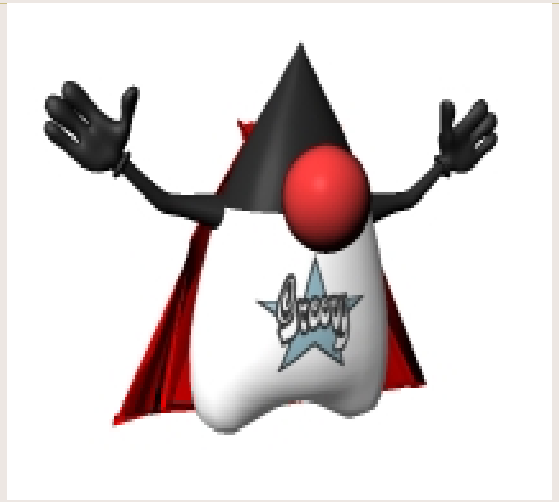
Copyright Notice

Welcome



Introduction to Groovy 3.0

Prakash Badhe
prakash.badhe@vishwasoft.in

About Me

- ✓Software Technologies consultant and mentor
- ✓Enjoying the IT for 20+ years
- ✓Passion for technologies and frameworks
- ✓Mentor and Promoter for Agile technical practices
- ✓Specialized in Java Technologies and Frameworks
- ✓Worked on medium and large scale dynamic enterprise application projects.
- ✓ Skilled in Agile Java frameworks
- ✓ Working with Groovy, Grails, Jenkins, Gradle.

About you..

- ✓**Prerequisite** : Well versed with Java technologies.
- ✓Skilled in Java and JavaEE web and enterprise projects
- ✓**Pl. Tell about..**
- ✓Current Job Role and skill-Sets
- ✓Objectives for Groovy Training
- ✓Your experience in JavaEE Web application development
- ✓Awareness about tools Ant, Maven, Jenkins etc.

Agenda

- Groovy Features
- Groovy Tools
- Groovy language Basics
- Groovy scripts
- Develop Groovy applications
- Groovy-Java integration
- Hands on with Groovy
- Operator overloading
- Database, JSON and YAML and XML Processing
- Groovy Builders
- Groovy Meta Programming
- Groovy usage
- Groovy Application development.

Setup

- Windows 10 64 bit
- Adobe PDF Reader
- JDK1.8 64 bit
- Groovy Grails Tool Suite : GGTS IDE(Eclipse)
- Groovy 3.x
- Firefox Mozilla browser
- Live Internet Connection

- SET JAVA_HOME=d:/JDK1.8/JRE
- SET GROOVY_HOME=d:/apache-groovy-binary-3.0.4
- Set PATH=%PATH%; d:/apache-groovy-binary-3.0.4/bin;

What is Groovy

- The Groovy is the server side scripting language
- Builds on the strengths of Java but has additional power features inspired by languages like Python, Ruby and Smalltalk.
- Groovy makes modern programming features available to Java developers with almost-zero learning curve.
- The syntax is very close to Java language

Why Groovy ?

- Java is the object oriented application development language, but still requires lot of coding and configurations with xml and or annotations.
- Number of Java technology Frameworks supporting higher level API for Web MVC, Data Base mapping like ORM with extensive configurations.
- The typical Java application has lot of dependencies on other third party jar libraries.

Need of Groovy

- Needs single platform that supports higher level API programming with minimum configurations and faster deployments.
- Need dynamic scripting for fast development and build up.

Usage of Groovy

- ✓It is meant for Java developers.
- ✓It is a powerful high level language on top of Java platform.
- ✓The Groovy code gets compiled to Java byte-code.

Tools on top of Groovy

Gradle : Higher level build automation framework based on groovy language.
Gradle works on top of Apache Ant and Maven tools.
Gradle utilizes groovy to define the build script for your build process.
No need to compile this script programs in Gradle.
Griffon : Groovy based framework for desktop applications.
Spock : Testing framework utilizing the groovy language.
SoapUI : Soap and REST web Services testing framework based on groovy

Grails on Groovy

Higher level framework based on groovy language.
Designed to fast develop and build J2EE web applications for enterprise.
Utilizes number of Java frameworks internally as Spring,Hibernate,JUnit,Ant, Maven etc.
Supports Agile development for Java applications.
Provides solutions for short comings over other frameworks.

Groovy Features

1. Groovy is an **agile** and **dynamic language for the Java platform.**
2. Groovy provides the ability to statically type check and statically compile your code
3. Supports Domain-Specific Languages and other compact syntax so the code becomes easy to read and maintain.
4. With groovy writing shell and build scripts easy with its powerful processing primitives, OO abilities.
5. Use groovy compiler just like *javac* to produce byte-code at runtime.
6. Groovy integrates with java code easily.

Groovy advantage

- Increases developer productivity by reducing scaffolding code when developing web, GUI, database or console applications.
- Simplifies testing by supporting unit testing and mocking out-of-the-box.
- Groovy seamlessly integrates with all existing Java classes and libraries.
- Groovy code is compiled straight to Java byte-code so you can use it anywhere you can use Java.
- The performance improvements in execution as well as lower memory consumption.

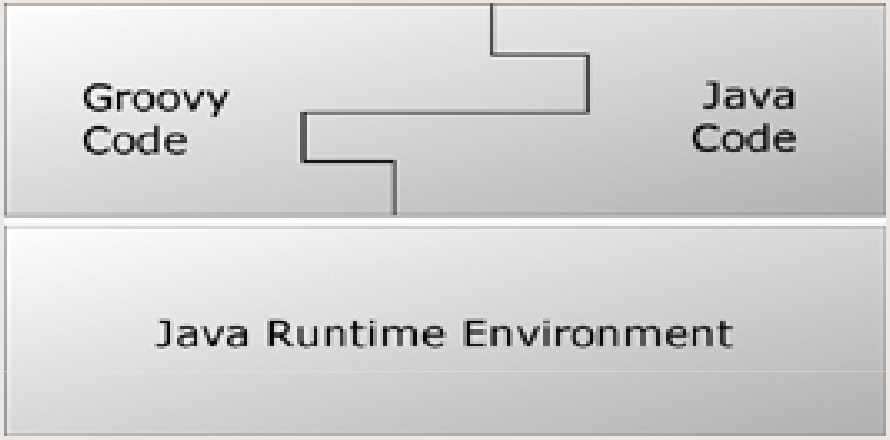
Groovy with Java

- Groovy is like a super version of Java.
- It can leverage Java's enterprise capabilities but also has cool productivity features like closures, builders and dynamic typing.
- If you are a developer, tester or script guru, you have to love Groovy.

Groovy Syntax

- Same java keywords and additional java keywords
- Same Java syntax for comments.
- No semicolon required for end of line, but multiple declaration statements on same line should be separated by the semicolon.
- Data-typing not needed on variable declarations.
- Same keyword ‘def’ for variable and method declarations.
- More defaults and conventions for groovy programs.

Groovy RunTime



- ✓Groovy runs with Java.
- ✓Groovy extends java library and adds more powerful features required for creating scalable, fast applications in short time.

Groovy Scripting

- Supports all java features as it is.
- The short and concise syntax
- Dynamic data-typing for the variables
- Methods with variable arguments
- Methods with different argument types
- Overloaded operators
- Customizable DSL implementation
- Groovy code re-usable in java and java code re-usable in groovy.

Groovy Setup

Set the User variables on the system

- JAVA_HOME=C:\Program Files\Java\jdk1.8.25\jre
- GROOVY_HOME=D:\apache-groovy-binary-3.0.4
- PATH=C:\Program Files\Java\jdk1.8\jre\bin;D D:\apache-groovy-binary-3.0.4\bin;

Groovy Tools

- The groovy interpreter tools
- The groovy interpreter : groovy.exe
- The groovy compiler : groovyc
- The groovyConsole : Interactive graphical Groovy Editor
- Java2groovy : groovy converter
- Groovydoc: Generates the documentation for groovy classes and scripts.
- Grape : Groovy cache management tool

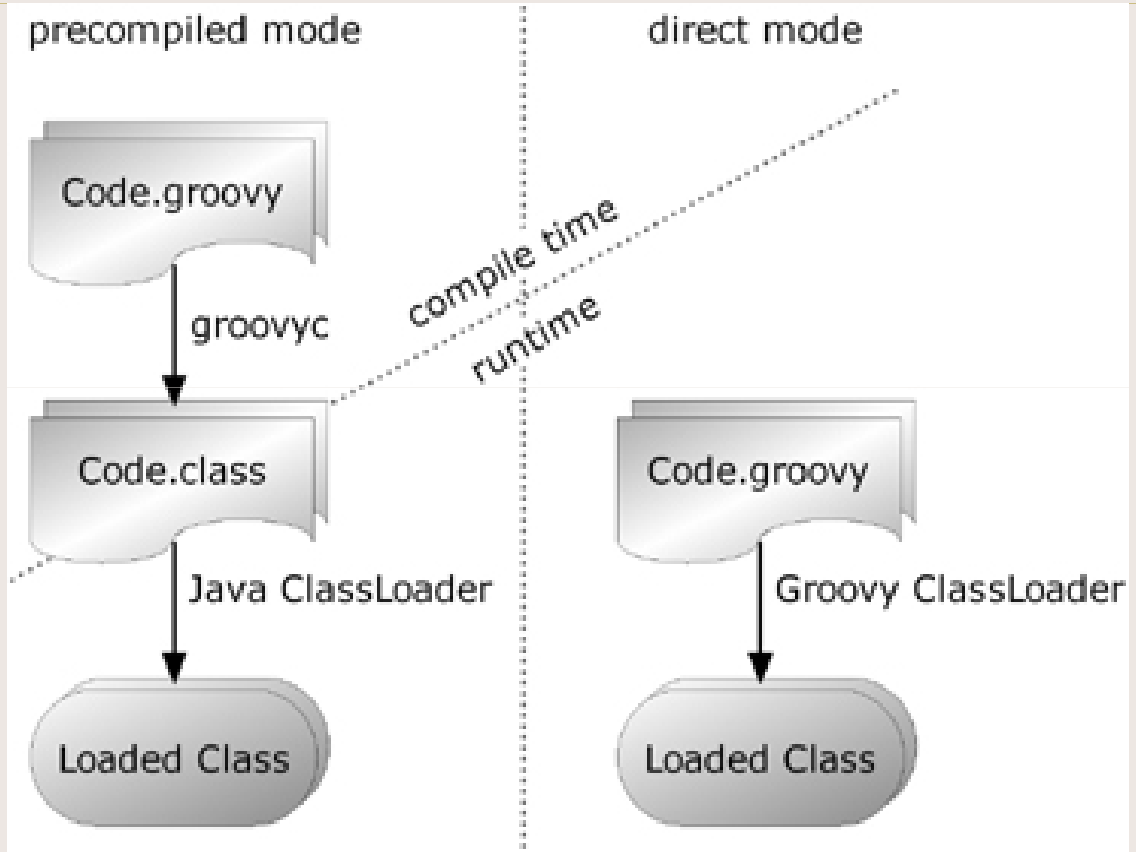
Groovy Keywords

Groovy Language Keywords			
as	assert	break	case
catch	class	const	continue
def	default	do	else
enum	extends	false	finally
for	goto	if	implements
import	in	instanceof	interface
new	null	package	return
super	switch	this	throw
throws	trait	true	try
while			

Groovy is Java

- Any java class source program you can be saved as .groovy file and run as groovy file.
  : groovy MyJavaClass.groovy.
- The groovy interpreter generates the byte code at runtime and runs the byte code.
- Although compilation is not needed, to use it other java programs to compile groovy code to java byte code.
  Compile with groovyc and re-use the groovy class in other java applications.
- The command groovyc MyGroovyClass.groovy produces MyGroovyClass.class output file.
- The groovy language supports shorthand synatx for writing scripts and classes.

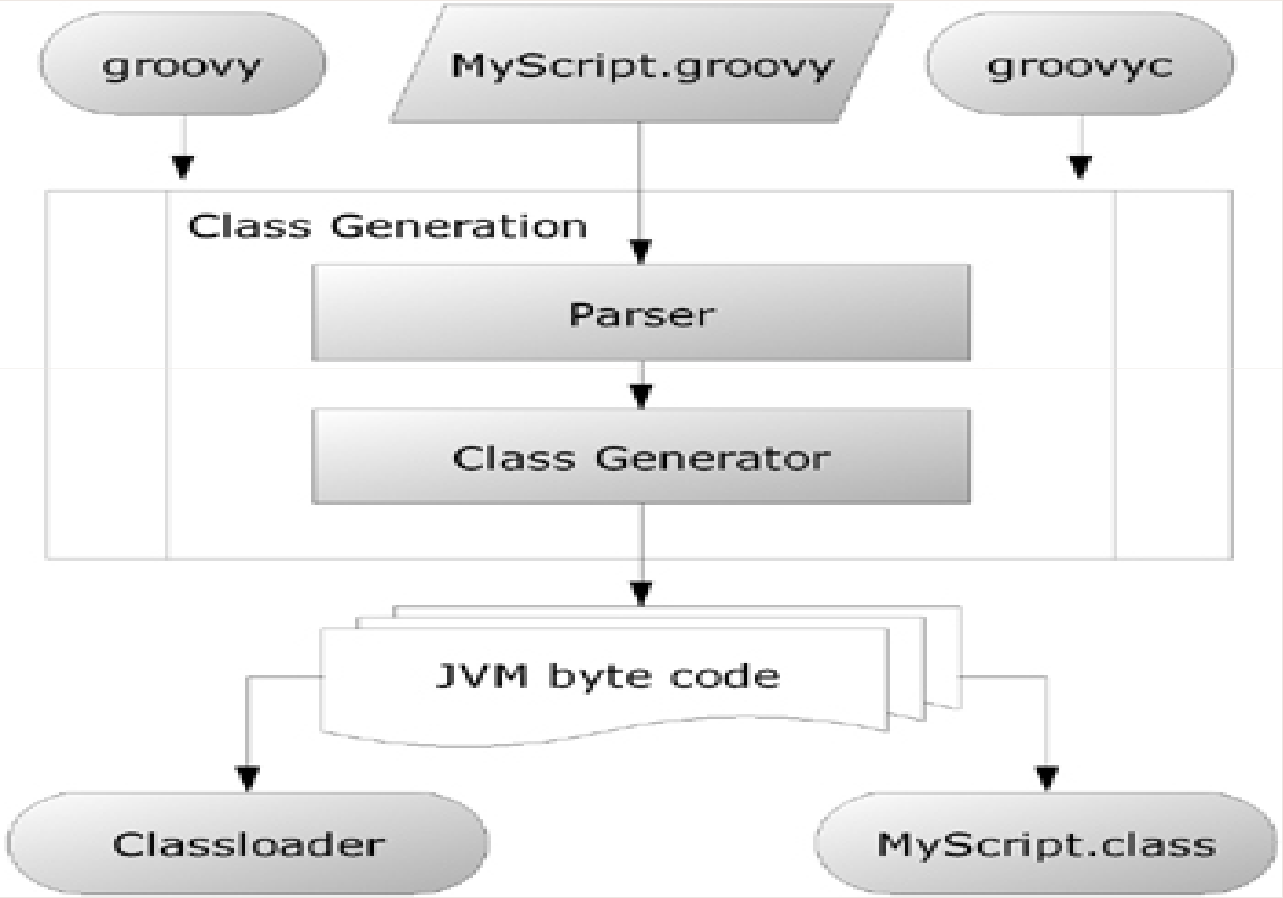
Groovy as Java code



Groovy Interpreter and compiler

- The groovy command dynamically generate the byte code without saving tit in physical .class file and passes it to JVM.
- when groovyc is used the groovy code is physically compiled into java .class files.

Groovy Process



Groovy Script

- It is not necessary to write all the necessary java code into the groovy file.
- Simply add the execution statements in groovy file like : println “Welcome User”.
- This is interpreted as script and generates the required code to run including the static main method in the corresponding java class.
- Behind the scene the groovy class is defined with main method.

Groovy Class

- The groovy class can have main method
- The groovy class can have helper methods.
- Groovy generates additional code into the classes at runtime.
- The groovy looks for static void main in the class being executed and invokes it.
- You can define multiple classes in one groovy file.
- The default access is public.

Comments in Groovy

```
// single line comment
  /* multi- line comments ..
     Here and
       there
  */
```

- #!/usr/bin/env groovy
- The #! *shebang comment is allowed only in the first line. The shebang allows Unix shells to locate the Groovy bootstrap script and run code with it.*
- Javadoc-like comments in /** ... */ markers are treated the same as other multiline comments, but are processed by the *groovydoc Ant task.*

Groovy Defaults

Groovy supports defaults and conventions in every groovy program.

- No semicolon required at the end of statement
- The semicolon when multiple statements written on same line
- More default imports get added in every groovy program.
- Generates setter and getter methods for Groovy Bean class fields.
- Supports built-in operator overloading.
- Supports all features of java.

Groovy Defaults..

- In groovy code the default access is public unless specified otherwise.
- The method calls in Groovy can omit the parenthesis if there is at least one parameter and there is no ambiguity.
- The print “user” is same as print(“User”) or System.out.print(“User”)

Default Imports

The below imports are added as default in every groovy program:

```
import java.lang.*
import java.util.*
import java.io.*
import java.net.*
import groovy.lang.*
import groovy.util.*
import java.math.BigInteger
import java.math.BigDecimal
```

Groovy Dynamics

Groovy brings new dynamic capabilities.

- The data types are supported same as from java language.
- Dynamic data types based on value assigned.
- Groovy supports operator overloading.
- Integrates with java code.
- Dynamic methods, fields can be added at runtime to the groovy classes.

Groovy Data Types

- The same data types as java are supported in
- groovy.
- The groovy scripting supports dynamic types such as
- X = 144
- Uname= “Vijay”
- The data types are identified at runtime depending on the value assigned on right hand side.
- In the above code the X is identified as integer and Uname as String.
- To change the type simply assign another type value.
- Uname = false.. Now it becomes Boolean type.
- Identify the data type: getClass().getName() returns the current type.

Groovy: Primitives are objects

- In Groovy *primitives are first-class objects rather than primitive types as in Java .*
- In Java, you cannot invoke methods on primitive types.
- If x is of primitive type int , you cannot write x.toString() .
- On the other hand, if y is an object, you cannot use 2*y.
- In Groovy, both are possible. You can use numbers with numeric operators, and you can also call methods on number instances.

Constructors

- Groovy supports default and parameterized constructors.
- In Groovy there are two ways to invoke constructors:
- **As positional parameters**: Like Java constructors
- **As named parameters :** Allows to specify the parameter names when invoking the constructor.

```
def person1 = new Person’(Vijay', 18)

def person2 = [‘Maya', 22] as Person
```

Here constructor is invoked using coercion with **as** keyword

```
Person person3 = [‘Babu', 33]
```

Here constructor is invoked using coercion in assignment

Named argument constructor

- It is possible to invoke then constructors by passing parameters in the form of a map (property/value pairs).
- This can be in handy in cases to allow several combinations of parameters. Otherwise, by using traditional positional parameters it would be necessary to declare all possible constructors.

- Here all are default constructors
- def person5 = new Person(name: 'Maya')
- def person6 = new Person(age: 16)
- def person7 = new Person(name: ‘Baburam', age:22)

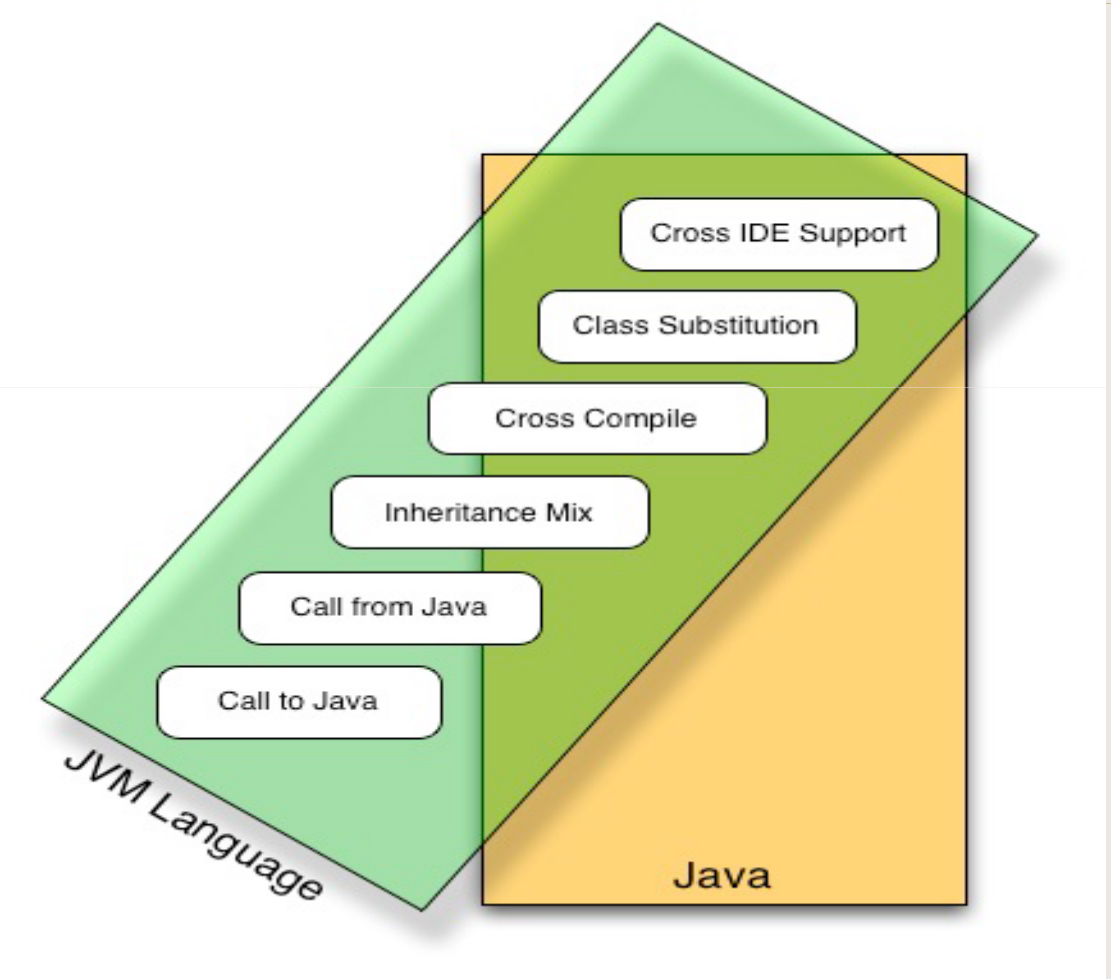
Groovy Usage

- High power scripting in java applications.
- Used internally in Grails and other high level tools as SoapUI, Gradle etc.
- Dynamically injects required features into application objects.
- Flexible and configurable.
- Write less and get more out of programming.
- Used to implement Domain Specific Language .
- Utilized in testing tools like SoapUI.

Groovy Scaffolding

- Compile the groovy program to java .class using groovyc
- Run JavaP on the .class code: javap MyClass
- The javap tool generates and displays the structure of the .class
- The code including the class name, variables and method names.

Groovy Java Integration



Operator overloading

- Groovy supports operator overloading which makes working with Numbers, Collections, Maps and various other data structures easier to use.
- Various operators in Groovy are mapped onto regular Java method calls on objects.
- This allows the developer to provide own Java or Groovy objects which can take advantage of operator
- overloading.

Overloaded operators

- a + b = a.plus(b)
- a - b = a.minus(b)
- a * b = a.multiply(b)
- a ** b = a.power(b)
- a / b = a.div(b)
- a % b = a.mod(b)
- a | b = a.or(b)
- a & b = a.and(b)
- a ^ b = a.xor(b)
- a++ or ++a = a.next()
- a-- or --a = a.previous()
- a[b] = a.getAt(b)
- a[b] = c = a.putAt(b, c)

More Overloaded operators

- a << b = a.leftShift(b)
- a >> b = a.rightShift(b)
- switch(a) { case(b) : } = b.isCase(a)
- ~a = a.bitwiseNegate()
- -a = a.negative()
- +a = a.positive()
- a == b = a.equals(b) or a.compareTo(b) == 0 **
- a != b = !a.equals(b)
- A <=> = b a.compareTo(b)
- a > b = a.compareTo(b) > 0
- a >= b = a.compareTo(b) >= 0
- a < b = a.compareTo(b) < 0
- a <= b = a.compareTo(b) <= 0

➢The comparison operators handle nulls gracefully avoiding the throwing of java.lang.NullPointerException.

Spread operator

- Spread Operator (*.)
- The Spread Operator is used to invoke an action on all items of an aggregate object.
- It is equivalent to calling the collect method like here:
- parent*.action
- //equivalent to:
- parent.collect{ child -> child?.action }

The Elvis operator

- **Elvis Operator (?:)**
- The "Elvis operator" is a compact way of shortening of Java's ternary operator .
- This is handy is for returning a 'sensible default' value if an expression resolves to false or null .
- def displayName = user.name ? user.name : "Anonymous"
- //traditional ternary operator
- def displayName = user.name ?: "Anonymous"

Safe Navigation operator

- **Safe Navigation Operator (?.)**
- The Safe Navigation operator is used to avoid a NullPointerException. Typically when you have a reference to an object you might need to verify that it is not null before accessing methods or properties of the object.
- To avoid this, the safe navigation operator will simply return null instead of throwing an exception
- def user = User.find("admin")
- //this might be null if 'admin' does not exist
- def streetName = user?.address?.street
- //streetName will be null if user or
- user.address is null - no NPE thrown

Object operators

- **Object-Related Operators**
- invokeMethod and get/setProperty (.)
- Java field (.@)
- The spread java field (*.@)
- Method Reference (.&)
- 'as' - "manual coercion" - asType(t) method
- Groovy == (equals()) behavior.
- "is" for identity
- The instanceof operator (as in Java)

Groovy advanced Features

- Invoke java capabilities in JRE
- Callable from java environment
- Generates java byte code on the fly
- Adds scripting flavour to java
- Supports jdbc
- XML processing
- Testing support
- Meta programming
- Template based code generation
- Extensible

Groovy Performance

- Groovy brings noticeable performance improvements.
- Groovy has lower memory consumption.
- Groovy is faster and leaner in most situations.

Groovy Agility

- Groovy helping developers to be more productive and more agile.
- Focusing more on the task at hand than on boiler-plate technical code
- Leveraging existing Enterprise APIs rather than reinventing the wheel
- Improving the overall performance and quality of the language
- Enabling developers to customize the language at will to derive their own Domain-Specific Languages

Groovy Usage

- Groovy's flexibility and expressivity and its scripting capabilities open the doors to advanced build scripting.
- In infrastructure systems for the continuous integration practices and project build solutions, such as Gant and Gradle .
- At the tooling level, Groovy also progresses, for instance with its groovydoc Ant task to generate proper JavaDoc covering, documenting and interlinking both your Groovy and Java source files for the Groovy/Java mixed projects.
- The IDEs support full editing capabilities with groovy.

Ranges in Groovy

- Groovy supports additionally ranges as internal loop constructs in literal formats.

```
def x = 1..10   //Range declaration
assert x.contains(5)
assert x.contains(15) == false
assert x.size() == 10
assert x.from == 1
assert x.reverse() == 10..1
```

The ..< range operator specifies a half-exclusive range--that is, the value on the right is not part of the range:

Ranges are objects

```
def result = “ “
(5..9).each { element ->
result += element
}
assert 5 in 0..10 //Range for classification
assert (0..10).isCase(5)
def age = 36
switch(age){
case 16..20 : insuranceRate = 0.05 ; break
case 21..50 : insuranceRate = 0.06 ; break
case 51..65 : insuranceRate = 0.07 ; break
default: throw new IllegalArgumentException()
}
```

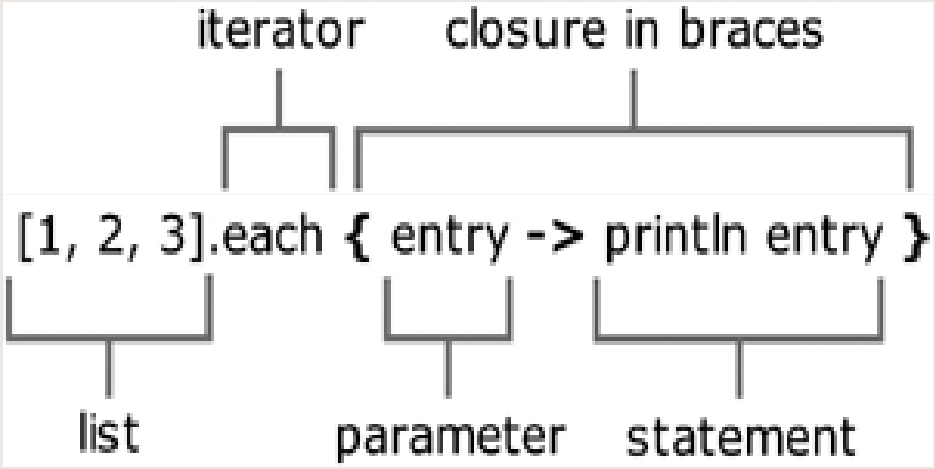
Groovy with Collections

- Groovy makes collection handling simple, with added support for operators, literals, and extra methods beyond those provided by the Java standard libraries.
-
- In groovy arrays are treated as lists and dynamically resized.
- The lists can be specified as literals directly in the code.
- i.e. def roman = ['', 'I', 'II', 'III', 'IV', 'V', 'VI', 'VII']
- The Maps are specified with literals and providing suitable operators to work with them.

Iteration over the Collection

Print each item in the list

```
 list.each { item ->
    println item
 }
```



The List.each method takes a single parameter--a closure. It executes that closure for each of the elements in the list, passing in that element as the argument to the closure. The main body of the closure is a statement to print out whatever is passed to the closure, namely the parameter we've called entry.

Other Groovy Features

- The static imports
- Import aliasing
- Generic collections
- Var args
- Regular Expressions
- Interpolation for string embedded expressions
- XML and JSON processing
- Database and file io operations processing
- Dynamic builds
- Template based code generation

Groovy Closure

- Groovy support functions that used as first class objects.
- That is you can define a chunk of code and then pass it around as if it were a string or an integer. Objects.
- square = { it * it }
- The curly braces tells the Groovy compiler to treat this expression as code. In the software world, this is called a "closure".
- In this case, the field "it" refers to whatever value is passed to the function.
- This compiled function is assigned to the variable "square" as above.
- So now to use it : square(11), square 101 etc.

Closure Usage

- Use the above closure in the "collect" method on arrays.
- Def array = [1, 2, 3, 4]
- Array.collect(square)
- It says, create an array with the values 1,2,3 and 4, then call the "collect" method, passing in the
- closure we defined above.
- The collect method runs through each item in the array, calls the closure on the item,
- then puts the result in a new array, resulting in:
- [1, 4, 9, 16].
- The Closures can be passed into methods like any other object.
- def closure = { param -> param + 1 }
- def answer = [1, 2].collect(closure)
- assert answer == [2, 3]

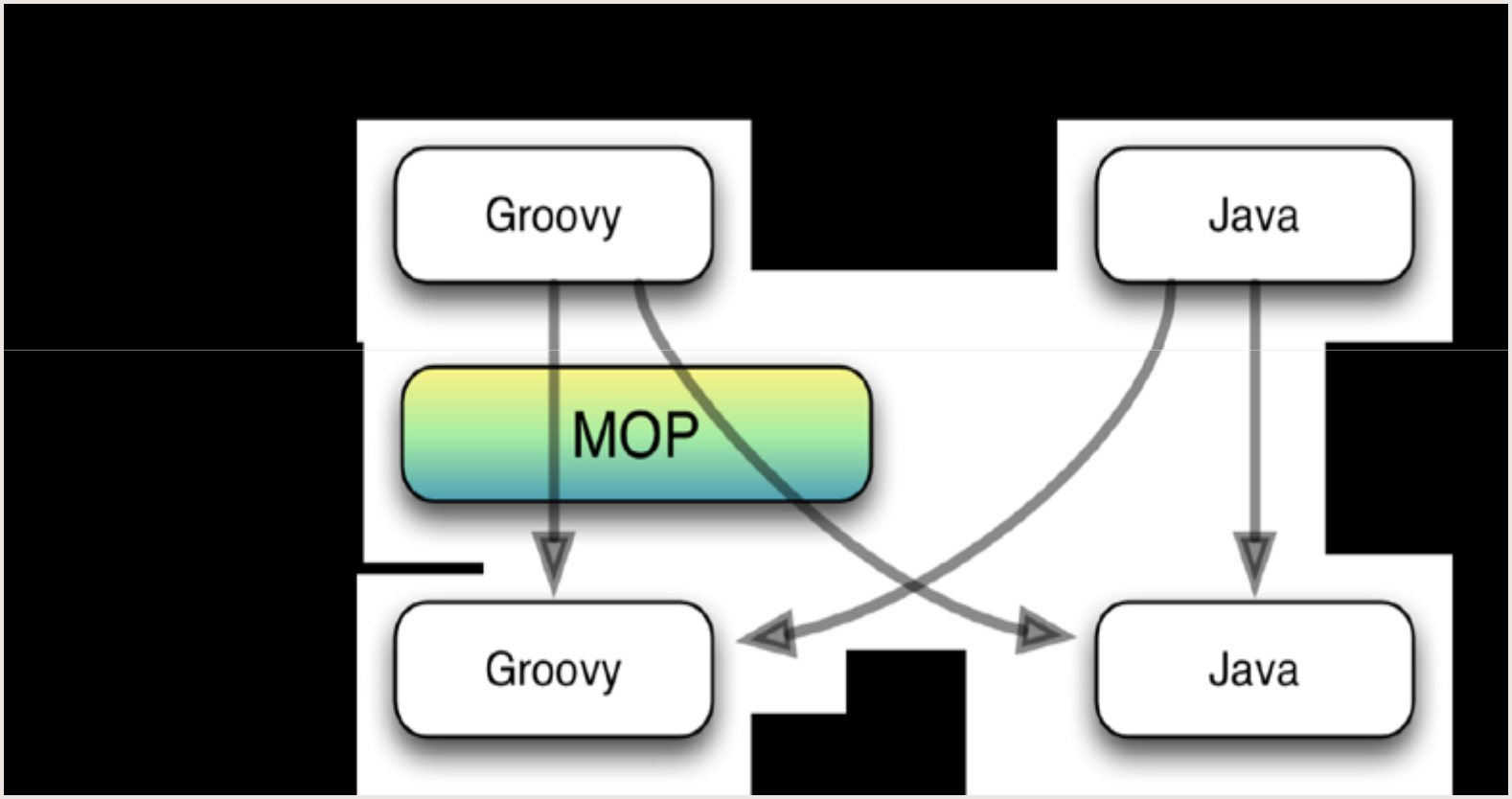
Groovy Meta Object Protocol

- Whenever "Groovy calls a method" it doesn't call it directly but asks an intermediate layer to do so on its behalf.
- The intermediate layer provides hooks that allow us to influence its inner workings.
- A protocol is a collection of rules and formats. The Meta-Object-Protocol (MOP) is the collection of rules of how a request for a method call is handled by the Groovy runtime system and how to control the intermediate layer.

Groovy Meta Object Protocol

- The "format" of the protocol is defined by the respective APIs.
- When Groovy calls a method, Groovy compiler generates byte-code that calls into the MOP.
- println 'Hello' // Groovy
- The resulting byte-code that Groovy produces is equivalent to the following Java code
- InvokerHelper.invokeMethod(this, "println", {"Hello"});

The Groovy MOP



Use of MOP

- Every method call that you write in Groovy is actually a call into the Groovy MOP.
- This is regardless of whether the call target has been compiled with Groovy or Java.
- This applies equally to static and instance method calls, constructor calls, and property access, even if the target is the same object as the caller.
- This allows to add additional code into groovy classes at runtime.
- You can add a new field, a new method or a constructor at run time in to the groovy class code!

Groovy DSL

The basic idea of a Domain Specific Language (DSL) is a computer language that's targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem.
Domain specific languages have been talked about, and used for almost as long as computing has been done.
Common DSL examples include CSS, regular expressions, make, rake, ant, SQL, HQL, many bits of Rails etc.

Groovy DSL..

DSL's are meant to target a particular type of problem. They are short expressive means of programming that fit well in a narrow context.
For example with GORM in Grails , you can express hibernate mapping with a DSL rather than XML.

```
static mapping = {
    table 'person'    columns
   {      name column:'name'    }
  }
```

Java build Tools

- Apache Ant for automation of java and build tasks
- Apache Maven for centrally managing the application dependencies on different multiple library files.
- Jenkins for continuously monitoring the code base and running the build jobs automatically with build actions whenever a change is detected.
- Testing tools JUnit, TestNG,Cucumber etc.
- Static Code analysis tools PMD,checkStyle,FindBugs, Sonar etc.

Gradle : Build Automation Tool

- A very flexible general purpose build tool like Ant.
- Switchable, build-by-convention frameworks like Maven.
- Powerful support for multi-project builds.
- Application dependency management (with Apache Ivy).
- Full support for the existing Maven or Ivy repository infrastructure.
- Support for transitive dependency management
- Ant tasks and builds as first class citizens.
- *Gradle runs on top of groovy* build scripts.
- A rich domain model of java code for configuring the build process.

Thank You!