Comparativa sintáctica entre los lenguajes de programación java y groovy

Comparação sintática entre linguagens de programação Java e groovy

Rosa Isela Zarco Maldonado
Universidad Autónoma del Estado de México, México
zamrtct@hotmail.com

Joel Ayala de la Vega
Universidad Autónoma del Estado de México, México
joelayala2001@yahoo.com.mx

Oziel Lugo Espinosa
Universidad Autónoma del Estado de México, México
ozieluz@hotmail.com

Alfonso Zarco Hidalgo
Universidad Autónoma del Estado de México, México
azarcox@hotmail.com

Hipólito Gómez Ayala
Universidad Autónoma del Estado de México, México
escribeme88@gmail.com

Resumen

Uno de los lenguajes que lleva varios años de vida y que permanece como uno de los más importantes debido a sus diversas características que permiten la creación de aplicaciones de software, es el lenguaje Java. Java es un lenguaje que permite el desarrollo para aplicaciones de dispositivos móviles, de escritorio, corporativas y de igual manera para el entorno web. Por otro lado, el área de desarrollo de lenguajes de programación se mantiene en un gran dinamismo. En el 2003 aparece el lenguaje de programación Groovy, este lenguaje conserva una sintaxis familiar a Java pero con características particulares. Tanto Groovy como Java son Lenguajes Orientados a Objetos y se ejecutan sobre una Máquina Virtual. La intención de éste escrito es realizar una comparativa sintáctica de los lenguajes Java y Groovy para observar las particularidades de cada uno, y de esta manera, facilitar a los programadores la implementación de sus proyectos

Palabras clave: Programación Orientada a Objetos, modularidad, métodos, polimorfismo, encapsulamiento, jerarquía, tipificación, concurrencia, persistencia.

Resumo

Uma das línguas que tem vários anos de vida e continua sendo um dos mais importantes devido às suas várias características que permitem a criação de aplicações de software é a linguagem Java. Java é uma linguagem que permite o desenvolvimento de aplicações para dispositivos móveis, desktops corporativos e também para o ambiente web. Além disso, o campo de desenvolvimento de linguagens de programação continua muito forte. Em 2003, a linguagem de programação Groovy aparece nesta língua mantém uma sintaxe familiar para Java, mas com características particulares. Groovy e Java são ambos linguagens orientadas a objeto e executado em uma máquina virtual. A intenção deste trabalho é realizar uma sintaxe comparativa de Java e linguagens Groovy para observar as particularidades de cada um, e, assim, tornar mais fácil para os programadores para implementar seus projetos

Palavras-chave: OOP, modulares, métodos, polimorfismo, encapsulamento, de hierarquia, de digitação, de concorrência, de persistência.

Fecha recepción: Enero 2015 **Fecha aceptación:** Julio 2015

Introdução

As linguagens de programação já existem há várias décadas. Cada nova linguagem fornece novas ferramentas para a vida diária. Uma das línguas que leva anos de vida e conseguiu manter-se uma das mais importantes é a linguagem Java, Java foi desenvolvida pela Sun Microsystems em 1991, com a idéia principal de criar uma linguagem para unificar equipamentos eletrônicos para o consumo doméstico. Na primeira, foi chamado Oak, mas em 1995 mudou seu nome para Java e de 2009, é de propriedade da Oracle Corporation.

Por outro lado você tem que Groovy, criado por James Strachan e Bob McWhirter em 2003. Groovy é baseado em Smalltalk, Python e linguagens Ruby, mas também disse que ele é baseado em Perl. Groovy mantém uma sintaxe familiar para Java, tornando-se assim que os programadores Java trabalhando é fácil para eles para se familiarizar com Groovy.

Uma vez que estas duas línguas são consideradas Orientada a Objetos para realizar benchmark foi selecionada com base na definição de Grady Booch sobre o conceito de Programação Orientada a Objetos:

> *Programação orientada a objetos é um modelo de implantação em que os programas são organizados como cooperativas coleções de objetos, cada um dos quais representa uma instância de alguma classe, e cujas aulas são todos os membros de uma hierarquia de classes em conjunto por relações de herança*

As linguagens orientadas a objeto deve atender quatro elementos-chave deste modelo:

- Abstração
- Encapsulation
- A modularidade
- Hierarquia

Dizer fundamentais, dizer que uma linguagem sem qualquer um destes elementos não é orientada a objetos.

Há três filhos do modelo orientado a objetos:

- Tipos (digitação)
- Concorrência
- Persistência

No lado isso significa que cada um é uma peça útil na linguagem orientada a objetos, mas não é essencial (Booch, 1991).

Pela extensão da linguagem Java, apenas a esses pontos foram escolhidos para comparar, portanto, ao longo deste artigo tentará fazer uma comparação de línguas no âmbito do regime acima referido, incluindo, naturalmente, a análise dos operadores controle e uso da máquina virtual

COMPARAÇÃO

A. MODULARIDADE

Uma classe vem para representar a definição de um módulo de programa, o qual, por sua vez define atributos e métodos comuns.

Um objeto é uma instância da classe, esta instância contém atributos e métodos com valores de dados específicos. (Prieto & Ramos Rodríguez Echeverría, 2004).

Um método é escrito dentro de uma classe e determina como você deve agir quando o objeto recebe a mensagem associada a esse método. Por sua vez, um método pode enviar mensagens para outros objetos que requerem ação ou informações. Os atributos definidos na classe permitem armazenar informação para o objecto. (Ceballos, 2010).

Quando o pensamento é modelado objetos, você deve ter as características e propriedades de um corpo real, e trazê-lo para um objeto. O termo refere-se à ênfase no "o que faz?" ao invés do "como?" (Recurso caixa preta) (Di Serio, 2011).

A modularidade em Java e Groovy logicamente organizados em classes e pacotes e aptidão através de arquivos.

Classes:

• Encapsular os atributos e os métodos de um tipo de objeto em um compartimento.

• Hiding por especificadores de acesso, elementos internos não destinadas a publicação no exterior.

Pacotes:

- As unidades são agrupamento lógico de aulas.
    - ou classes públicas são parte da interface do pacote e são visíveis do lado de fora.
    - ou classes que não são públicos só são visíveis dentro da própria embalagem.

Registros:

- Entre os arquivos podem residir várias classes com certas restrições que serão vistos mais tarde. (TutorialesNET, 2014)

O módulo de gestão de estrutura permite um melhor programa reduz a complexidade da mesma, cria uma série de limites bem definidos, o que aumenta a sua compreensão.

Java é desejável que cada classe é colocado em um arquivo. O nome do arquivo é o nome da classe, se você tem mais de uma classe em um arquivo, o nome do arquivo será a classe que tem o principal método.

Pode haver apenas uma classe pública por arquivo, o nome deve coincidir com a classe pública. Pode haver mais de uma classe padrão no mesmo arquivo.

Os nomes de classe devem ser substantivos, em caso misturado com a primeira letra de cada palavra interna capitalizados. (Oracle, 2014).

Esse mecanismo é chamado de nome CamelCase (por exemplo: ClassName, CuentaUsuario, fatura).

Groovy, em comparação com Java, permite scripts. Scripts são programas, normalmente pequenas ou simples, geralmente para executar tarefas muito específicas. Eles são geralmente um conjunto de instruções armazenadas em um arquivo de texto que deve ser interpretado linha por linha e em tempo real para a execução; Ele distingue este programa (compilado), uma vez que estes devem ser convertidos em um arquivo binário executável (por exemplo executável .exe, etc.) para executá-los. (Alegsa)

O que é o mesmo, não há necessidade de colocar todo o código em uma classe Groovy. Assim, em Groovy, se o código para implementar é algumas linhas, você pode executar um script simples, criando a classe vir de acordo com o tamanho do programa no momento em que são necessários a maioria das variáveis, instâncias, etc. ., que é quando a realização de uma classe (Kousen, 2014, p. 19) é necessária.

Duas diferenças adicionais observadas no groovy do tratado em sintaxe Java:

- *As vírgulas são opcionais.*
- *Parêntesis são frequentemente opcional. Não é ruim se você quiser incluir.*

Para Groovy nomenclatura depende de algumas características com base em classes de gerenciamento, ou ambos os scripts em um único arquivo. Em seguida, a fundação é realizar esta parte:

Arquivo de relação de classe Groovy

A relação entre arquivos e declarações de classe não é tão fixo como em Java. Groovy arquivos podem conter qualquer número de classes de declarações públicas de acordo com as seguintes regras:

- Se um arquivo Groovy contém a declaração da classe, ele lida como um script; ou seja, que envolve de forma transparente em uma classe de tipo Script. Esta classe gerado

automaticamente tem o mesmo nome como o nome do arquivo de origem do roteiro (sem a extensão). O conteúdo do arquivo é envolto em um método de execução, e um método principal adicional é facilmente construído a partir do script.

- Se um arquivo contém exatamente uma declaração de classe Groovy com o mesmo nome do arquivo (sem a extensão), então é a mesma proporção de um-para-um em Java.
- Um arquivo Groovy pode conter várias instruções de qualquer tipo de visibilidade, e não há nenhuma regra imposta que qualquer um deles deve coincidir com o nome do arquivo. O compilador alegremente groovyc cria arquivos .class para todas as classes declaradas no arquivo.
- Um arquivo pode misturar declarações de classe Groovy e código de script. Neste caso, o código de script irá tornar-se a classe principal a ser executado (König & Glover, 2007, pp. 188-189).
- Se o arquivo contém uma classe com um script, e classe corresponde exatamente ao nome do arquivo, o compilador Groovy educadamente sugere renomear qualquer script ou o nome da classe, porque ele não pode gerar todos os arquivos classes necessárias.

Como regra geral, recomenda-se usar apenas essa codificação de modo misto quando você está escrevendo roteiros separados. Groovy para escrever código que é integrado em uma aplicação maior, é melhor ficar com o caminho Java de fazer as coisas, de modo que os nomes de arquivo fonte correspondente aos nomes das classes que são implementados como parte do seu aplicativo. (Dearle, 2010, pp. 34-35). Um ponto importante é que, como Java, o nome do arquivo é necessário para iniciar com uma letra maiúscula.

A. MÉTODOS

Classes consistem em variáveis de instância e métodos. O conceito é muito amplo, porque método Java lhes dá grande poder e flexibilidade.

Em Java os únicos elementos necessários de uma declaração de método são o tipo do método de retorno, nome, um par de parênteses "()", e um corpo entre chaves "{}".

De modo mais geral, as declarações método tem os seguintes componentes:

• Os modificadores - públicas, privadas e outras.

• O tipo de retorno - tipo do valor retornado pelo método de dados, ou nulo se o método não retorna um valor.

• O nome do método.

• Lista de Parâmetros entre parênteses uma lista delimitada por vírgulas de parâmetros de entrada, precedida por seus tipos de dados, entre parênteses (). Se não houver parâmetros, você deve usar parênteses vazios.

• O corpo do método, fechado em código-chaves método, incluindo a declaração de variáveis locais.

• modificadores, tipos de retorno e parâmetros. (Oracle)

Em relação às habituais modificadores Groovy Java pode ser usado; declarar um tipo de retorno é opcional; e se não modificadores tipo de retorno ou fornecida, a palavra-chave def preenche o buraco. Quando def palavra-chave é usada, o tipo de retorno é considerado sem tipo. Neste caso, debaixo dos lençóis, o tipo de retorno é java.lang.Object. Visibilidade padrão é métodos públicos. (König & Glover, 2007, p. 180)

Você deve usar a palavra-chave def para definir um método que tem um tipo de retorno de dinâmica.

Em troca Groovy palavra não é usada como faz Java para retornar um valor de referência ou objeto. Groovy por padrão sempre retorna a última linha do corpo, por isso não é necessário especificá-la explicitamente como em Java. Você pode explicitamente lidar sem problemas, mas Groovy dá conforto omiti-lo.

Método Java principal é o seguinte:

public static void main(String args[]) { }

Nesta linha começa a execução do programa. Todos os programas Java começar a correr com a chamada ao método main ().

O intérprete ou máquina virtual Java chama main () antes de qualquer objeto é criado. O vazio de palavras-chave simplesmente diz ao compilador que main () não retorna nenhum valor. (Schildt, 2009, p. 23)

No método principal () há apenas um parâmetro, embora complicado. Args String [] declara um parâmetro denominado args, que é uma matriz de instâncias da classe String (os arranjos são coleções de objetos semelhantes). String objetos armazenados cordas. Neste caso, args recebe os argumentos que estão presentes na linha de comando, quando o programa é executado.

O último caractere da linha é {. Este personagem sinaliza o início do main () corpo do método. Todos incluídos num código método deve situar-se entre a chave de abertura do método e a torneira de passagem correspondente. O método main () é simplesmente um ponto de partida para o programa. Um programa complexo pode ter um monte de aulas, mas é necessário apenas que a pessoa tem o método main () para iniciar o programa. (Schildt, 2009, 24 p.)

Groovy manipula o principal método de uma forma mais simples. Sua forma é o seguinte:

static main (args){ }

O principal método tem alguns toques interessantes. Em primeiro lugar, o modificador pública pode ser omitido, uma vez que o valor predeterminado é. Em segundo lugar, args geralmente tem que ser do tipo String [], a fim de tornar o principal método para iniciar a execução da classe. Graças à entrega de Groovy, trabalhar de qualquer maneira, mas agora é args tipo java.lang.Object implicitamente estática. Em terceiro lugar, porque os tipos de retorno não utilizado para o transporte, você pode omitir a declaração vazio. (König & Glover, 2007, p. 180)

Definições

Considerando Figura 1, podemos ver que Groovy não precisa declarar o tipo de dados com a variável de obter, enquanto que em Java devem ser definidos de forma estrita.

código Groovy:

```
def comer(alimento){
    "le gusta comer ${alimento}"
}
```

Figura 1. Parâmetro nenhuma variável

A Figura 2 mostra como a Java:

```
public String comer(String alimento){
    return "le gusta comer" + alimento;
}
```

Figura 2. Código Java con tipo de parámetro necesario

No código Groovy só é necessário para declarar a variável de obter. Não esquecendo a enfatizar a dinâmica, no exemplo pode executar tanto um inteiro como uma seqüência sem problemas.

A. Polimorfismo

O polimorfismo é a forma de invocar uma acção ter comportamentos diferentes dependendo do contexto em que é usada. Mais simples, "polimorfismo é a possibilidade de que um método (função) podem ter um comportamento diferente dos parâmetros que são enviados (sobrecarga), ou a partir do meio que é invocado (na escrita)". (Hdeleon, 2014)

Qualquer objeto que passa o IS-Um teste pode ser polimórfico.

Sobrecarga de método

É uma característica que torna os programas mais legíveis. Ele é re-declarar um método já declarados com um número diferente e / ou tipo de parâmetros. Um método sobrecarregado não pode apenas diferem no tipo de resultado, mas deve também diferem no tipo e / ou o número de parâmetros formais. (Ceballos, 2010)

Builders

Para criar um objeto a palavra-chave "novo" é necessário, por exemplo:

variable = new *nombre_de_clase();*

É mais evidente a necessidade de parênteses após o nome da classe. O que realmente acontece é que ele está chamando o construtor da classe.

Um construtor é usado para criar um objeto que é uma instância de uma classe. Normalmente, ele realizou o necessário para inicializar o objeto antes que os métodos são chamados ou acessado campos de operações. Os construtores não herdado. (Oracle)

O construtor tem o mesmo nome da classe em que reside e sintaticamente similar a um método. Uma vez definido, o construtor é chamado automaticamente depois do objecto é criada e antes do final do novo operador. Os construtores são um pouco diferentes com métodos convencionais, porque eles não retornam qualquer tipo, nem mesmo vazio.

Quando não definir explicitamente um construtor de classe, Java cria um construtor de classe padrão.

Para as classes simples, basta usar o construtor padrão, mas não para as classes mais sofisticados. Uma vez definido o construtor de si mesmo, o construtor padrão não é mais usado. (Schildt, 2009, pp. 117-119). O precedente é referenciado para Java e Groovy.

A declaração de um construtor construtor padrão diferente, exige ser atribuído o mesmo identificador como a classe e não indica explicitamente um tipo de valor de retorno. Seja ou não parâmetro é opcional. Por outro lado, a sobrecarga permite que vários construtores que podem ser designados (com o mesmo identificador como a classe), desde que disponha de um tipo e / ou número de vários parâmetros. (Garcia & Beltran Arranz)

Em groovy gosta de métodos, o construtor padrão é público. Você pode chamar o construtor de três maneiras diferentes: como Java normal (uso normal construtor Figura 3), o tipo de restrição imposto usando a palavra-chave como, e tipo de restrição implícita.

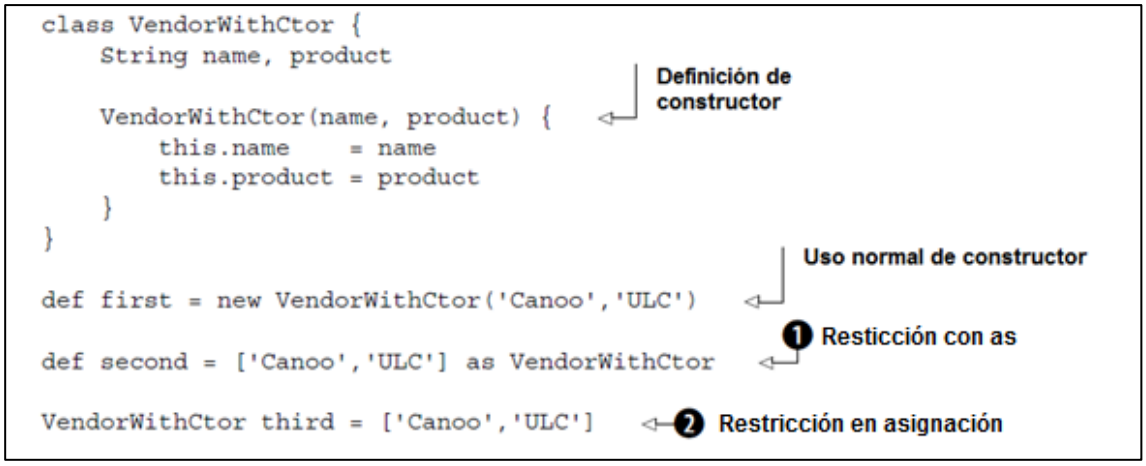

Figura 3. construtores chamada com parâmetros posicionais

A restrição nos números 1 e 2 pode ser impressionante. Quando Groovy vê a necessidade de restringir a lista de algum outro tipo, tente entrar em contato com o fabricante do tipo com todos os argumentos apresentados pela lista, pela ordem indicada. Essa necessidade pode ser imposta para restringir a palavra-chave ou pode surgir como atribuições a tipagem estática referências.

AS: Alterar o tipo de objeto.

Parâmetros nomeados são construtores úteis. Um caso de uso que surge frequentemente é a criação de classes imutáveis que têm alguns parâmetros que são opcionais. Usando parâmetros de posição rapidamente tornar-se complicada porque eles têm que ter construtores que permitem todas as combinações de parâmetros opcionais.

Por exemplo, se dois construtores são desejados argumento tipo string, eles poderiam não tem construtor com um único argumento, porque não distingue se a colocar o nome do produto ou atributo (ambos são seqüências). Um argumento adicional para a distinção seria procurado, ou

deve escrever fortemente parâmetros. Para evitar isso, o Groovy é baseada em parâmetros nomeados. A Figura 4 mostra como usar parâmetros nomeados com uma versão simplificada da classe de fornecedor. Baseia-se o construtor padrão implícito. (König & Glover, 2007, págs. 186-187)

```
class Vendor {
    String name, product
}

new Vendor()
new Vendor(name:    'Canoo')
new Vendor(product:'ULC')
new Vendor(name:    'Canoo', product:'ULC')

def vendor = new Vendor(name: 'Canoo')
assert 'Canoo' == vendor.name
```

Figura 4. Parâmetros nomeados

O exemplo na Figura 4 mostra como os parâmetros nomeados são flexíveis para os seus construtores. (König & Glover, 2007, pp. 186-187)

Da mesma forma faz para um mapa para o construtor de um feijão que contém os nomes das propriedades, juntamente com um valor de inicialização associado:

```
map = [id: 1, name: "Barney, Rubble"]
customer1 = new Customer( map )
customer2 = new Customer( id: 2, name: "Fred, Flintstone")
```

Figura 5. Pasar un mapa a un constructor

Passando "mapa" diretamente para o construtor do cliente, é permitido omitir "mapa" de parênteses, como mostrado na inicialização customer2.

cada GroovyBean[1] com essa conta pelo construtor padrão Mapa construído. Esse construtor funciona repetindo o objeto mapa e chama a propriedade correspondente *setter*[2] para cada entrada no mapa. Qualquer entrada de mapa que não corresponde a uma propriedade real feijão fará com que uma exceção seja lançada. (Dearle, 2010, p. 41).

[1] Un **Bean** es un componente software que tiene la particularidad de ser reutilizable y así evitar la tediosa tarea de programar los distintos componentes uno a uno.
[2] Los getters y setters son métodos de acceso lo que indica que son siempre declarados públicos. Los setters nos sirve para asignar un valor inicial a un atributo, pero de forma explícita, además el Setter nunca retorna nada. Los getters nos sirve para obtener (recuperar o acceder) el valor ya asignado a un atributo y utilizarlo para cierto método

Groovy para declarar dois construtores com o mesmo tipo de argumentos, e apoio através dos parâmetros são nomeados para não causar erros. Além disso, você pode dizer quando a sua definição é possível omitir o construtor.

Em Java, um JavaBean é uma classe que implementa métodos getters e setters para todos ou alguns dos seus campos de instância. Groovy gera automaticamente getters e setters campos de instância em uma classe e ter a visibilidade pública padrão. Ele também gera o construtor padrão. Instância campos com getters e setters gerados automaticamente são conhecidos em Groovy como propriedade, e refere-se a essas classes como GroovyBeans, ou POGO coloquial (Plain Old Groovy Object).

Os fechamentos são fragmentos de código anônimos que podem ser atribuídos a uma variável e têm características que os fazem parecer como um método para a medida em que permite passar parâmetros para eles, e eles podem retornar um valor. No entanto, ao contrário dos métodos, os encerramentos são anônimos, não são coladas a um objeto. Um fechamento é apenas um trecho de código que pode ser atribuído a uma variável e executar mais tarde. (Dearle, 2010)

Não precisa ser declarada dentro de aulas, eles podem ser declaradas em qualquer lugar.

Um código Groovy encerramento está envolvido como um objeto do tipo groovy.lang.Closure, definido e reconhecido por chaves {// código aqui}. Um fecho é muito semelhante aos métodos Java, manuseamento parâmetro pode ser feita por tipagem dinâmica. Esta característica do Groovy é uma das línguas mais importantes, o que faz a diferença em termos de funcionalidade em relação a Java. Um fecho pode ser transmitido como um parâmetro para um outro fecho, opção Java que permite trabalhar com os seus métodos, deve notar-se que, quando uma cápsula é transmitido como um parâmetro dentro dos parênteses, deve ser declarado que o último argumento.

Você pode determinar se o fecho foi fornecida. Caso contrário, você pode decidir usar uma implementação padrão para lidar com o caso. (Subramaniam, 2013)

```
3  class CDinamico {
4      static main(args){
5          doSomeThing() { println "Use specialized implementation" }
6          doSomeThing({ println "Use specialized implementation" })
7          doSomeThing()
8      }
9      def static doSomeThing(closure) {
10         if (closure) {
11             closure()
12         } else {
13             println "Using default implementation"
14         }
15     }
16 }
```

Figura 6. dinâmico Encerramento

O código na Figura 6 mostra o anterior, onde a convocação de fechamento 3 vezes, o primeiro e segundo com um valor, eo terceiro sem ele, a chamada irá receber se as linhas 5 e 6, enquanto a corrida mais linha 7 chamar porque ele não envia qualquer valor.

Na linha 5 verifica-se que está a aplicar o corpo de um método, mas não é, linha 6 passa o valor entre os parênteses, ambas as linhas são equivalentes.

A. ENCAPSULAMENTO

Encapsulação pode controlar como os dados e métodos utilizados. Você pode usar modificadores de acesso para evitar métodos ou métodos de classe externo executar ler e modificar os seus atributos. Para permitir que outros objetos para consultar ou modificar os atributos de objetos, as classes têm, frequentemente, métodos de acesso. (UNAM, 2013)

Os seguintes níveis de acesso (sendo de baixa para alta) deve:

- **private:** acesso apenas a partir da própria classe.
- **default:** acessível a partir da própria classe e a partir do mesmo pacote.
- **protected:** acessível a partir da própria classe, pacote e subclasse.
- **public:** acesso a partir de qualquer pacote.

Nota: Em Java, existem 4 níveis de acesso, mas apenas três modificadores de acesso. Classes só pode ser público ou padrão. (Horna, 2010)

Dentro do código Java somente a classe principal deve ser público, quando mais de uma classe em um arquivo.

Um objeto fornece uma fronteira bem definida em torno de uma única abstração, encapsulamento e tanto a modularidade proporcionada como barreiras em torno essa abstração. (Booch, 1991)

Java exige sempre expressar explicitamente a visibilidade pública de uma classe. Por padrão, a menos que você especifique o contrário, padrão em todas as classes Groovy, propriedades e métodos são de acesso público, independentemente de qual tem o método principal. (Judd & Faisal Nusairat, 2008, p. 23).

A. RANK

Um conjunto de abstrações muitas vezes formam uma hierarquia, e identificar essas hierarquias no projeto simplifica muito a compreensão do problema.

(Booch, 1991) define o termo como se segue:

> *La jerarquía es una clasificación u ordenación de abstracciones.*

A hierarquia de herança é das classes mais importantes e é um elemento essencial de sistemas orientados a objetos. (Booch, 1991)

Se uma classe só pode receber mais características de classe base, a hereditariedade é chamado de herança simples. Mas se uma classe recebe propriedades de mais de uma classe base, herança é chamado de herança múltipla. (Di Serio, 2011).

Na terminologia do Java, uma classe que é herdada é chamada de superclasse. A classe é chamada de subclasse herda. Portanto, uma subclasse é uma versão especializada de uma superclasse, ele herda todas as variáveis de instância e métodos definidos pela superclasse e adiciona seus próprios elementos.

Para herdar uma classe, simplesmente uma definição de classe é incorporado a outra usando a palavra-chave extends. (Schildt, 2009, p. 157)

Super palavra-chave tem duas formas gerais. O primeiro chama o construtor da superclasse. O segundo é usado para acessar um membro da superclasse que tem sido escondida por um membro de uma subclasse.

A lista de parâmetros especifica qualquer acção a necessidade de o construtor da superclasse. super () deve ser sempre a primeira instrução é executada dentro de um construtor da subclasse. (Schildt, 2009, pp. 162-163)

Às vezes é necessário para chamar um método da superclasse. Isso é feito com a palavra-chave super. Se isso se refere à classe atual, super refere-se ao respeito superclasse para a classe atual, que é um método essencial para métodos de acesso substituídas por herança.

Por padrão Java executa estas ações:

> • Se a primeira declaração em um construtor de uma subclasse é uma frase que não é super-nem ESTA, invisivelmente acrescenta Java e super implícita () chamada para o construtor padrão da superclasse, em seguida, começa variáveis e subclasse em seguida, continua com a execução normal.
> • Se estiver usando super (...) na primeira instrução, então você chama o construtor da superclasse selecionado, em seguida, inicia as propriedades da subclasse e, em seguida, continua com outros acórdãos do construtor.

Finalmente, se a primeira instrução é esta (...), então você chama o construtor selecionado pelo ESTA, e, em seguida, continua com os acórdãos do construtor. A inicialização de variáveis de ter concluído o construtor que foi chamado por este. (Marin, 2012)

Todos os recursos de herança Java (incluindo classes abstratas) estão disponíveis em Groovy.

Como o Java não suporta o conceito de herança múltipla, como outras línguas, orientada a objeto, ele faz uma simulação, introduz o conceito de interfaces como uma alternativa à herança múltipla, eles são bastante diferentes, embora as interfaces pode resolver problemas semelhantes . Em particular:

> • Uma vez que uma interface, uma classe herda única constante.
> • Uma vez que uma interface, uma classe não pode herdar definições de método.
> • A hierarquia de interfaces é independente da hierarquia de classe. Várias classes podem implementar a mesma interface e não pertencem à mesma hierarquia de classes. Mas quando falamos de herança múltipla, todos pertencem à mesma hierarquia de classe. (Ceballos, 2010)

Em groovy para lidar com a herança múltipla ele faz uso de um conceito chamado "Mixins". Mixins são usados para injectar o comportamento (métodos) de uma ou mais classes. Normalmente eles usaram anotação withMixin. (Emalvino, 2013)

Sobre métodos de escrita

Quando usar a herança, para herdar de uma classe que pode usar seus métodos, e pode haver ocasiões em que o método não é o pai de nossa utilidade e devemos criar um novo com o mesmo nome, usamos o Sobreecritura. (Hdeleon, 2014)

Dentro de Java para substituir um método, você pode querer usar theOverride anotação que instrui o compilador que você pretende substituir um método na superclasse. Se, por algum motivo, o compilador detecta que o método não existe numa das superclasses, então um erro é gerado. (Oracle, 2014)

Ao contrário de Java, Groovy faz anotação não handleOverride para substituir esse usando a propriedade metaclass eo operador "=". (EduSanz, 2010) Usando a propriedade metaclass você pode adicionar novos métodos ou substituir os métodos existentes da classe. Se você quiser adicionar novos métodos que têm o mesmo nome, mas com argumentos diferentes, você pode usar uma notação de atalho. Groovy usos para este LeftShift () (<<). (Klein, 2009)

A. TIPOS (digitação)

As línguas O.Ö. fazer (para simplificar) que não há virtualmente nenhuma diferença entre os tipos e classes, no entanto, formalmente diferenciação geralmente é feito salienta que um tipo especificado e uma estrutura semântica e uma classe é uma implementação concreta de um tipo.

Em P.O.O. Classes normalmente definir um tipo, e existem outros tipos básicos. Em Java, para que possamos tratá-los como classes quando precisamos dela (por exemplo, para colocá-los onde for necessário Object), classes container (classes de mensagens publicitárias em Inglês) são definidos. (Castro Souto, 2001)

As vantagens de ser rigoroso com os tipos são:

- Confiabilidade, como para detectar erros em tempo de compilação são corrigidos.
- Legibilidade, fornecendo alguma "documentação" para o código.
- Eficiência, otimizações podem ser feitas. (Castro Souto, 2001)

Com base no conceito de tipos, alguns conceitos são mais apoio para classificar as línguas são manipulados. Estes são verificados e tipos de ligação, que são a seguir definidos:

Em tipo de verificação garante que o tipo de coincidir com a construção planejada no contexto. Há duas maneiras de lidar com a verificação de tipo:

- Rigidez de tipos: requer de todas as expressões do tipo são consistentes em tempo de compilação. As línguas que lidam com este tipo são Java, C++, Object Pascal. (Castro Souto, 2001)
- Verificação tipos fracos: Todos os tipos são feitas verificações em tempo de execução (dado, por conseguinte, um maior número de exceções). Isto é, a flexibilidade é muito maior, mas encontram muitas mais problemas ao executar. Ejemplo: Smalltalk. (Castro Souto, 2001)

Existem várias vantagens importantes derivados da utilização da linguagem com tipos rigorosas.

> • "Nenhuma verificação de tipo, um programa pode explodir misteriosamente na execução.
> • Na maioria dos sistemas, o ciclo de edição-compilação-depuração é tão tediosa que a detecção precoce de erros é indispensável.
> • A declaração do tipo de ajuda a documentar os programas.
> • A maioria dos compiladores pode gerar código mais eficiente se você declarou tipos." (Tesler, 1981)

Com o exposto, e com base no conhecimento prévio de como trabalhar Java e Groovy este aspecto, pode-se concluir que Java é uma linguagem de tipagem forte, porque quando vai codificação, e se ele veio para atribuir um tipo de valor diferente em relação ao tipo, marca imediatamente compilador erro, avisando que algo está errado e deve ser revisto ou não a execução não é possível.

Groovy, pelo contrário, é uma linguagem com fraco verificação de tipo, atribuí-la a um valor tipo diferente atribuídos, respectivamente, percebe nada e tempo de execução é quando você envia erros.

Em suma, enquanto Java alerta sobre erros em tempo de compilação tipos, Groovy faz em tempo de execução.

Tipos de ligação

Ligadura é o processo de associar um atributo para um nome no caso de funções, o termo de ligação (binding) refere-se à conexão ou ligação entre uma chamada de função eo código real executado como um resultado da chamada. (Joyanes Aguilar, 1996)

Ligadura é classificada em duas categorias: vinculação estática e dinâmica de ligação. (Booch, 1991)

- Tubal estático: alocação estática, também conhecido como tipos estáticos temprana-ligadura ou ligadura refere-se ao tempo em que os nomes estão ligados a seus tipos. Meios de ligação estáticas que os tipos de todas as variáveis e expressões em tempo de compilação estão definidos.
- Ligadura Dinâmico: também chamada de ligação tardia, isso significa que os tipos de variáveis e expressões não são conhecidos até a execução.

Polimorfismo existe quando interagindo características de herança e ligação dinâmica. É talvez o recurso mais poderoso de linguagens orientadas a objeto após a sua capacidade de suportar abstração, e é o que distingue programação orientada a objetos com tipos de dados abstratos mais tradicionais de programação. (Booch, 1991)

Dito isto, Java e Groovy são línguas com tanto polimorfismo ligação tardia, apoio.

Digitado estática

Nesta caracterização, cada variável deve ser declarado com um tipo associado a ele.

Java é uma linguagem fortemente tipificada. Parte de sua segurança e robustez é devido a este fato.

Em primeiro lugar, cada variável e cada expressão tem um tipo, e cada tipo é rigidamente definido. Em segundo lugar, em todas as atribuições, expressa ou via passagem de parâmetro na chamada do método, a compatibilidade de tipo está marcada. Em Java não há nenhuma conversão automática de tipos incompatíveis como alguns outros idiomas. O compilador Java verifica todas as expressões e parâmetros para assegurar que os tipos são suportados.

Java define oito tipos primitivos: byte, short, int, long, char, float, duplos e boolean. Tipos primitivos também são chamados de tipos simples. (Schildt, 2009)

Qualquer variável usada em um código Java necessário para declarar um tipo de dados, caso contrário, o compilador vai fazer o seu trabalho destacando esse erro quando a codificação.

Tipagem dinâmica

Este tipo de crime não exige que a variável é atribuído um tipo de dados que pode ser int, String, etc., como Java seria simplesmente atribuir valores a essas variáveis e seu tipo de dados será considerada dependendo do valor que você atribui à variável.

SEGURANÇA GROOVY

Independentemente do facto de o tipo de uma variável é declarado explicitamente ou não, o sistema é tipo seguro. Diferentemente das linguagens sem tipo não groovy permitir o tratamento de um objeto de um tipo como uma instância de um tipo diferente, sem uma conversão bem definido disponível. Você nunca iria tratar um valor java.lang.String "1", como se fosse um java.lang.Number. Tal comportamento seria perigoso - é a razão que Groovy não permite isso é mais do que Java. (König & Glover, 2007)

Groovy para definir um tipo de dados primitivo de uma variável como faz Java é muito válido, sem alterações, para tornar tudo mais fácil fácil manuseio, Groovy, sem esquecer o tema principal refere-se à maneira pela qual esta linguagem opera principalmente como um objeto.

Desenhadores Groovy decidiu primitivas terminar. Quando Groovy precisa armazenar valores que usaram os tipos primitivos de Java, Groovy já usa as classes de mensagens publicitárias fornecidos pela plataforma Java.

Sempre que você vê o que parece ser um valor primitivo (por exemplo, o número 5 no código-fonte Groovy, esta é uma referência a uma instância da classe wrapper apropriado). Por razões de brevidade e familiaridade, Groovy pode declarar variáveis como se fossem variáveis de tipos primitivos.

A conversão de um único valor em uma instância de um tipo de wrapper é chamado de boxe em Java e outras linguagens que suportam a noção. A ação reversa - tomando um exemplo de um wrapper e recuperar o valor primitivo - é chamado unboxing. Groovy executa essas operações automaticamente, se necessário. Este boxe automática e unboxing é chamado autoboxing.

Tudo isso é transparente - você não precisa fazer nada no código Groovy para habilitá-lo.

Devido a isso, podemos dizer que Groovy é mais orientada a objeto- Java. (König & Glover, 2007)

A. CONCURSO

Para certos tipos de problema, um sistema automatizado pode precisar para lidar com eventos diferentes ao mesmo tempo. Outros problemas podem envolver muitos cálculos que excedem a capacidade de qualquer único processador. Em ambos os casos, é natural considerar o uso de um conjunto distribuído de computadores para implementar processadores perseguido ou usar capazes de multitarefa. Um único thread de controle - processo chamado é a raiz a partir da qual as acções dinâmicas independentes ocorrendo dentro do sistema.

Os sistemas que executam em múltiplas CPUs permitir threads simultâneos verdadeiramente controlar enquanto os sistemas que funcionam com uma única CPU só pode alcançar a ilusão de threads simultâneos de controle.

Existem 2 tipos de competição, pesado e competição por luz. Um processo de construção é tratado de forma independente pelo sistema operativo alvo e inclui o seu próprio espaço de endereço. Um processo leve normalmente existem dentro de um único processo do sistema operacional na companhia de outros processos leves que compartilham o mesmo espaço de endereço e muitas vezes envolvem dados compartilhados.

Enquanto programação orientada a objetos concentra-se em dados abstração, encapsulamento e herança, a simultaneidade abstração enfoca o processo e calendário. O objeto é um conceito que unifica esses dois pontos de vista: cada objeto (elaborado a partir de uma abstração do mundo real) pode representar um segmento separado de controle (uma abstração de um processo). (Booch, 1991)

Com base no acima simultaneidade Grady Booch definidos como se segue:

> *"A competição é a propriedade que distingue um objeto ativo de alguém que não está ativo."*

Como pode ser visto, o fio e conceitos de processo que são definidas abaixo são manipulados para entender melhor o problema.

> • Hilo: também conhecida como luz ou de fios de processo. Ou processos independentes são pequenos pedaços de um processo. Você também pode dizer que um segmento é um único fluxo de execução dentro de um processo.

• Processo: é um programa executado em seu próprio espaço de endereço. (Cisneros, 2010)

Java suporta o conceito de fio da própria linguagem, com algumas classes e interfaces definidas no pacote java.lang e métodos específicos para a manipulação de threads na classe Object.

Você pode definir e instanciar uma thread (thread) de duas maneiras:

• Estender java.lang.Thread classe
• Implementar a interface Runnable

Desde Groovy pode utilizar todas as instalações normais que Java fornece para a simultaneidade, só facilita o trabalho Groovy como será visto a seguir. (König & Glover, 2007)

A primeira e principal característica do Groovy é que o suporte para multithreading é implementado Encerramento Runnable. Isto permite que as definições de fios simples quanto:

```
t = new Thread() { /* Closure body */ }
t.start()
```

Isto mesmo pode ser simplificado com novos métodos estáticos na classe Thread:

```
Thread.start { /* Closure body */ }
```

Java Tem o conceito de um segmento do daemon e, portanto, o mesmo acontece com Groovy. Um segmento daemon pode ser iniciado através de:

```
Thread.startDaemon { /* Closure body */ }
```

B. PERSISTÊNCIA

Persistência é a propriedade de um objeto através do qual sua existência transcende o tempo e / ou espaço. Isto significa que um objeto persistente ainda existir depois de ter completado o programa que o criou e também pode ser transferida de local de memória no qual ele foi criado. (Ortiz, 2014)

Sugere-se que existe um contínuo de existência do objecto, que varia de objectos transientes, que aparecem na avaliação de uma expressão para os objectos na base de dados que sobrevivem a execução de um único programa. Este aspecto da persistência abrange:

- resultados temporários na avaliação de expressões.
- As variáveis locais nos procedimentos de ativação.
- variáveis próprias, variáveis globais e elementos da pilha (heap) cuja duração difere de seu espaço.
- dados entre execuções de um programa.
- dados entre várias versões de um programa.
- Dados sobreviver ao programa.

As linguagens de programação tradicionais geralmente lidam apenas com os três primeiros tipos de objetos persistentes; a persistência dos últimos três tipos pertence tradicionalmente ao domínio dos bancos de dados de tecnologia. (Booch, 1991)

O atributo de persistência só devem estar presentes nesses objetos que um aplicativo requer para manter entre as execuções, caso contrário, seria provavelmente armazenar uma enorme quantidade de objetos desnecessários. Persistência é conseguido através do armazenamento de um dispositivo de armazenamento secundário (disco rígido, memória flash) as informações necessárias de um objeto para restaurá-lo mais tarde. Persistência normalmente tem sido o domínio da tecnologia de banco de dados, de modo que esta propriedade não foi até recentemente que foi incorporado à arquitetura básica de linguagens orientadas a objeto.

A linguagem de programação Java permite serializar objetos em um fluxo de bytes. Este fluxo pode ser gravada em um arquivo no disco e, em seguida, ler e desserializado para reconstruir o objeto original. Isso é feito com o que é chamado de "luz persistente" (persistência lightwigth em Inglês). (Ortiz, 2014)

Serialização de um objecto é a obtenção de uma sequência de bytes que representam o estado do objecto. Esta sequência pode ser usado de várias maneiras (podem ser enviados através da rede, salvos em um arquivo para uso posterior, usado para reconstruir o objeto original, etc.).

Objeto Serializable

Um objecto seriável é um objecto que pode ser convertido em uma sequência de bytes. Para um objeto pode ser serializado, você deve implementar a interface java.io.Serializable. Esta interface não define nenhum método. Apenas utilizado para 'marcar' as classes cujas instâncias podem ser convertidos em seqüências de bytes (e mais tarde reconstruído). Como objetos comuns, como String, Vector ou ArrayList implementar Serializable, para que eles possam ser serializado e reconstruída mais tarde. (Miedes, 2014)

Serialização e desserialização pode ser realizada tanto Java e Groovy.

Em Java, manipulação de arquivos é feito usando o pacote java.io classe File, que presta apoio através de métodos para operações / O.

O foco em como Groovy torna o gerenciamento de arquivo, em comparação com Java, faz pouca diferença. Primeira língua utiliza o mesmo pacote com o trabalho Java, só que acrescenta vários métodos de conveniência e como sempre, faz redução de código necessário para linhas de trabalho.

O Java JDK, aborda esta necessidade com seus pacotes e java.IO de java.net. Ele fornece suporte feito com o arquivo, URL e numerosas versões de córregos, leitores e escritores aulas.

Mais sem mudança, Groovy estende o Java JDK seu próprio GDK (Groovy Development Kit). GDK tem uma série de métodos que fazem a magia para trabalhar com mais facilidade.

Em Groovy características que podem ser vistos a olho nu são eles:

- Não há necessidade de fazer o tratamento de exceção.
- O pacote java.io é automaticamente importado pela GDK, não necessitando o uso explícito.
- Gestão BufferedReader e FileReader não é necessário.
- Somente a classe de ficheiros é utilizado, que é apenas a representação de um arquivo e diretório caminhos.

A eliminação das características acima em várias linhas reduz o código, permitindo mais rápido e mais legível codificação de código.

El JDK Groovy se puede consultar en su sitio oficial. (Groovy)

C. EXCEÇÕES DE GESTÃO

Uma excepção é um evento que ocorre durante a execução de um programa e interrompe o fluxo normal de instruções. (Oracle)

O tratamento de exceções em Java é executada por cinco palavras-chave: try, catch, throw, e, finalmente, atira. A seguir descreve resumidamente como funciona: As instruções do programa que deseja monitorar, estão incluídas em um bloco try. Se ocorrer uma exceção dentro do bloco try, ele é jogado. O código pode pegar essa exceção usando captura, e gerenciá-lo racionalmente. Excepções gerados pelo sistema são enviados automaticamente pelo intérprete

de Java. Para enviar uma exceção lançar manualmente palavra-chave é usada. Deve ser especificado pela cláusula throws qualquer exceção que é enviado de um método para o método externo que o chamou. Você deve colocar qualquer código que o programador deseja sempre correr atrás de um bloco try for concluída, o bloco finally. (Ceballos, 2010).

Em Groovy, manipulação de exceção é exatamente o mesmo que em Java e segue a mesma lógica. Você pode especificar uma seqüência completa de blocos try-catch-finally, ou apenas try-catch, ou simplesmente tentar-finally. Note que as chaves ao contrário de outras estruturas de controle necessários ao redor dos corpos de bloco se eles contêm mais de uma instrução. A única diferença entre Java e Groovy em termos de exceções é que as comunicações de acusações na assinatura do método é opcional exceções, até mesmo xadrezes. (König & Glover, 2007, p. 171)

O Groovy não exigem exceções identificador que não deseja gerenciar ou são inadequados no nível de código atual. Qualquer exceção não tratada passa automaticamente para o próximo nível (Subramaniam, 2013, p. 17).

Em geral, todas as exceções não são comprovados Groovy, ou melhor, a sua gestão é opcional.

D. PALABRAS RESERVADAS

Java define uma série de palavras da língua para a identificação de operações, métodos, classes, etc., a fim de que o compilador pode compreender os processos que estão sendo desenvolvidos. Estas palavras não pode ser usado pelo desenvolvedor para nomear métodos, variáveis ou classes, porque, como mencionado, cada um tem um alvo dentro da linguagem.

Se um identificador para definir qualquer uma das palavras-chave do compilador avisa que o erro para o programador fazer algo sobre isso.

A Tabela 1 mostra as palavras de Java reservados:

Tabla 1. Palabras reservadas Java (Schildt, 2009)

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

A linguagem Groovy torna a implementação de novas palavras-chave, que ajudam a gerir determinadas características de uma forma mais simples, tornando o código mais fácil de escrever, ler e entender.

def e estão entre as novas palavras-chave Groovy. def define os métodos, propriedades e variáveis locais. usado em loops para especificar o intervalo de um loop, como em (i em 1..10).

O uso dessas palavras-chave como nomes ou nomes de método variável pode levar a problemas, especialmente quando o código Java existente é usado como código Groovy.

Também não é uma boa idéia para definir uma variável chamou. Groovy não Reclamar embora, se um campo com este nome é usado em um gabinete, o nome do parâmetro refere-se ao encerramento e não um campo dentro da classe. (Subramaniam, 2013)

Outra nova palavra que traz-lo como Groovy, que permite que você altere o tipo de objeto, o exemplo prático foi o Builders sujeitos.

E. Estruturas de Controle

Groovy suporta quase a mesma estruturas lógicas de controle Java, fazendo exceção às normas ou implementações aceites mesmos. Dentro do booleana teste condicional avaliar e tomar uma decisão com base em se o resultado era verdadeiro ou falso. Nenhuma dessas estruturas deve ser uma experiência completamente nova para qualquer desenvolvedor de Java, mas é claro Groovy acrescenta alguns toques.

Em Groovy um teste expressão booleana pode ser de qualquer tipo (não vazio). Ele pode ser aplicado a qualquer objecto. Groovy decide se considerar a expressão como verdadeira ou falsa, aplicando as regras mostradas na Figura 8 (König & Glover, 2007):

Tipo	Criterio de evaluación requerido a true
Boolean	Correspondiente valor booleano es true
Matcher	El comparador cuenta con una coincidencia
Collection	La colección no está vacía
Map	El mapa no está vacío
String, Gstring	La cadena no está vacía
Number, Character	El valor es distinto de cero
Ninguna de las anteriores	La referencia del objeto no es null

Figura 8. Regras de sequência usado para avaliar um teste booleano.

if/ if-else

Ele funciona exatamente da mesma forma como faz Groovy para Java.

Como em Java, expressão de teste booleano deve ser colocada entre parênteses. O bloco condicional geralmente entre chaves. Estas chaves são opcionais se o bloco é composto por uma única instrução.

A única diferença está na forma como Groovy interpreta estas condições. Groovy não pode promover uma série de condições para booleano verdadeiro ou falso. Por exemplo, um número diferente de zero é sempre verdadeira. (Dearle, 2010)

Elvis operador ternário (? :)

Em Java, o operador ternário pode dizer que é um if-else abreviado como proprietária da linguagem de programação C Groovy suporta o caminho Java padrão para lidar com isso, mas tem um operador similar chamado Elvis, que destaca mais praticidade durante a programação, a sua estrutura é (a: b) ser o ternário equivalente (um para b?). (Dearle, 2010). Os símbolos "?" Não deve ir com um espaço entre elas, ao contrário do que marcará erro.

Elvis funciona através da manutenção de uma variável local oculto, que armazena o resultado inicial. Se esse resultado é determinado de acordo com as regras da Figura 8, que, em seguida, o valor é devolvido, caso contrário, o valor alternativo é usado. (Dearle, 2010)

Switch

Em Java, como na linguagem C, a instrução switch permite certa variável sendo testado por uma lista de condições manipulados no caso.

Groovy, pelo contrário, é mais amigável para os tipos de dados dentro do caso. Os elementos do caso não são do mesmo tipo que recebe o interruptor pode lidar com ambos inteiros, listas, cordas, faixas, etc. (Dearle, 2010)

While

Este ciclo, em relação à estrutura e sintaxe lógico é o mesmo em ambos os idiomas.

Note que Groovy não aceita o do circuito {} while (), completamente desconhecido o que essa frase significa que se alguém tentar utilizar.

Bucle for

Quando Groovy foi criado, e durante algum tempo depois, ele não suportar o padrão Java para loop:

for (int i = 0; i < 5; i++) { ... }

Na versão 1.6, no entanto, foi decidido que era mais importante que suporta construções Java que tentam manter uma linguagem livre que pouco sintaxe Java desconfortável que ele herdou de seus antecessores (Como a linguagem C).

Da mesma forma suporta Java para cada-. (Kousen, 2014)

O para-cada um é usado para iterar sobre os elementos de coleções que são matrizes, ArrayList, HashMap, ...

No entanto, nenhum desses laços é a forma mais comum de loop no Groovy. Em vez de escrever um laço explícito, como nos exemplos acima, Groovy preferir uma aplicação mais direta do padrão de design iterador. Groovy acrescenta a cada método, que leva um fecho como um argumento para coleções.

(0..5).each { println it }

A cada um dos métodos é a construção do circuito mais comum em Groovy.

Groovy outro loop é executado pelo para-in, você pode usar o termo "in" para repetir toda a coleção.(Dearle, 2010)

F. LA JAVA VIRTUAL MACHINE (JVM)

Um dos pontos interessantes é que você tem Groovy usa a máquina virtual Java para executar o código. Como é bem conhecido, Java é uma linguagem compilada e interpretados, enquanto Groovy é jogado, embora também seja possível utilizar o compilador para ele.

A especificação JVM fornece definições específicas para a execução do seguinte: um conjunto de instruções (equivalente a uma unidade central de processamento [CPU]), um conjunto de registros, o formato de arquivo de classe, uma pilha de execução, pilha coletor de lixo, uma área de memória, mecanismo de relatórios de erros fatais, e apoiar sincronismo de alta precisão.

O formato do código JVM bytecode é compacto e eficiente. Representado por programas de bytecode JVM deve manter uma disciplina tipo adequado. Mais verificação de tipo é feito em tempo de compilação.

Qualquer intérprete compatível tecnologia Java deve ser capaz de executar qualquer programa com arquivos de classe que estejam em conformidade com o formato de arquivo de classe especificado na especificação do Java Virtual Machine. (Oracle e / ou suas afiliadas, 2010)

A filosofia da máquina virtual é como se segue: o código fonte é compilado, detectar os erros de sintaxe, e um tipo de executável é gerado com o código de uma máquina destinada a uma máquina imaginário, com um processador central imaginário. Neste tipo de código de máquina é chamada de código intermediário, ou às vezes também linguagem intermediária, p-code, ou byte-code.

Como máquina imaginário que não existe, você pode executar o arquivo executável, um intérprete é construído. Este intérprete é capaz de ler cada uma das instruções de código de máquina de imaginários e executá-los em plataforma real. Este intérprete é chamado o intérprete da máquina virtual. (Vic, 2013)

Actualmente, como é bem conhecido, a JVM já não é apenas para Java, existem outros idiomas que podem ser compilados na máquina virtual e estão disponíveis para uso. Estas línguas compilar para bytecode em arquivos de classe, que podem ser executados pela JVM.

Groovy é uma linguagem de programação orientada a objeto que é executado na JVM. Ele também mantém a plena interoperabilidade com Java.

o seguinte sobre a execução de classes de Groovy ou scripts são mencionados e concluiu o autor dessa carta que é uma linguagem compilada:

"*Java compilado com javac e execute o bytecode Java resultante. Groovy pode ser compilado e executado com groovyc groovy, mas realmente não tem que compilar primeiro. O comando Groovy pode funcionar com um argumento de código fonte e compilá-lo executado pela primeira vez em quando. Groovy é uma linguagem compilada, mas você não tem que separar as etapas, embora a maioria das pessoas fazem. Quando um IDE é usado, por exemplo, cada vez que você salvar um script ou classe Groovy é compilado*". (Kousen, 2014)

Conclusiones

Foi possível confirmar que as duas são linguagens que atendam satisfatoriamente os elementos necessários para ser considerado Orientada a Objetos, lembrando-se de que eles são apenas quatro elementos primários a considerar, mas da mesma forma conhecer o lado, estes elementos fazem parte do Orientada Modelo Objeto Graddy proposto por Booch.

Ambas as línguas têm sintático como uma grande scripts, Groovy faz uso de pequenos programas, enquanto que para um sistema maior é recomendado usar classes como Java faz, apenas para ter um desenvolvimento melhor e isso permite que ele seja modificado ou melhorado de uma maneira simples.

Uma das características mais notáveis é o código que lida com a simplicidade da nova língua, é mais fácil programar em Groovy, vindo a trabalhar com codificação Java é mais conveniente, lembrando que 99% do código Java reconhece Groovy, mas também o último só que usa o que é preciso para trabalhar. O que poderia trabalhar com Java 5 linhas no simples "Olá mundo" Groovy é feito em apenas um usando scripts, ou com os mesmos 5 linhas usando Java, mas omitindo parte do ponto de manuseio sintaxe forçado que deu origem a desenvolvimento deste trabalho.

Você pode simplesmente destacar o seguinte groovy do manuseio sobre comparando sintaxe Java:

• As classes, variáveis e métodos são públicos por padrão.

• respeitando Groovy lida com hierarquia de classes herança múltipla.

• ponto e vírgula, no final de uma frase omitida.

• Os parênteses são opcionais.

• Groovy nenhuma necessidade de definir um tipo de dados variáveis, já que um dos seus pontos fortes é que ele é uma linguagem dinâmica.

• Algumas das frases mais curtas.

• importações atrás biblioteca e não exige explicitamente expressa.

• A palavra-chave return é opcional.

• Para implementar Cordas oferece novas maneiras de fazer além de trabalhar com a forma tradicional de Java.

• Reduza o código para criar getters e setters variáveis implicitamente ao público por padrão.

• No que diz respeito estruturas de controle que Java manipula o mesmo, exceto para os do-while não reconhece, mas todos os outros trabalhos com a mesma lógica, e até mesmo faz algumas implementações da mesma.

• Trabalhar um recurso chamado Closure, que oferece a mesma funcionalidade que um método, mas a diferença é que os encerramentos são anônimas, ou seja, eles não estão ligados a um objeto.

• O manuseio de blocos try-catch e tudo o que implica a manipulação de exceção torna opcional.

• Groovy é puro objeto-orientado, aparentemente trabalhando com variáveis primitivas como Java, mas debaixo das cobertas usa wrappers para converter a objetos.

Java e processos de execução Groovy em uma máquina virtual, que funciona com um compilador e um intérprete, a primeira língua é compilado e interpretada, enquanto o segundo é jogado, mas também permite que você compilar.

É certo que, Java continua a excel e conseguiu ser um dos melhores línguas porque oferece mais ferramentas para atender às necessidades de programadores, permitindo-lhes satisfazer as necessidades dos utentes. A Groovy carece de abordar alguns pontos que permitem o uso de maneira mais satisfatória, pelo menos para trabalhar até o seu homólogo, mas lembre-se que as ferramentas gradualmente ele próprio tem melhorado e também foram criando que ajudam a cobrir com a demanda. Java, da mesma forma como ele está melhorando aspectos, mostrando

que não é uma língua fácil de substituir ou jogo, o viu têm falhas, mas isso o ajudou a trabalhar nele e melhorar.

Seria interessante mais tarde rever a questão ora tratada e ver como eles evoluíram ambas as línguas, tenha em mente que o desenvolvimento é um tema da actualidade, com grande futuro, não haverá novas linguagens, outros irão desaparecer, talvez, e muitos mais irão melhorar as suas características, tais como Tem sido visto.

> *"O sucesso de Groovy é principalmente devido a sua familiaridade com Java. Os desenvolvedores querem fazer as coisas, querem dominar a língua rapidamente, eles querem características de potência. Groovy é uma mais produtivo do que Java, mas ainda com uma sintaxe Java-like que já é bem conhecida linguagem. Em cima disso, Groovy simplifica as tarefas diárias que costumavam ser complexo para desenvolver."* (White & Maple, 2014)

Bibliografía

Booch, G. (1991). *Object Oriented Design with Aplications.* Redwood City, California 94065: The Benjamin/cummings Publishing Company, Inc.

Castro Souto, L. (2001). *Programación Orientada a Objetos.* Retrieved Enero 15, 2015, from Programación Orientada a Objetos: http://quegrande.org/apuntes/EI/OPT/POO/teoria/00-01/apuntes_completos.pdf

Ceballos, F. J. (2010). *Java 2, Curso de Programación.* México: RA-MA.

Cisneros, O. (2010, Agosto). *Tópicos Selectos de Programación*. Retrieved Enero 17, 2015, from Programación concurrente multihilo: http://topicos-selectosdeprogramacion-itiz.blogspot.mx/p/unidad-3-programacion-concurrente.html

Dearle, F. (2010). *Groovy for Domain-Specific Languages.* Birmingham: Packt Publishing.

Di Serio, A. (2011, Marzo 18). *Programación Orientada a Objetos.* Retrieved Octubre 28, 2014, from LDC: Noticias: http://ldc.usb.ve/~adiserio/Telematica/HerramientasProgr/ProgramacionOONotes.pdf

EduSanz. (2010). *Añadiendo o sobreescribiendo métodos*. Retrieved Febrero 15, 2015, from Groovy & Grails: http://beginninggroovyandgrails.blogspot.mx/p/groovy.html

Egiluz, R. (2011, Septiembre 27). Groovy hacia un JVM políglota.

emalvino. (2013, Mayo 21). *Hexata Architecture Team*. Retrieved Febrero 3, 2015, from Hexata Architecture Team: http://hat.hexacta.com/agregando-funcionalidad-a-objetos-de-dominio-en-grails/

ESCET. (2009, Agosto 18). *Escuela Superior de Ciencias Experimentales y Tecnología.* Retrieved Octubre 23, 2014, from Introducción a POO y Java: http://www.escet.urjc.es/~emartin/docencia0809/poo/1.-Introduccion-4h.pdf

Groovy. (n.d.). *Class File*. Retrieved Diciembre 2014, from Method Summary: http://groovy.codehaus.org/groovy-jdk/java/io/File.html

Hdeleon. (2014, Mayo 12). *5.- POLIMORFISMO – CURSO DE PROGRAMACIÓN ORIENTADA A OBJETOS.* Retrieved Diciembre 4, 2014, from Curso Programación Orientada a Objetos POO: http://hdeleon.net/5-polimorfismo-curso-de-programacion-orientada-objetos-en-10-minutos-5/

Horna, M. (2010). *Resumen de objetivos para el SCJP 6.0.*

Joyanes Aguilar, L. (1996). *Programación Orientada a Objetos.* España: McGraw-Hill.

Judd, C. M., & Faisal Nusairat, J. (2008). *Beginning Groovy and Grails.* United States of America: Apress.

Klein, H. (2009, Diciembre 22). *Groovy Goodness: Implementing MetaClass Methods with Same Name but Different Arguments*. Retrieved Febrero 15, 2015, from HaKi: http://mrhaki.blogspot.mx/2009/12/groovy-goodness-implementing-metaclass.html

König, D., & Glover, A. (2007). *Groovy in Action.* United States of America: Manning.

Kousen, K. A. (2014). *Making Java Groovy.* United States of America: Manning.

Marin, F. (2012). *Herencia y Polimorfismo JAVA.* Retrieved Septiembre 25, 2014, from issuu: http://issuu.com/felixmarin/docs/herencia_y_polimorfismo_java

Miedes, E. (2014). *Serialización de objetos en Java* . Retrieved Diciembre 3, 2014, from javaHispano: http://www.javahispano.org/storage/contenidos/serializacion.pdf

Oracle. (2014, Septiembre 16). *Naming Convertions*. Retrieved from Oracle Technology network-Java: http://www.oracle.com/technetwork/java/codeconventions-135099.html

Oracle. (2014). *Overriding and Hiding Methods*. Retrieved Diciembre 10, 2014, from Java Documentation: https://docs.oracle.com/javase/tutorial/java/IandI/override.html

Oracle and/or its affiliates. (2010, Junio). *Java Programming Language, Java SE 6.* United States of America.

Oracle. (n.d.). *What Is an Exception?* Retrieved Septiembre 27, 2014, from Java Documentation: http://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html

Ortiz, A. (2014). *Persistencia en Java*. Retrieved Diciembre 3, 2014, from Estructura de datos: http://webcem01.cem.itesm.mx:8005/apps/s201411/tc1018/notas_persistencia/

Rodríguez Echeverría, R., & Prieto Ramos, Á. (2004). *Programación Orientada a Objetos.*

Schildt, H. (2009). *Java, Manual de Referencia (7a ed.).* México: McGraw-Hill.

Subramaniam, V. (2013). *Programming Groovy 2.* United States of America: The Pragmatic Bookshelf.

Tesler, A. (1981). *The Smalltalk Environment.*

TutorialesNET. (2014, Abril 16). *Java - 33: Modularidad.* Retrieved Enero 9, 2015, from TutorialesNet: http://tutorialesnet.net/cursos/curso-de-java-7

UNAM. (2013, Febrero 12). *Análisis Orientado a Objetos.* Retrieved Octubre 28, 2014, from Repositorio digital de la Facultad de Ingeniería - UNAM: http://www.ptolomeo.unam.mx:8080/xmlui/bitstream/handle/132.248.52.100/175/A6%20Cap%C3%ADtulo%203.pdf?sequence=6

Vic. (2013, Febrero 28). *La tecla de ESCAPE.* Retrieved Octubre 15, 2014, from La tecla de ESCAPE: http://latecladeescape.com/t/Compiladores,+int%C3%A9rpretes+y+m%C3%A1quinas+virtuales

White, O., & Maple, S. (2014). *10 Kickass Technologies Modern Developers Love.* Retrieved Octubre 30, 2014, from RebelLabs: http://pages.zeroturnaround.com/Kickass-Technologies.html?utm_source=10%20Kickass%20Technologies&utm_medium=reportDL&utm_campaign=kick-ass-tech&utm_rebellabsid=89