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

JOÃO VICTOR DE ARAÚJO SILVA
RODRIGO LOIOLA DE FARIAS

ESPECIFICAÇÃO DO PROJETO

CAMPINA GRANDE
06 de Abril 2024

1. **Considerações Iniciais**
   1.1. Classificação (nível, geração, paradigmas): Groovy é uma linguagem de alto nível, e uma linguagem de terceira geração e multiparadigmas, significa que suporta vários estilos de programação, incluindo: Orientação a Objetos, programação Funcional, programação Imperativa e programação Declarativa
   1.2. Breve Histórico (incluindo comentários sobre outras linguagens que a influenciaram): Criada em 2003 por James Strachan e Bob McWhirter, Groovy é uma linguagem ágil e dinâmica para a plataforma Java com recursos baseados em linguagens com Python, Ruby e Smalltalk. Apesar de ainda não ter tanto suporte como outras linguagens mais usadas, cada vez mais Groovy aumenta em popularidade.
   1.3. Popularidade (em relação a rankings famosos: TIOBE, IEEE Spectrum, etc.):
   IEEE Spectrum
   TIOBE
   1.4. Uso de compilador? Interpretador? Ou ambos? Ambos, pode ser compilada pelo groovy ou interpretada como um script.
   1.5. Aplicações da linguagem (Para que ela é usada? Quem usa? etc): As aplicações da linguagem Groovy é Desenvolvimento Web, Scripting e Automação, Testes Automatizados, Desenvolvimento de Plugins e Extensões e Desenvolvimento de Jogos e Aplicativos Móveis. algumas empresas conhecidas que usam Groovy em suas aplicações incluem Netflix, LinkedIn, Sky, Alibaba, Canoo, Grails, e muitas outras. Groovy é uma escolha popular em empresas que operam em ambientes Java e precisam de uma linguagem mais expressiva e produtiva.

2. Conceitos

2.1. Nomes

2.1.1. A linguagem possui palavras-chave e palavras reservadas? Quais?

Possui sim. São elas: abstract, assert, break, case, catch, class, const, continue, def, default, do, else, enum, extends, final, finally, for, goto, if, implements, import, instanceof, interface, native, new, null, non-sealed, package, public, protected, private, return, static, strictfp, super, switch, synchronized, this, threadsafe, throw, throws, transient, try e while. Dessas, apenas const, goto, strictfp e threadsafe não estão em uso. Há também as palavras-chaves contextuais (que são reservadas apenas em alguns contextos): as, in, permits, record, sealed, trait, var e yields. Algumas outras são: true, false, boolean, char, byte, short, int, long, float e double.

2.2. Vinculações (Bindings)

2.2.1. Vinculação de tipos estática (explícita? implícita?) ou vinculação de tipos dinâmica? A linguagem é considerada dinamicamente tipada. Isso significa que as variáveis em Groovy não têm um tipo de dados fixo e podem ser associadas dinamicamente a diferentes tipos de dados durante a execução do programa.

2.2.2. Vinculações de Armazenamento e Tempo de Vida

2.2.2.1. Como as variáveis são armazenadas na memória? As variáveis em Groovy são armazenadas na memória de forma dinâmica, conforme necessário durante a execução do programa.

```groovy
def x = 10  // Variável x armazenada na memória
```

2.2.2.2. Que tipo de variável fica em que tipo de seção da memória? Em Groovy, não há uma divisão explícita de seções de memória para diferentes tipos de variáveis, como em linguagens de baixo nível. No entanto, as variáveis locais geralmente são armazenadas na pilha de execução. Isso inclui variáveis declaradas dentro de métodos ou blocos de código, que são descartadas quando o método ou bloco termina de executar.

2.3. Escopo

```groovy
// Variável global
def globalVar = "Sou global"

def someMethod() {
    // Variável local
    def localVar = "Sou local"
    println(globalVar)  // Acesso a variável global dentro do método
}

someMethod()
```

2.3.1. Variáveis globais? Variáveis locais? Groovy permite a declaração de variáveis globais, que podem ser acessadas de qualquer lugar no código. As variáveis locais têm escopo limitado ao bloco onde são declaradas.

2.3.2. Escopo estático? Escopo dinâmico? Groovy geralmente segue um escopo dinâmico. Isso significa que a resolução de nomes de variáveis e métodos é feita em tempo de execução. O escopo estático em Groovy pode ser alcançado usando blocos de código estático ou métodos

estáticos, onde as variáveis ou métodos estão associados à classe em vez de instâncias de objetos.

2.4. Tipos de dados

2.4.1. Quais os tipos de dados suportados pela linguagem?

```groovy
def number = 10          // Inteiro
def floating = 10.5      // Ponto flutuante
def string = "Groovy"    // String
def bool = true          // Booleano
```

2.4.1.1. Quais são tipos primitivos e quais são tipos compostos? Groovy usa os mesmos tipos primitivos do Java, que são: byte, boolean, char, short, int, long, float e double.

2.4.1.2. Quais são e como se diferenciam os tipos numéricos? Os tipos numéricos são: byte com 8 bits, short com 16 bits, int com 32 bits, long com 64 bits, float e double

2.4.1.3. Como a linguagem trata strings? Em Groovy, as strings são tratadas como objetos.

2.4.1.3.1. Tipo primitivo? Tipo composto? Do tipo composto.

2.4.1.3.2. Tamanho estático? Tamanho dinâmico (limitado)? Tamanho dinâmico e não são limitados

2.5. Checagem de Tipos

2.5.1. Tipagem estática ou dinâmica? Tipagem dinâmica, no groovy para declarar uma variável só é necessário identificar com def e o groovy se encarrega de associar com o tipo mais próximo.

2.5.2. Fracamente ou fortemente tipada? Fracamente Tipada

2.6. Completude de Tipo

2.6.1. A linguagem adota valores de primeira classe? E valores de segunda classe? Ele só trata valores de primeira classe de forma adequada. Isso significa que os valores podem ser passados como argumentos, atribuídos a variáveis e armazenados em estruturas de dados de forma semelhante a outros tipos de dados.

2.6.2. A linguagem viola o Princípio da Completude de Tipo? O Princípio da Completude de Tipo (ou Princípio de Substituição de Liskov) estabelece que objetos de um tipo específico podem ser substituídos por instâncias de seus subtipos sem afetar a corretude do programa. Groovy, como uma linguagem dinâmica, não faz uma verificação estrita de tipos em tempo de compilação, o que significa que ela pode violar esse princípio em certas situações.

2.7. Expressões

```groovy
// Expressão condicional
def age = 20
def status = age >= 18 ? "Adulto" : "Menor"

// Expressões iterativas
def list = [1, 2, 3, 4, 5]
list.each { println(it) }  // Iteração sobre uma lista
```

2.7.1. Literais? O Groovy suporta uma variedade de literais, incluindo literais para números (inteiros e de ponto flutuante), strings, booleanos, listas, mapas, conjuntos e expressões regulares.

2.7.2. Agregados? O Groovy permite a criação e manipulação de agregados, como listas, mapas e conjuntos, usando literais e operadores específicos.

2.7.3. Chamadas de Função? Em Groovy suporta chamadas de função, onde você pode chamar funções definidas pelo usuário ou funções embutidas na linguagem. Além disso, Groovy permite o uso de sintaxe de closure para definir funções de forma mais concisa.

2.7.4. Expressões condicionais? Groovy suporta expressões condicionais, como if, else, switch e operador ternário “? :”.

2.7.5. Expressões iterativas? Groovy oferece suporte a expressões iterativas, como for, while e each. Além disso, Groovy fornece métodos de alta ordem, como findAll, collect, each, entre outros, para trabalhar com coleções de forma mais expressiva.

2.7.6. Acessos a constantes e variáveis? Groovy permite o acesso a constantes e variáveis definidas no escopo atual ou em escopos externos. Você pode acessar variáveis locais, variáveis de instância, variáveis estáticas, bem como constantes.

2.8. Sistema de tipos

2.8.1. Sobrecarga? Groovy suporta sobrecarga de métodos, o que significa que você pode ter múltiplas definições de um método com o mesmo nome.

2.8.2. Polimorfismo Paramétrico? Groovy não oferece suporte nativo ao polimorfismo paramétrico

2.8.3. Polimorfismo por Inclusão? Groovy suporta polimorfismo por inclusão, o que significa que você pode usar herança e interfaces para criar hierarquias de classes

2.8.4. Coerção? Coerção em Groovy refere-se à capacidade de converter automaticamente um tipo de dados em outro quando necessário, de forma implícita.

2.9. Armazenamento

2.9.1. Variáveis

```groovy
// Arrays
def numbers = [1, 2, 3, 4, 5]  // Array dinâmico
println(numbers[2])  // Acesso a elementos do array

// Semântica de referência
def array1 = [1, 2, 3]
def array2 = array1  // array2 referencia o mesmo objeto que array1
array1[0] = 5
println(array2[0])  // Saída será 5, pois array2 aponta para o mesmo objeto que array1
```

2.9.1.1. Variáveis Temporárias: Variáveis temporárias em Groovy são aquelas que são usadas para armazenar valores durante a execução do programa, mas não são persistentes. Elas são geralmente definidas dentro de métodos ou blocos de código e existem apenas enquanto o escopo em que foram definidas estiver ativo. Quando o escopo termina, as variáveis temporárias são destruídas e a memória associada a elas é liberada.

2.9.1.2. Variáveis Persistentes: Em Groovy, as variáveis persistentes podem ser criadas usando diversas técnicas de persistência de dados, como bancos de dados, arquivos, ou serviços de

armazenamento na nuvem. Estas variáveis mantêm seus valores mesmo após o encerramento do programa e são acessíveis em execuções futuras.

2.9.1.3. Storables: Não há um conceito nativo de "Storables" em Groovy. No entanto, você pode implementar persistência de objetos utilizando serialização ou outras técnicas de persistência de dados disponíveis na linguagem.

2.9.1.4. Variáveis Compostas

2.9.1.4.1. Atualização de variáveis

2.9.1.4.1.1. Atualização total? Atualização seletiva? Ele utiliza a total. Uma atualização total de uma variável substitui completamente seu valor anterior pelo novo valor atribuído.

2.9.1.4.2. Arrays

2.9.1.4.2.1. Estático? Dinâmico? Flexível? Em Groovy, os arrays são dinâmicos e flexíveis. Eles podem aumentar ou diminuir de tamanho dinamicamente conforme necessário.

2.9.1.5. Semântica de Cópia e Semântica de Referência? Em Groovy, os arrays têm semântica de referência.

2.9.2. Comandos

2.9.2.1. Skip: Em Groovy, não há um comando específico chamado "Skip", mas você pode usar a instrução "continue" dentro de loops para alcançar um comportamento semelhante.

```groovy
for (int i = 0; i < 5; i++) {
    if (i == 2) {
        continue; // Pula para a próxima iteração quando i for igual a 2
    }
    println(i);
}

```

2.9.2.2. Atribuição? Para atribuição de valores a variáveis usa-se o operador de atribuição ‘=’.

```groovy

    def x = 'raiz'

```

2.9.2.3. Chamada de procedimento? Em Groovy, chamada de procedimento é uma chamada de função/método.

```groovy
println("Hello, World!")

```

2.9.2.4. Comando sequencial? Como são apenas uma sequência de instruções executadas em ordem, em Groovy há comando sequencial.

```groovy
def nome = "Klaudio"
println(nome + ", bote um 10 pra gente, prfv :)")

```

2.9.2.5. Comando colateral? Há comando colateral, por exemplo, na imagem anterior o comando ‘println’ produzirá uma saída no console.

2.9.2.6. Comando condicional? É possível utilizar ‘if’, ‘else if’ e ‘else’.

```
def nota = 10
if (nota >= 9) {
    println("Melhor professor!!")
} else if (nota >=7){
    println("Ainda estamos na média!")
} else {
    println("Pra que esse odio no coração?")
}

```

2.9.2.7. Comando interativo? Comando interativo é o que interage diretamente com o usuário, isso é possível a partir da entrada de dados que pode ser feita com o comando 'System.in.read()'

2.9.2.8. Expressão de Comando (expressões com efeitos colaterais)? Há suporte para expressão de comando.

2.9.2.9. Comando bloco e Expressão bloco? Os blocos de código são delimitados por chaves '{}' e podem conter várias instruções.

```
def result = {
    def x = 5
    def y = 10
    x + y
}()
println(result)
```

2.10. Abstrações

2.10.1. Abstração de função? Abstração de procedimento? Groovy suporta abstração de função através da definição de métodos. Assim como no Java, Groovy não faz uma distinção explícita entre procedimentos e funções, no entanto, os procedimentos podem ser simulados através de métodos sem retorno de um valor específico.

2.10.2. Parametrização de abstrações

2.10.2.1. Passagem de parâmetro por cópia? (valor, resultado, valor/resultado?) Não se aplica ao Groovy, pois mesmo para tipos primitivos a passagem de parâmetro é por referência.

2.10.2.2. Passagem de parâmetro por referência? Qualquer modificação feita no objeto dentro da função afetará o objeto fora da função, pois toda passagem é por referência.

2.10.3. Unidades de Programa

2.10.3.1. Pacotes? Groovy suporta a organização de código em pacotes.

2.10.3.2. Tipos Abstratos de Dados? Groovy permite a definição de tipos abstratos de dados por meio de classes e interfaces.

2.10.3.3. Objetos e Classes? Groovy é uma linguagem orientada a objetos e com isso, suporta classes e suas instâncias, objetos.

2.10.4. Unidades Genéricas? Há suporte a genéricos, permitindo que um código possa ser escrito utilizando diferentes tipos de dados.

2.11. Sequenciadores

2.11.1. Desvios Incondicionais? Há suporte para desvios incondicionais como 'break' e 'continue', também é possível usar return para sair de métodos e funções.

2.11.2. Escapes? É possível utilizar escapes, os mesmos da linguagem Java, como o '\n' para pular uma linha, por exemplo.

2.11.3. Exceções? Para usar exceções no Groovy, utilizam-se os blocos ‘try’, ‘catch’ e ‘finally’. O ‘try’ é usado quando se pretende lançar uma exceção, ‘catch’ é usado para capturar e tratar exceções e ‘finally’ usado para executar um código sem importar se uma exceção será lançada ou não. Também há o ‘throw’ para lançar exceções personalizadas.

3. Considerações Finais

3.1. Dificuldades e Lições 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.

```
// Exemplo de gerenciamento de estado de jogo
class Game {
    def players = []
    def stack = []
    def playedCards = []

    // Outros métodos e atributos
}

```

3.2. Análise da linguagem com base em critérios de avaliação

```
// Exemplo de legibilidade e capacidade de escrita
def sum(a, b) {
    a + b
}

def result = sum(5, 10)
println(result)  // Saída: 15

```

3.2.1. Legibilidade: 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.

3.2.2. Capacidade de Escrita: 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.

3.2.3. Confiabilidade: 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.

3.2.4. Custo: 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

4.1. https://groovy-lang.org/single-page-documentation.html