Dynamic, Functional and Script Language - F1

Learn once, code anywhere.

Contato: **fernandofranzini@gmail.com**

Observação

Qual é o objetivo desse curso?
Capacitar um desenvolvedor Java de "**nível intermediário"** a usar os novos conceitos, filosofias e recursos da linguagem de programação groovy.

Qual a dinâmica desse curso?
Ensinar "**somente groovy"** sobre a linguagem Java, OOP, polimorfismo. design patterns e produtos básicos da JDK: Collections, IO, Swing, e etc.

Download Material

Download material do curso e explicação das pastas.

- groovy-m1-win64.zip

Introdução

Introdução

Java é indiscutivelmente a maior e mais usada plataforma do mercado atual. Isso tem acontecido por causa de alguns motivos:

1. A **JVM** tem evoluído com tempo e tem atendido às novas realidades e necessidades.

2. A **JDK** tem evoluído com tempo e tem atendido às novas realidades e necessidades.

3. **Novos produtos** (API, frameworks, IDE, ferramentas, tecnologias) de terceiros e ou proprietários e especificações JCP tem evoluído e surgido para atender novas realidades e necessidades.

Introdução

Ficando para trás apenas **1 problema**: a **linguagem de programação Java** que por motivos de retro compatibilidade não consegue evoluir e nem atender novas formas de produtividade.

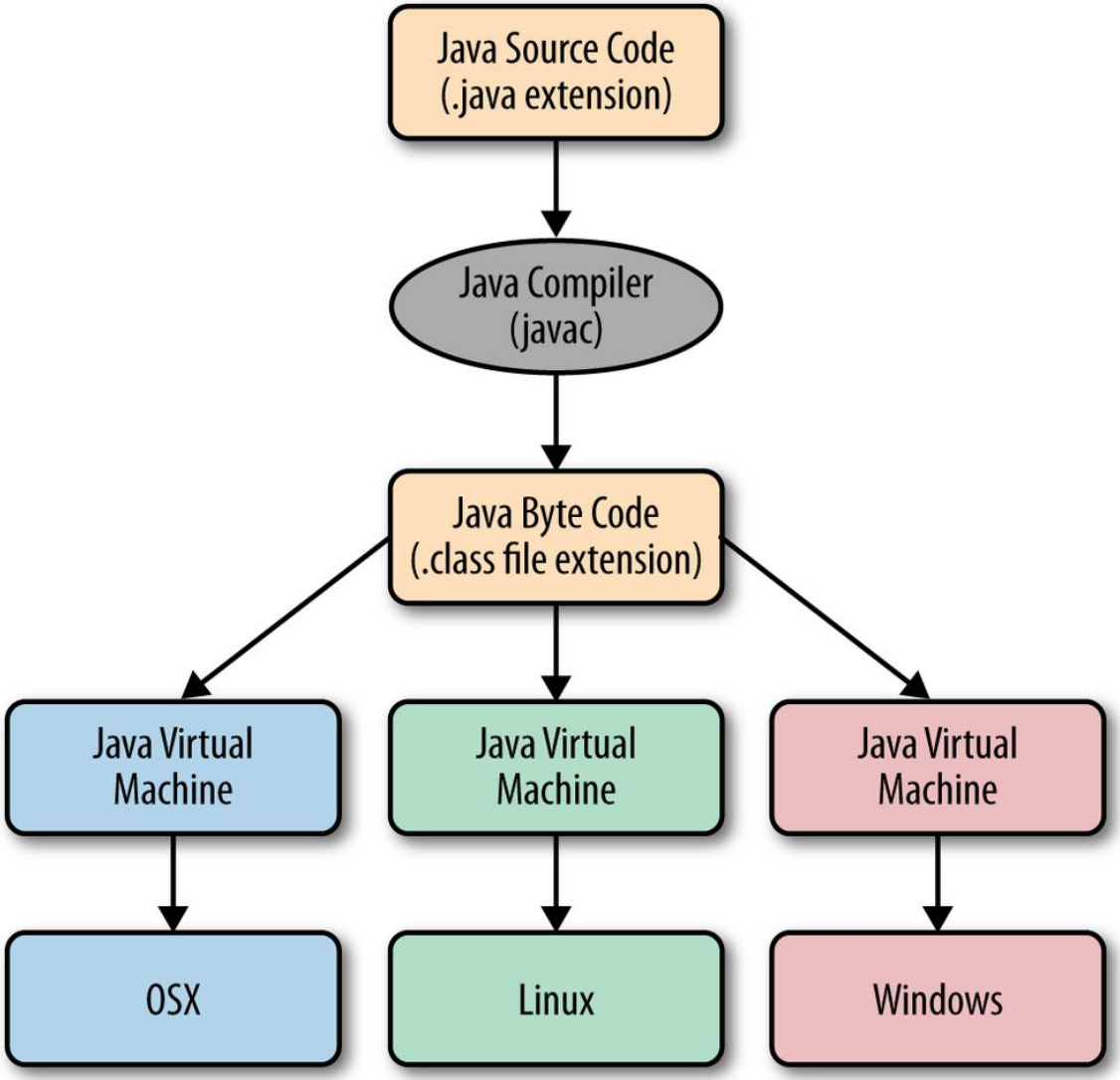
Linguagens modernas chamadas de ágeis, dinâmicas, funcionais e com capacidades de meta programação como Ruby, Python e Scala **começaram se destacar** sobre a velha e antiga linguagem Java [1994].

Introdução

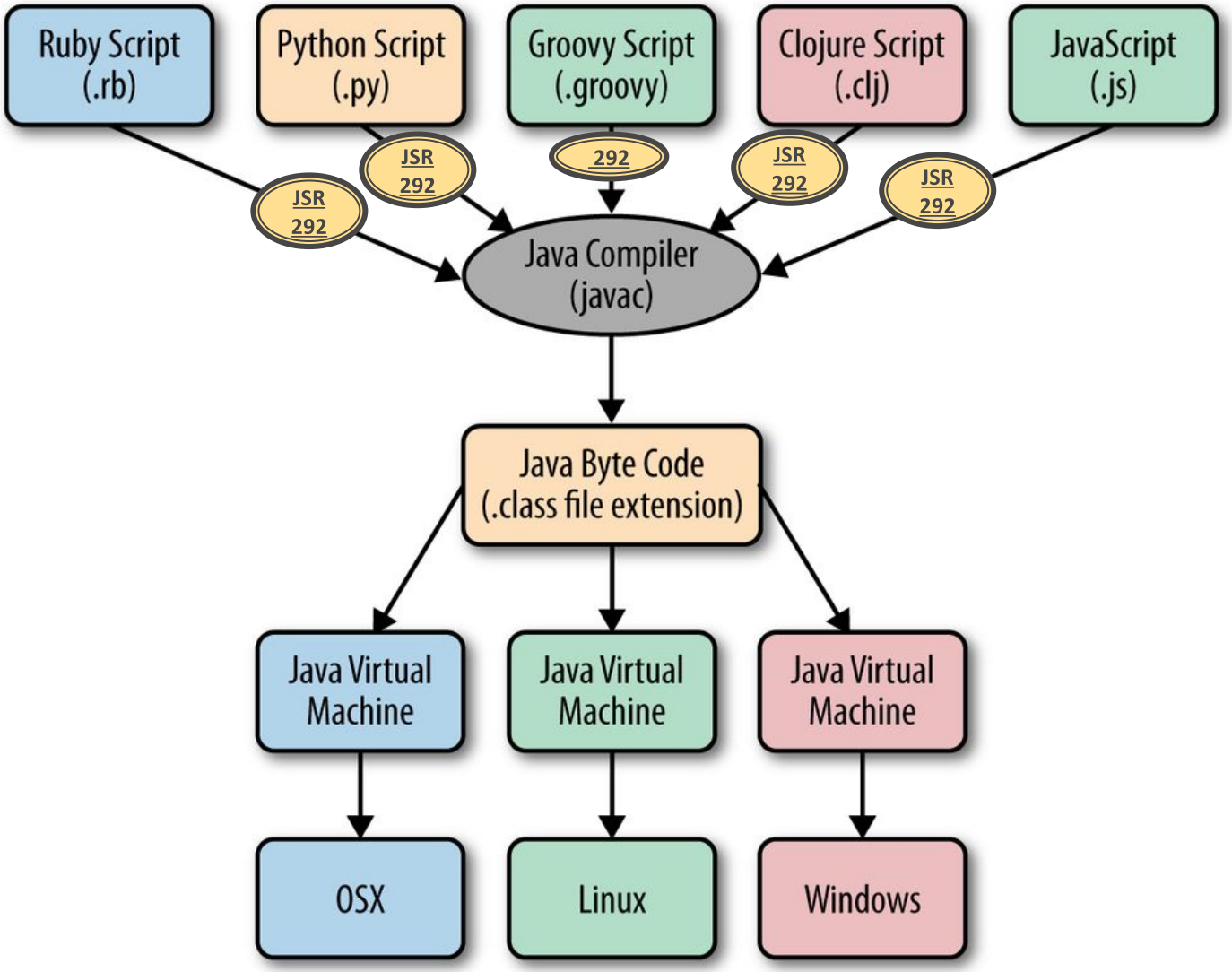
Já sabendo dessa limitação, em 2006 foi estrategicamente criado e aprovado uma especificação chamada de *"JSR 292 – Da Vinci Machine"* **responsável por abrir a JVM para ser utilizada com outras linguagens de programação**.

A partir disso, a comunidade Java agora tinham a **liberdade de escolher qual linguagem** de programação usar para escrever suas soluções em Java.

Introdução


Java Source Code
(.java extension)
Java Compiler
(javac)
Java Byte Code
(.class file extension)
Java Virtual
Machine
Java Virtual
Machine
Java Virtual
Machine
OSX
Linux
Windows

Introdução – JSR 292



Introdução

E por isso, hoje, quem usa JVM, tem a **versatilidade** de selecionar e ou combinar diferentes tipos de linguagens de programação, atualmente com **mais de 240** diferentes linguagens. Veja catálogo:

http://www.is-research.de/info/vmlanguages/

Introdução

As mais “badaladas” são:

- Groovy - http://www.groovy-lang.org/
- JRuby - http://jruby.org/
- Scala - http://www.scala-lang.org/
- Python - http://www.jython.org/
- Clojure - http://clojure.org/

Introdução

Se nenhuma agradar, crie a sua própria linguagem sobre a JVM!!!!

Muitas empresas têm feito isso...

- Google = **Go - https://golang.org**
- Red Hat = **Ceylon - https://ceylon-lang.org**

Introdução

Nesse ponto da história, alguns da comunidade estavam "**começando a pensar em abandonar o plataforma Java"** em virtude de não ter como usar uma linguagem de programação nova e moderna.

O que certamente não aconteceu!!!!

A partir da JSR 292, foi possível continuar usando a melhor e maior plataforma da história e escolher qual linguagem utilizar.

Introdução

Discussão, comentários e dúvidas?

Groovy

Groovy

Criado em 2003, é uma linguagem de programação desenvolvida para a plataforma Java como alternativa à “velha” linguagem de programação tradicional.

Ela foi aprovado pela Java Community Process (JSR 241) como linguagem dinâmica oficial da plataforma Java. A JSR está congelada atualmente.

Groovy

Hoje está sob Apache License, Version 2.0.

Foi criado com o objetivo de fornecer aos programadores Java todas as características das linguagens modernas chamadas de **dinâmicas**, **funcionais** e com capacidades de **metaprogramação**.

Groovy = Dinâmica + Funcional + Recursos de Metaprogramação.

Groovy

Filosofia Groovy:

"Ser a linguagem padrão Java do século 21, evoluída e acrescida com os determinados recursos modernos que não podem ser adicionados ou alterados na linguagem atual devido a retrocompatibilidade...."

Groovy

Filosofia Groovy:

"Foi criado por experientes programadores Java, para programadores Java, com único e exclusivo objetivo de facilitar, melhorar e aumentar a produtividade na atividade de programação cotidiana."

Groovy

Por que usar Groovy e não outra?

Porque Groovy?

1)Nenhuma Curva de Aprendizado

Groovy é exatamente a linguagem Java, "**acrescida"** dos recursos encontradas nestas outras linguagens modernas, ágeis, produtivas, dinâmicas, funcionais e com recursos de metaprogramação.

A curva de aprendizado para um programador Java é extremamente baixa.

Porque Groovy?

1)Nenhuma Curva de Aprendizado

Diferentemente se fosse aprender por exemplo Ruby que tem **uma sintaxe totalmente nova**. Por isso, um programador já acostumado com Java vai naturalmente se adaptar a Groovy muito rápido, fazendo com que ele não perca a sua bagagem.

Diferentemente se ele fosse aprender uma outra, teria que começar do zero, com uma **grande curva** de aprendizado.

Porque Groovy?

2) Groovy tem o GDK = JDK + Groovy API

Groovy estende inúmeras api's da JDK padrão acrescentando **centenas de recursos e otimizações** que vão facilitar e aumentar a produtividade em praticamente todas as api's e ou frameworks como por exemplo: testes, xml, json, io, datas, jdbc, swing, web services, strings, collections, expressão regular, entre muitas outros....

Porque Groovy?

3) Novos Recursos Groovy

Groovy oferece novos recursos de linguagens que que melhoram o paradigma de programação:

- Pogo's, AST Transformations, Operator Overloading, Builders, Closures e Metaprogramming.

Veremos cada um deles durante o curso.

Porque Groovy?

4)Alta Produtividade

Baixa curva de aprendizado + novos recursos oferecidos na linguagem groovy + as facilidades do GDK = **programadores Groovy escrevem em torno de 50% a menos de código**, elevando exponencialmente a produtividade na entrega de soluções Java.

Porque Groovy?

5)Integrar Java e Groovy no mesmo projeto

No mesmo projeto, pode-se criar e usar classes Java ou classes em Groovy. As classes feitas em Java, podem usar objetos de classe feitas em Groovy e vice versa.

Compilador “groovyc” compila ambas, não tendo diferença no bytecode java ou groovy.

Comece hoje usar groovy em projeto antigos ou aplique groovy em camadas específicas. Use o melhor de cada um, no lugar correto.

Porque Groovy?

6) Tipagem Dinâmica por Padrão mas com Tipagem Estática como Opção

Em groovy é possível fazer o melhor dos dois mundos:

1. Programação fortemente tipada, como a linguagem Java é.
2. Programação dinâmica (fracamente tipada) como as linguagens modernas são hoje.
3. Independente da sua opinião sobre, no groovy tem a opção de utilizar as duas opções, quando quiser e ou até misturar.

Porque Groovy?

7)Usar Groovy com qualquer outro framework Java

Classes groovy podem ser utilizada com qualquer outro tipo de framework Java, sendo especificação ou proprietário. Swing, IReport, JSF, JPA, Spring, JUnit, VRaptor, Hibernate etc...

Alguns deles com nenhum tipo de configuração especial.

Alguns outros com algumas configurações próprias para groovy e ou com otimizações específicas.

Porque Groovy?

8)Usar Groovy para criar Apps Android

Desde 2014, a partir da versão 2.4x do groovy é possível desenvolver native apps android usando a linguagem groovy.

Porque Groovy?

9) Usar ecossistema Groovy

A linguagem cresceu tanto, a comunidade também, que hoje temos um **ecossistema** completo de produtos em volta do groovy.

Veja:
http://www.groovy-lang.org/ecosystem.html

Groovy

Discussão, comentários e dúvidas?

Ambiente de Desenvolvimento

Ambiente Desenvolvimento Groovy

Ferramentas Utilizadas:

- JDK8x.
- Eclipse: https://eclipse.org/
- Eclipse Groovy Plugin - https://github.com/groovy/groovy-eclipse/wiki

Não precisa fazer download, todas já estão no zip do curso e plugin do groovy já está configurado no eclipse.

Ambiente Desenvolvimento Groovy

- Download material do curso.
- Instalar JDK8x.
- Eclipse Neon.
- Tunning de Eclipse.

Vamos groovar?

Do Java ao Groovy

Regra Geral

Groovy é Java, com as seguintes alterações:

1) Um arquivo/classe em groovy deve ter a extensão "NomeClasse.groovy".

2) Você pode programar Java no Groovy usando a sintaxe padrão.

3) Groovy usa como palavras reservadas: **def**, **in** e **it**, e assim, não podem ser usadas como identificadores.

Do Java ao Groovy

Regra Geral
Groovy é Java, com as seguintes alterações:

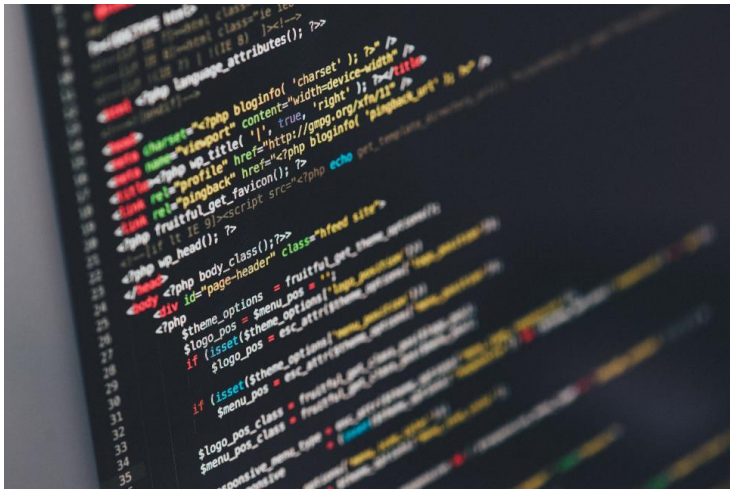
4) Segue abaixo os itens da linguagem Java padrão que **NÃO EXISTE** no groovy:

- Sintaxe de inicialização de array de primitivos.
- Bloco de escopo lógico anônimo de instância.
- Instrução de loop: “do/while”.
- Visibilidade “package default” na ausência de modificar.
- Expressões lambdas do Java 8.

Ambiente Desenvolvimento Groovy

Exercício 1: Criar projeto:

- Adicionar jar's externos.
- MBD -> New Groovy Project -> aula.
- Criar uma classe groovy com main "Ola Groovy" e executar.



Recursos Groovy – Java tem “Dry”

DRY é uma abreviação para o inglês **Don't Repeat Yourself**, "Não se Repita". É o primeiro princípio do desenvolvimento de software mencionado por Andy Hunt e Dave Thomas no clássico livro O Programador Pragmático.

O princípio basicamente declara que não se pode fazer coisas repetidas ou duplicadas ao longo do desenvolvimento de software. Isso ocasiona **alta manutenção**, **aumenta a complexidade**, **reduz a qualidade da solução final** e deixa **chato** e **cansativo** de fazer.

Recursos Groovy – Java tem “Dry”

Java é uma linguagem repletas de DRY!

A OOP prega que todos os atributos de uma classe tem que ser privado segundo as regras de encapsulamento.

Por que a linguagem java assumiu como padrão atributos default e não privado, sendo que eles são os mais usados e os indicados?

Recursos Groovy – Java tem “Dry”

Quantos “private” vc declarou na sua vida como atributos usando a linguagem Java?

Isso não faz sentido...Isso é puro dry....

Groovy resolve todos os problemas de DRY da linguagem java.

Exemplo: todos os atributos em groovy são private por padrão e não precisam ser digitados.

Recursos Groovy – Default Imports

Em groovy, os pacotes abaixo são importados automaticamente por default. Por isso, não é necessário digitar:

- java.io.*
- java.lang.*
- java.math.BigDecimal;
- java.math.BigInteger;
- java.net.*
- java.util.*
- groovy.lang.*
- groovy.util.*

Recursos Groovy – Tipos Primitivos

Em groovy não existe tipos primitivos, tudo é objeto!

Todos os literais primitivos digitado diretamente no código fonte **será automaticamente mapeado e convertido para seu objeto wrapper java padrão.**

Type	Example literals
java.lang.Integer[a]	15, 0x1234ffff, 0b00110011, 100_000_000
java.lang.Long	100L, 200l[b]
java.lang.Float	1.23f, 4.56F
java.lang.Double	1.23d, 4.56D
java.math.BigInteger	123g, 456G
java.math.BigDecimal	1.23, 4.56, 1.4E4, 2.8e4, 1.23g, 1.23G

Recursos Groovy – Tipos Primitivos

Em groovy não existe tipos primitivos, tudo é objeto!

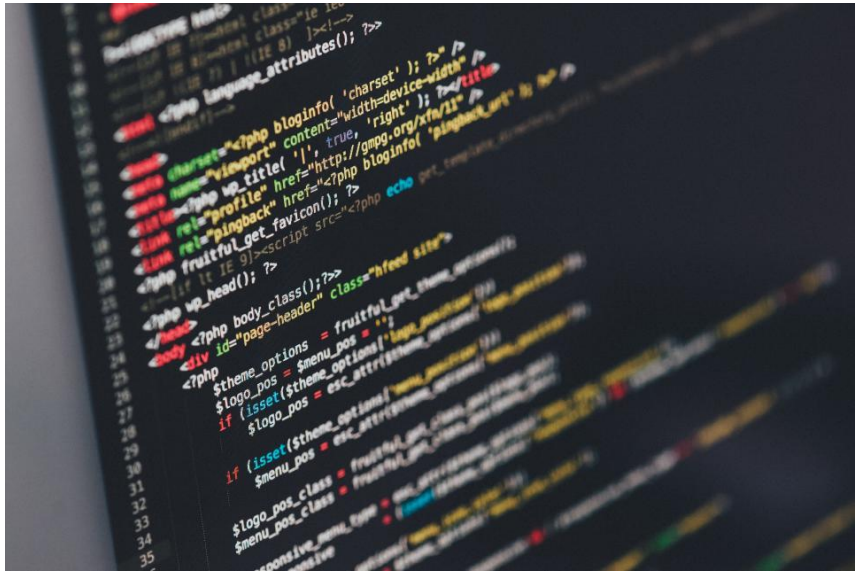
Toda referência a qualquer tipo primitivo, **será automaticamente convertido para seu objeto wrapper padrão.**

Primitive type	Wrapper type	Description
byte	java.lang.Byte	8-bit signed integer
short	java.lang.Short	16-bit signed integer
int	java.lang.Integer	32-bit signed integer
long	java.lang.Long	64-bit signed integer
float	java.lang.Float	Single-precision (32-bit) floating-point value
double	java.lang.Double	Double-precision (64-bit) floating-point value
char	java.lang.Character	16-bit Unicode character
boolean	java.lang.Boolean	Boolean value (true or false)

Recursos Groovy

A partir de agora, usarmos os exercícios em modelo de testes com JUnit. Adicione os jars, crie uma única classe e acumule todos os exercícios.

Exercícios 1.2: Siga o instrutor:



Recursos Groovy - POGO

Visibilidade “**public**” é o padrão para nível de classes e métodos. Por isso, não é necessário digitar.

Visibilidade “**private**” é o padrão para nível de atributos. Por isso, não é necessário digitar.

Recursos Groovy - POGO

Ponto e vírgula “;” **é opcional** para comando em 1 linha e ou quando **não houver ambiguidade na instrução**. Para vários comandos na mesma linha, é obrigatório delimitação com “;”.

Parentes para chamadas de métodos com 1 único parâmetro **é opcional** e ou **quando não houver ambiguidade na instrução**.

Por isso, na maioria dos casos, não é necessário digitar.

Recursos Groovy - POGO

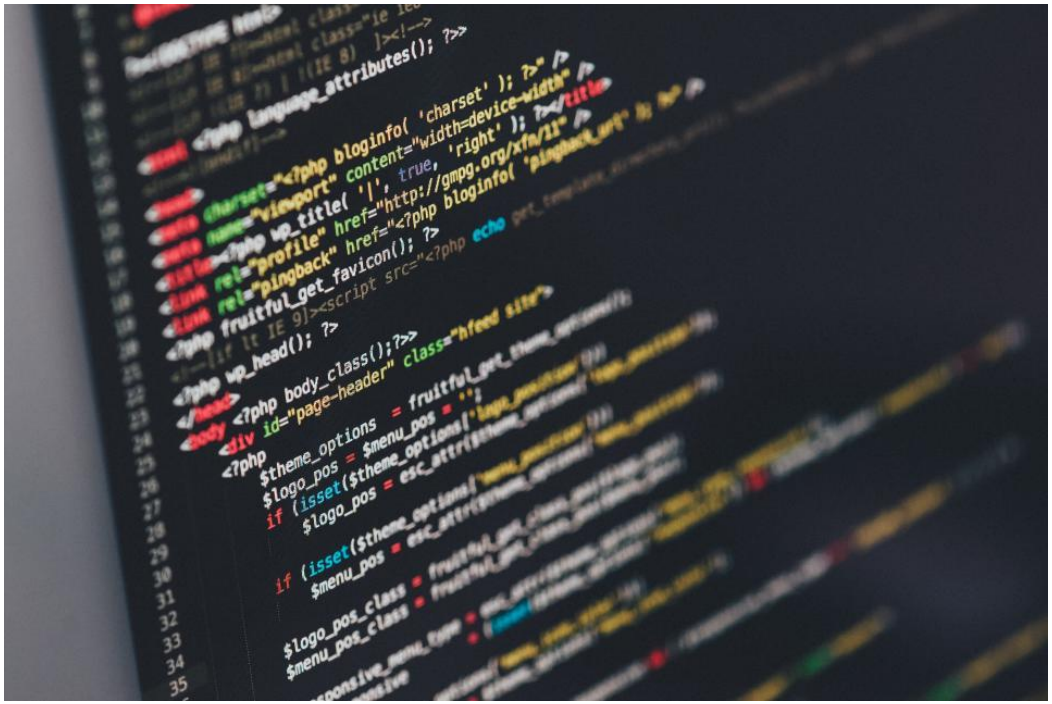
Não é necessário criar os get's e set's, uma vez que eles serão automaticamente e dinamicamente gerados pelo groovy na compilação.

Nos métodos com retornos, não é necessário colocar a palavra "**return**", pois a última linha de qualquer método groovy faz um **return** implícito.

Simplesmente coloque o valor/objeto de retorno como última instrução, última linha.

Recursos Groovy

Exercícios 2: Siga o instrutor:



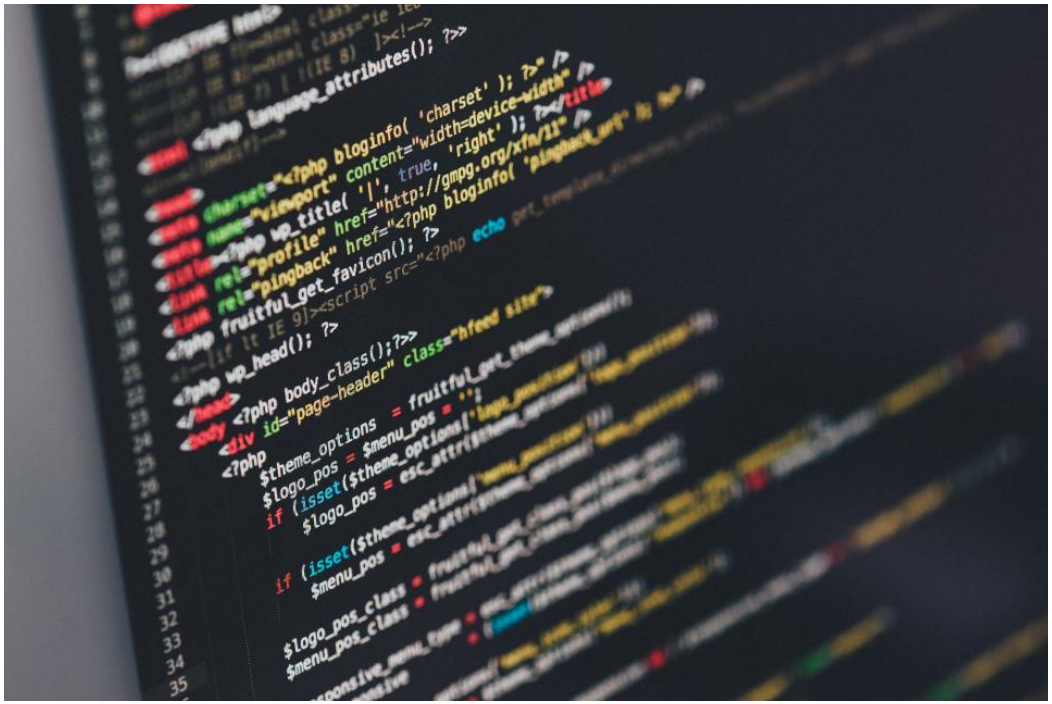
Recursos Groovy - POGO

Não é necessário criar vários construtores sobrecarregados com os parâmetros diversos que preenchem os atributos de um objeto.

Em groovy existe um recursos chamado "**Constructor Names Arguments"** que gera dinamicamente todas as combinações de construtores em forma de "mapa" que atribui valores padrões para cada atributo da classe.

Recursos Groovy

Exercícios 3: Siga o instrutor:



Recursos Groovy

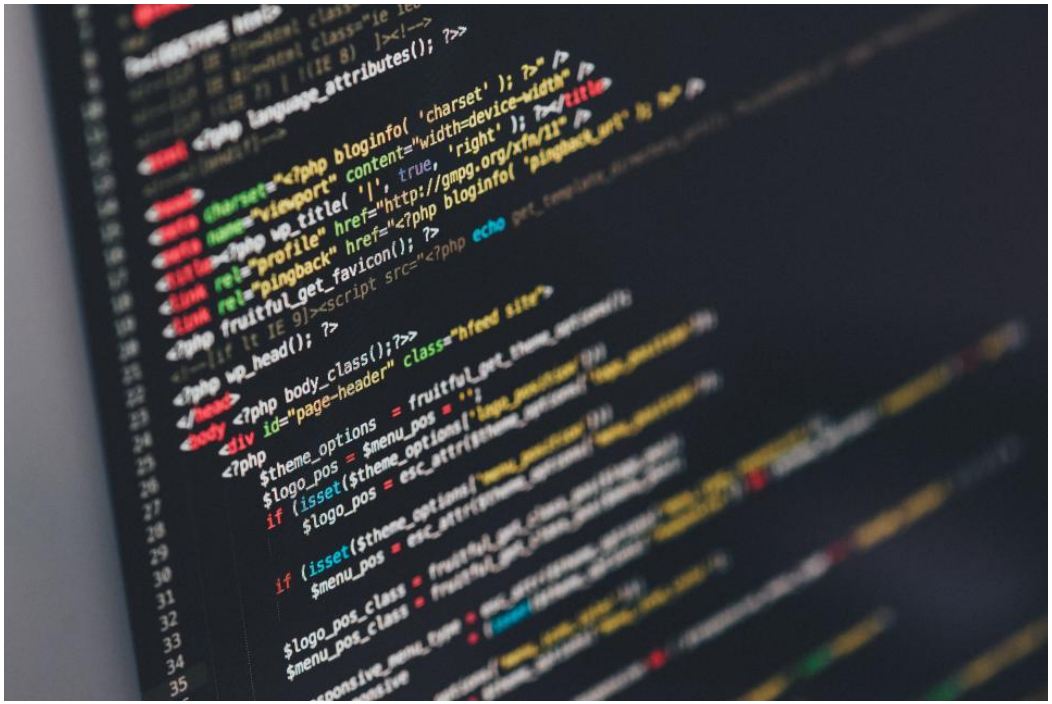
Em groovy existe um recursos chamado "**Subscript Operator**" que dinamicamente usa operador objeto["nomeDoAtributo"] para acessar e manipular os atributos de um objeto.

Ele acessa o get e set da classe.

Muito utilizado para gerar **estruturas dinâmicas** para atribuição de valores em soluções como por exemplo: Swing, Web Apps etc.

Recursos Groovy

Exercícios 4: Siga o instrutor:



Recursos Groovy

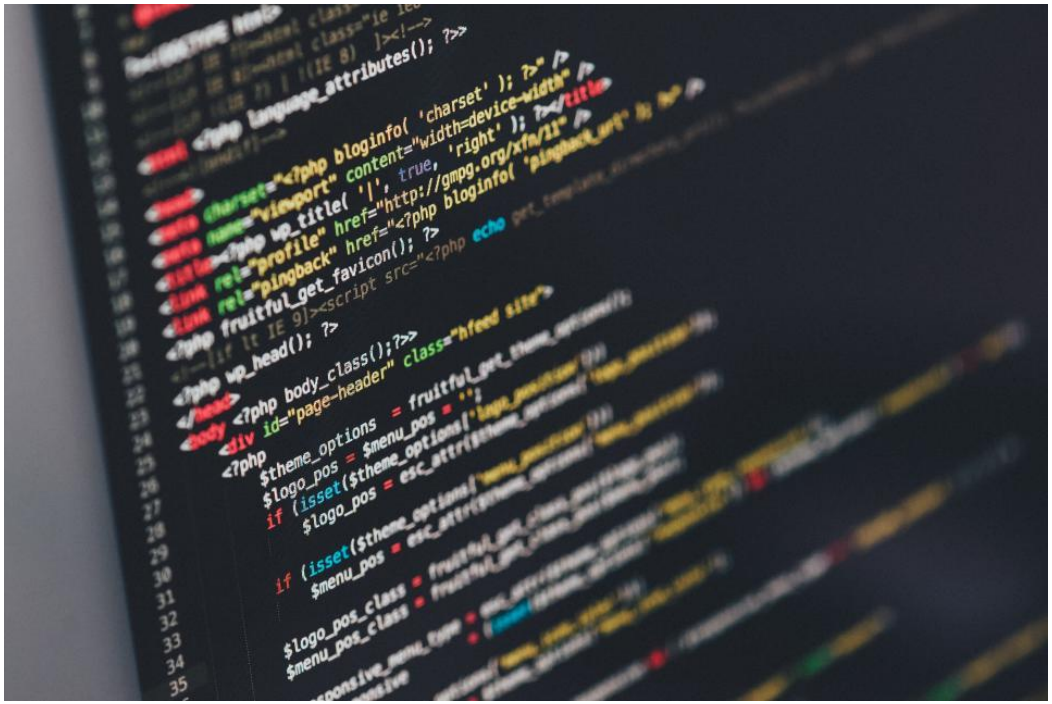
Em groovy existe um recursos chamado "**Direct Field Access Operator**" que usa operador Objeto.atributo para acessar e manipular os atributos de um objeto.

Dinamicamente, é invocado o get e set da classe.

Utilizado para otimizar chamadas de get's e set's Por isso, não é necessário digitar setBlaBla(Bla) e getBlaBla().

Recursos Groovy

Exercícios 5: Siga o instrutor:



Recursos Groovy

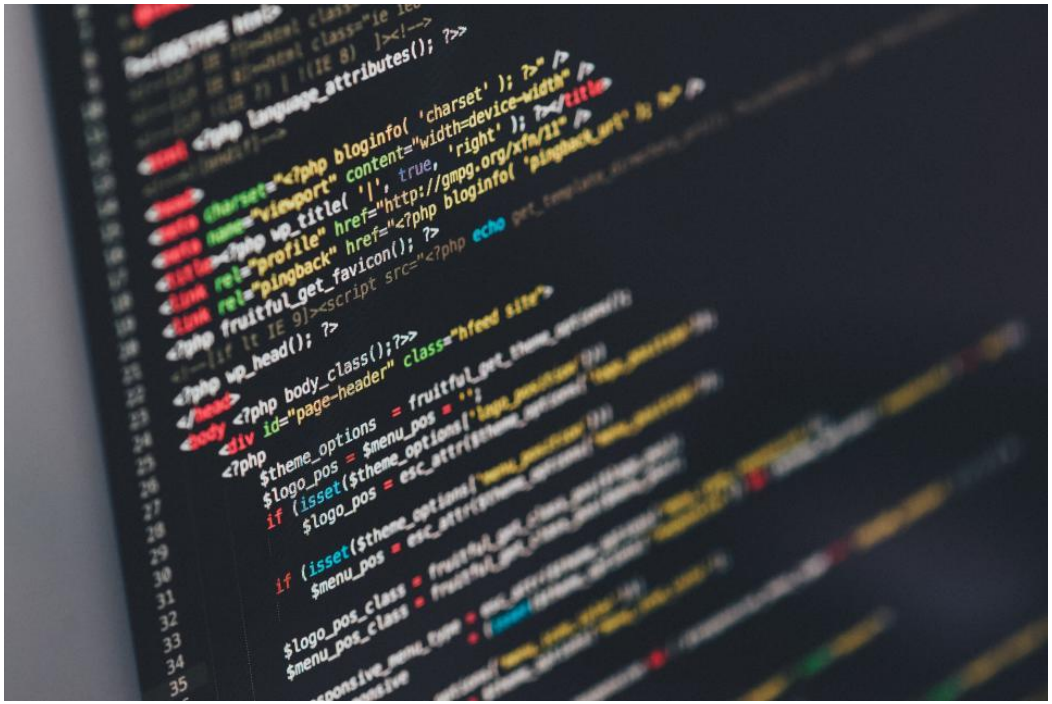
Em groovy existe o operador "**as**" utilizado para criar "alias" reduzidos para nomes de classes e recursos estáticos.

Ele é utilizado para várias *** outras coisas....

Use para reduzir qualquer digitação de código repetida.

Recursos Groovy

Exercícios 6: Siga o instrutor:



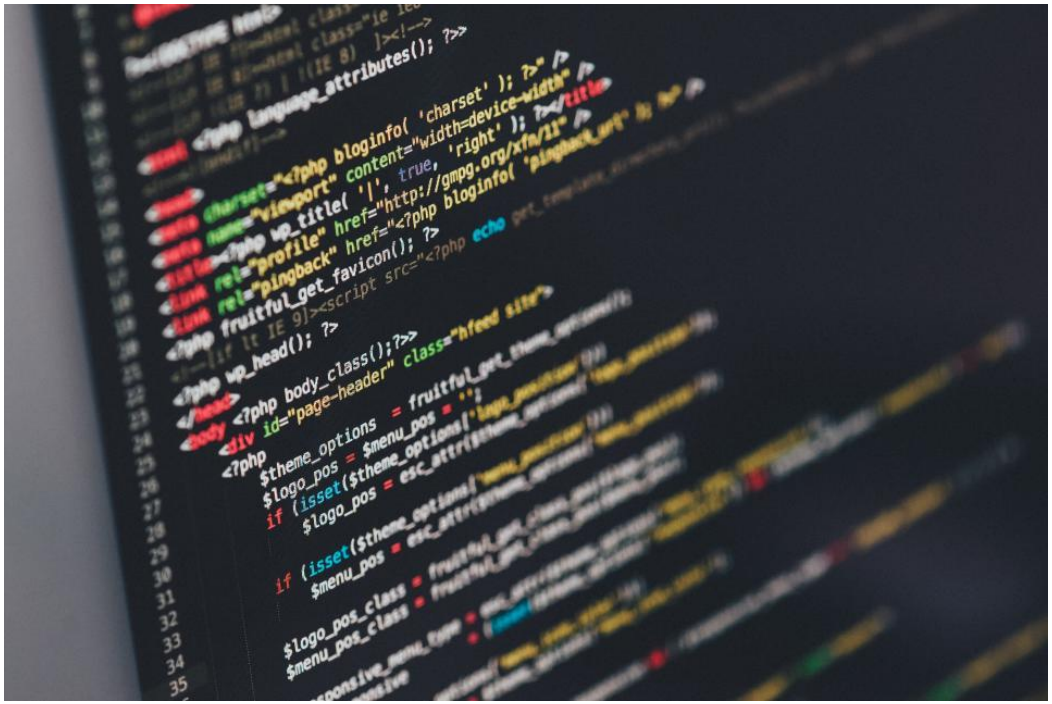
Recursos Groovy

Em groovy existe um recurso chamado de "**Optional Parameters**" utilizado para que os parâmetros de um método assuma valores padrões em caso de não ser devidamente informados na sua chamada.

Groovy vai gerar dinamicamente todas as sobrecargas necessários para cumprir os parâmetros opcionais. Por isso, não é necessário digitar.

Recursos Groovy

Exercícios 7: Siga o instrutor:



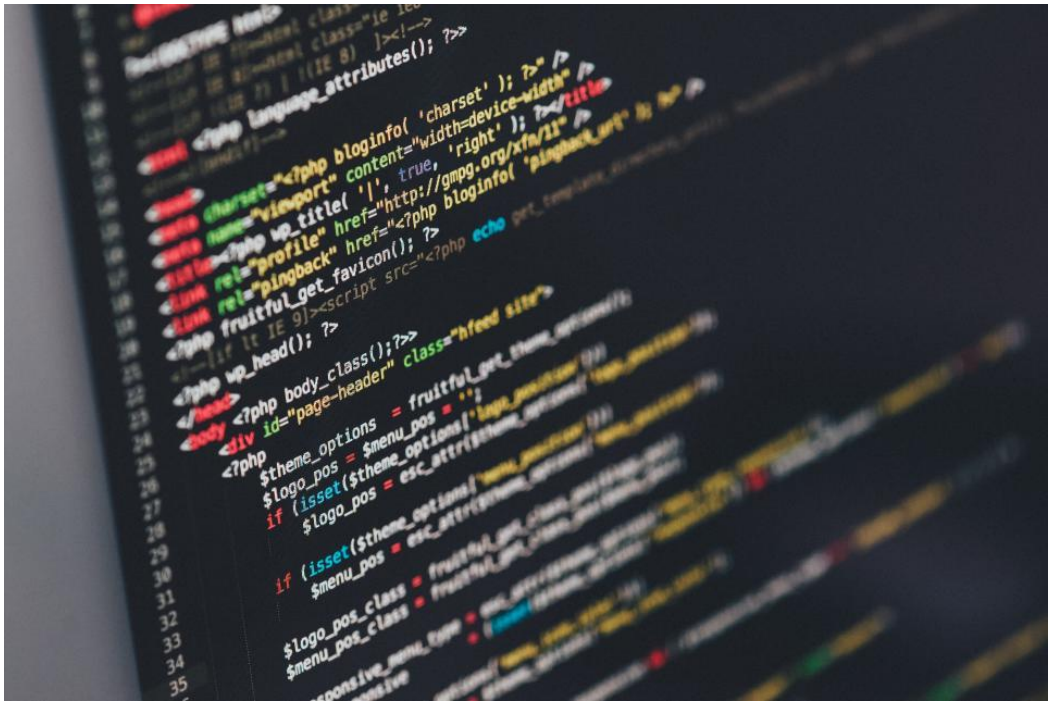
Recursos Groovy

Em groovy existe um recurso chamado de "**Array Optional Parameters**" utilizado para que os parâmetros de um método como array possa ser passado de forma simples, utilizando vírgulas.

Groovy vai gerar dinamicamente a criação do array e a passagem correta do método. Por isso, não é necessário digitar.

Recursos Groovy

Exercícios 8: Siga o instrutor:



Recursos Groovy

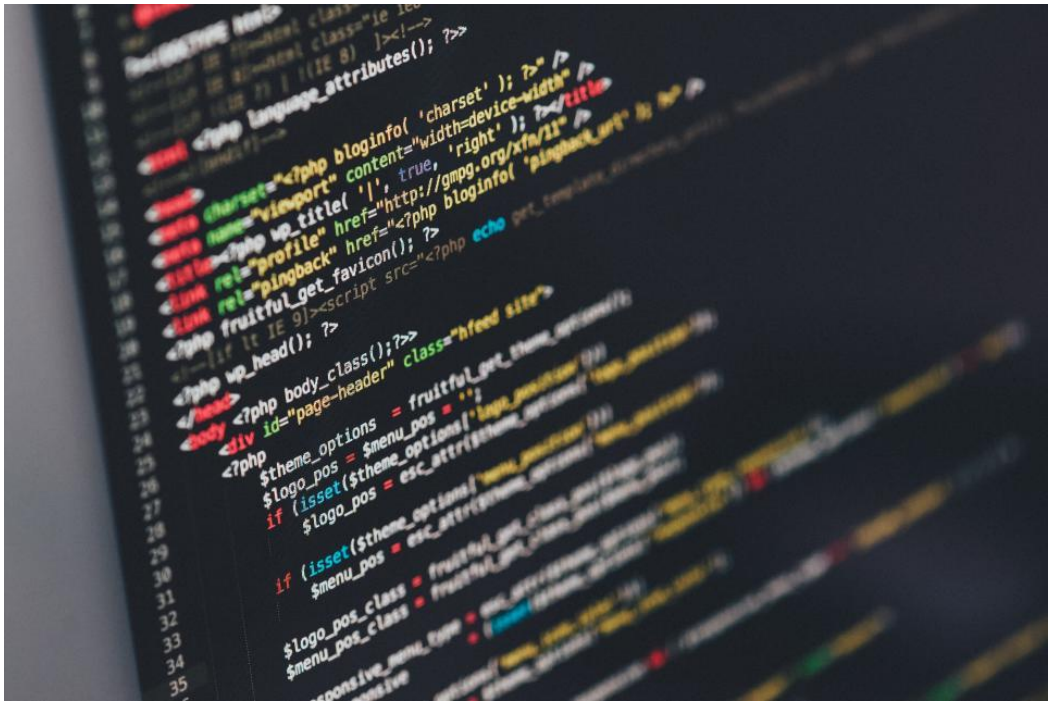
Em groovy existe um recurso chamado de "**Safe Navegator Operator**" utilizado para evitar o famoso NullPointerException quando referências as objetos não estão devidamente apontadas.

- objeto?.executar()..... objeto?.setNome("bla").....

Groovy, dinamicamente vai fazer uma verificação de "if" como null e não executar a operação. Por isso, não é necessário digitar.

Recursos Groovy

Exercícios 9: Siga o instrutor:



Recursos Groovy

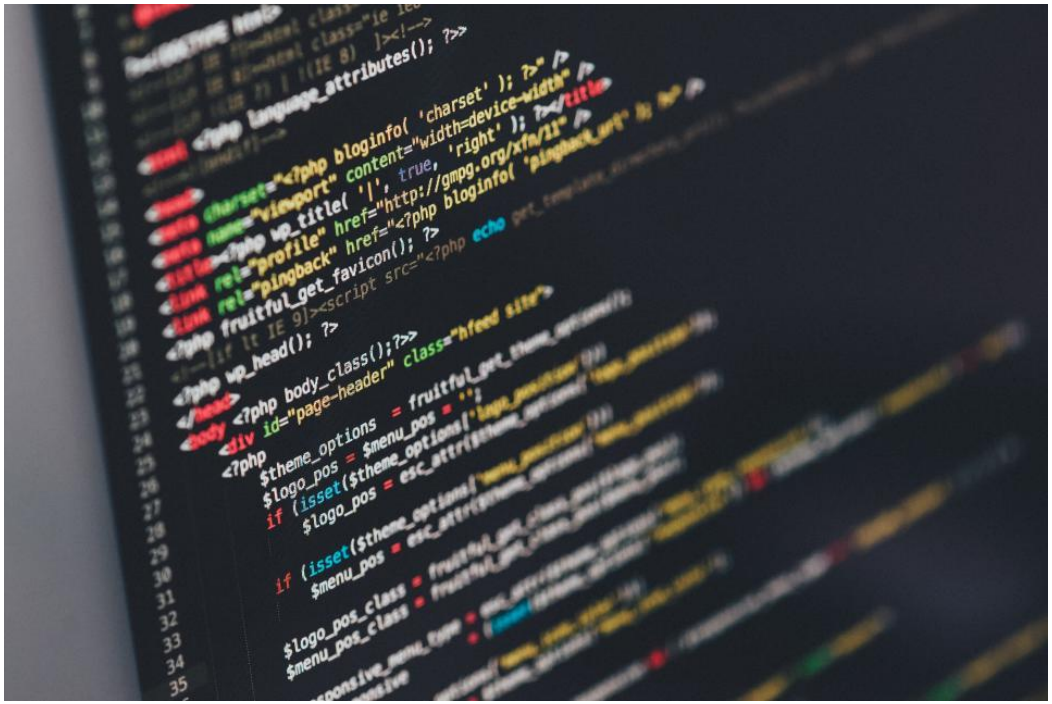
Em groovy existe um recurso chamado de "**Spread Operator**" utilizado para executar comportamentos em bloco sobre coleções.

- colecao*.executar()
- colecao*.setNome("bla")

Groovy vai dinamicamente iterar na coleção e executar a determinada operação em bloco. Spread usa o operador "safe ?" por padrão. Por isso, não é necessário digitar.

Recursos Groovy

Exercícios 10: Siga o instrutor:



Recursos Groovy

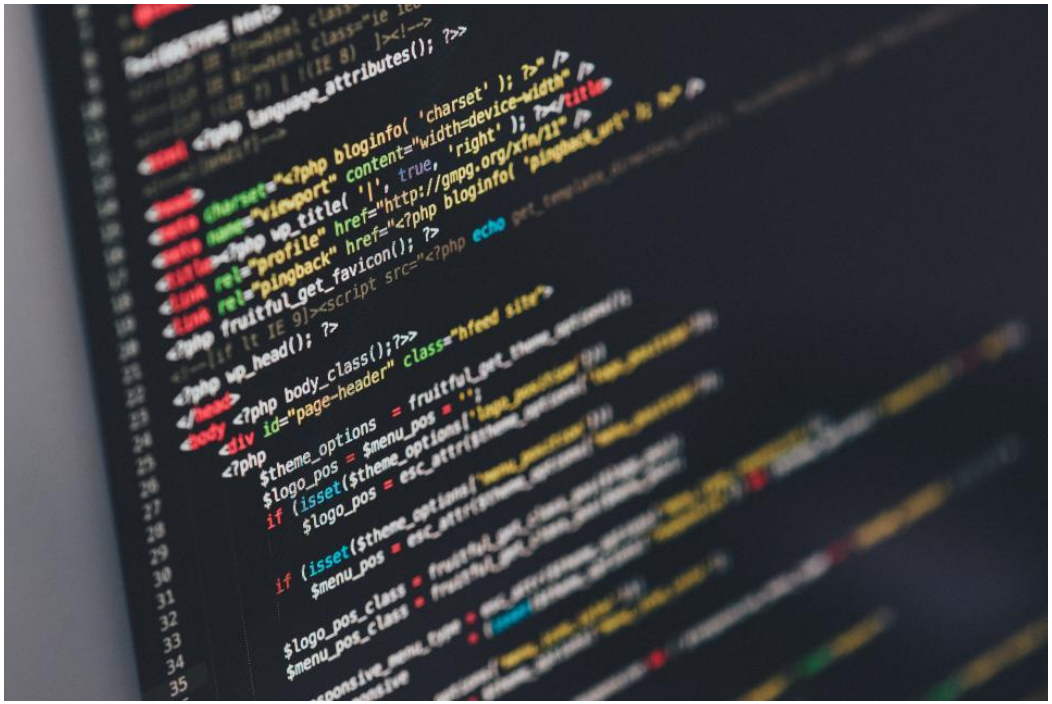
Em groovy as “checked exceptions” **são opcionais**. Ou seja, o desenvolvedor não é obrigado a digitar try/catch para executar as operações checadas.

Em caso de acontecer a exceção, ela será lançada para o método chamador até estourar a pilha de execução da JVM. Por isso, não é necessário digitar.

Cada um fica livre em tratar ou não. Equipes ágeis com cobertura de **testes automatizados** não precisam usar try/catch, reduzindo e muito o código da solução final.

Recursos Groovy

Exercícios 11: Siga o instrutor:



Recursos Groovy

Em groovy existe um recurso chamado de "**Boolean Avaluation**" utilizado para avaliar qualquer tipo de expressão como resultado boolean.

De forma que não existe necessidade de digitar qualquer coisa extra para avaliar a expressão para boolean.

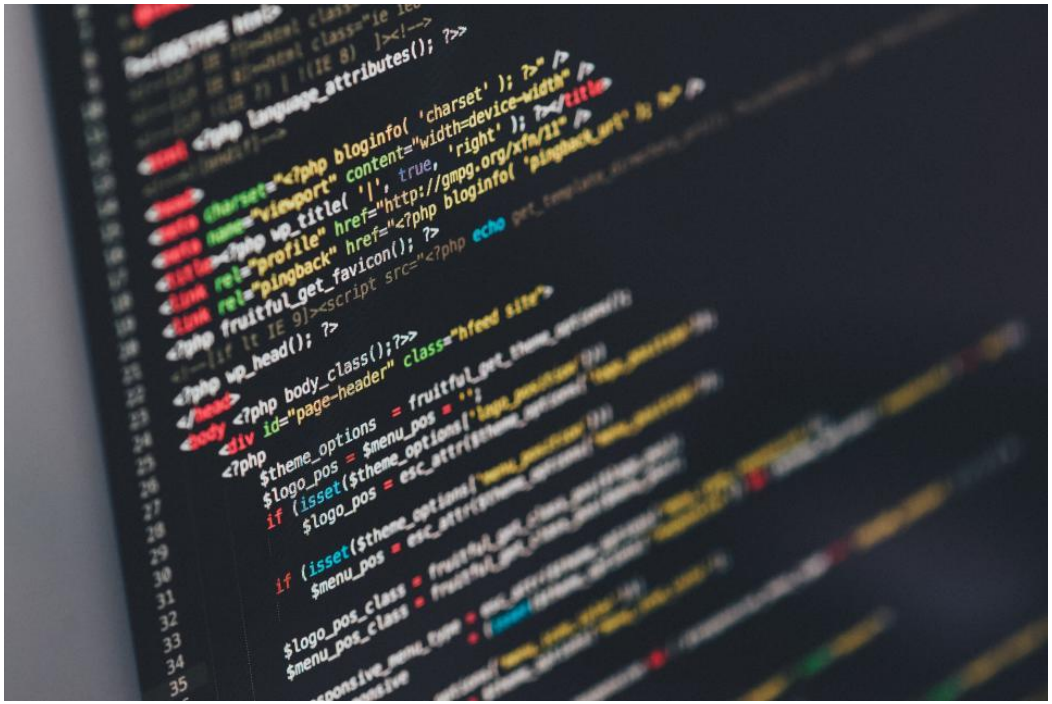
Segue a listagem das regras:

Recursos Groovy

Type	Condition for Truth
Boolean	True
Collection	Not empty
Character	Value not 0
CharSequence	Length greater than 0
Enumeration	Has more elements
Iterator	Has next
Number	Double value not 0
Map	Not empty
Matcher	At least one match
Object[]	Length greater than 0
Any other type	Reference not null

Recursos Groovy

Exercícios 12: Siga o instrutor:



Recursos Groovy

Em groovy existe um recurso chamado de "**Operator Overloading**" utilizado para criar expressões de operadores com qualquer tipo de objeto.

Utilizado para criar uma melhor **expressividade** e **otimização** na utilização de objetos com os operadores.

Segue a listagem das opções:

Recursos Groovy

Operator	Method	Operator	Method
`+`	a.plus(b)	`a[b]`	a.getAt(b)
`-`	a.minus(b)	`a[b] = c`	a.putAt(b, c)
`*`	a.multiply(b)	`<<`	a.leftShift(b)
`/`	a.div(b)	`>>`	a.rightShift(b)
`%`	a.mod(b)	`++`	a.next()
`**`	a.power(b)	`--`	a.previous()
`|`	a.or(b)	`+a`	a.positive()
`&`	a.and(b)	`-a`	a.negative()
`^`	a.xor(b)	`~a`	a.bitwiseNegative()

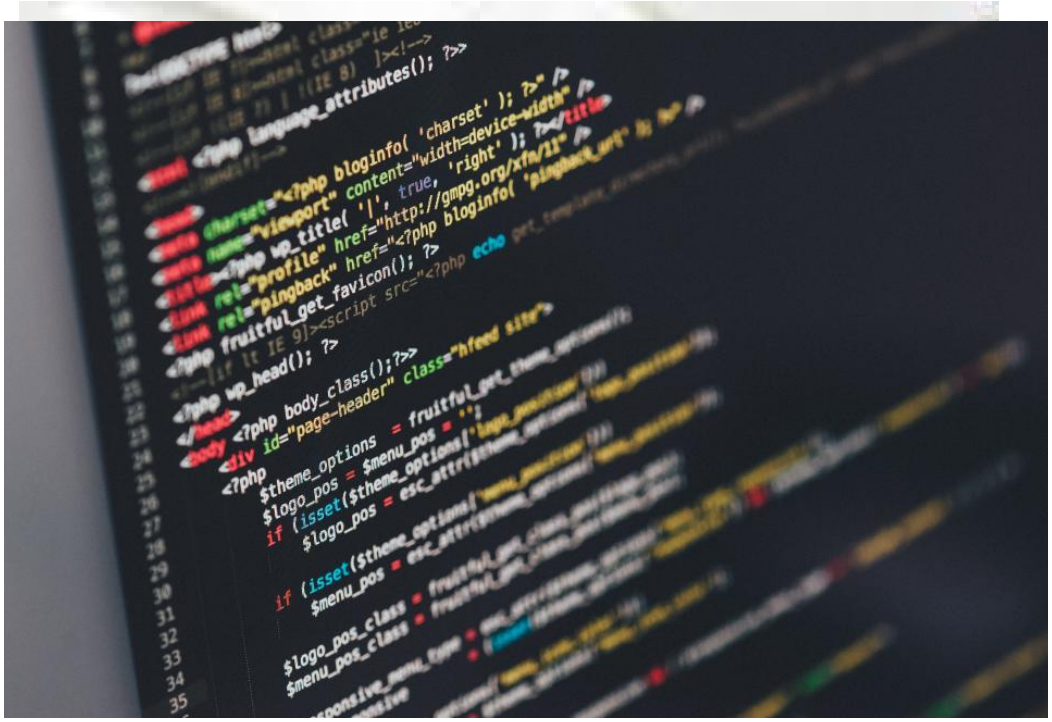
Recursos Groovy

Métodos padrões do "**Operator Overloading**":

- Devem ser públicos.
- Devem ser nomeados exatamente como indicado na tabela.
- Devem retornar o mesmo objeto do tipo operado a esquerda.
- O parâmetro depende do tipo do objeto a direita do operador, de forma que é possível operar com objetos diferentes.

Recursos Groovy

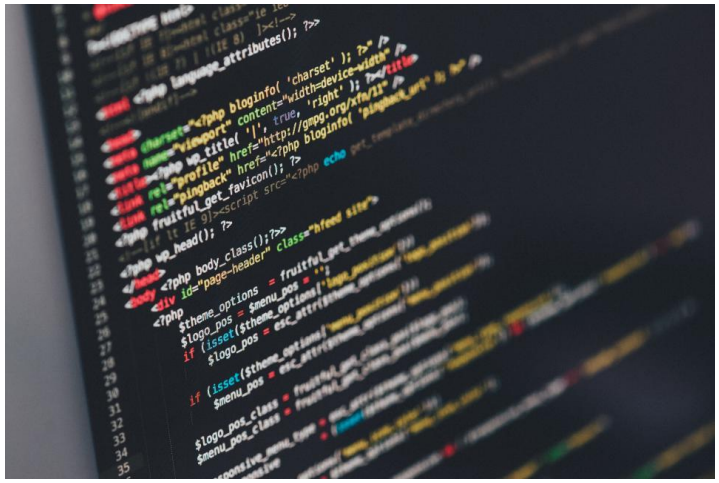
Exercícios 13: Siga o instrutor:



Recursos Groovy

Várias classes da JDK foram implementadas para automaticamente funcionar com a maioria dos operadores.

Exercícios 13.2: Siga o instrutor:



Recursos Groovy

Trait é uma nova construção estrutural em groovy que mistura regras de **interface** + classes **concretas abstratas.**

Trait é uma interface que pode ter métodos abstratos e é também uma classe abstrata que pode conter estado e métodos concretos.

As outras classes usam o comando **implements** para implementar uma trait herdando todos os seus comportamentos.

As traits podem herdar outras traits e outras interfaces usando comando **extends**.

Recursos Groovy

Todas as regras de **herança, sobreposição** e **polimorfismo** da linguagem Java padrão se aplica da mesma forma.

As traits vieram para facilitar a criação de estruturas OO que eram implementados usando um padrão verboso, burocrático e trabalhoso chamado de "**Java Skeletal Implementations**"´[Java Effective by Joshua Blosh].

Veja: https://10kloc.wordpress.com/2012/12/03/abstract-interfaces-the-mystery-revealed/

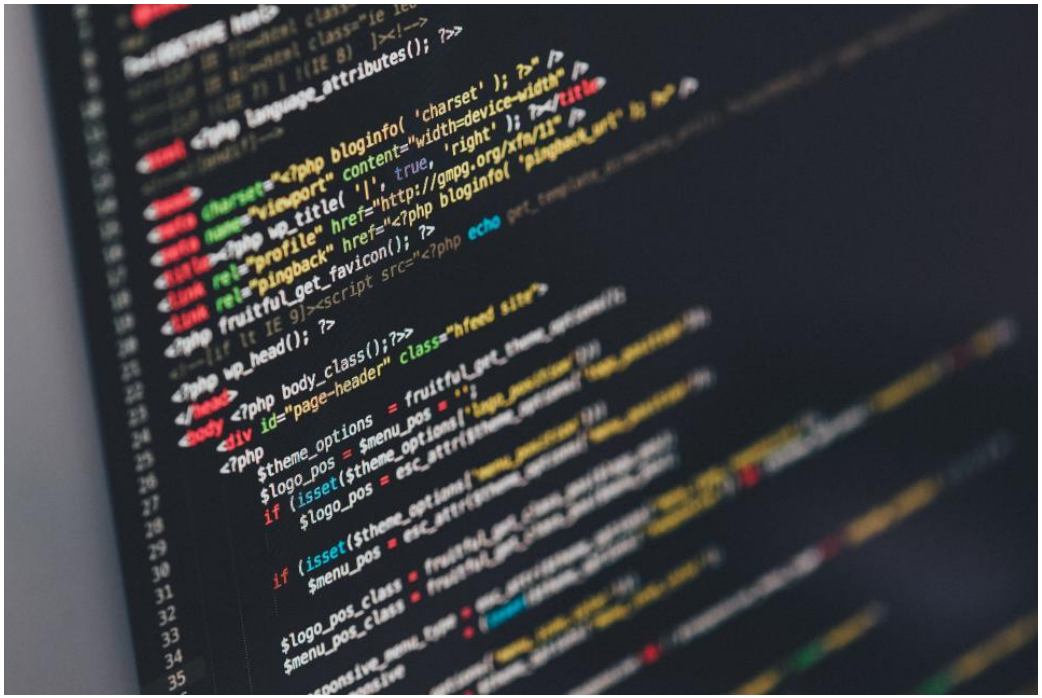
Recursos Groovy

Usando **trait** em groovy é possível fazer **herança múltiplas** no qual uma classe pode implementar várias traits diferentes herdando métodos concretos de varios lados.

Na linguagem Java até a versão 7, não existia recursos para herança múltiplas.

Recursos Groovy

Exercícios 13.3: Siga o instrutor:



Recursos Groovy

Herança múltipla com trait gera novas regras de OOP inexistentes por programadores Java como por exemplo "**Conflito de Herança Múltiplas**".

Para todas as informações, acesse o manual oficial do groovy:

http://www.groovy-lang.org/objectorientation.html#_traits

Recursos Groovy

Na linguagem Java padrão, o nível de visibilidade conhecido como **package** é indicado com a **ausência** de qualquer modificador de acesso. No groovy isso não funciona em virtude dos recursos de POGO.

Sendo assim, no código groovy **não existe mais esse nível de visibilidade!!!**

Linguagem Dinâmica

Linguagem Dinâmica

A linguagem java é estática e fortemente tipada:

É uma linguagem que faz a **verificação** dos tipos de dados (objetos) para **garantir** que o programa execute corretamente os comportamentos esperados em todas as situações. Esta verificação é feita **previamente** no código fonte pelo processo de "compilação".

O compilador **fornece garantias** que problemas não poderão ocorrer em tempo de execução, após o programa passar por esta verificação, ou seja, erros são detectados logo, antes do programa ser efetivamente executado.

Linguagem Dinâmica

Groovy é uma linguagem dinâmica:

Linguagem dinâmica é um termo usado na engenharia de software para descrever uma classe de linguagens de programação que, em tempo de execução conseguem **acrescentar dinamicamente** comportamentos não existentes no programa, **adicionando, substituindo ou removendo comportamentos**, estendendo classes e objetos, ou modificando o sistema de tipos.

Linguagem Dinâmica

Diferentemente das antigas linguagens “fortemente tipadas”, as linguagens dinâmicas **não possuem garantia de tipos e comportamentos** na fase de compilação, utilizada anteriormente para restringir e garantir o sistema tipos em tempo de execução.

A principal diferença das linguagens dinâmicas é que o **processo de verificação** é feita em **tempo de execução**. Isto é feito através de uma infraestrutura auxiliar como por exemplo: uma máquina virtual ou uma biblioteca normalmente chamada de “runtime”.

Linguagem Dinâmica

Objetivo dessa nova classe de linguagens é oferecer: mais **flexibilidade**, mais **agilidade** e um **novo paradigma** na criação e manutenção de **soluções** e suas respectivas arquiteturas.

Linguagem Dinâmica

Programação dinâmica em groovy é oferecida através de vários níveis de recursos diferentes:

1. ATS Transformations.
2. Operador def.
3. Closures.
4. Metaprogramação (MOP).

Estudaremos cada um deles!

ATS Transformations

É um recurso utilizado para **gerar dinamicamente** determinados tipos de comportamentos repetitivos nas classes em tempos de compilação, com base em certas anotações.

A cada nova versão do groovy, novas anotações de ATS são criadas para facilitar a vida dos desenvolvedores.

Vejamos as mais usadas:

ATS Transformations

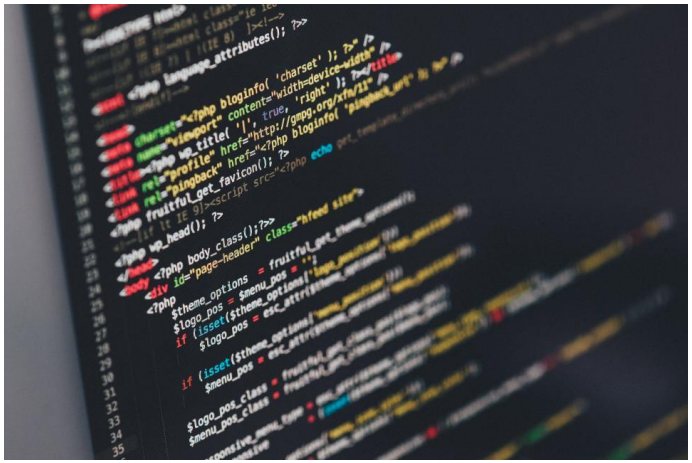
@ToString: utilizada para gerar a sobreposição automática do método toString.

Exercícios 14: Siga o instrutor:

ATS Transformations

@EqualsAndHashCode: utilizada para gerar a sobreposição automática do método equals e hashCode.

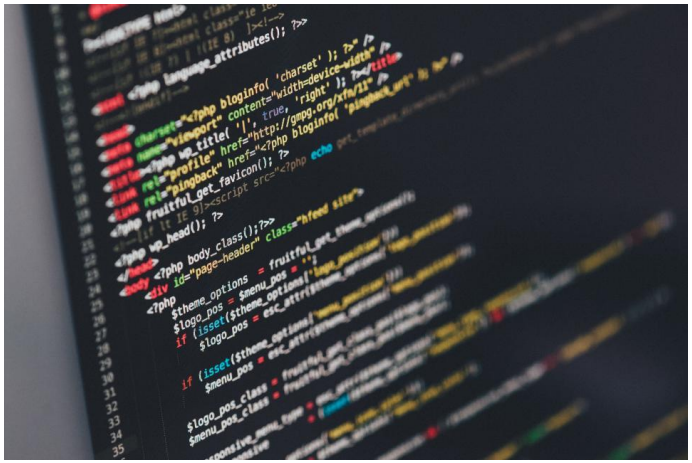
Exercícios 15: Siga o instrutor:



ATS Transformations

@Immutable: utilizada para o design pattern de objetos imutáveis. A classe é dinamicamente gerada somente com atributos get's + toString + equals e hashCode. Qualquer tentativa de alteração gera erro.

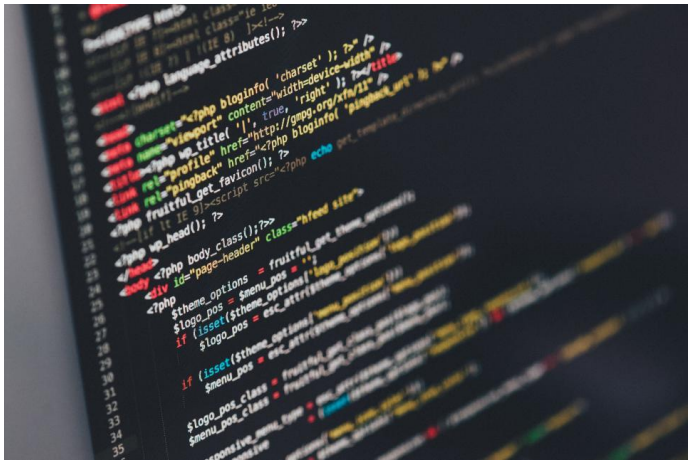
Exercícios 16: Siga o instrutor:



ATS Transformations

@Singleton: utilizada para o design pattern de objetos singleton. A classe é dinamicamente gerada apenas com 1 única instância chamada de “instance”. Qualquer tentativa de criar uma instância, gerará erro.

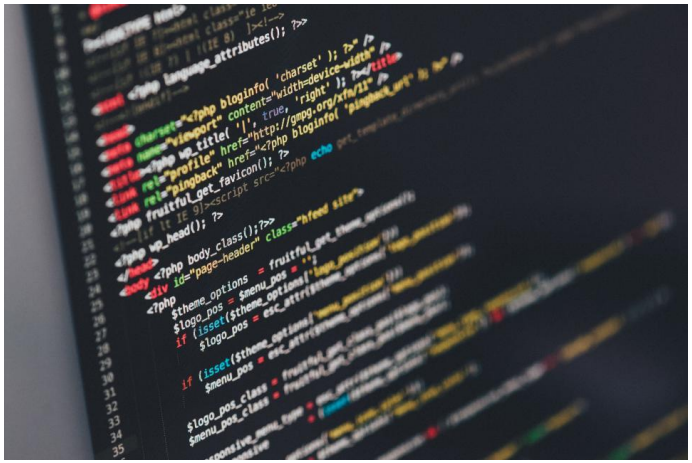
Exercícios 17: Siga o instrutor:



ATS Transformations

@Builder: utilizada para o design pattern de objetos Builder. A classe é dinamicamente gerada seguindo o padrão, sem acesso set's aos atributos.

Exercícios 18: Siga o instrutor:



ATS Transformations

Para todas as opções de anotações ATS, consulte a documentação:

http://www.groovy-lang.org/metaprogramming.html#_available_ast_transformations

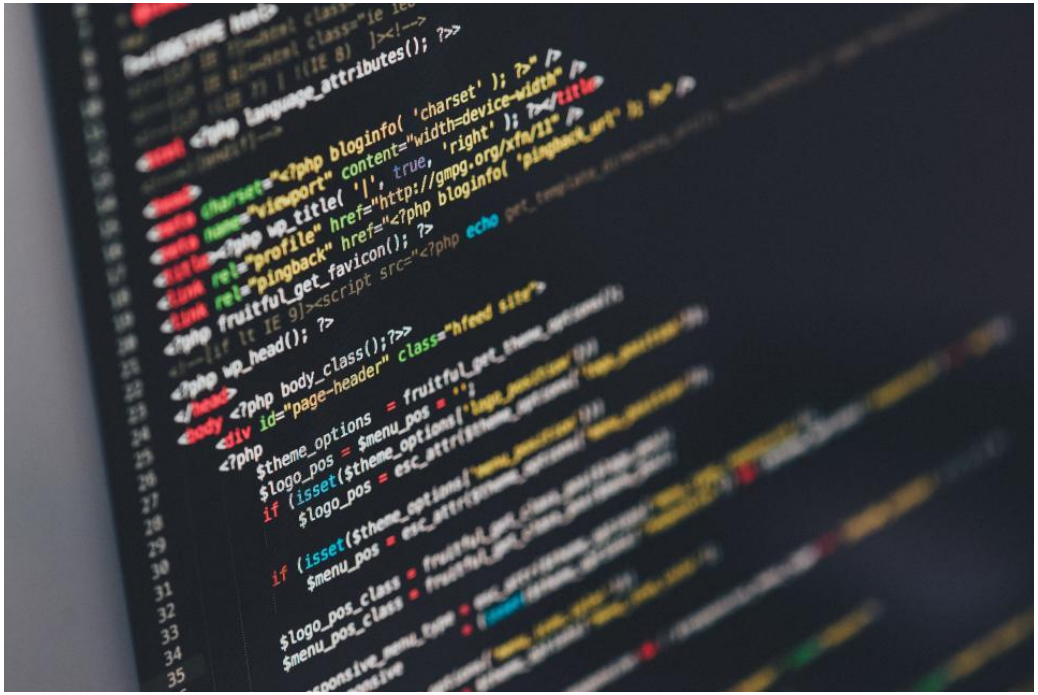
Operador def

Operador utilizado para criar objetos de um **tipo indeterminado**. O objeto assume várias tipagens diferente ao longo da execução da solução.

Em linguagens dinâmicas, não existe **a limitação da preocupação** com tipagem de dados, deixando as referências com **tipagem dinâmicas** ao longo da execução do programa.

Operador def

Exercícios 19: Siga o instrutor:



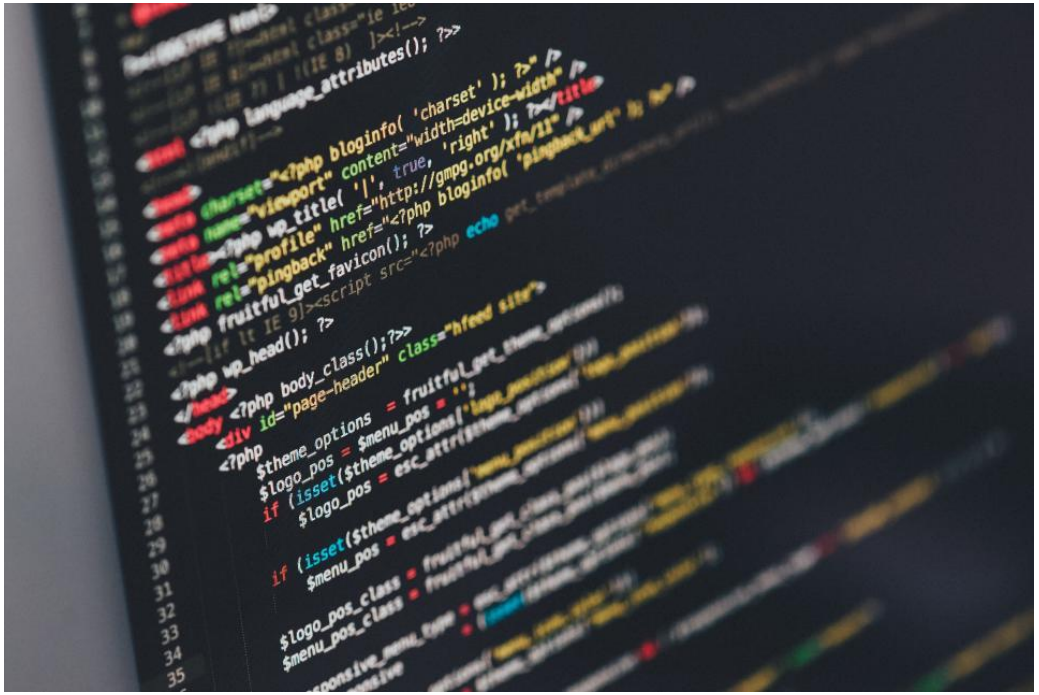
Operador def

Utilizados para passagem de parâmetros e retorno de métodos.

Ausência de tipo na passagem de parâmetros do método, já indica que é do tipo dinâmico. Por isso, não precisa digitar.

Operador def

Exercícios 20: Siga o instrutor:



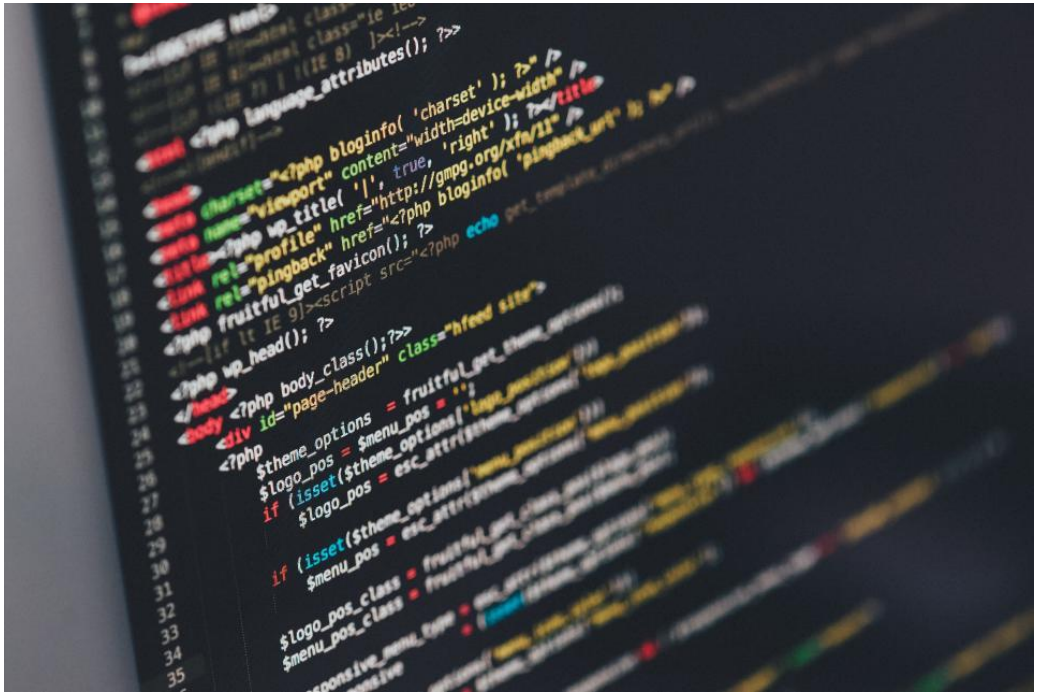
Operador def

Existe um comando **for** chamado de **FOR-IN** no groovy específico para iterar sobre coleções de objetos sem tipo usando **def**.

Dessa forma, é possível fazer a iteração com tipagem dinâmica.

Operador def

Exercícios 21: Siga o instrutor:



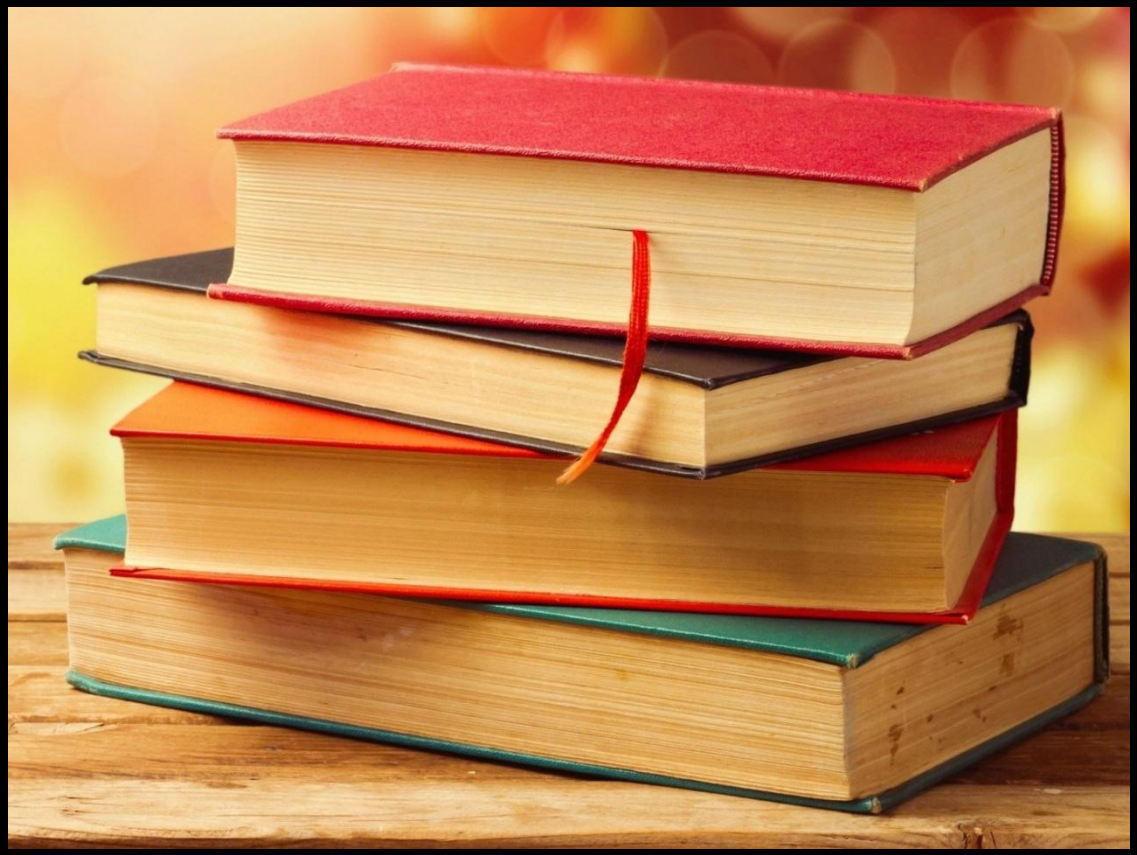
Operador def

Não existe validação de tipagem na compilação que garanta que um objeto tenha aquele determinado comportamento!!!!

Operador def

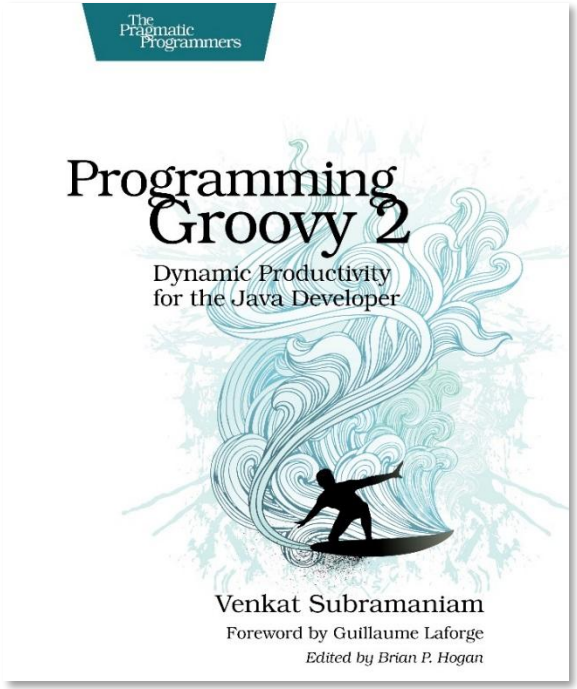
Não confie mais no "code completion", uma vez que não existe nenhuma garantia da existência de metodos ou atributos.

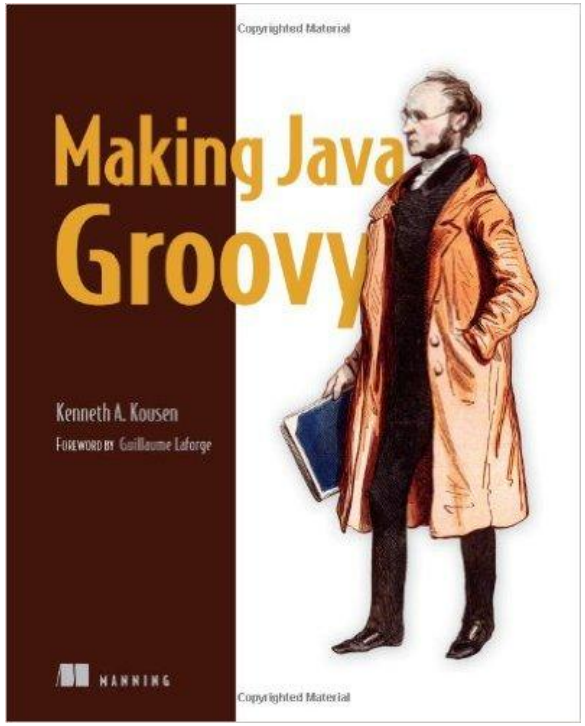
Em caso de um objeto não conseguir executar, será gerado um erro de execução.

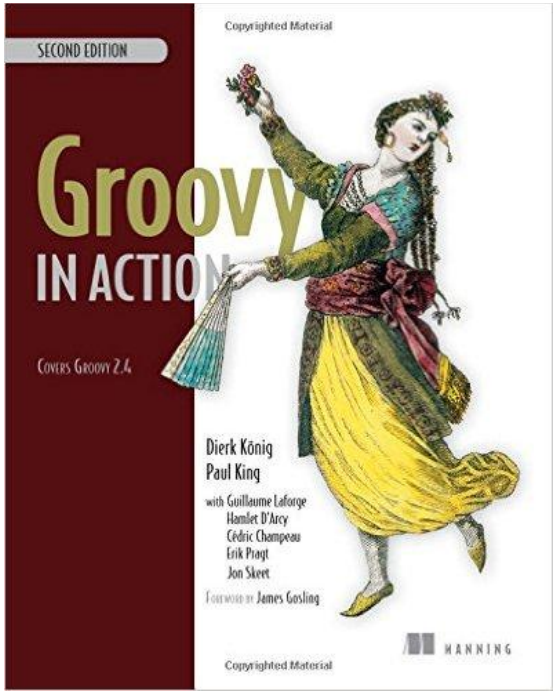


Referência Bibliográficas

Referência Bibliográfica







Groovy

Discussão, comentários e dúvidas?