groovy
// Bloco de Comandos
{
    def x = 10
    println(x)
}

// Expressão Bloco
def incrementar = { x -> x + 1 }
println(incrementar(5)) // Saída: 6
```

## 2.11. Abstrações

**Abstração de Função:** Refere-se à capacidade de definir uma função que recebe zero ou mais argumentos e retorna um valor.

**Abstração de Procedimento:** Refere-se à capacidade de definir um bloco de código que pode ser chamado e executado, mas não retorna um valor explicitamente.

```
// Abstração de Função
def soma(a, b) {
    return a + b
}

// Abstração de Procedimento
def imprimirMensagem() {
    println("Olá, mundo!")
}
```

A parametrização de abstrações permite que você generalize funções ou procedimentos para que possam ser reutilizados com diferentes conjuntos de dados ou em diferentes contextos.

- **Passagem de Parâmetro por Cópia:** O valor do parâmetro é copiado para a função ou procedimento, de modo que as alterações no parâmetro dentro da função ou procedimento não afetam a variável original.

```
// Definindo uma função que recebe um parâmetro por cópia
def incrementarPorCopia(int valor) {
    valor++
    println("Dentro da função: $valor")
}

// Chamando a função com um valor
def numero = 5
incrementarPorCopia(numero)
println("Fora da função: $numero") // Saída: Fora da função: 5
```

- **Passagem de Parâmetro por Referência:** Uma referência ao objeto é passada para a função ou procedimento, de modo que as alterações no parâmetro dentro da função ou procedimento afetam a variável original.

```
// Definindo uma função que recebe um parâmetro por referência
def incrementarPorReferencia(int[] valores) {
    valores[0]++
    println("Dentro da função: ${valores[0]}")
}

// Chamando a função com um array
def numeros = [5]
incrementarPorReferencia(numeros)
println("Fora da função: ${numeros[0]}") // Saída: Fora da função: 6
```

Unidades de Programa

- **Pacotes:** São unidades de organização de código em Groovy. Um pacote é um diretório que contém classes e outros recursos relacionados.
- **Tipos Abstratos de Dados (TAD):** Em Groovy, os tipos abstratos de dados são geralmente representados por classes, que encapsulam dados e comportamentos relacionados.
- **Objetos e Classes:** Em Groovy, objetos são instâncias de classes. As classes definem a estrutura e o comportamento dos objetos.

```groovy
class Pessoa {
    def nome
    def idade

    Pessoa(nome, idade) {
        this.nome = nome
        this.idade = idade
    }

    def saudacao() {
        println("Olá, meu nome é ${nome} e eu tenho ${idade} anos.")
    }
}

def pessoa = new Pessoa("João", 30)
pessoa.saudacao() // Saída: Olá, meu nome é João e eu tenho 30 anos.
```

Unidades genéricas referem-se à capacidade de criar abstrações que funcionam com tipos de dados genéricos. Isso permite que o mesmo código seja reutilizado com diferentes tipos de dados. Em Groovy, você pode usar generics para criar classes, interfaces e métodos genéricos.

```groovy
class Caixa<T> {
    T valor

    Caixa(T valor) {
        this.valor = valor
    }

    def getValor() {
        return valor
    }
}

def caixaString = new Caixa<String>("Olá")
println(caixaString.getValor()) // Saída: Olá

def caixaInteiro = new Caixa<Integer>(42)
println(caixaInteiro.getValor()) // Saída: 42
```

## 2.12. Sequenciadores

Os desvios incondicionais são usados para alterar o fluxo de execução do programa sem levar em consideração nenhuma condição.

- **Instrução break:** Utilizada para sair de um loop antes que a condição de término seja satisfeita.
- **Instrução continue:** Utilizada para interromper a iteração atual de um loop e continuar com a próxima iteração.

```
def numeros = [1, 2, 3, 4, 5]

// Instrução break
for (num in numeros) {
    if (num == 3) {
        break
    }
    println(num)
}
// Saída: 1
// Saída: 2

// Instrução continue
for (num in numeros) {
    if (num == 3) {
        continue
    }
    println(num)
}
// Saída: 1
// Saída: 2
// Saída: 4
// Saída: 5
```

Os escapes são utilizados para controlar o fluxo de execução do programa em situações específicas.

- **Instrução return:** Utilizada para sair de uma função ou método e retornar um valor opcional.
- **Instrução throw:** Utilizada para lançar uma exceção manualmente em um ponto específico do código.

```groovy
// Instrução return
def soma(a, b) {
    return a + b
}

def resultado = soma(3, 5)
println(resultado) // Saída: 8

// Instrução throw
def validarIdade(idade) {
    if (idade < 0) {
        throw new IllegalArgumentException("Idade não pode ser negativa")
    }
    println("Idade válida: $idade")
}

try {
    validarIdade(-5)
} catch (Exception ex) {
    println("Erro: ${ex.message}")
}
// Saída: Erro: Idade não pode ser negativa
```

As exceções são usadas para lidar com condições excepcionais que ocorrem durante a execução do programa. Você pode usar o bloco try-catch para capturar e tratar exceções.

```groovy
def dividir(a, b) {
    try {
        return a/b
    } catch (ArithmeticException ex) {
        println("Erro: Divisão por zero")
    }
}

def resultado = dividir(10, 0)
println(resultado)
```

## 3. Considerações finais

Aprendidas com o desenvolvimento do jogo e do relatório em geral: Uma das dificuldades encontradas pode ter sido o gerenciamento do estado do jogo, especialmente quando se trata de acompanhar a mão de cada jogador, a pilha de cartas, cartas jogadas, etc. Uma lição aprendida seria a importância de um design sólido de classes e uma estrutura de dados eficiente para lidar com esses aspectos.

Groovy geralmente é considerado uma linguagem muito legível devido à sua sintaxe concisa e expressiva. No entanto, a legibilidade pode ser comprometida em certos casos, especialmente se os recursos avançados da linguagem não forem usados com moderação.

Recursos como coerção dinâmica de tipos, sintaxe simplificada para listas e mapas, e a facilidade de interagir com classes Java tornam Groovy uma ótima escolha para desenvolvimento rápido de protótipos e projetos ágeis.

A natureza dinâmica de Groovy pode introduzir alguns desafios de confiabilidade, especialmente em projetos maiores onde a tipagem estática pode ser desejada para evitar erros comuns em tempo de execução.

Groovy é uma linguagem de código aberto e gratuita, o que significa que não há custo associado à sua utilização. No entanto, ao considerar o custo total de propriedade em um projeto de desenvolvimento de software, é importante levar em conta outros fatores, como treinamento da equipe, manutenção do código e suporte da comunidade.

## 4. Referências


**Groovy Languagem Documentation.** Disponível em:

<https://docs.groovy-lang.org/next/html/documentation/>

Acesso: 25 de Março, 2024

**Groovy in Action.** Disponível em:

<https://www.manning.com/books/groovy-in-action>

Acesso: 04 de Abril, 2024

**Linguagem de Programação Groovy: Introdução.** Disponível em:

<https://www.devmedia.com.br/linguagem-de-programacao-groovy-introducao/34099>

Acesso: 04 de Abril, 2024

**Groovy: O que é e para que serve?.** Disponível em:

<https://www.hostgator.com.br/blog/o-que-e-groovy/>

Acesso: 05 de Abril, 2024

**Estruturas básicas e expressões do Groovy.** Disponível em:

<https://training.nextflow.io/pt/basic_training/groovy/>

Acesso: 06 de Abril, 2024