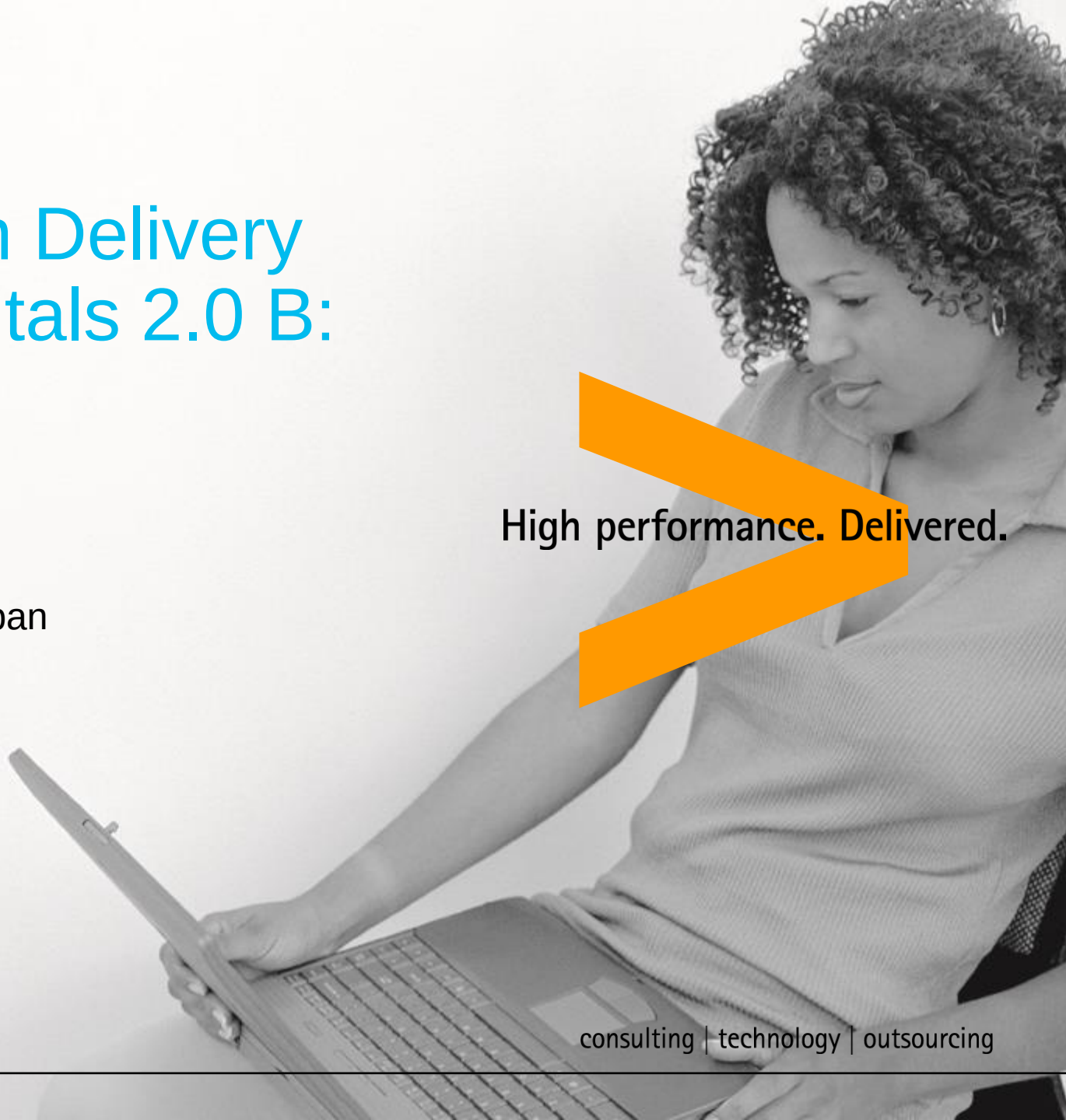
Application Delivery Fundamentals 2.0 B:

Groovy



Parameswari Ettiappan

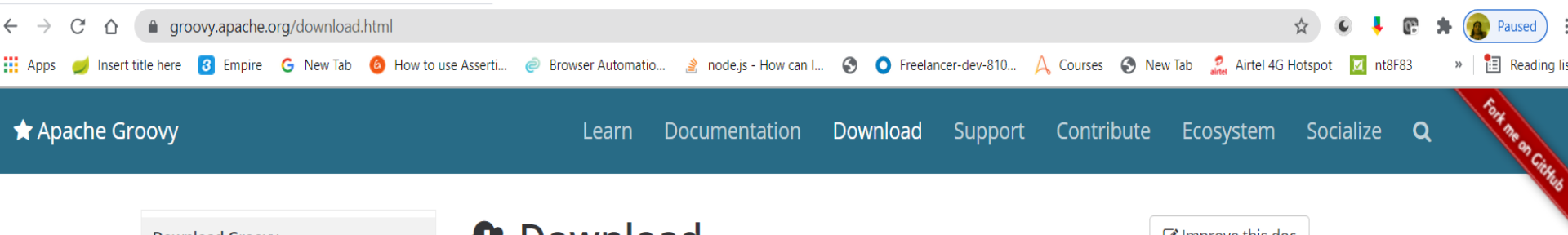
Goals

- Groovy Fundamentals
- Object-Oriented Programming In Groovy
- Groovy Collections
- Closures in Groovy
- Groovy Control Structures
- Miscellaneous Groovy operators
- Unit Testing in Groovy
- Abstract Syntax Tree (AST) Transformations

Goals

- Database Access with Groovy
- The Spring Framework
- RESTful web services
- The Gradle Build Tool

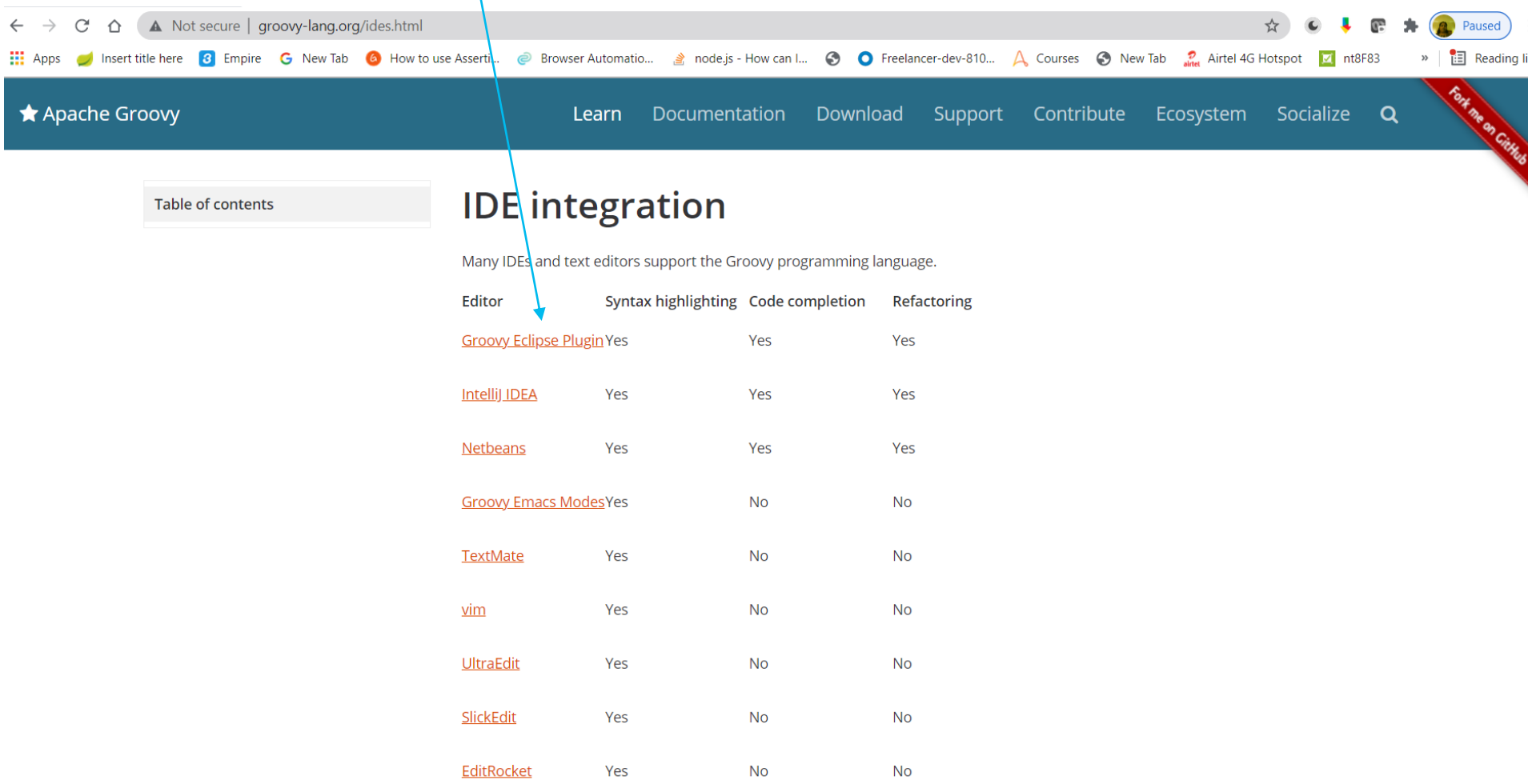
Groovy installation



Groovy installation

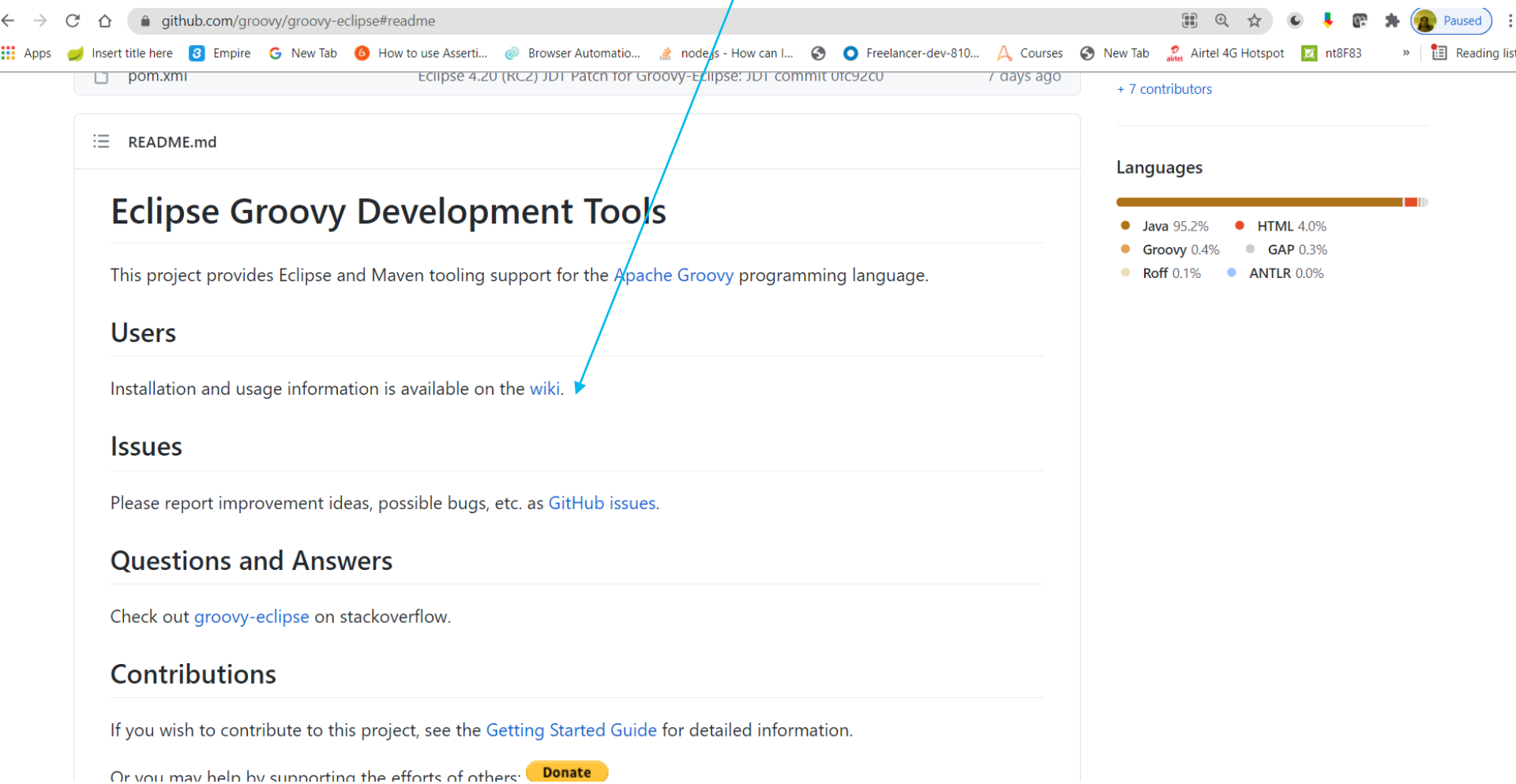

```
Microsoft Windows [Version 10.0.22621.1265]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>groovy -v
Groovy Version: 4.0.0 JVM: 11.0.15 Vendor: Oracle Corporation OS: Windows 11

C:\Windows\System32>
```

Groovy installation



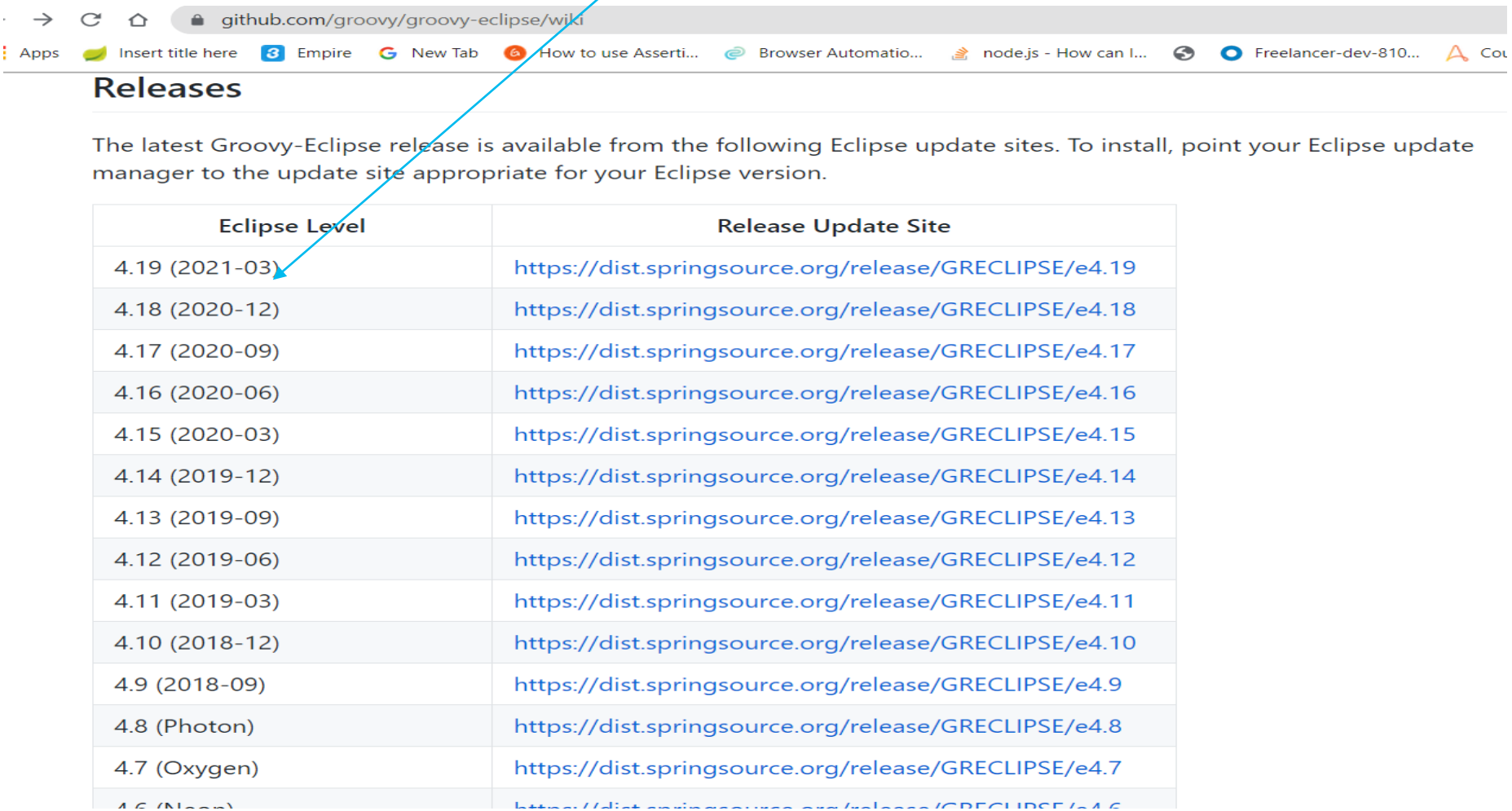
Groovy installation

Go to Documentation



Groovy installation

Go to Releases

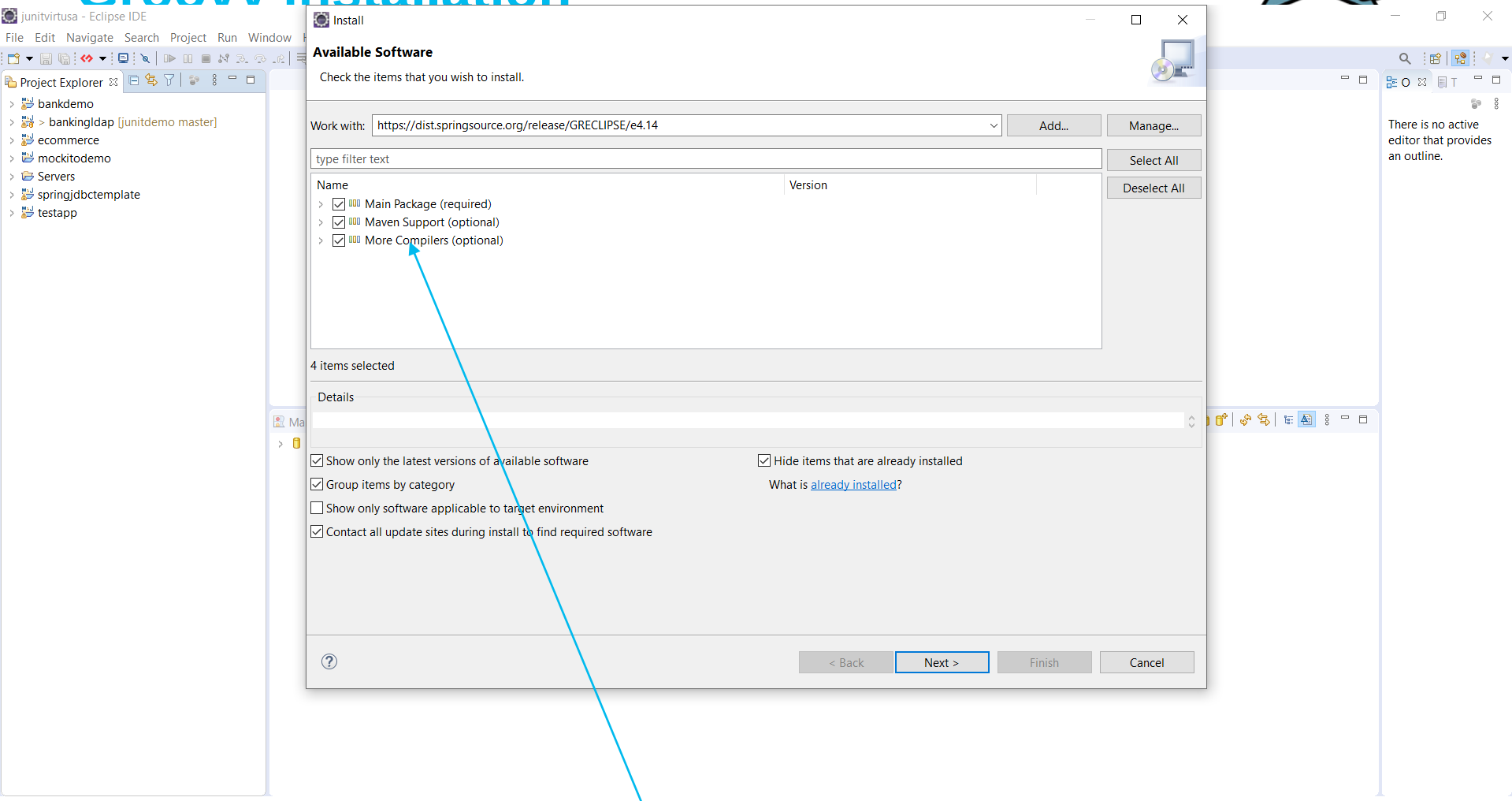


Releases

The latest Groovy-Eclipse release is available from the following Eclipse update sites. To install, point your Eclipse update manager to the update site appropriate for your Eclipse version.

Eclipse Level	Release Update Site
4.19 (2021-03)	https://dist.springsource.org/release/GRECLIPSE/e4.19
4.18 (2020-12)	https://dist.springsource.org/release/GRECLIPSE/e4.18
4.17 (2020-09)	https://dist.springsource.org/release/GRECLIPSE/e4.17
4.16 (2020-06)	https://dist.springsource.org/release/GRECLIPSE/e4.16
4.15 (2020-03)	https://dist.springsource.org/release/GRECLIPSE/e4.15
4.14 (2019-12)	https://dist.springsource.org/release/GRECLIPSE/e4.14
4.13 (2019-09)	https://dist.springsource.org/release/GRECLIPSE/e4.13
4.12 (2019-06)	https://dist.springsource.org/release/GRECLIPSE/e4.12
4.11 (2019-03)	https://dist.springsource.org/release/GRECLIPSE/e4.11
4.10 (2018-12)	https://dist.springsource.org/release/GRECLIPSE/e4.10
4.9 (2018-09)	https://dist.springsource.org/release/GRECLIPSE/e4.9
4.8 (Photon)	https://dist.springsource.org/release/GRECLIPSE/e4.8
4.7 (Oxygen)	https://dist.springsource.org/release/GRECLIPSE/e4.7
4.6 (Neon)	

Groovy installation



Select only groovy 4.0 compiler

Groovy installation


```
Administrator: Command Prompt
Terminate batch job (Y/N)? y

C:\WINDOWS\system32>gradle -version

------------------------------------------------------------
Gradle 6.8.1
------------------------------------------------------------

Build time:   2021-01-22 13:20:08 UTC
Revision:     31f14a87d93945024ab7a78de84102a3400fa5b2

Kotlin:       1.4.20
Groovy:       2.5.12
Ant:          Apache Ant(TM) version 1.10.9 compiled on September 27 2020
JVM:          1.8.0_251 (Oracle Corporation 25.251-b08)
OS:           Windows 10 10.0 amd64

C:\WINDOWS\system32>
```

Groovy installation

Administrator: Command Prompt - gradle init

```
F:\groovyvebws>gradle init

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4] 2

Select implementation language:
  1: C++
  2: Groovy
  3: Java
  4: Kotlin
  5: Scala
  6: Swift
Enter selection (default: Java) [1..6]
```

Groovy installation

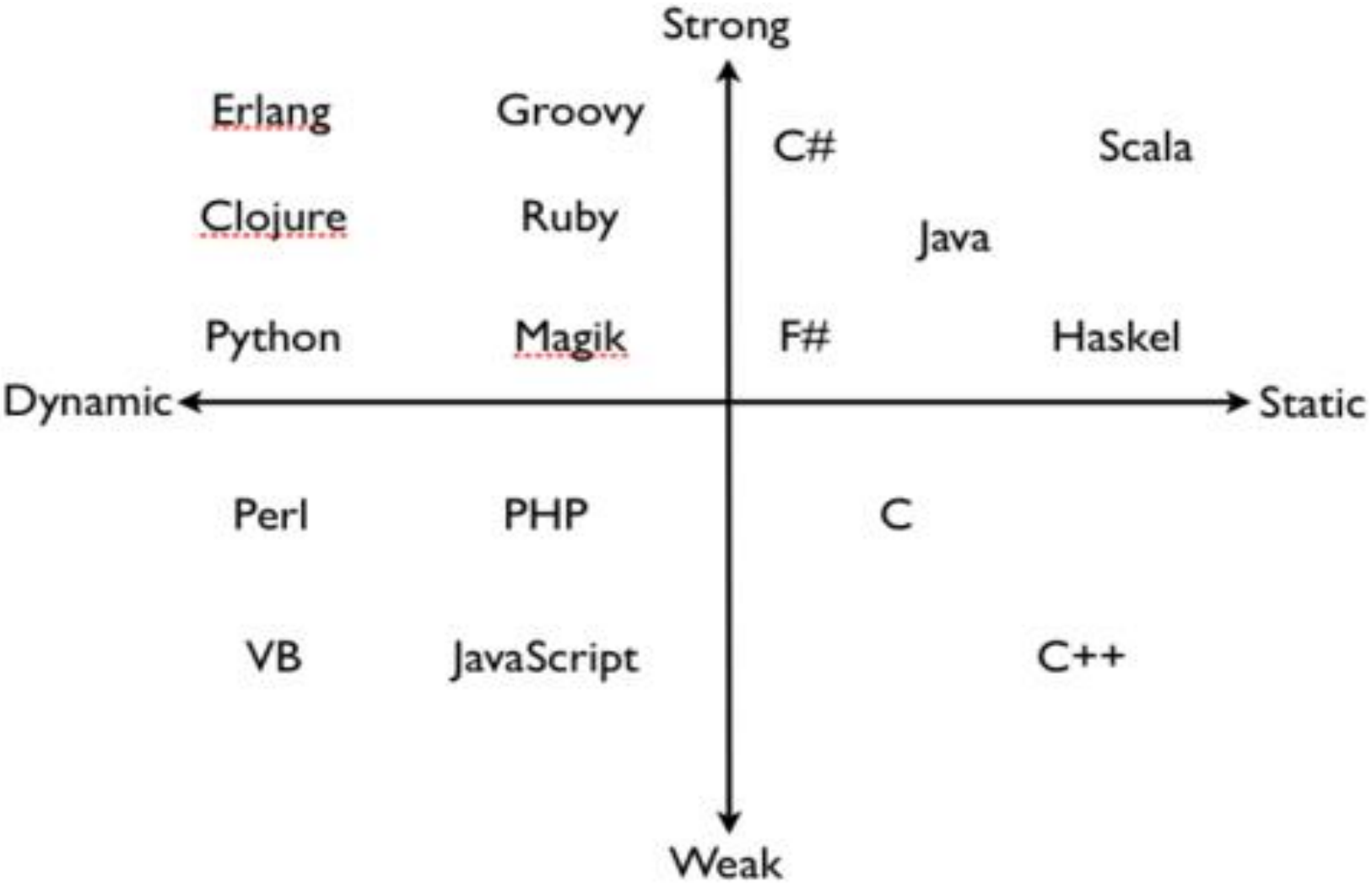
Administrator: Command Prompt - gradle init

```
OS:           Windows 10 10.0 amd64

C:\WINDOWS\system32>f:

F:\>cd groovyvebws

F:\groovyvebws>gradle init

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4] 2

Select implementation language:
  1: C++
  2: Groovy
  3: Java
  4: Kotlin
  5: Scala
  6: Swift
Split functionality across multiple subprojects?:
  1: no - only one application project
  2: yes - application and library projects
Select build script DSL:
  1: Groovy
  2: Kotlin---> 50% EXECUTING [53s]
Enter selection (default: Groovy) [1..2] 1

Project name (default: groovyvebws): groovygradledemo
```

Next JVM


2%
Xtend
2%
Ceylon
2%
Kotlin
4%
Others
12%
Clojure
47%
Scala
31%
Groovy

Static Vs Dynamic Languages



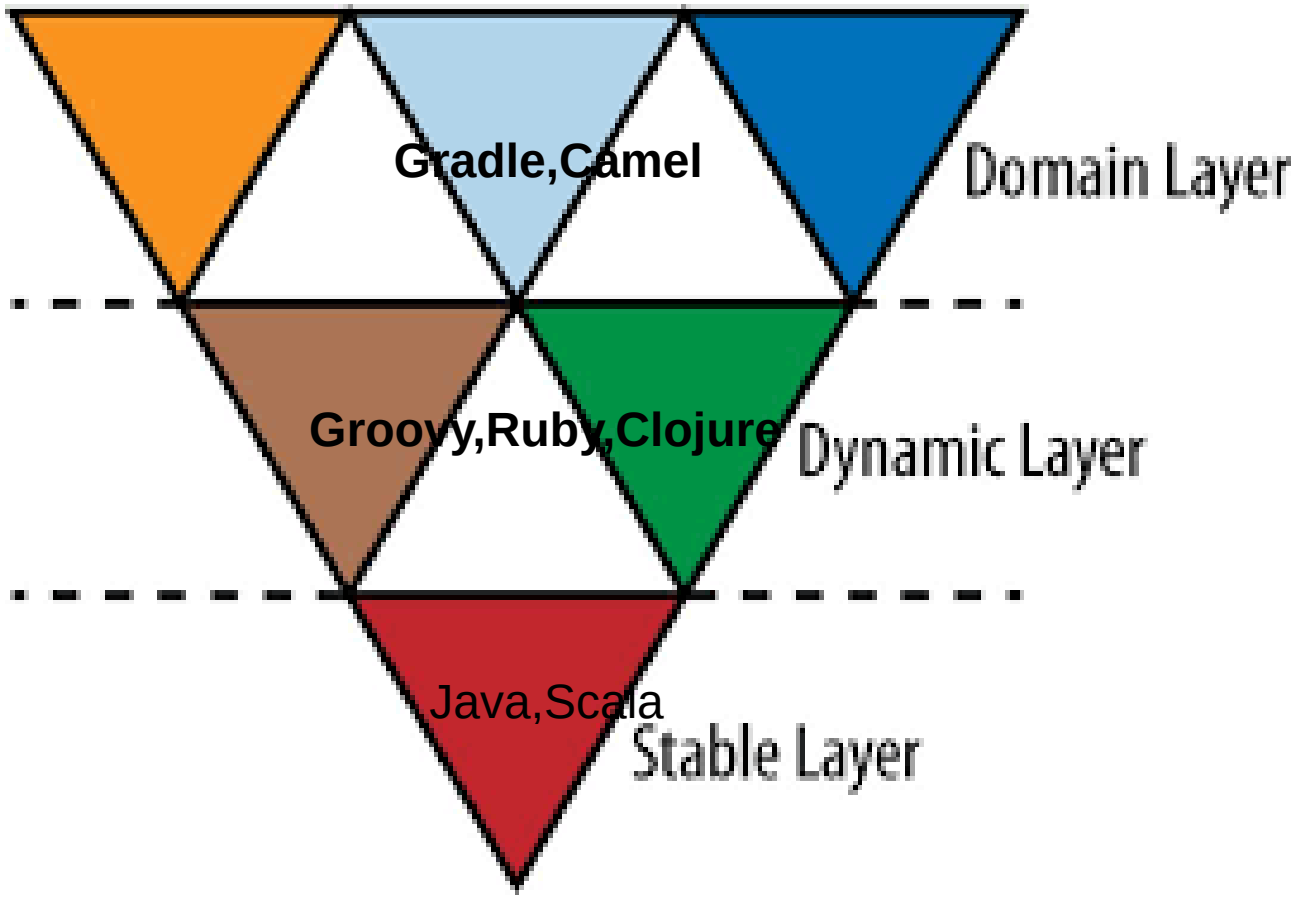
Static Vs Dynamic Languages

- Static Languages enforce type constraints at compile Time
- Many bugs caught early in development yields performance gains
- Can introduce friction in development
- ill suited for flexible data formats

Static Vs Dynamic Languages

- Dynamic Languages enforce type constraints at run time
- Can lead to more rapid development
- Well suited for flexible data formats
- Some bugs may not be found until later in development
- Performance tends to be slower

Polygot Programming



Groovy

- Groovy was developed by Jochen Theodorou, Guillaume Laforge, Cedric Champeau, and Paul King.
- Its typing discipline is strong, static and dynamic.
- It was licensed under the Apache 2.0 license.
- It first appeared in the year 2003.
- Its file extension is .groovy and was designed by James Strachan.
- An agile dynamic language for the Java Platform with many features that are inspired by languages like Python, Ruby and Smalltalk, making them available to Java developers using a Java-like syntax.

Why groovy

- Groovy is an agile and dynamic language
- Seamlessly integration with all existing Java objects and libraries
- Feels easy and natural to Java developers
- More concise and meaningful code compares to Java
- You can use it as much or as little as you like with Java apps

Why groovy

- *"Groovy is like a **super version of Java**. It can leverage Java's enterprise capabilities but also has cool productivity features like closures, DSL support, builders and dynamic typing."*

```
Groovy = Java – boiler plate code
               + mostly dynamic typing
               + closures
               + domain specific languages
               + builders
               + metaprogramming
               + GDK library
```

Where does Groovy FIT

- Web Application Development
- Building Domain Specific Language
- Writing acceptable test cases
- Scripting in JVM Environment

Scripting Languages The old new new thing…

- Productivity / Rapid Development / Agile
- Interpreted (No Compile Cycle)
- Expressive - Shorter, Less Verbose Syntax
- Feature Rich yet Simple and Productive
- Shell / System Integration / Systems “Glue”
- Dynamic
- Open Source – also code available by default
- Advanced languages features Closures / Mix Ins

Script Languages Reputation

- Weakly Typed
- Procedural
- Scalability / Enterprise Ready
- Not Maintainable – (Perl)
- Lack of threading support
- Lack of IDE / Debugger support
- Cross Platform Support
- Performance
- Specialized / Not general purpose

Groovy Overview

- Syntactically very close to Java
- Leverages investment in Java
- Interpreted-ish. .groovy files fully parsed and classes are generated on the fly, via a custom class loader.
- Can be compiled and fully integrated with traditional Java application.
- Language level support for lists, maps, regular expressions.
- Supports closures, dynamic typing, meta object protocol.

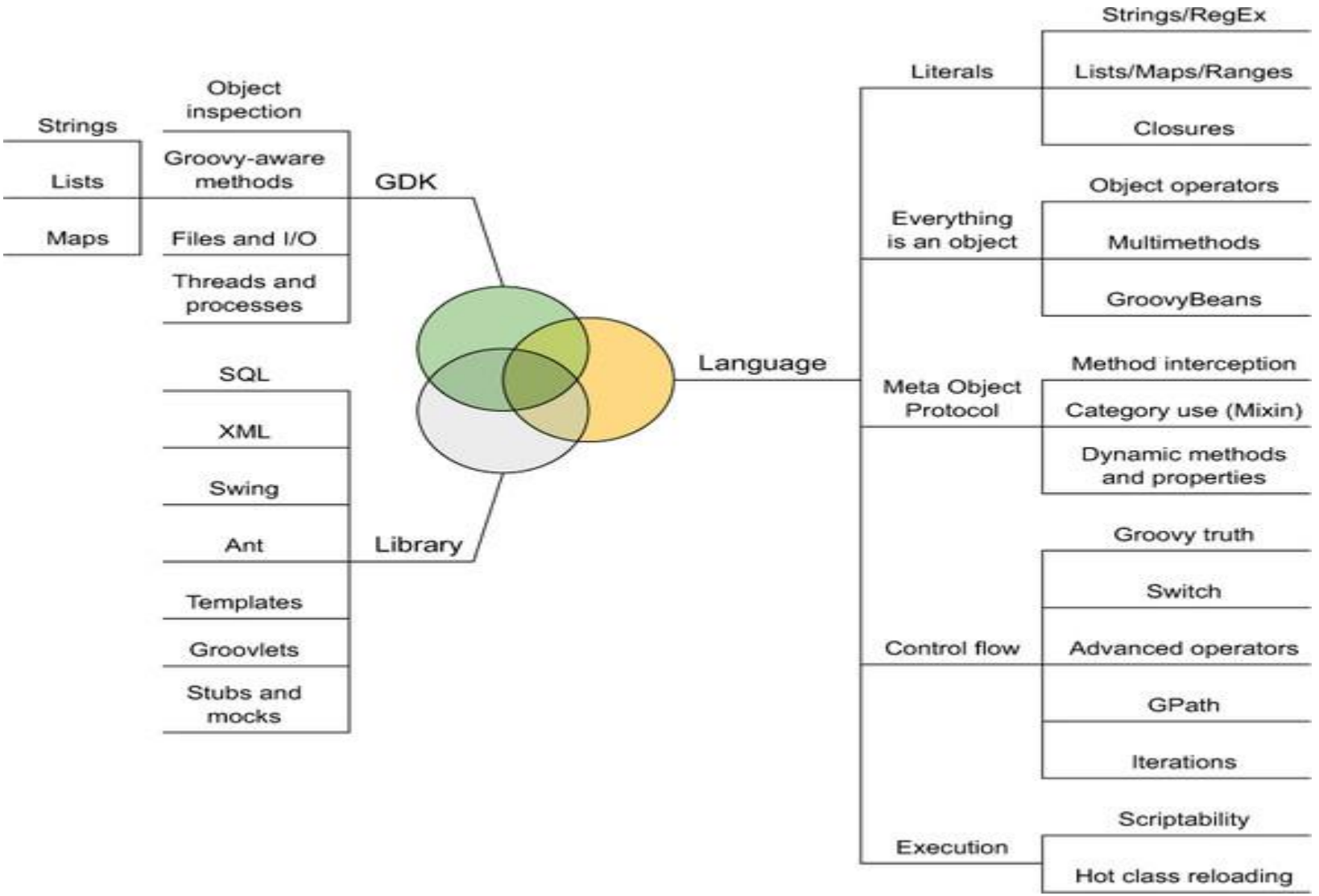
Groovy & Java – Seamless Integration

- Groovy code co-exists with Java Code and runs in the JRE.
- Syntax nicely aligned with Java.

Power in your code: a feature-rich language

- Groovy has three main enhancements over and above those of Java: language features, libraries specific to Groovy, and additions to the existing Java standard classes (known as the Groovy Development Kit, or GDK)

Power in your code: a feature-rich language

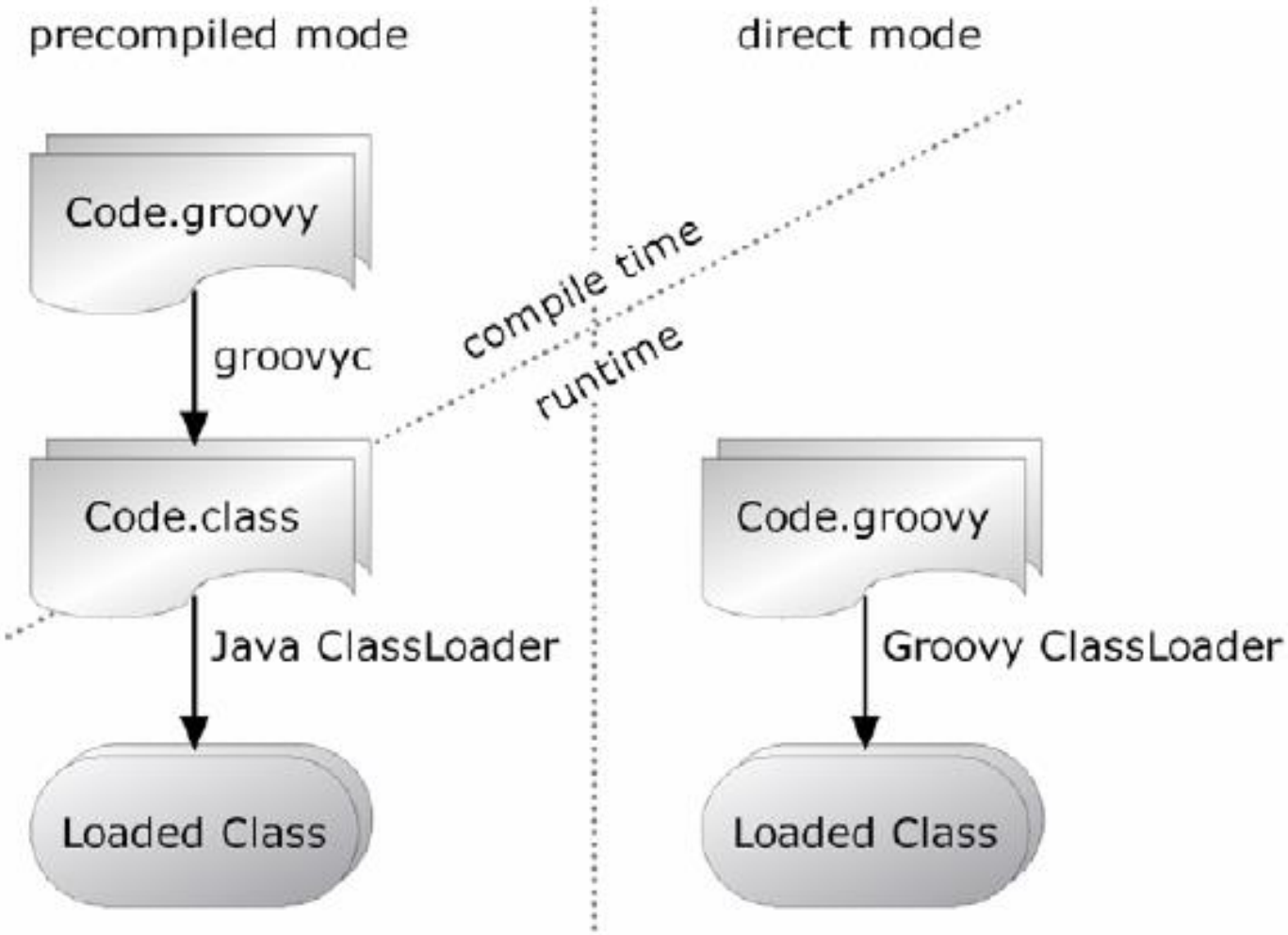


Direct vs. Precompiled Mode in Groovy

- Direct mode. Code is "directly executed, without producing any executable files".

- Precompiled mode. The second way, which involves taking the code and "compiling it to Java bytecode and running it as regular Java application code within a Java Virtual Machine (JVM)".

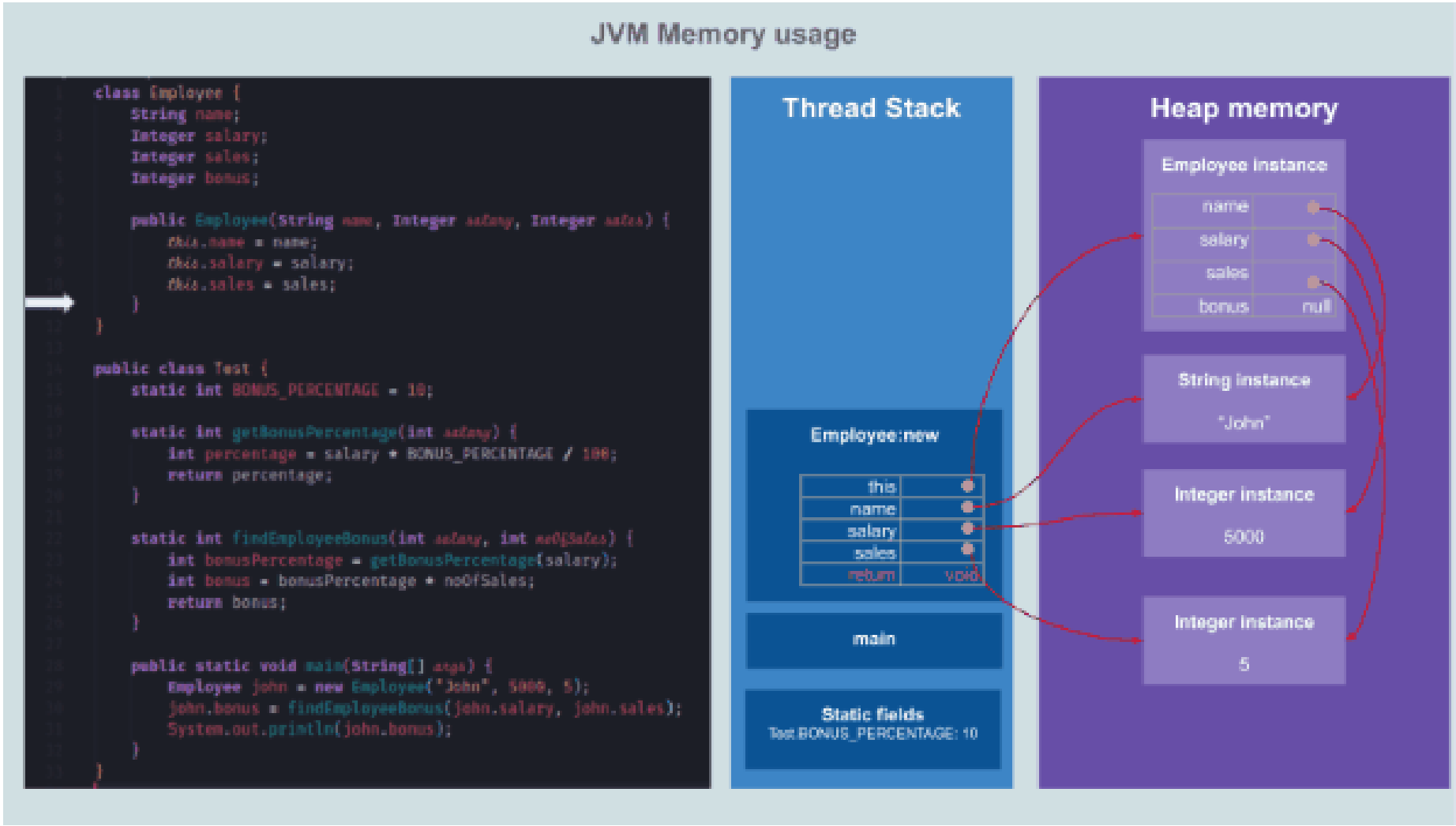
Direct vs. Precompiled Mode in Groovy



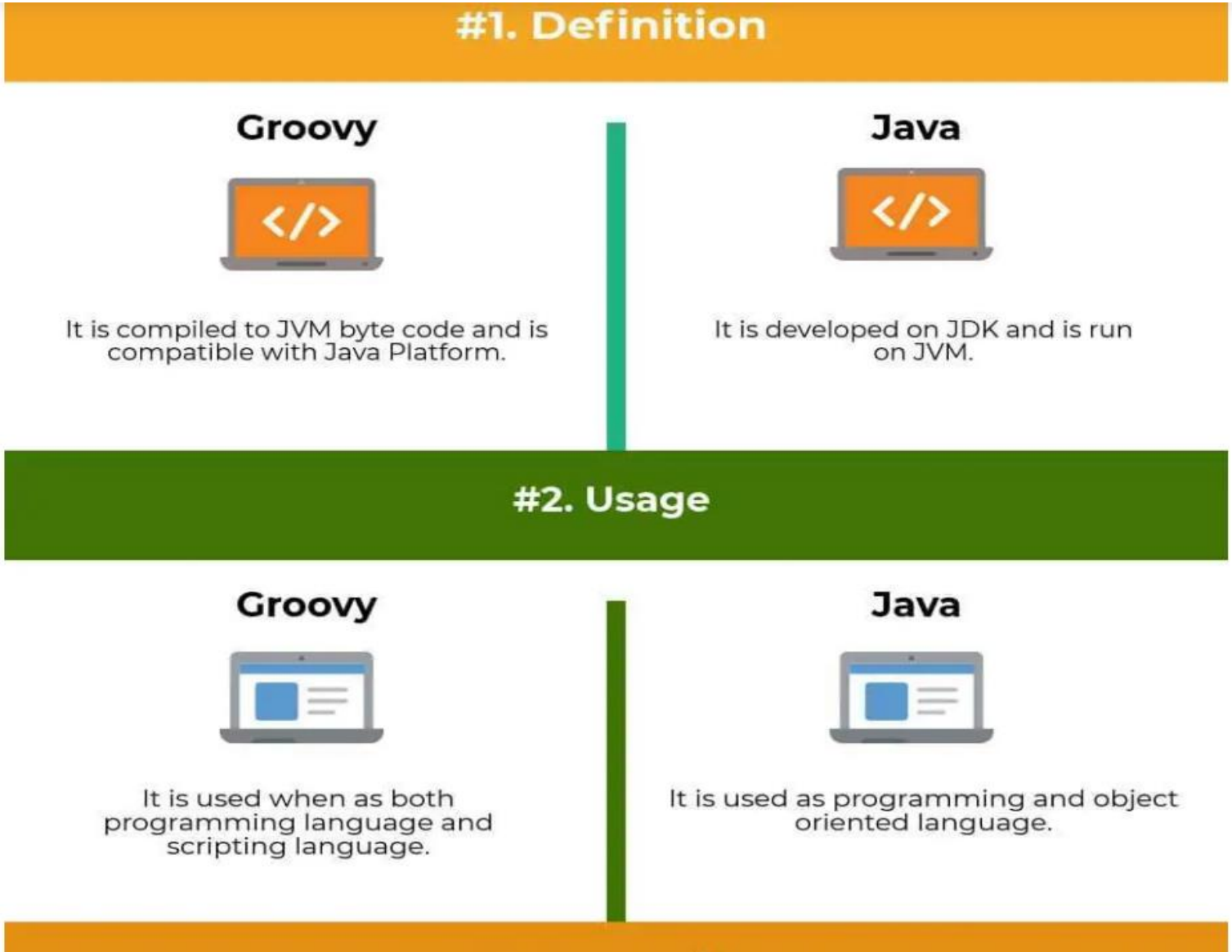
Compiling and executing Groovy programs

- groovyc - the Groovy compiler
- groovyc is the Groovy compiler command line tool.
- It allows you to compile Groovy sources into bytecode.
- It plays the same role as javac in the Java world.
- The easiest way to compile a Groovy script or class is to run the following command:
  - **groovyc MyClass.groovy**

Visualizing memory management in JVM(Java, Kotlin, Scala, Groovy, Clojure)



Groovy vs Java



Groovy vs Java

#3. Integration

Groovy

It can be integrated along with any web applications and scripting applications.

Java

It can also be integrated with any object oriented application as it is compatible with any JVM based applications.

#4. Platform

Groovy

It supports any operating systems or platforms.

Java

It supports cross platform operating systems.

Groovy vs Java

#5. Syntax

Groovy

The syntax is very similar to Java syntax.

Java

It has strong disciple in its syntax.

#6. Community

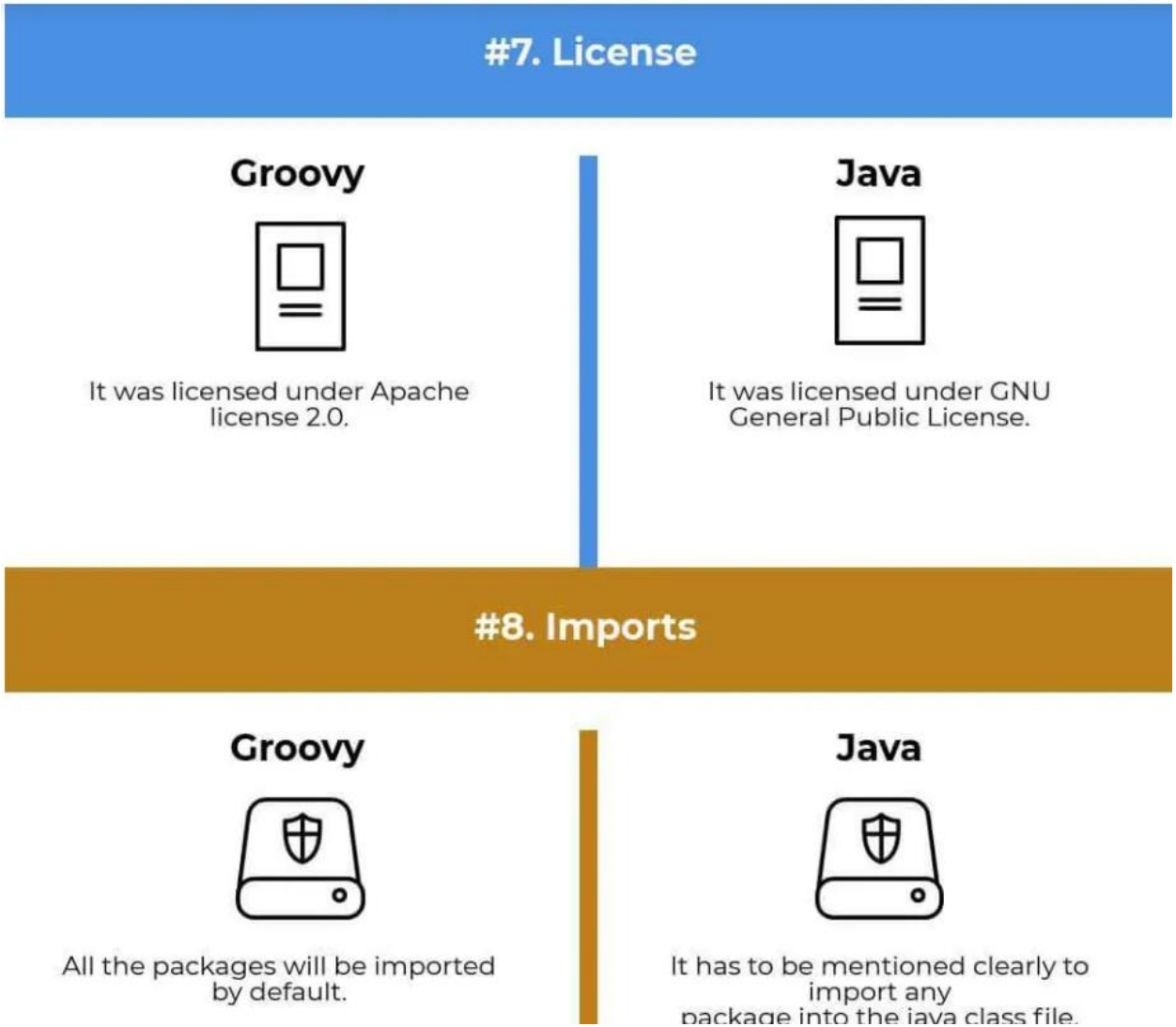
Groovy

It has been submitted to JCP for specification request.

Java

It has larger community called Java Community process i.es JCP being maintained by large group of highly qualified technical

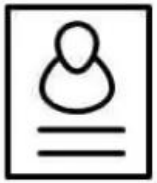
Groovy vs Java



Groovy vs Java

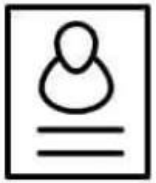
#9. Documentation

Groovy



It provides single page documentation.

Java



It has documentation in the form specification given by JCP.

Groovy vs Java

Java	Groovy
Groovy code does not compile within a Java project	All valid Java code works in Groovy as well
Only *java.lang* package is imported by default	Lot of Standard Java packages such as *java.lang*, *java.io*, *java.net*, *java.util* are imported into groovy code by default
Method to be executed is determined at compile time	Method to be executed is determined at runtime
Braces { and } are used for Array initialization	Square brackets [and] are used for Array initialization
Not specifiying a scope will result in default scope being considered	Not specifying a scope will result in private scope being considered
Java 8 supports lambdas and method references	Groovy supports closures
Spring boot provides convention-by-configuration for Java language	Grails provides convention-by-configuration for Groovy language
Spring *ORM* provides support for *Hibernate* for Java language	Grails *GORM* provides support for *Hibernate* for Groovy language
Java *Strings* are created with double quotes	Groovy *Strings* are created with single or double quotes

Groovy vs Java

Java *Strings* do not support templating by default	Groovy *GStrings* provide templating ability by default
Java not suitable for scripting	Since Groovy has a lot of defaults like no need of class name, no need of imports for some of java packages, simple declarations of maps and arrays, it is suitable for scripting
Java does not autowrap primitive unless actually needed. Java prefers widening of primitive data type over autoboxing. For example, *int* is preferably converted to *long* over converting to *Integer*	Groovy autowraps all primitives by default. *int* is converted to *Integer* by default
For Java objects, == checks only for identity and does not check for equality	For Groovy objects,== checks for comparison if the object supports *Comparable* interface and checks for equality otherwise
In Java, we have to use *Object* class to refer to anything	In Groovy, *def* keyword can be used
Java does not have easy support for meta-programming compared to Groovy language	Groovy supports meta-programming with ease

Groovy vs Java

Java	Groovy
Every statement must end with a semicolon.	Semicolon is not needed in Groovy.
The default access for an identifier is "package" except for members of interfaces, which are public.	Groovy considers its package protected classes and methods public by default.
The Java exception-handling scheme distinguishes between checked and unchecked exceptions.	There is literally no difference between checked and unchecked exceptions in Groovy.
Specifying type information is mandatory in Java.	Declaring a return type is optional in Groovy.
Java programs run well in Groovy environment.	Groovy programs may or may not run well in Java runtime environment.
To access the state of the managed bean, you need to provide getter and setter methods for that state.	Getters and Setters are automatically generated for class members.
Methods are chosen at compile-time.	Methods are chosen at run-time.

Groovy vs Java

Java	Groovy
It is developed on JDK and is run on JVM	It is compiled to JVM Byte code and It is compatible with **Java** platform
It is used as programming and object oriented Language	It is used as both programming and scripting Language
In **Java** default access modifier **package**	In **Groovy** default access modifier **public**
In **Java**, you need to provide getters and setters method for fields, especially if you are following **Java** Beans naming convention	In **Groovy**, getters and setters are automatically generated for class members
In **Java** semicolons are compulsory	In **Groovy** semicolons are optional
In **Java** only **Java**.lang.* package is imported by default	In **Groovy** commonly used packages are imported by default
Java has primitive data types and wrapper classes to perform boxing and unboxing implicitly or explicitly	In **Groovy** everything is object and uses only object hence no concept of autoboxing or unboxing
Java has a requirement of the main method inside a class to run the program	**Groovy** does not require any main method or entry point of a method to run the class or any program

Default Imports

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

Groovy vs JavaScript

Groovy VS JavaScript Feature comparision

Feature	Groovy	JavaScript
Scripting	✓	✓
Compiler	✓	✓
Interpreter	✓	✓
Object-oriented Language	✓	✓
High Level	✓	✓
Dynamic typing	✓	✓
Garbage Collection	✓	✓
Asynchronous	✓	✓
Interpreted language	✓	✓
Programmers Documentation	✓	✓
Coding	✓	✓

Groovy vs JavaScript

Cross-platform	✓	✓
Blocks	✓	✓
Package Manager	✓	✓
Text processing	✓	✓
Compiled Language	✓	✓
Scripting language	✓	✓
Configuration	✓	✓
Contract programming	✓	✗
Functional Language	✓	✓
Multiparadigm	✓	✗
Imperative Language	✓	✗
Prototype-based inheritance	✗	✓
Web Development	✗	✓
Shell integration	✗	✓
Support for Javascript	✗	✓
Javascript scripting	✗	✓

Groovy Comments

- #!/usr/bin/groovy
- // some line comment
- /*
- some multiline
- comment
- */

Gradle Project

C:\Windows\System32\cmd.exe - gradle init

```
(c) Microsoft Corporation. All rights reserved.

F:\groovyws2022>gradle init
Starting a Gradle Daemon, 4 incompatible Daemons could not be reused, use --status for de
tails

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4] 2

Select implementation language:
  1: C++
  2: Groovy
  3: Java
  4: Kotlin
  5: Scala
  6: Swift
Enter selection (default: Java) [1..6] _
```

Gradle Project

C:\Windows\System32\cmd.exe - gradle init

```
  4: Gradle plugin
Enter selection (default: basic) [1..4] 2

Select implementation language:
  1: C++
  2: Groovy
  3: Java
  4: Kotlin
  5: Scala
  6: Swift
Enter selection (default: Java) [1..6] 2

Split functionality across multiple subprojects?:
  1: no - only one application project
  2: yes - application and library projects
Enter selection (default: no - only one application project) [1..2] 1

Select build script DSL:
  1: Groovy
  2: Kotlin
Enter selection (default: Groovy) [1..2] 1
```

Gradle Project

C:\Windows\System32\cmd.exe

```
  1: no - only one application project
  2: yes - application and library projects
Enter selection (default: no - only one application project) [1..2] 1

Select build script DSL:
  1: Groovy
  2: Kotlin
Enter selection (default: Groovy) [1..2] 1

Project name (default: groovyws2022): gradledemo
Source package (default: gradledemo): com.gradledemo

> Task :init
Get more help with your project: https://docs.gradle.org/7.0.2/samples/sample_building_gr
oovy_applications.html

BUILD SUCCESSFUL in 2m 11s
2 actionable tasks: 2 executed
F:\groovyws2022>
```

Groovy Script


```
println 'Hello Groovy'
```

To run type ‘groovy hello.groovy’

- Parenthesis are optional
- Ending semi-colons optional
- Class declaration optional
- Main entry point optional
- Single Quotes or Double quotes for Strings. Groovy has several string styles.
- System.out is assumed - Groovy automatically imports the packages groovy.lang.*, groovy.util.*, java.lang.*, java.util.*, java.net.*, and java.io.*

Groovy Types

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)

Groovy Types

Statement	Type of value	Comment
`def a = 1`	`java.lang.Integer`	Implicit typing
`def b = 1.0f`	`java.lang.Float`	
`int c = 1`	`java.lang.Integer`	Explicit typing using the Java primitive type names
`float d = 1`	`java.lang.Float`	
`Integer e = 1`	`java.lang.Integer`	Explicit typing using reference type names
`String f = '1'`	`java.lang.String`	

Groovy Types

- Everything is an object no primitive types.
- Primitive types are auto boxed
- Optional Typing – If not explicitly specified assumed to be java.lang.Object
- Type safe – Unlike some scripting languages, Groovy doesn’t allow one type to be treated as another without a well defined conversion being available.

Def keyword

- The def keyword is used to define an untyped variable or a function in Groovy, as it is an optionally-typed language.
- When we're unsure of the type of a variable or field, we can leverage def to let Groovy decide types at runtime based on the assigned values:
- Def readln=<Swing Dialog Box>
- def firstName = readln “Enter name”
- def listOfCountries = ['USA', 'UK', 'FRANCE', 'INDIA']
- Here, firstName will be a String, and listOfCountries will be an ArrayList.
- We can also use the def keyword to define the return type of a method:
- def multiply(x, y) {
- return x*y
- }

Types Example


```
a = 1          // Implicit typing to Integer
b = 'howdy'    // Implicit typing to String
int c = 33     // Explicit typing to Integer
def d = 5.2f // def keyword means any type

println 'a is ' + a.class.name
println 'b is ' + b.class.name
println 'c is ' + c.class.name
println 'd is ' + d.class.name  // class name as property
```

Output

```
a is java.lang.Integer
b is java.lang.String
c is java.lang.Integer
d is java.lang.Float
```

Operator Overloading

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

Operator	Method
a << b	a.leftShift(b)
a >> b	a.rightShift(b)
a >>> b	a.rightShiftUnsigned(b)

switch(a){
case b:
}
	b.isCase(a)

a == b	a.equals(b)
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

Strings

- Single Quotes – java.lang.String
- Double Quotes – groovy.lang.GString – Allow replacement of ${vars}
- Triple Single Quotes – Multi Line Strings, Newlines always \n, Whitespace preserved.
- Triple Double Quotes – Multi Line GStrings
- Forward Slash – Escape backslashes ignored, makes Regular Expressions more readable

Strings

Start/end characters	Example	GString aware?	Backslash escapes?
Single quote	`'hello Dierk'`	No	Yes
Double quote	`"hello $name"`	Yes	Yes

Triple single quote (`'''`)	`'''------------`
	`Total: $0.02`
	`------------'''`
		No	Yes

Triple double quote (`"""`)	`"""first line`
	`second line`
	`third line"""`
		Yes	Yes

Forward slash	`/x(\d*)y/`	Yes	Occasionally[a]

Strings


```
me = 'Tarzan'
you = 'Jane'
line = "me ${me} - you $you"
assert line == 'me Tarzan - you Jane'
// Note abbrev dollar syntax $you
// Note == is equality not identity

date = new Date(0)
out = "Year $date.year Month $date.month Day $date.date"
assert out == 'Year 70 Month 0 Day 1'
out = "Date is ${date.toGMTString()} !"
assert out == 'Date is 1 Jan 1970 00:00:00 GMT !'
// Note $date.month access month property
```

Strings

```
// Multi line
sql = """
SELECT FROM MyTable
WHERE Year = $date.year
"""

// Literal dollar sign
out = "my 0.02\$"

// Slashy Strings, don't need to escape
assert "abc" == /abc/
assert "\\d" == /\d/
```

Regular Expressions

- Find Operator: =~
- Creates a matcher.
- Match Operator: ==~
- Matches a regular expression.
- Pattern Operator: ~String
- Creates a pattern from a string.

Regular Expressions


```
// Find operator
assert "aaa1bbb2ccc3" =~ /bbb/

// Match operator - must match entire string
assert "aaa1bbb2ccc3" ==~ /(...\d)*/

// Patterns
def p = ~/a*b/
assert p instanceof Pattern
def m = p.matcher("aaaaab");
assert m.matches()
```

Ranges

- Specify an upper and lower bound of some sequence.
- Can be used to replace:
- if (a >= 0 && a <= upperBound) {
- // do something with a
- }
- Ranges are Objects
- lowerbound..upperbound: inclusive
- Lowerbound..<upperbound: half inclusive
- Supports Numbers, Strings, Dates, Reverse Ranges.
- Can create your own, by overriding next(), previous() and implements Comparable

Ranges Example

```
assert (0..10).contains(0)
assert (0..10).contains(5)
assert (0..10).contains(10)
assert (0..10).contains(-1) == false
assert (0..10).contains(11) == false
assert (0..<10).contains(9)
assert (0..<10).contains(10) == false
assert ('a'..'c').contains('b')
log = ''

// Replacement for Java for statements.
for (element in 5..9){
log += element
}
assert log == '56789'
```

Lists and Maps

- Groovy has language level support for maps
- Lists specified with [item, item, item] syntax
- java.util.ArrayList under the hood
- Maps specified with [name1:value, name2:value, name3:value] syntax
- java.util.HashMap under the hood.
- Use [ ] operator to access.
- Define closures to ease iteration.

Lists - Specifying


```
myList = [1,2,3]
assert myList.size() == 3
assert myList[0] == 1
assert myList instanceof ArrayList

emptyList = []
assert emptyList.size() == 0

longList = (0..1000).toList()
assert longList[555] == 555

explicitList = new ArrayList()
explicitList.addAll(myList)
assert explicitList.size() == 3
explicitList[0] = 10
assert explicitList[0] == 10
```

Lists – Overloaded Subscript Operator


```
myList = ['a','b','c','d','e','f']

assert myList[0..2] == ['a','b','c'] // getAt(Range)

assert myList[0,2,4] == ['a','c','e'] // getAt(Index Collection)

// putAt(Range)
myList[0..2] = ['x','y','z']
assert myList == ['x','y','z','d','e','f']

myList[3..5] = []
assert myList == ['x','y','z']

myList[1..1] = ['y','1','2']
assert myList == ['x','y','1','2','z']
```

Lists – Negative Indexes

- Count from the back of the list. The last index is -1.
- Can be used with ranges.
- list[-3..-1] gives you the last three entries.
- list[1..-2] to cut away the first and the last entry.

Lists – Adding and Removing


```
myList = []

myList += 'a' // plus(Object)
assert myList == ['a']

myList += ['b','c'] // plus(Collection)
assert myList == ['a','b','c']

myList = []
myList << 'a' << 'b' // leftShift is like 'append'
assert myList == ['a','b']

assert myList - ['b'] == ['a'] // Minus to remove
```

Lists – GDK Methods

- Groovy Development Kit adds many methods to the collection classes.
- Many take closures to perform some operation on the collection
- Examples: min, max, sort, sum, collect, each, find, unique etc…

Lists Iteration with Closures


```
def list = [1,2,3]

// Find even numbers
def even = list.find {item -> item % 2 == 0}
assert even == 2

// Is every number less than 5?
assert list.every { item -> item < 5}

// Is any number less than 2
assert list.any { item -> item < 2}

// Append each item to store
def store = ''
list.each { item -> store += item }
assert store == '123'
```

Maps - Specifying


```
def myMap = [a:1, b:2, c:3]
assert myMap instanceof HashMap
assert myMap.size() == 3
assert myMap['a'] == 1

def emptyMap = [:]
assert emptyMap.size() == 0

def explicitMap = new TreeMap()
explicitMap.putAll(myMap)
assert explicitMap['a'] == 1
```

Maps - Access


```
def myMap = [a:1, b:2, c:3]

// retrieve
assert myMap['a'] == 1
assert myMap.a == 1
assert myMap.get('a') == 1

assert myMap.d == null
assert myMap.get('d',0) == 0
assert myMap.d == 1 // now it's there

// assign
myMap['d'] = 1
assert myMap.d == 1
myMap.d = 2
assert myMap.d == 2
```

Maps - Iteration


```
def myMap = [a:1, b:2, c:3]

// Dump contents by each entry
myMap.each {entry ->
    println "$entry.key = $entry.value"
}

// Dump contents by each key/value
myMap.each {key, value ->
    println "$key = $value"
}

// Dump contents by for
for (key in myMap.keySet())
    println "$key = " + myMap[key]
```

Closures

- A closure in Groovy is an open, anonymous, block of code that can take arguments, return a value and be assigned to a variable.
- Closure may reference variables declared in its surrounding scope.
- In opposition to the formal definition of a closure, Closure in the Groovy language can also contain free variables which are defined outside of its surrounding scope.
- https://groovy-lang.org/closures.html

Closures

- A closure definition follows this syntax:
- { [closureParameters ->] statements }
- Where [closureParameters->] is an optional comma-delimited list of parameters, and statements are 0 or more Groovy statements.
- The parameters look similar to a method parameter list, and these parameters may be typed or untyped.

Closures


```groovy
{ item++ }                                          // 1

{ -> item++ }                                       // 2

{ println it }                                      // 3

{ it -> println it }                                // 4

{ name -> println name }                            // 5

{ String x, int y ->                                // 6
    println "hey ${x} the value is ${y}"
}

{ reader ->                                         // 7
    def line = reader.readLine()
    line.trim()
}
```

1. A closure referencing a variable named `item`
2. It is possible to explicitly separate closure parameters from code by adding an arrow (`->`)
3. A closure using an implicit parameter (`it`)
4. An alternative version where `it` is an explicit parameter

Closures

A closure is an instance of the `groovy.lang.Closure` class, making it assignable to a variable or a field as any other variable, despite being a block of code:

```
def listener = { e -> println "Clicked on $e.source" }      (1)
assert listener instanceof Closure
Closure callback = { println 'Done!' }                      (2)
Closure<Boolean> isTextFile = {
    File it -> it.name.endsWith('.txt')                     (3)
}
```

1. You can assign a closure to a variable, and it is an instance of `groovy.lang.Closure`
2. If not using `def` or `var`, use `groovy.lang.Closure` as the type
3. Optionally, you can specify the return type of the closure by using the generic type of `groovy.lang.Closure`

Closures

- A powerful and integral feature of Groovy. You won't be able to avoid them.
- Essentially a block of code wrapped in an object.
- Similar to anonymous inner classes in Java but less verbose and more flexible.
- Aka C# Delegate, Lisp Lambda, C function pointers.
- Useful for iteration, using resources safely.
- Avoids interface proliferation (Runnables, Observers, Listeners, Visitors, Comparators, Strategies, Commands, Controllers)
- Have access to local variables without having to make them final as Java anonymous inner classes require.
- Proposal from Gosling and others to add Closures to Java 7.

Closures - Specifying


```
def list = [1,2,3]

// Closure to print contents of a list
def closure = { x -> println x }
list.each(closure)

// Simplify 1 - Create closure within each
list.each({ x -> println x })

// Simplify 2 - () not required if closure last param
list.each { x -> println x }

// Simplify 3 - 'it' is default parameter name if one param
list.each { println it }
```

Closures - Calling


```
// Create a closure
def adder = { x, y -> return x + y }

// Call it using the call method
assert adder.call(2, 6) == 8

// Abbreviated call
assert adder(4, 3) == 7

// In a method
void adjustSalary(Closure adjustment) {
    for(e in employees) {
        adjustment(e)
    }
}
```

Closures


```
map = ['a': 1, 'b': 2]
map.each {key, value -> map[key] = value * 2}
assert map == ['a': 2, 'b': 4]

doubler = {key, value -> map[key] = value * 2}
map.each(doubler)
assert map == ['a': 4, 'b': 8]

def doubleMethod(entry) {
    map[entry.key] = entry.value * 2
}
doubler = this.&doubleMethod
map.each(doubler)
assert map == ['a': 8, 'b': 16]
```

Normal Parameters

- def closureWithOneArg = { str -> str.toUpperCase() }
- assert closureWithOneArg('groovy') == 'GROOVY'
- def closureWithOneArgAndExplicitType = { String str -> str.toUpperCase() }
- assert closureWithOneArgAndExplicitType('groovy') == 'GROOVY'
- def closureWithTwoArgs = { a,b -> a+b }
- assert closureWithTwoArgs(1,2) == 3
- def closureWithTwoArgsAndExplicitTypes = { int a, int b -> a+b }
- assert closureWithTwoArgsAndExplicitTypes(1,2) == 3
- def closureWithTwoArgsAndOptionalTypes = { a, int b -> a+b }
- assert closureWithTwoArgsAndOptionalTypes(1,2) == 3
- def closureWithTwoArgAndDefaultValue = { int a, int b=2 -> a+b }
- assert closureWithTwoArgAndDefaultValue(1) == 3

Implicit parameter

- When a closure does not explicitly define a parameter list (using ->), a closure always defines an implicit parameter, named it. This means that this code:
- def greeting = { "Hello, $it!" }
- assert greeting('Patrick') == 'Hello, Patrick!’

- is strictly equivalent to this one:
- def greeting = { it -> "Hello, $it!" }
- assert greeting('Patrick') == 'Hello, Patrick!'

Varargs

It is possible for a closure to declare variable arguments like any other method. *Vargs* methods are methods that can accept a variable number of arguments if the last parameter is of variable length (or an array) like in the next examples:

```
def concat1 = { String... args -> args.join('') }          (1)
assert concat1('abc','def') == 'abcdef'                    (2)
def concat2 = { String[] args -> args.join('') }           (3)
assert concat2('abc', 'def') == 'abcdef'

def multiConcat = { int n, String... args ->               (4)
    args.join('')*n
}
assert multiConcat(2, 'abc','def') == 'abcdefabcdef'
```

1. A closure accepting a variable number of strings as first parameter
2. It may be called using any number of arguments **without** having to explicitly wrap them into an array
3. The same behavior is directly available if the *args* parameter is declared as an array
4. As long as the **last** parameter is an array or an explicit vargs type

Groovy Truth

- Groovy evaluates conditions in if, while and for statements the same way as Java does for standard Java conditions : in Java you must provide a boolean expression and the result is the result of the evaluation.
- In Groovy , the result is the same as in Java for those conditions.

Groovy Truth

- The other truthfulness evaluation mechanism shown by examples can be summarized as:
- numbers: a zero value evaluates to false, non zero to true.
- objects: a null object reference evaluates to false, a non null reference to true.
- Character : a character with a zero value evaluates to false, true otherwise.
- String : a string evaluates to true if not null and not empty, false if null or empty (applies to GStrings and CharSequences too).
- Collections and Maps (including subclasses List, Map, Set, HashSet ...) : also takes into account the size, evaluates to true if the collection is not null and not empty, false if null or empty.
- Enumerations and Iterators evaluates to true if not null and they are some more elements (groovy evaluates hasMoreElements or hasNext on the object), false if null or no more elements.
- Matcher : a matcher evaluates to true if there is at least one match, false if not match is found.
- Closure : a closure evaluates to the evaluation of the result returned by the closure.

Groovy Truth

Broader than Java which just uses Boolean tests to determine truth.

Boolean	Corresponding boolean value is true
Matcher	The matcher has a match
Collection	The collection is non-empty
Map	Whether the map is non-empty
String	The string is non-empty
Number	The number is non-zero
None of the above	The object reference is non-null

Groovy Truth


```
assert true
assert !false
```
Boolean values are trivial

```
assert ('a' =~ /./)
assert !('a' =~ /b/)
```
Matchers must match

```
assert [1]
assert ![]
```
Collections must be non-empty

```
assert ['a':1]
assert ![:]
```
Maps must be non-empty

```
assert 'a'
assert !''
```
Strings must be non-empty

```
assert 1
assert 1.1
assert 1.2f
assert 1.3g
assert 2L
assert 3G
assert !0
```
Numbers (any type) must be nonzero

```
assert new Object()
assert !null
```
Any other value must be non-null

Groovy Truth


```
if (true)          assert true
else               assert false

if (1) {
    assert true
} else {
    assert false
}

if ('non-empty') assert true
else if (['x'])  assert false
else             assert false

if (0)           assert false
else if ([])     assert false
else             assert true
```

Groovy Truth

```
def a = 1
def log = ''
switch (a) {
    case 0  : log += '0'
    case 1  : log += '1'
    case 2  : log += '2' ; break
    default : log += 'default'
}
assert log == '12'
```

Fall through

Switch Statement

- An object that defines isCase method can be the candidate of a switch statement.
- All of the standard Java classes have isCase defined.

Object	a.equals(b)
Class	a.isInstance(b)
Collection	a.contains(b)
Range	a.contains(b)
Pattern	a.matcher(b.toString()).matches()
String	(a==null && b==null) || a.equals(b)
Closure	a.call(b)

Switch Statement Example


```
switch (10)
{
   case 0 : assert false ; break
   case 0..9 : assert false ; break
   case [8,9,11] : assert false ; break
   case Float : assert false ; break
   case {it%3 == 0}: assert false ; break
   case ~/../ : assert true ; break
   default : assert false ; break
}
```

Looping

- Mostly the same as Java.
- Standard Java for not supported.

  Use for(x in 0..9) instead
- Enhanced for loop, like Java 5 but uses in rather than ':' and doesn't requires lists to be templated.
- In many cases, closures can be used instead.

Regular Expression

Symbol	Meaning
.	Any character
^	Start of line (or start of document, when in single-line mode)
$	End of line (or end of document, when in single-line mode)
\d	Digit character
\D	Any character except digits
\s	Whitespace character
\S	Any character except whitespace
\w	Word character
\W	Any character except word characters
\b	Word boundary
()	Grouping
(x|y)	x or y as in (Groovy|Java|Ruby)
x*	Zero or more occurrences of x
x+	One or more occurrences of x
x?	Zero or one occurrence of x
x{m,n}	At least m and at most n occurrences of x
x{m}	Exactly m occurrences of x
[a-f]	Character class containing the characters *a, b, c, d, e, f*
[^a]	Character class containing any character except *a*
(?=regex)	Positive lookahead
(?<=text)	Positive lookbehind

Regular Expression


```groovy
assert "Hello World!" =~ /Hello/        // Find operator
assert "Hello World!" ==~ /Hello\b.*/   // Match operator
def p = ~/Hello\b.*/                    // Pattern operator
assert p.class.name == 'java.util.regex.Pattern'

def input = "Voting is open between 01 Nov 2008 and 04 Nov 2008"
def dateFormat = /\d\d? ... \d{4}/
def matches = (input =~ dateFormat)

assert matches[1] == '04 Nov 2008'

matches.each { m -> println m }

input.eachMatch(dateFormat) { m -> println m }

assert input.findAll(dateFormat) == ['01 Nov 2008', '04 Nov 2008']

println input.replaceAll(dateFormat, '?/?/?')
```

```
01 Nov 2008
04 Nov 2008
```

```
Voting is open between ?/?/? and ?/?/?
```

Regular Expression


```groovy
// "Voting is open between 01 Nov 2008 and 04 Nov 2008"
def dateFormatGroups = /((\d\d?) (...) (\d{4})/
def matchingGroups = (input =~ dateFormatGroups)

assert matchingGroups[1][1] == '04'

matchingGroups.each { all, d, m, y -> println "$d/$m/$y" }

input.eachMatch(dateFormatGroups) { all, d, m, y ->
    println "$d/$m/$y"
}

assert input.findAll(dateFormatGroups) {
    full, d, m, y -> "$m $d of $y"
} == ['Nov 01 of 2008', 'Nov 04 of 2008']

println input.replaceAll(dateFormatGroups) {
    full, d, m, y -> "$m $d ????"
}
```

```
01/Nov/2008
04/Nov/2008
```

```
Voting is open between Nov 01 ???? and Nov 04 ????
```

Exceptions

- Mostly the same as java with the following differences.
- Declaration of exceptions in method signatures is optional. (Even for checked exceptions)
- You therefore don't have to catch checked exceptions, they just go up the call stack

Markup Builder

- **Markup Builder**

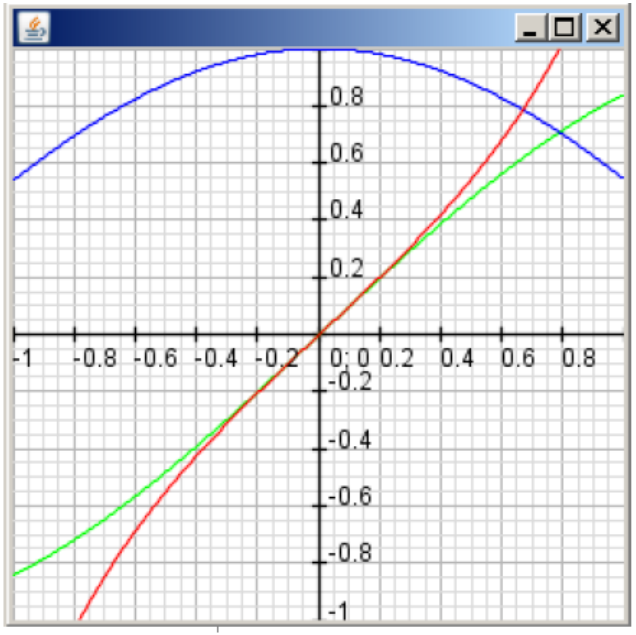
```
import groovy.xml.*
def page = new MarkupBuilder()
page.html {
  head { title 'Hello' }
  body {
    ul {
      for (count in 1..5) {
        li "world $count"
} } } }
```

```
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <ul>
      <li>world 1</li>
      <li>world 2</li>
      <li>world 3</li>
      <li>world 4</li>
      <li>world 5</li>
    </ul>
  </body>
</html>
```

Swing Builder


```
import java.awt.FlowLayout
builder = new groovy.swing.SwingBuilder()
langs = ["Groovy", "Ruby", "Python", "Pnuts"]

gui = builder.frame(size: [290, 100],
          title: 'Swinging with Groovy!') {
    panel(layout: new FlowLayout()) {
        panel(layout: new FlowLayout()) {
            for (lang in langs) {
                checkBox(text: lang)
            }
        }
        button(text: 'Groovy Button', actionPerformed: {
            builder.optionPane(message: 'Indubitably Groovy!').
                createDialog(null, 'Zen Message').show()
        })
        button(text: 'Groovy Quit',
                actionPerformed: {System.exit(0)})
    }
}
gui.show()
```

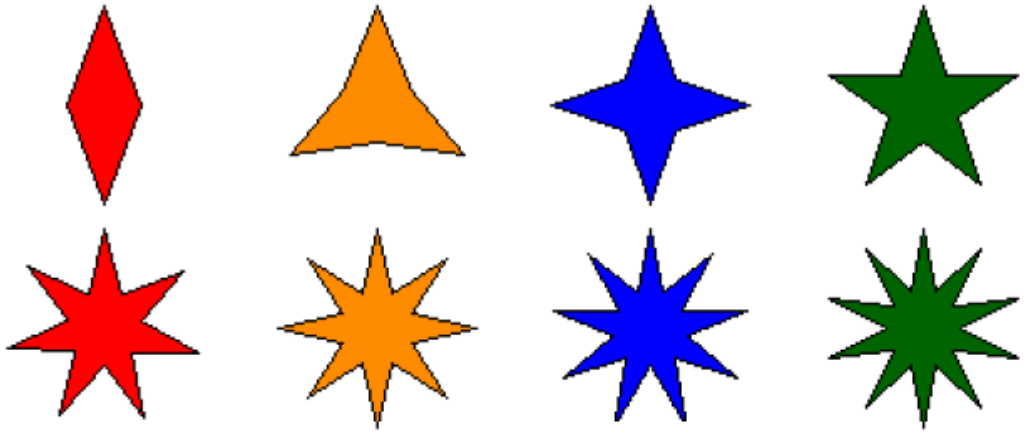
SwingX Builder




```
import groovy.swing.SwingXBuilder
import static java.awt.Color.*
import static java.lang.Math.*

def swing = new SwingXBuilder()
def frame = swing.frame(size: [300, 300]) {
    graph(plots: [
        [GREEN, {value -> sin(value)}],
        [BLUE,  {value -> cos(value)}],
        [RED,   {value -> tan(value)}]
    ])
}.show()
```

Graphics Builder


```
def colors = ['red','darkOrange','blue','darkGreen']
(0..3).each { index ->
  star( cx: 50 + (index*110), cy: 50, or: 40, ir: 15,
    borderColor: 'black', count: 2+index, fill: colors[index] )
  star( cx: 50 + (index*110), cy: 140, or: 40, ir: 15,
    borderColor: 'black', count: 7+index, fill: colors[index] )
}
```



Ant Builder


```groovy
def ant = new AntBuilder()

ant.echo("hello")    // let's just call one task

// create a block of Ant using the builder pattern
ant.sequential {
    myDir = "target/AntTest/"
    mkdir(dir: myDir)
    copy(todir: myDir) {
        fileset(dir: "src/test") {
            include(name: "**/*.groovy")
        }
    }
    echo("done")
}

// now let's do some normal Groovy again
file = new File("target/test/AntTest.groovy")
assert file.exists()
```

Ant Builder


```groovy
def ant = new AntBuilder()
ant.echo(file:'Temp.java', '''
class Temp {
  public static void main(String[] args) {
    System.out.println("Hello");
  }
}
''')
ant.with {
  javac(srcdir:'.', includes:'Temp.java', fork:'true')
  java(classpath:'.', classname:'Temp', fork:'true')
  echo('Done')
}
// =>
//     [javac] Compiling 1 source file
//      [java] Hello
//      [echo] Done
```

Ant Builder


```
import static groovy.xml.NamespaceBuilder.newInstance as namespace
def ant = new AntBuilder()
def mvn = namespace(ant, 'antlib:org.apache.maven.artifact.ant')
def antunit = namespace(ant, 'antlib:org.apache.ant.antunit')

direct = [groupId:'jfree', artifactId:'jfreechart', version:'1.0.9']
indirect = [groupId:'jfree', artifactId:'jcommon', version:'1.0.12']

// download artifacts
mvn.dependencies(filesetId:'artifacts') { dependency(direct) }

// print out what we downloaded
ant.fileScanner { fileset(refid:'artifacts') }.each { println it }

// use AntUnit to confirm expected files were downloaded
def prefix = System.properties.'user.home' + '/.m2/repository'
[direct, indirect].each { item ->
    def (g, a, v) = [item.groupId, item.artifactId, item.version]
    antunit.assertFileExists(file:"$prefix/$g/$a/$v/$a-${v}.jar")
}
```

```
C:\Users\paulk\.m2\repository\jfree\jcommon\1.0.12\jcommon-1.0.12.jar
C:\Users\paulk\.m2\repository\jfree\jfreechart\1.0.9\jfreechart-1.0.9.jar
```

OOPS

- **Similar capabilities to Java**
  - Define classes, interfaces, enums, annotations
- **Differences to Java**
  - Classes (and interfaces etc.) public by default
  - Methods public by default
  - Property support within classes (auto-setters/getters)
  - Multimethods (runtime dispatch – "duck typing")

```
class BikeGroovyBean {
    String manufacturer, model, serialNo, status
    Integer frame
    Double weight
    BigDecimal cost
}
```

Object Orientation – Runtime Interfaces

- **Java has these methods in java.util.Collections:**

> `min(Collection<? extends T> coll, Comparator<? super T> comp)`
> Returns the minimum element of the given collection, according to the order induced by the specified comparator.

> `max(Collection<? extends T> coll, Comparator<? super T> comp)`
> Returns the maximum element of the given collection, according to the order induced by the specified comparator.

- **In Groovy, no need to create a class implementing `Comparator` having a `compare` method, just use a map of Closures instead** (though Groovy has special support for this example which is even more concise):

```
def animals = ['Antelope', 'Bee', 'Zebra']
def comparator = [
    compare:{ a, b -> a.size() <=> b.size() }
] as Comparator
assert comparator instanceof Comparator
assert Collections.min(animals, comparator) == 'Bee'
assert Collections.max(animals, comparator) == 'Antelope'
```

Classes

- Classes defined very much like in Java.
- Public by default
- Default visibility (not public, protected or private) will automatically adds bean methods.
- Defining member variable type is optional however can't stand alone.
- Fields can be referenced with subscript operator myclass['myfield']
- Can also customize the field access operator '.' via get / set overrides.

Class – Field Access


```
class Counter {
    public count = 0
}
def counter = new Counter()

// Using field access operator
counter.count = 1

// using subscript operator
def fieldName = 'count'
counter[fieldName] = 2
```

Class – Field Access

```
class SomeClass {

    public    fieldWithModifier
    String    typedField
    def       untypedField
    protected field1, field2, field3
    private   assignedField = new Date()

    static    classField

    public static final String CONSTA = 'a', CONSTB = 'b'

    def someMethod(){
        def localUntypedMethodVar = 1
        int localTypedMethodVar = 1
        def localVarWithoutAssignment, andAnotherOne
    }
}

def localvar = 1
boundvar1 = 1

def someMethod(){
    localMethodVar = 1
    boundvar2 = 1
}
```

Class – Overriding Field Access


```
class PretendFieldCounter {
   public count = 0
   Object get (String name) { return 'pretend value' }
   void set (String name, Object value) { count++ }
}
def pretender = new PretendFieldCounter()

// Call get method via overridden '.' operator
assert pretender.isNoField == 'pretend value'

// Call set method via overriden '.' operator
pretender.isNoFieldEither = 'just to increase counter'
assert pretender.count == 1
```

Method Declaration

- Public by default
- void returns can be omitted
- Types in argument list can be omitted, Object assumed.
- Dynamic dispatch used to resolve methods.
- Defaults allowed in argument list.
- Optionals parameters if last argument is Object[]
- Common practice to use Maps for 'named' arguments.

Method Example

```
class SomeClass {

    static void main(args) {
        def some = new SomeClass()
        some.publicVoidMethod()
        assert 'hi' == some.publicUntypedMethod()
        assert 'ho' == some.publicTypedMethod()
        combinedMethod()
    }

    void publicVoidMethod(){
    }

    def publicUntypedMethod(){
        return 'hi'
    }
    String publicTypedMethod(){
        return 'ho'
    }

    protected static final void combinedMethod(){
    }
}
```

❶ Implicit public

Call static method of current class

Method Example

```
class Summer {
    def sumWithDefaults(a, b, c=0){
        return a + b + c
    }
    def sumWithList(List args){
        return args.inject(0){sum,i -> sum += i}
    }
    def sumWithOptionals(a, b, Object[] optionals){
        return a + b + sumWithList(optionals.toList())
    }
    def sumNamed(Map args){
        ['a','b','c'].each{args.get(it,0)}
        return args.a + args.b + args.c
    }
}

def summer = new Summer()

assert 2 == summer.sumWithDefaults(1,1)
assert 3 == summer.sumWithDefaults(1,1,1)

assert 2 == summer.sumWithList([1,1])
assert 3 == summer.sumWithList([1,1,1])
```

1. Explicit arguments and a default value
2. Define arguments as a list
3. Optional arguments as an array
4. Define arguments as a map

Method Example


```
class Summer {
    def sumWithDefaults(a, b, c=0) { return a + b + c }

    def sumWithOptionals(a, b, Object[] optionals) {
       return a + b + optionals.inject(0) { sum, i -> sum += i }
    }

    def sumNamed(Map args) {
        ['a','b','c'].each{ args.get( it, 0 ) }
        return args.a + args.b + args.c
    }
}
def summer = new Summer()
assert 2 == summer.sumWithDefaults(1,1)
assert 3 == summer.sumWithDefaults(1,1,1)
assert 2 == summer.sumWithOptionals(1,1)
assert 3 == summer.sumWithOptionals(1,1,1)
assert 2 == summer.sumNamed(a:1, b:1)
assert 1 == summer.sumNamed(c:1)
```

Record

A record is a class that:

- Is implicitly final (so can't be extended)
- Has a private final field for each component, e.g. color
- Has an accessor method for each component of the same name, e.g. color()
- Has a default Point(int, int, String) constructor
- Has a default serialVersionUID of 0L and special serialization code
- Has implicit toString(), equals() and hashCode() methods
- Implicitly extends the java.lang.Record class (so can't extend another class but may implement one or more interfaces)

Record

```
record Point(int x, int y, String color) { }
```

It is equivalent to the following traditional declaration:

```
@RecordType
class Point {
    int x
    int y
    String color
}
```

Record

```
record Point(int x, int y, String color) { }
```

It is equivalent to the following traditional declaration:

```
@RecordType
class Point {
    int x
    int y
    String color
}
```

Record

```
@RecordBase
@RecordOptions
@TupleConstructor(namedVariant = true, force = true, defaultsMode = AUTO)
@PropertyOptions
@KnownImmutable
@POJO
@CompileStatic
class Point {
    int x
    int y
    String color
}
```

Record

```
@Sortable
record Point(int x, int y, String color) { }

var points = [
    new Point(0, 100, 'red'),
    new Point(10, 10, 'blue'),
    new Point(100, 0, 'green'),
]

println points.toSorted(Point.comparatorByX())
println points.toSorted(Point.comparatorByY())
println points.toSorted(Point.comparatorByColor())
```

Category


```
@Category(Integer)
 class IntegerOps {
     def triple() {
         this * 3
     }
 }

 use (IntegerOps) {
     assert 25.triple() == 75
 }
```

Category


```
@Category(List)
 class Shuffler {
     def shuffle() {
         def result = new ArrayList(this)
         Collections.shuffle(result)
         result
     }
 }
```

Category


```
class Sentence extends ArrayList {
    Sentence(Collection initial) { super(initial) }
}
Sentence.mixin Shuffler

def words = ["The", "quick", "brown", "fox"]
println new Sentence(words).shuffle()
// => [quick, fox, The, brown]       (order will vary)
```

Newify

```
 @Newify([Author, Book])
def createKingPython() {
    Author(name: 'Stephen King', books: [
        Book(title: 'Carrie'),
        Book(title: 'The Shining'),
        Book(title: 'It')
    ])
}
```

Safe Dereferencing

- The ?. operator can be used to safely dereference member variables without worring about null pointer exceptions.
- When the reference before that operator is a null reference, the evaluation of the current expression stops and null is returned.

```
def map = [a:[b:[c:1]]]

// Normal . operator
assert map.a.b.c == 1

// Safe dereferencing
assert map?.a?.x?.c == null
```

Constructors

- Public by default
- If implicit can construct
  - Empty constructor
  - Named params
- If explicit can construct
  - Traditional Java
  - Using as keyword
  - Implicit type conversion

```
emp = new Employee()
emp = new Employee(name: 'Bob', age: 32)

class Employee {
   def name
   def age
}
```

```
emp = new Employee('Bob', 32)
emp2 = ['Joe', 41] as Employee
Employee emp3 = ['Tom', 50]

class Employee {
   Employee(name, age) {
    this.name = name; this.age = age
   }
   def name
   def age
}
```

Constructors

```
class VendorWithCtor {
    String name, product

    VendorWithCtor(name, product) {
        this.name    = name
        this.product = product
    }
}

def first = new VendorWithCtor('Canoo','ULC')

def second = ['Canoo','ULC'] as VendorWithCtor

VendorWithCtor third = ['Canoo','ULC']
```

Constructor definition

Normal constructor use

❶ Coercion with as

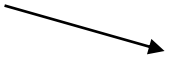
❷ Coercion in assignment

.groovy Files

- If a .groovy file contains no class, it becomes a class of type Script using the filename as the class name. main automatically added.
- If it contains exactly one class with same name as file, then 1 to 1 relationship as in Java.
- Can contain multiple classes of any visibility
- Can mix scripting code and classes. No need to worry about ordering. Classes can be referenced before declared.
- Groovy classloader will search for .groovy files on the classpath. Will only find classes where name matches file.

Imports

- By default groovy imports the following packages into all .groovy files
  - 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
- Import can be used for 'type aliasing' using as keyword



```
import thirdparty.Element as Element2
```

Multimethods

- Groovy takes dynamic type of arguments into account when resolving methods. Java uses static type.

```
def oracle(Object o) { return 'object' }
def oracle(String o) { return 'string' }

Object x = 1
Object y = 'foo'

assert 'object' == oracle(x)
assert 'string' == oracle(y) // would return 'object' in Java
```

GroovyBeans

- Allows property access for all JavaBeans, defined in Java or Groovy. (`myclass.myproperty`)
- Auto Generates get / set methods for properties members in groovy classes if member variables have default visibility.
- Also get / set only generated if you have not already defined an accessor. This allows you to create read only or write only properties.

GroovyBeans Example


```
class MyBean implements java.io.Serializable {
    String foo, bar
    // Make bar read only
    String getBar() { return bar }
}

bean = new MyBean(foo: 'foo', bar: 'bar')
println bean.foo
bean.bar = 'error' // this won't be allowed

d = new Date()
println d.time // can access this as a property
```

GroovyBean Properties

- All properties in a bean can be accessed with property semantics.

```
class SomeBean {
    int var1
    String var2
    Date var3
}

def bean = new SomeBean(var1: 1, var2: "foo", var3: new Date());

bean.properties.each { println it.key + " = " + it.value }
```

Spread Dot Operator

- '.*' use to access a property on each element in a list.

```
class Person {
    Person(def name) { this.name = name }
    def name
}

def list = [new Person('Bob'), new Person('Tom'),
new Person('Sal')]

// Java approach
for(Iterator i = list.iterator(); i.hasNext();)
    System.out.println(((Person) i).getName());

// Closure
println list.collect{ p -> p?.name }

// Spread dot operator
println list*.name
```

GDK Methods File Management

```groovy
def out = new File('result.txt')
out.delete()
new File('..').eachFileRecurse {file ->
  if (file.name.endsWith('.groovy')) {
    file.eachLine { line, num ->
      if (line.toLowerCase().contains('groovy'))
        out <<
          "File '$file' on line $num\n$line\n\n"
    }
  }
}
```

```
File '..\files\src\PrintGroovySourceLines.groovy' on line 4
  if (file.name.endsWith('.groovy')) {

File '..\files\src\PrintGroovySourceLines.groovy' on line 6
      if (line.toLowerCase().contains('groovy'))

File '..\jdbc\src\JdbcGroovy.groovy' on line 1
import groovy.sql.Sql
…
```

ESDC 2010 - 74

GDK Methods Process Management


```groovy
// windows version (minimal error/output)
def p = "cmd /c dir *.xml".execute()
p.waitFor()
if (p.exitValue()) println p.err.text
else println p.text
```

```
----- stdout -----
Volume in drive D is DATA
Volume Serial Number is FC5C-B39D

Directory of D:\svntrunk-groovy\groovy-core

27/06/2009  10:13 AM        38,478 build.xml
29/06/2009  06:53 PM        27,617 pom.xml
               2 File(s)         66,095 bytes
               0 Dir(s)  15,503,597,568 bytes free
```

```groovy
// unix version
def p = "ls *.xml".execute()
def err = new StringBuffer()
def out = new StringBuffer()
// could ignore/use Stream below
p.consumeProcessOutput(out, err)
def exitValue = p.waitFor()
if (err) println """----- stderr -----
Return Code: $exitValue
$err"""
if (out) println "----- stdout -----\n$out"
```

```
----- stderr -----
Return Code: 1
ls: *.xml: No such file or directory
```

GDK Methods Process Management

```
sout: tstfl.grvy
serr:
```

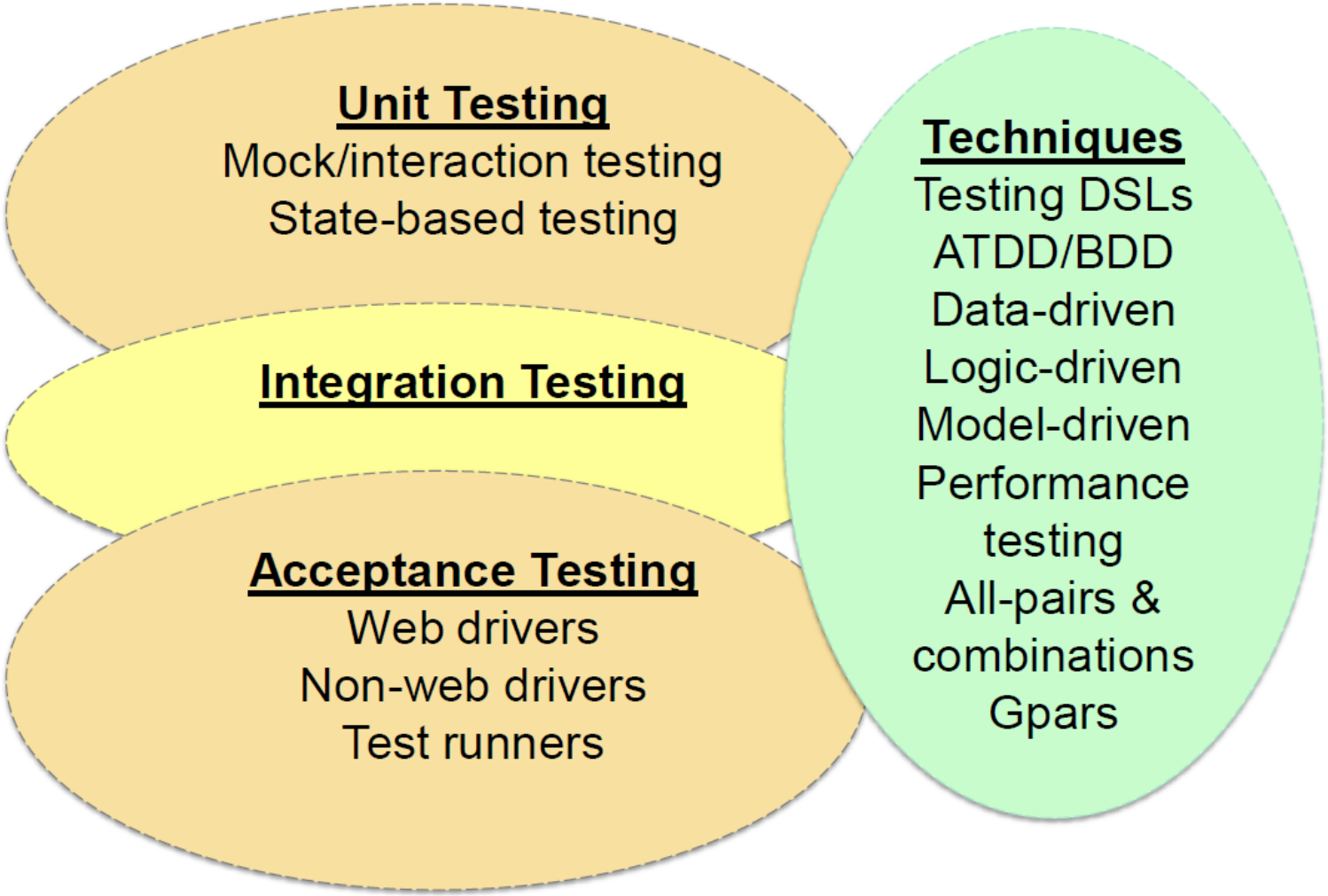
```groovy
def sout = new StringBuffer()
def serr = new StringBuffer()
proc1 = 'tr -d o'.execute()
proc2 = 'tr -d e'.execute()
proc3 = 'tr -d i'.execute()
proc3.consumeProcessOutput(sout, serr)
proc1 | proc2 | proc3
[proc1, proc2].each{ it.consumeProcessErrorStream(serr) }
proc1.withWriter { writer ->
    writer << 'testfile.groovy'
}
proc3.waitForOrKill(1000)
println 'sout: ' + sout
println 'serr: ' + serr
```

GDK Methods Thread Management


```
Thread.start{
  4.times {
    println 'Hello from thread 1'
    sleep 400
  }
}
Thread.start{
  5.times {
    println 'Hello from thread 2'
    sleep 300
  }
}
7.times {
  println 'Hello from main thread'
  sleep 200
}
```

```
Hello from thread 2
Hello from thread 1
Hello from main thread
Hello from main thread
Hello from thread 2
Hello from thread 1
Hello from main thread
Hello from thread 2
Hello from main thread
Hello from thread 1
Hello from main thread
Hello from thread 2
Hello from main thread
Hello from thread 2
Hello from thread 1
Hello from main thread
```

Testing


Unit Testing
Mock/interaction testing
State-based testing
Integration Testing
Acceptance Testing
Web drivers
Non-web drivers
Test runners
Techniques
Testing DSLs
ATDD/BDD
Data-driven
Logic-driven
Model-driven
Performance testing
All-pairs & combinations
Gpars

Testing

- **Unit testing**
  - Built-in asserts, support for JUnit 3&4 and TestNG, GroovyTestCase with shouldFail and other methods
  - Built-in mocking and compatible with Java mocking
- **Integration testing**
  - Metaprogramming allows various kinds of IOC like intercepting and hooking up of components
  - Wealth of GDK methods for Ant, Processes, Files, Threads, etc. make the automating part much simpler
- **Acceptance Testing and Generally**
  - Allows creation of English-like testing DSLs using Closures, builders, metaprogramming
  - Simpler syntax great for non hard-core testers
  - Grapes make tests easier to share

XML Management

records.xml

```
<records>
   <car name='HSV Maloo' make='Holden' year='2006'>
      <country>Australia</country>
      <record type='speed'>Production Pickup Truck with speed of 271kph</record>
   </car>
   <car name='P50' make='Peel' year='1962'>
      <country>Isle of Man</country>
      <record type='size'>Smallest Street-Legal Car at 99cm wide and 59 kg weight</record>
   </car>
   <car name='Royale' make='Bugatti' year='1931'>
      <country>France</country>
      <record type='price'>Most Valuable Car at $15 million</record>
   </car>
</records>
```

XML Management

```
def records = new XmlParser().parse("records.xml")
records.car.each {
    println "year = ${it.@year}"
}
```

```
year = 2006
year = 1962
year = 1931
```

XML Management

- **Reading XML**
  - Special Groovy support: XmlParser, XmlSlurper, DOMCategory
    - *All support GPath, XmlSlurper features Lazy loading*
  - Or Groovy sugar for your current favorites: DOM, SAX, StAX, DOM4J, JDom, XOM, XPath, XSLT, XQuery, etc.
- **Creating XML**
  - Special Groovy support: MarkupBuilder and StreamingMarkupBuilder
  - Or again, enhanced syntax for your current favorites
- **Updating XML**
  - Using above: read followed by create
  - Can be done with XmlParser, XmlSlurper, DOMCategory
  - Or with your Java favorites
- **Verifying XML**
  - Also DTD, W3C XML Schema, Relax NG in a similar fashion to Java mechanisms for these features

XML Management


```
def records = new XmlSlurper().parse("records.xml")
assert  3 == records.car.size()                                  // 3 records in total
assert  10 == records.depthFirst().collect{ it }.size()          // 10 nested nodes
// test properties of the first record
def firstRecord = records.car[0]
assert  'car' == firstRecord.name()
assert  'Holden' == firstRecord.@make.toString() && 'Australia' == firstRecord.co
// 2 cars have an 'e' in the make
assert  2 == records.car.findAll{ it.@make.toString().contains('e') }.size()
// 2 cars have an 'e' in the make
assert  2 == records.car.findAll{ it.@make =~ '.*e.*' }.size()
// makes of cars that have an 's' followed by an 'a' in the country
assert  ['Holden', 'Peel'] == records.car.findAll{ it.country =~ '.*s.*a.*' }.@make.col
// types of records
assert  ['speed', 'size', 'price'] == records.depthFirst().grep{ it.@type != '' }.'@type'
assert  ['speed', 'size', 'price'] == records.'**'.grep{ it.@type != '' }.'@type'*.toStrin
// check parent() operator
def countryOne = records.car[1].country
assert  'Peel' == countryOne.parent().@make.toString() && 'Peel' == countryOne.'
// names of cars with records sorted by year
def names = records.car.list().sort{ it.@year.toInteger() }.'@name'*.toString()
assert  ['Royale', 'P50', 'HSV Maloo'] == names
```

XML Management


```groovy
import java.text.SimpleDateFormat
import com.thoughtworks.xstream.XStream

class Athlete {
    String firstname, lastname, gender, country, dateOfBirth
}

def asDate(dateStr) { new SimpleDateFormat("yyyy-MM-dd").parse(dateStr) }

def athleteList = [
    new Athlete(firstname: 'Paul', lastname: 'Tergat',
                dateOfBirth: '1969-06-17', gender: 'M', country: 'KEN'),
    new Athlete(firstname: 'Khalid', lastname: 'Khannouchi',
                dateOfBirth: '1971-12-22', gender: 'M', country: 'USA'),
    new Athlete(firstname: 'Sammy', lastname: 'Korir',
                dateOfBirth: '1971-12-12', gender: 'M', country: 'KEN'),
    new Athlete(firstname: 'Ronaldo', lastname: 'da Costa',
                dateOfBirth: '1970-06-07', gender: 'M', country: 'BRA'),
    new Athlete(firstname: 'Paula', lastname: 'Radcliffe',
                dateOfBirth: '1973-12-17', gender: 'F', country: 'GBR')
]

// create XML as input
def input = new XStream().toXML(athleteList)
```

XML Management

```
def athletes = new XmlSlurper().parseText(input).Athlete

def bornSince70 = {
    asDate(it.dateOfBirth.text()) > asDate('1970-1-1') }
def excludingKh = { it.lastname.text() <= 'Kg' ||
                    it.lastname.text() >= 'Ki' }
def byGenderDescThenByCountry = { a, b ->
    a.gender.text() == b.gender.text() ?
        a.country.text() <=> b.country.text() :
        b.gender.text() <=> a.gender.text() }

def someYoungsters = athletes.
    findAll(bornSince70).findAll(excludingKh).list().
    sort(byGenderDescThenByCountry)

someYoungsters.each {
    def name = "$it.firstname $it.lastname".padRight(25)
    println "$name ($it.gender) $it.country $it.dateOfBirth"
}
```

Object Management


```
import java.text.SimpleDateFormat

class Athlete {
    def firstname, lastname, gender, country, dateOfBirth
}

def asDate(dateStr) {
    new SimpleDateFormat("yyyy-MM-dd").parse(dateStr)
}

def athletes = [
    new Athlete(firstname: 'Paul', lastname: 'Tergat',
                dateOfBirth: '1969-06-17', gender: 'M', country: 'KEN'),
    new Athlete(firstname: 'Khalid', lastname: 'Khannouchi',
                dateOfBirth: '1971-12-22', gender: 'M', country: 'USA'),
    new Athlete(firstname: 'Sammy', lastname: 'Korir',
                dateOfBirth: '1971-12-12', gender: 'M', country: 'KEN'),
    new Athlete(firstname: 'Ronaldo', lastname: 'da Costa',
                dateOfBirth: '1970-06-07', gender: 'M', country: 'BRA'),
    new Athlete(firstname: 'Paula', lastname: 'Radcliffe',
                dateOfBirth: '1973-12-17', gender: 'F', country: 'GBR')
]
```

Object Management

```
def bornSince70 = {
    asDate(it.dateOfBirth) > asDate('1970-1-1') }
def excludingKh = { it.lastname <= 'Kg' ||
                    it.lastname >= 'Ki' }
def byGenderDescThenByCountry = { a, b ->
    a.gender == b.gender ?
        a.country <=> b.country :
        b.gender <=> a.gender }

def someYoungsters = athletes.
    findAll(bornSince70).
    findAll(excludingKh).
    sort(byGenderDescThenByCountry)

someYoungsters.each {
    def name = "$it.firstname $it.lastname".padRight(25)
    println "$name ($it.gender) $it.country $it.dateOfBirth"
}
```

SQL Connection

```groovy
import groovy.sql.Sql

def url      = 'jdbc:hsqldb:file:EMPDB'
def username = 'sa'
def password = ''
def driver   = 'org.hsqldb.jdbcDriver'

def db = Sql.newInstance(url, username, password, driver)

db.eachRow("SELECT id, firstname, lastname FROM Employees") {
    println "$it.id: $it.firstname $it.lastname"
}
```

SQL Connection

- **Using standard SQL statements**

```
import groovy.sql.Sql

def foo = 'cheese'
def db = Sql.newInstance("jdbc:mysql://localhost:3306/mydb",
                         "user", "pswd", "com.mysql.jdbc.Driver")

db.eachRow("select * from FOOD where type=${foo}") {
    println "Gromit likes ${it.name}"
}
```

- **Using DataSets**

```
import groovy.sql.Sql

def db = Sql.newInstance("jdbc:mysql://localhost:3306/mydb",
                         "user", "pswd", "com.mysql.jdbc.Driver")

def food = db.dataSet('FOOD')
def cheese = food.findAll { it.type == 'cheese' }
cheese.each { println "Gromit likes ${it.name}" }
```

SQL Connection

```
athleteSet = db.dataSet('Athlete')
youngsters = athleteSet.findAll{ it.dateOfBirth > '1970-1-1'}
paula = youngsters.findAll{ it.firstname == 'Paula'}

println paula.sql
// =>
// select * from Athlete where dateOfBirth > ? and firstname =

println paula.parameters
// =>
// [1970-1-1, Paula]

paula.each { println it.lastname }    // database called here
// =>
// Radcliffe
```

SQL Data Access


```
import groovy.sql.Sql

dbHandle = null

def getDb() {
    if (dbHandle) return dbHandle
    def source = new org.hsqldb.jdbc.jdbcDataSource()
    source.database = 'jdbc:hsqldb:mem:GIA'
    source.user = 'sa'
    source.password = ''
    dbHandle = new Sql(source)
    return dbHandle
}

db.execute '''
    DROP   INDEX athleteIdx IF EXISTS;
    DROP   TABLE Athlete IF EXISTS;
    CREATE TABLE Athlete (
        athleteId   INTEGER GENERATED BY DEFAULT AS IDENTITY,
        firstname   VARCHAR(64),
        lastname    VARCHAR(64),
        country     VARCHAR(3),
        gender      CHAR(1),
        dateOfBirth DATE
    );
    CREATE INDEX athleteIdx ON Athlete (athleteId);
'''
```

SQL Data Access


```
def athleteList = [
    [firstname: 'Paul', lastname: 'Tergat',
     dateOfBirth: '1969-06-17', gender: 'M', country: 'KEN'],
    [firstname: 'Khalid', lastname: 'Khannouchi',
     dateOfBirth: '1971-12-22', gender: 'M', country: 'USA'],
    [firstname: 'Sammy', lastname: 'Korir',
     dateOfBirth: '1971-12-12', gender: 'M', country: 'KEN'],
    [firstname: 'Ronaldo', lastname: 'da Costa',
     dateOfBirth: '1970-06-07', gender: 'M', country: 'BRA'],
    [firstname: 'Paula', lastname: 'Radcliffe',
     dateOfBirth: '1973-12-17', gender: 'F', country: 'GBR']
]

def athletes = db.dataSet('Athlete')
athleteList.each {a -> athletes.add(a)}
```

SQL Data Access

```
def bornSince70 = { it.dateOfBirth > '1970-1-1' }
def excludingKh = { it.lastname <= 'Kg' || it.lastname >= 'Ki' }

def someYoungsters = athletes.
    findAll(bornSince70).
    findAll(excludingKh).
    sort { it.gender }.reverse().
    sort { it.country }

println someYoungsters.sql + '\n' + someYoungsters.parameters

someYoungsters.each {
    def name = "$it.firstname $it.lastname".padRight(25)
    println "$name ($it.gender)   $it.country   $it.dateOfBirth"
}
/* =>
select * from Athlete where dateOfBirth > ? and (lastname <= ? or lastname
>= ?) order by gender DESC, country
["1970-1-1", "Kg", "Ki"]
Ronaldo da Costa          (M)   BRA   1970-06-07
Sammy Korir               (M)   KEN   1971-12-12
Paula Radcliffe           (F)   GBR   1973-12-17       */
```

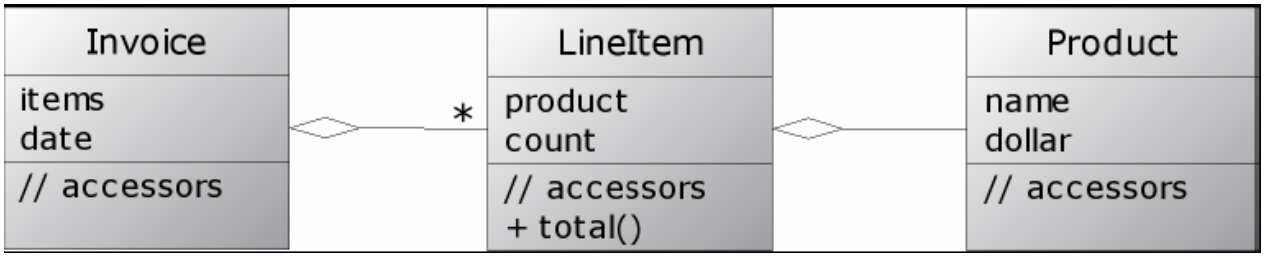
GPath

- GPath is a construction in Groovy code that powers object navigation, analogous to XPath
- Builds upon bean properties, closures, safe dereferencing operator and spread dot operator.

```
// Find all the 'get' methods in this class
println this.class.methods*.name.grep(~/get.*/).sort()

// Output
[getBinding, getClass, getMetaClass, getProperty]
```

GPath Example2

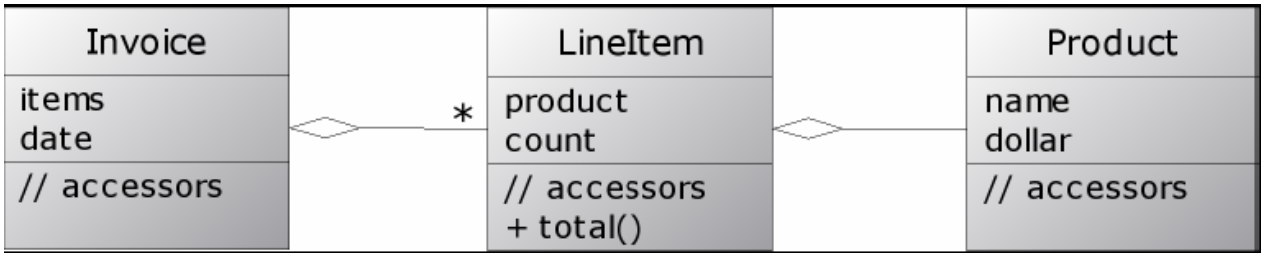
- Given the following model




```
class Invoice { List items; Date date }

class LineItem {
    Product product; int count;
    int total() {
        return product.dollar * count
    }
}

class Product { String name; def dollar}
```

GPath Example2

- You can perform GPath operations like the following.




```
// get the totals of all line items in all invoices
invoices.items*.total()

// find all product names with a line item totals > 7000
invoices.items.grep{it.total() > 7000}.product.name
```

Mix In Categories

- 'use' keyword
- Allows you to augment a class's available instance

```
class Person {
    Person(name, male) { this.name = name; this.male = male}
    private def name
    boolean male
}

// Add getTitle method to Person class
class PersonTitleCategory {
    static String getTitle(Person self) {
        if(self.male)
            return 'Mr. ' + self.name
        else
            return 'Ms. ' + self.name
    }
}
```

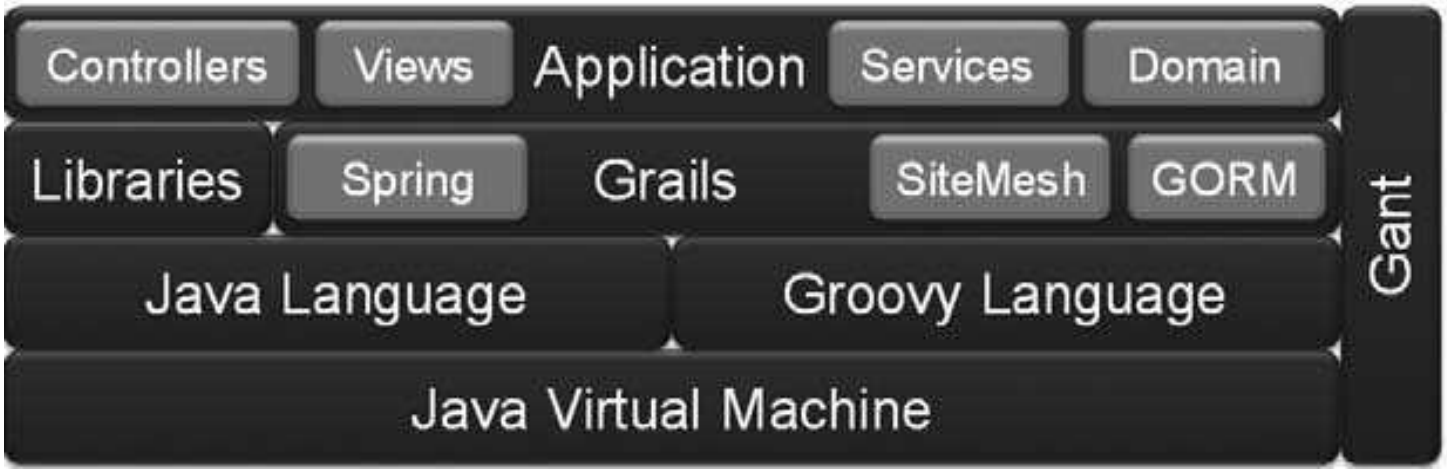
Mix In Categories


```
kim = new Person('Kim', false);
bob = new Person('Bob', true);

println kim.name
println bob.name

use (PersonTitleCategory) {
    println kim.title
    println bob.title
}

Output:
Kim
Bob
Ms. Kim
Mr. Bob
```

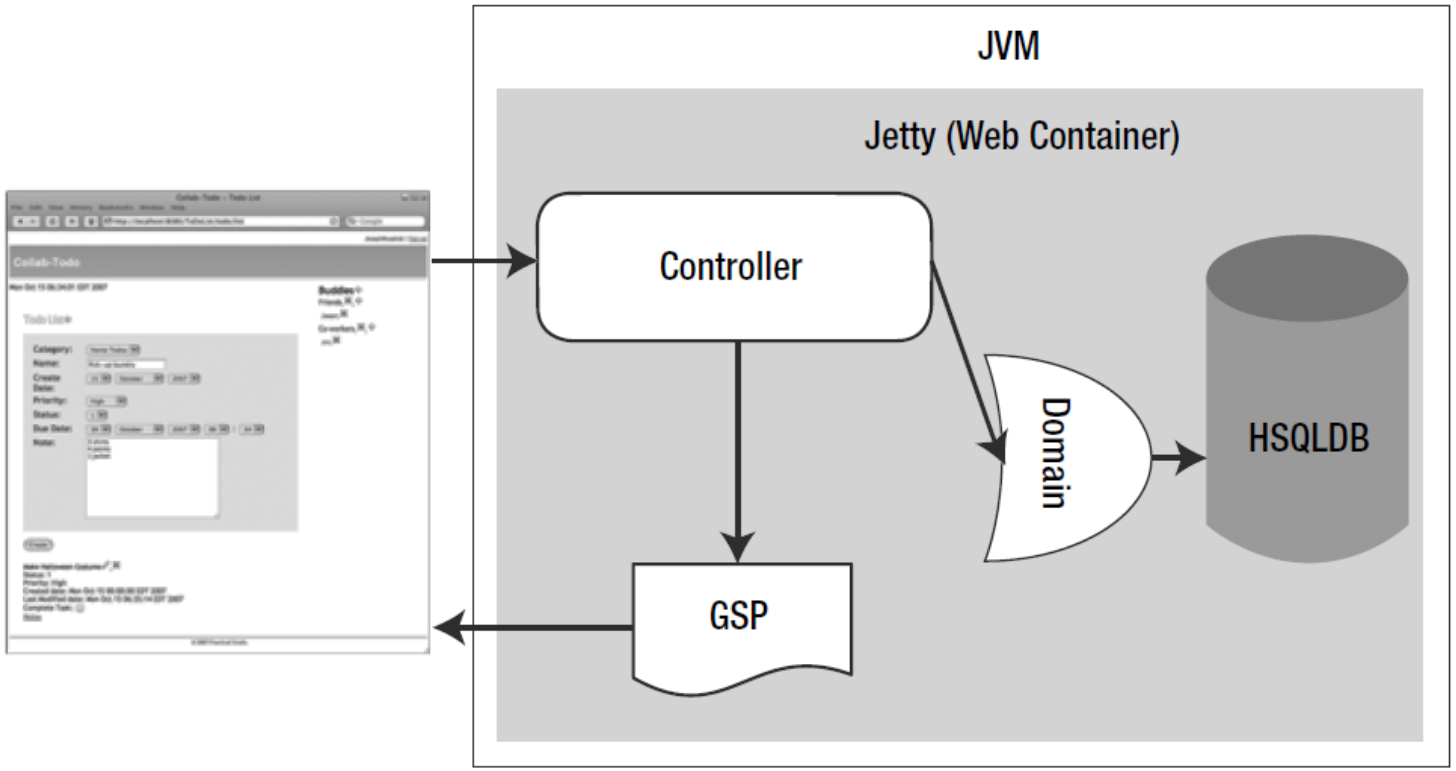
What is Grails?

- Grail is an open-source web application framework that uses the Apache Groovy programming language to develop real projects.
- It is a plug-in-based framework that uses its build system-(Gant) but plans to migrate to Gradle.
- The Grails homepage provides numerous pre-defined plugins which extend the Grails framework.
- The Grails framework allows for instance development without requiring any configuration.
- Grails uses JavaEE as the structural basis and Spring for structuring the application via dependency injection.

Grails Architecture



Grails Default Runtime



Understanding the Scaffolding Process

- Create an application.
- Run the application.
- Create the domain class.
- Implement the integration test.
- Run the test harness and update the domain classes until the tests pass.
- Create the controller.
- Repeat steps 3 through 6 until the domain class and the controllers are complete.

Features of Grails

- Grails framework is opted by the startup organizations to launch the product from the development mode to production mode.
- It ensures a different way of creating web apps to the standard Java mechanisms of servlet programming, Java Server Faces, JSPs, etc.

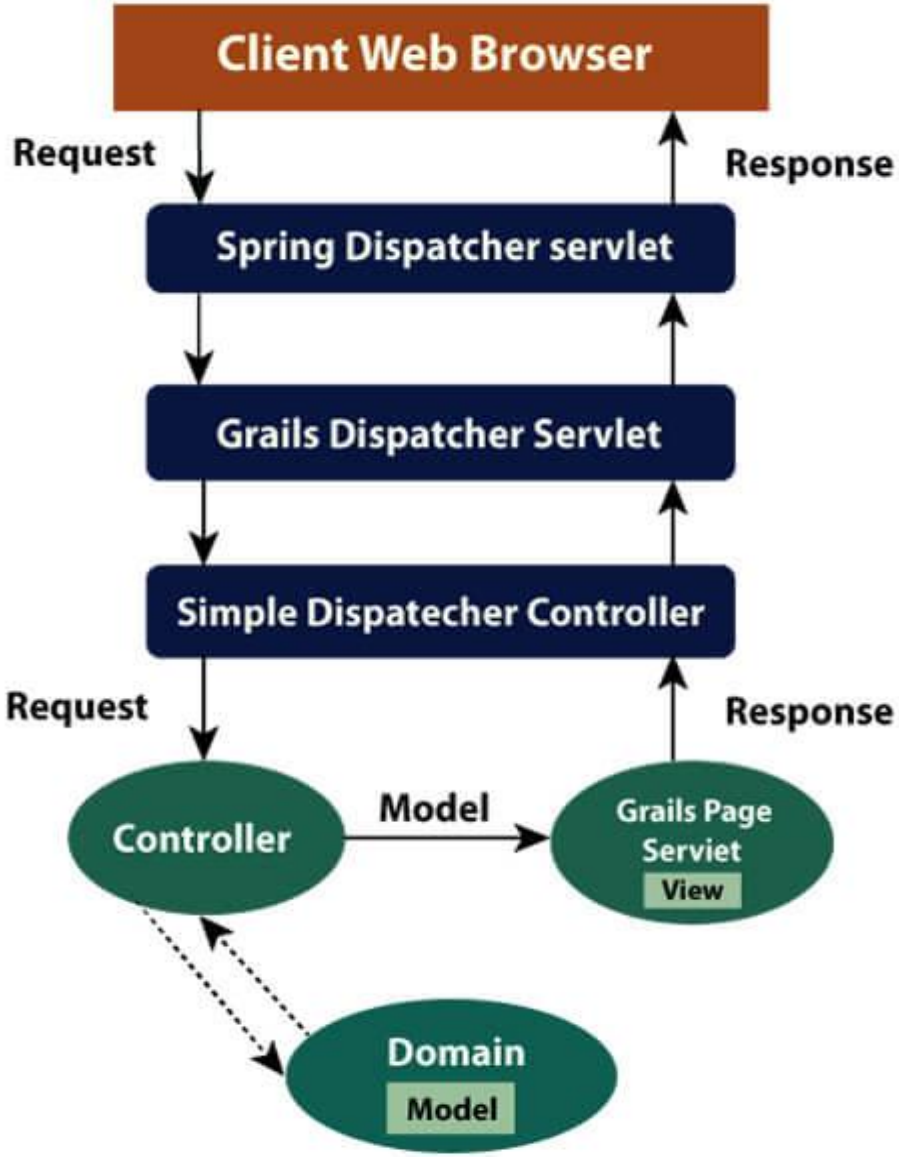
Features of Grails

- Smart Coding
- Vibrant and Supportive Community
- Grails Scaffolding
- Compatibility
- Extensive Plugins
- Minimal Server Restart
- Shorter Learning Curve
- https://guides.grails.org/creating-your-first-grails-app/guide/index.html

Difference between Grails and Groovy

Groovy	Grails
Groovy is an object-oriented programming language for the Java platform.	Grails (previously known as “Groovy on Grails”) is a programming framework based on Groovy and inspired by Ruby on Rails.
All groovy codes are compiled to Java byte codes which can be executed by JVM.	It’s a RAPID Web Application development framework which provides a tremendous boost in development by abstracting its common requirements in web development.
Groovy has a Java-like syntax and works seamlessly with Java bytecode.	Grail uses Spring , hibernate and a bunch of other libraries.
All groovy codes are compiled to Java byte codes which can be executed by JVM.	Grail is the advanced version of Groovy.

Grails MVC


Client Web Browser
Request
Response
Spring Dispatcher servlet
Grails Dispatcher Servlet
Simple Dispatecher Controller
Request
Response
Controller
Model
Grails Page
Serviet
View
Domain
Model

Grails Framework

Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>grails -v
| Grails Version: 4.0.10
| JVM Version: 1.8.0_251
C:\WINDOWS\system32>
```

Administrator: Command Prompt

```
C:\WINDOWS\system32>cd F:\groovyvebws

C:\WINDOWS\system32>f:

F:\groovyvebws>grails create-app RESTService
Resolving dependencies..
| Application created at F:\groovyvebws\RESTService
F:\groovyvebws>
```

Grails Framework

Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>grails -v
| Grails Version: 5.1.2
| JVM Version: 1.8.0_251
C:\WINDOWS\system32>cd F:\groovyoptumws

C:\WINDOWS\system32>f:

F:\groovyoptumws>grails create-app dentalcareapp
Resolving dependencies..
| Application created at F:\groovyoptumws\dentalcareapp
F:\groovyoptumws>grails list-profiles
Resolving dependencies....
| Available Profiles
-------------------
* angular - A profile for creating Grails applications with Angular 5+
* rest-api - Profile for REST API applications
* base - The base profile extended by other profiles
* plugin - Profile for plugins designed to work across all profiles
* profile - A profile for creating new Grails profiles
* react - A profile for creating a Grails application with a React frontend
* rest-api-plugin - Profile for REST API plugins
* vue - A profile for creating Grails applications with a Vue.js frontend
* web - Profile for Web applications
* web-plugin - Profile for Plugins designed for Web applications
F:\groovyoptumws>
```

Start

ENG IN 04:53 02/03/2022

Grails Framework


```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22621.1265]
(c) Microsoft Corporation. All rights reserved.

F:\groovyoptumfeb2023>grails -version
| Grails Version: 5.1.2
| JVM Version: 11.0.15
F:\groovyoptumfeb2023>grails create-app com.optum.grailsinsurance
Resolving dependencies..
| Application created at F:\groovyoptumfeb2023\grailsinsurance
F:\groovyoptumfeb2023>dir
 Volume in drive F is New Volume
 Volume Serial Number is 5641-E892

 Directory of F:\groovyoptumfeb2023

23/02/2023  07:09 AM    <DIR>          .
22/02/2023  10:48 PM    <DIR>          day1gradle
22/02/2023  03:52 PM    <DIR>          day1groovy
22/02/2023  05:46 PM    <DIR>          day2groovy
23/02/2023  07:09 AM    <DIR>          grailsinsurance
22/02/2023  10:16 AM           372,187 groovy-swing-4.0.2.jar
21/02/2023  04:14 PM         1,975,593 lombok-1.18.26.jar
22/02/2023  05:51 PM         1,006,904 mysql-connector-java-5.1.49.jar
21/02/2023  11:53 AM             1,836 test.class
21/02/2023  11:51 AM                31 test.groovy
22/02/2023  11:31 PM         1,092,359 testapp.zip
               6 File(s)      4,448,910 bytes
               5 Dir(s)  21,548,871,680 bytes free

F:\groovyoptumfeb2023>cd grailsinsurance

F:\groovyoptumfeb2023\grailsinsurance>
```

Grails Framework

C:\Windows\System32\cmd.e

```
24/02/2023  02:01 PM    <DIR>          grails-liveform-example-master
24/02/2023  02:01 PM           525,293 grails-liveform-example-master.zip
24/02/2023  10:12 AM    <DIR>          grails-sample-ajax-master
23/02/2023  07:49 AM    <DIR>          grailsinsurance
22/02/2023  10:16 AM           372,187 groovy-swing-4.0.2.jar
21/02/2023  04:14 PM         1,975,593 lombok-1.18.26.jar
22/02/2023  05:51 PM         1,006,904 mysql-connector-java-5.1.49.jar
21/02/2023  11:53 AM             1,836 test.class
21/02/2023  11:51 AM                31 test.groovy
22/02/2023  11:31 PM         1,092,359 testapp.zip
               8 File(s)      5,040,078 bytes
              16 Dir(s)  18,236,588,032 bytes free

F:\groovyoptumfeb2023>cd collab-todo

F:\groovyoptumfeb2023\collab-todo>dir
 Volume in drive F is New Volume
 Volume Serial Number is 5641-E892

 Directory of F:\groovyoptumfeb2023\collab-todo

18/04/2023  07:26 PM    <DIR>          .
18/04/2023  07:26 PM    <DIR>          ..
18/04/2023  07:26 PM                94 .gitignore
18/04/2023  07:26 PM             4,254 build.gradle
18/04/2023  07:26 PM    <DIR>          gradle
18/04/2023  07:26 PM               190 gradle.properties
18/04/2023  07:26 PM             5,766 gradlew
18/04/2023  07:26 PM             2,674 gradlew.bat
18/04/2023  07:26 PM    <DIR>          grails-app
18/04/2023  07:26 PM             5,507 grails-wrapper.jar
18/04/2023  07:26 PM             4,672 grailsw
18/04/2023  07:26 PM             2,375 grailsw.bat
18/04/2023  07:26 PM                33 settings.gradle
18/04/2023  07:26 PM    <DIR>          src
               9 File(s)         25,565 bytes
               5 Dir(s)  18,236,588,032 bytes free

F:\groovyoptumfeb2023\collab-todo>grailsw
Resolving dependencies.....................................................
```

Grails Framework


```
.
.
.
.
.


Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.2/userguide/command_line_interface.html#sec:command_line_warnings

CONFIGURE SUCCESSFUL in 3m 49s
| Starting interactive mode...
| Enter a command name to run. Use TAB for completion:
grails>
grails>
```

Grails Framework


```
| Resolving Dependencies. Please wait...

Deprecated Gradle features were used in this build, making it incompatible with Gradle 8.0.

You can use '--warning-mode all' to show the individual deprecation warnings and determine if they come from your own scripts or plugins.

See https://docs.gradle.org/7.2/userguide/command_line_interface.html#sec:command_line_warnings

| Starting interactive mode...
| Enter a command name to run. Use TAB for completion:
grails> help

Usage (optionals marked with *):'
grails [environment]* [target] [arguments]*'


| Examples:
$ grails dev run-app
$ grails create-app books

| Available Commands (type grails help 'command-name' for more info):
| Command Name                        Command Description
-------------------------------------------------------------------------------------------
assemble                              Creates a JAR or WAR archive for production deployment
bug-report                            Creates a zip file that can be attached to issue reports for the current project
clean                                 Cleans a Grails application's compiled sources
compile                               Compiles a Grails application
console                               Runs the Grails interactive console
create-command                        Creates an Application Command
create-controller                     Creates a controller
create-domain-class                   Creates a Domain Class
create-functional-test                Creates a Geb Functional Test
create-integration-test               Creates an integration test
create-interceptor                    Creates an interceptor
create-scaffold-controller            Creates a scaffolded controller
create-script                         Creates a Grails script
create-service                        Creates a Service
create-taglib                         Creates a Tag Library
create-unit-test                      Creates a unit test
```

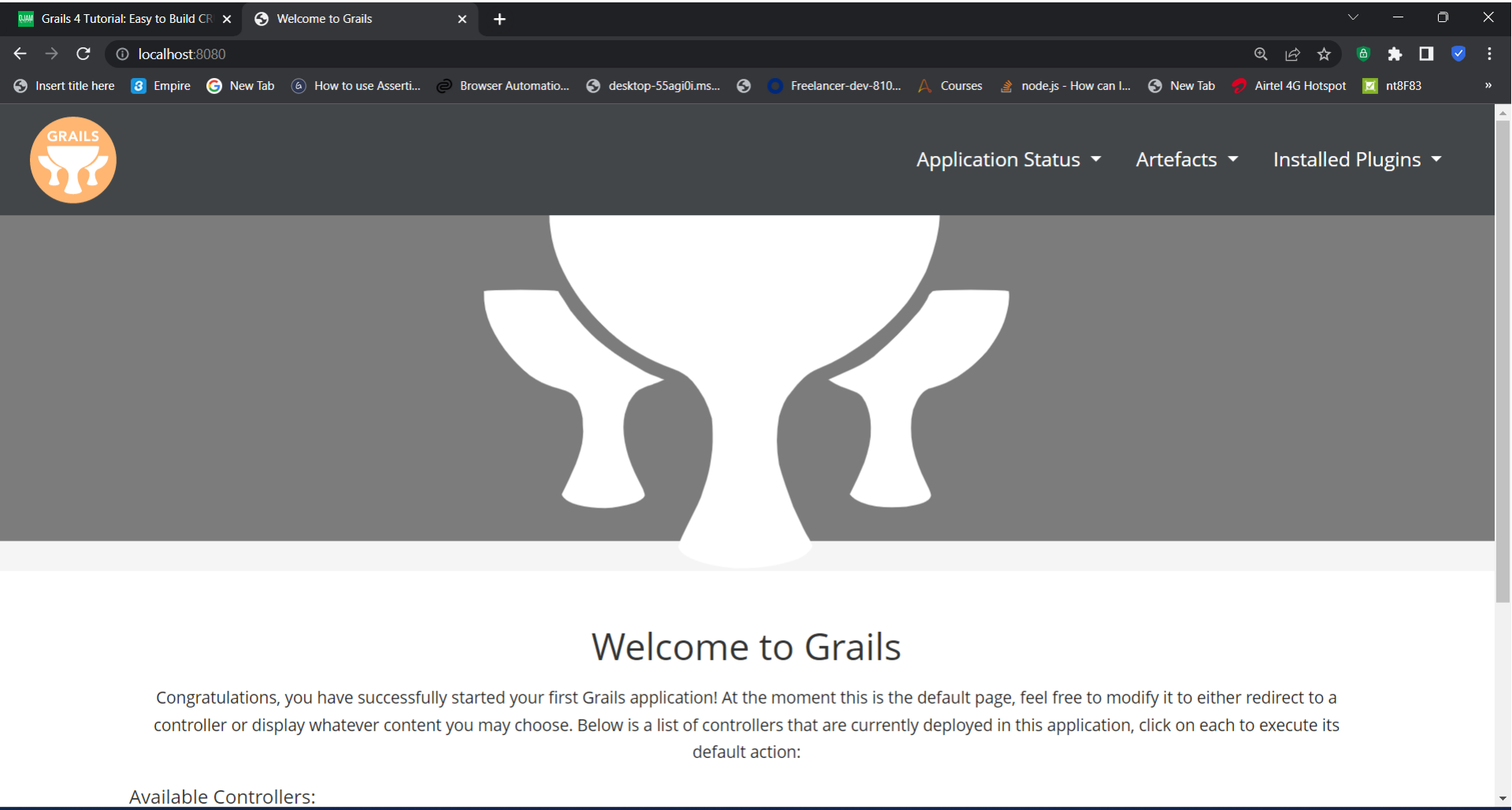
Grails Framework


```
schema-export                          Creates a DDL file of the database schema
shell                                  Runs the Grails interactive shell
stats                                  Prints statistics about the project
stop-app                               Stops the running Grails application
test-app                               Runs the applications tests
url-mappings-report                    Prints out a report of the project's URL mappings

| Detailed usage with help [command]
grails> create-domain-class com.optum.grailsinsurance.models.Team
| Created grails-app/domain/com/optum/grailsinsurance/models/Team.groovy
| Created src/test/groovy/com/optum/grailsinsurance/models/TeamSpec.groovy
grails> create-domain-class com.optum.grailsinsurance.models.Player
| Created grails-app/domain/com/optum/grailsinsurance/models/Player.groovy
| Created src/test/groovy/com/optum/grailsinsurance/models/PlayerSpec.groovy
grails> generate-all com.optum.grailsinsurance.Team
| Error Domain class not found for name com.optum.grailsinsurance.Team
| Error Domain class not found for name com.optum.grailsinsurance.Team
grails> generate-all com.optum.grailsinsurance.models.Team
| Rendered template Controller.groovy to destination grails-app\controllers\com\optum\grailsinsurance\models\TeamController.groovy
| Rendered template Service.groovy to destination grails-app\services\com\optum\grailsinsurance\models\TeamService.groovy
| Rendered template Spec.groovy to destination src\test\groovy\com\optum\grailsinsurance\models\TeamControllerSpec.groovy
| Rendered template ServiceSpec.groovy to destination src\integration-test\groovy\com\optum\grailsinsurance\models\TeamServiceSpec.groovy
| Scaffolding completed for grails-app\domain\com\optum\grailsinsurance\models\Team.groovy
| Rendered template index.gsp to destination grails-app\views\team\index.gsp
| Rendered template edit.gsp to destination grails-app\views\team\edit.gsp
| Rendered template create.gsp to destination grails-app\views\team\create.gsp
| Rendered template show.gsp to destination grails-app\views\team\show.gsp
| Views generated for grails-app\domain\com\optum\grailsinsurance\models\Team.groovy
grails> generate-all com.optum.grailsinsurance.models.Player
| Rendered template Controller.groovy to destination grails-app\controllers\com\optum\grailsinsurance\models\PlayerController.groovy
| Rendered template Service.groovy to destination grails-app\services\com\optum\grailsinsurance\models\PlayerService.groovy
| Rendered template Spec.groovy to destination src\test\groovy\com\optum\grailsinsurance\models\PlayerControllerSpec.groovy
| Rendered template ServiceSpec.groovy to destination src\integration-test\groovy\com\optum\grailsinsurance\models\PlayerServiceSpec.groovy
| Scaffolding completed for grails-app\domain\com\optum\grailsinsurance\models\Player.groovy
| Rendered template index.gsp to destination grails-app\views\player\index.gsp
| Rendered template edit.gsp to destination grails-app\views\player\edit.gsp
| Rendered template create.gsp to destination grails-app\views\player\create.gsp
| Rendered template show.gsp to destination grails-app\views\player\show.gsp
| Views generated for grails-app\domain\com\optum\grailsinsurance\models\Player.groovy
grails>
```

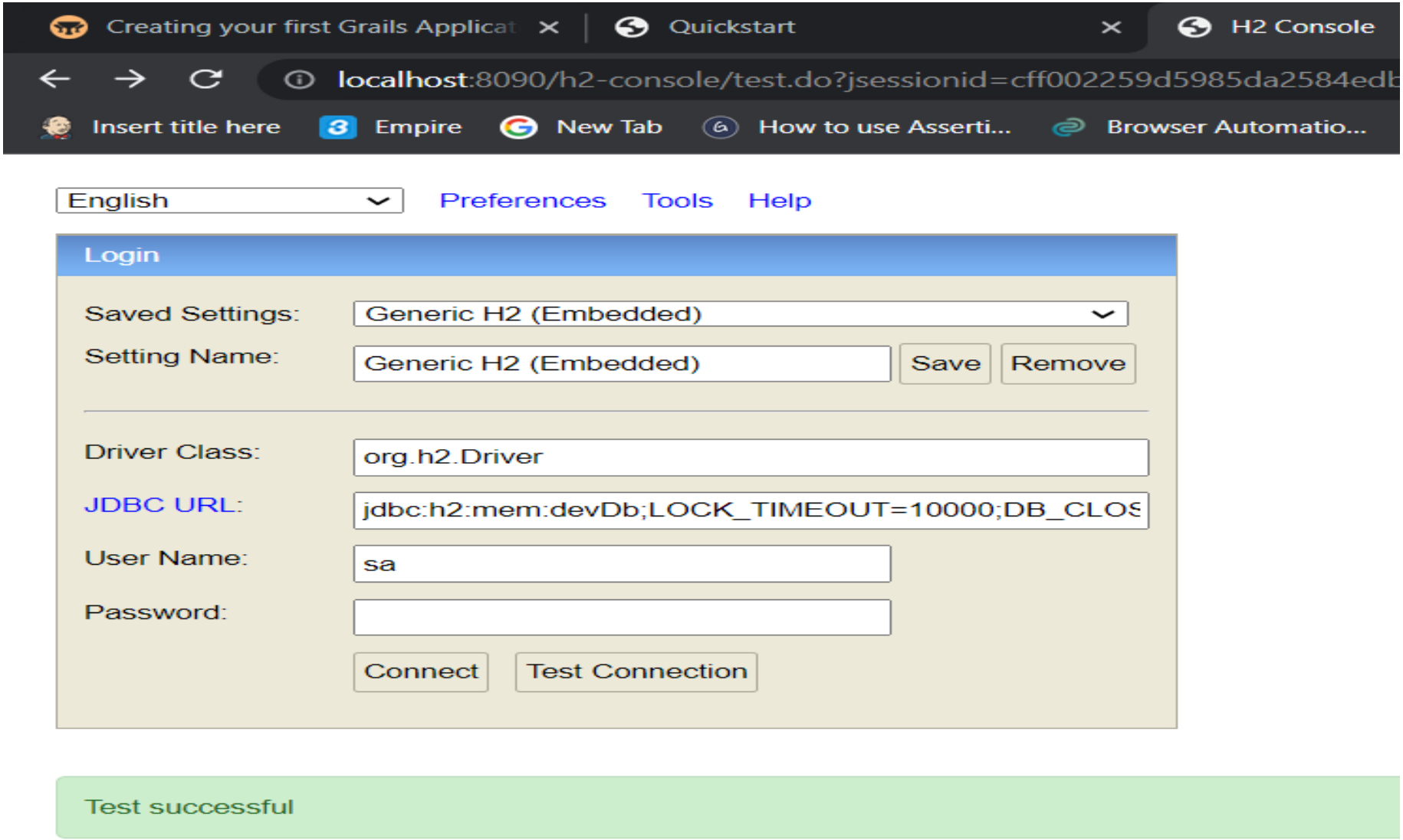
Grails Framework


```
grails> create-domain-class com.optum.grailsinsurance.models.Team
| Created grails-app/domain/com/optum/grailsinsurance/models/Team.groovy
| Created src/test/groovy/com/optum/grailsinsurance/models/TeamSpec.groovy
grails> create-domain-class com.optum.grailsinsurance.models.Player
| Created grails-app/domain/com/optum/grailsinsurance/models/Player.groovy
| Created src/test/groovy/com/optum/grailsinsurance/models/PlayerSpec.groovy
grails> generate-all com.optum.grailsinsurance.Team
| Error Domain class not found for name com.optum.grailsinsurance.Team
| Error Domain class not found for name com.optum.grailsinsurance.Team
grails> generate-all com.optum.grailsinsurance.models.Team
| Rendered template Controller.groovy to destination grails-app\controllers\com\optum\grailsinsurance\models\TeamController.groovy
| Rendered template Service.groovy to destination grails-app\services\com\optum\grailsinsurance\models\TeamService.groovy
| Rendered template Spec.groovy to destination src\test\groovy\com\optum\grailsinsurance\models\TeamControllerSpec.groovy
| Rendered template ServiceSpec.groovy to destination src\integration-test\groovy\com\optum\grailsinsurance\models\TeamServiceSpec.groovy
| Scaffolding completed for grails-app\domain\com\optum\grailsinsurance\models\Team.groovy
| Rendered template index.gsp to destination grails-app\views\team\index.gsp
| Rendered template edit.gsp to destination grails-app\views\team\edit.gsp
| Rendered template create.gsp to destination grails-app\views\team\create.gsp
| Rendered template show.gsp to destination grails-app\views\team\show.gsp
| Views generated for grails-app\domain\com\optum\grailsinsurance\models\Team.groovy
grails> generate-all com.optum.grailsinsurance.models.Player
| Rendered template Controller.groovy to destination grails-app\controllers\com\optum\grailsinsurance\models\PlayerController.groovy
| Rendered template Service.groovy to destination grails-app\services\com\optum\grailsinsurance\models\PlayerService.groovy
| Rendered template Spec.groovy to destination src\test\groovy\com\optum\grailsinsurance\models\PlayerControllerSpec.groovy
| Rendered template ServiceSpec.groovy to destination src\integration-test\groovy\com\optum\grailsinsurance\models\PlayerServiceSpec.groovy
| Scaffolding completed for grails-app\domain\com\optum\grailsinsurance\models\Player.groovy
| Rendered template index.gsp to destination grails-app\views\player\index.gsp
| Rendered template edit.gsp to destination grails-app\views\player\edit.gsp
| Rendered template create.gsp to destination grails-app\views\player\create.gsp
| Rendered template show.gsp to destination grails-app\views\player\show.gsp
| Views generated for grails-app\domain\com\optum\grailsinsurance\models\Player.groovy
grails> run-app
| Running application...
Grails application running at http://localhost:8080 in environment: development
<=======<===========--> 85% EXECUTING [57s]
> IDLE
> IDLE
> IDLE
> :bootRun
grails>
```

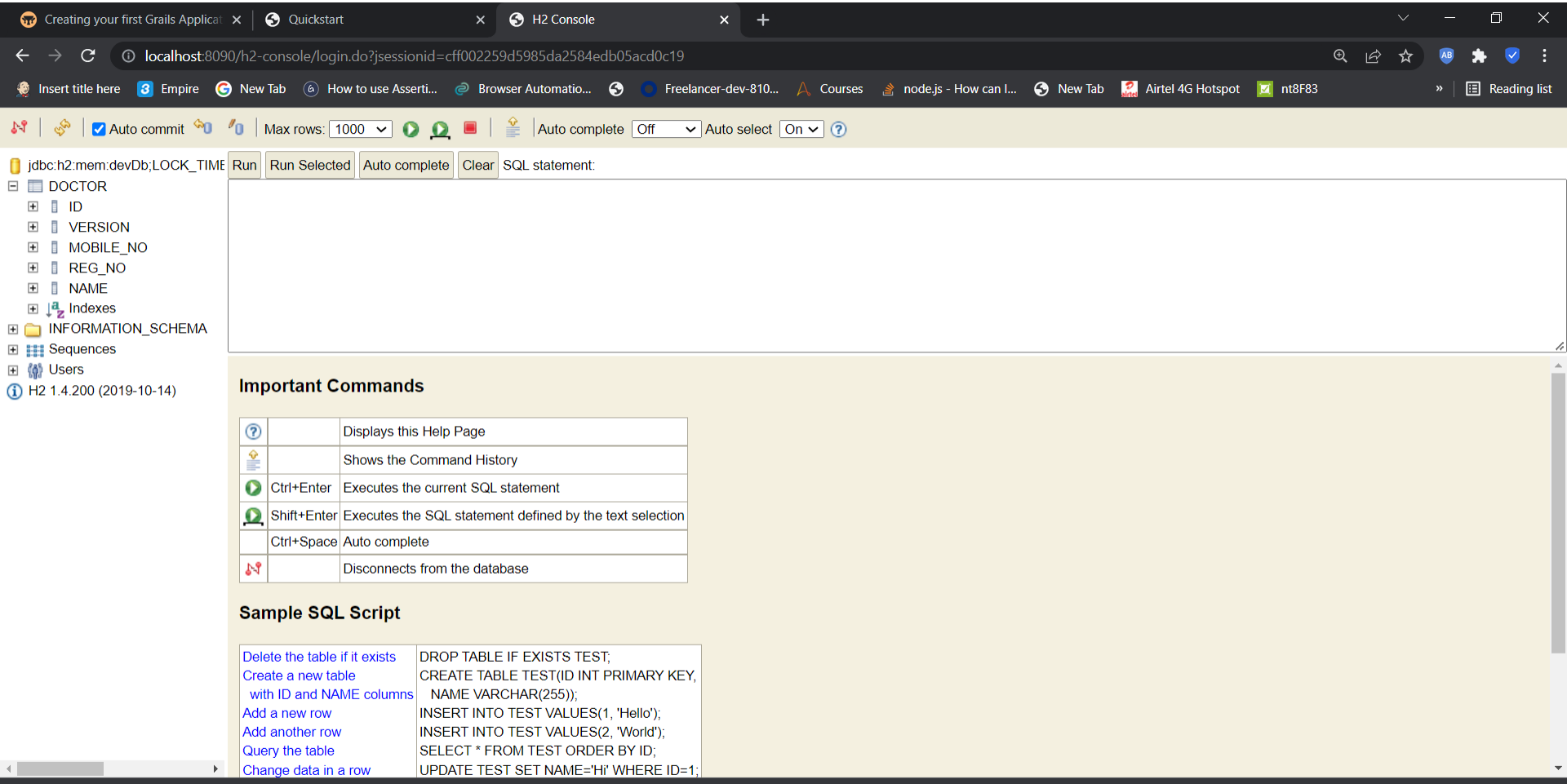
Grails Framework



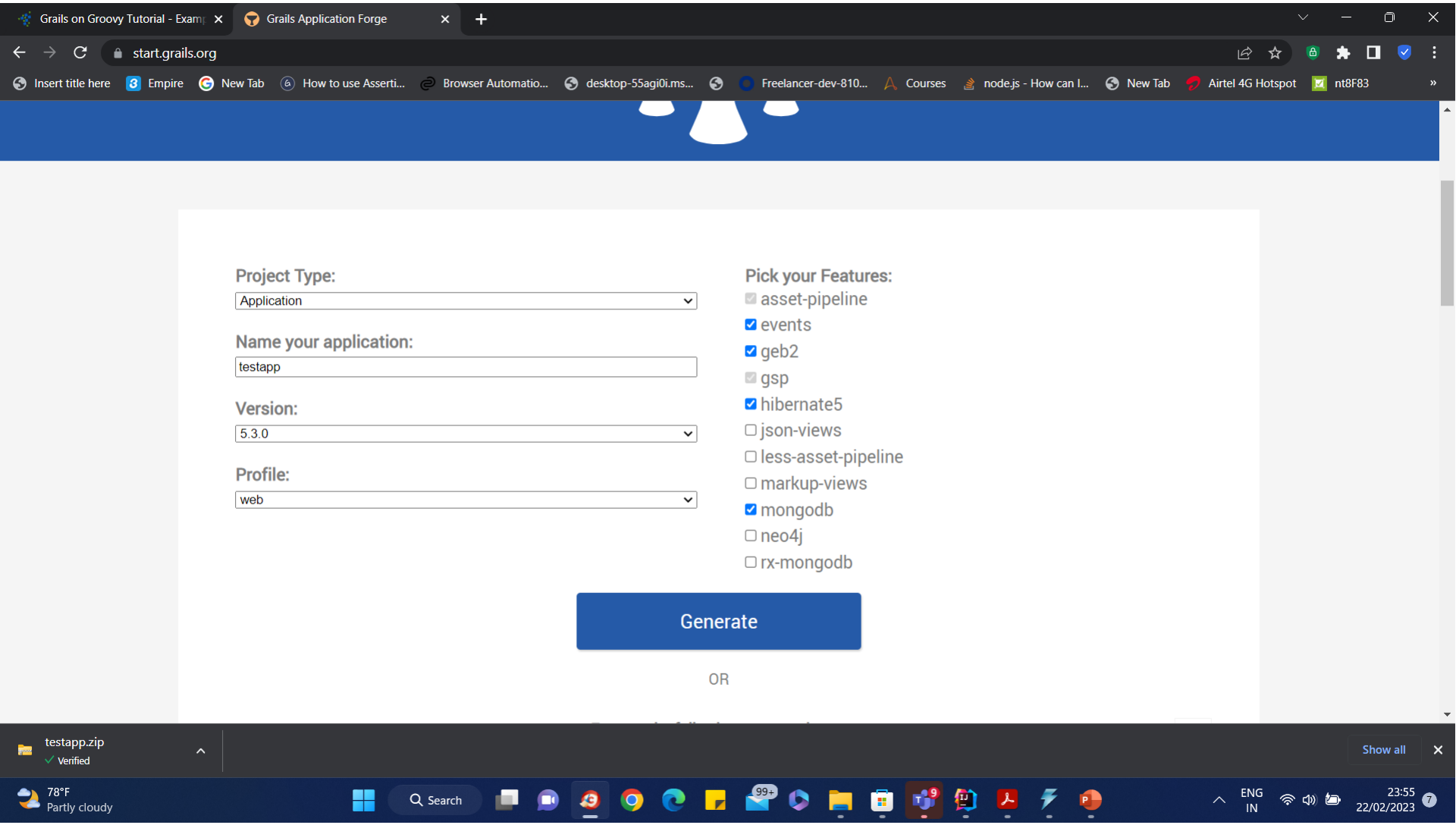
Grails Framework



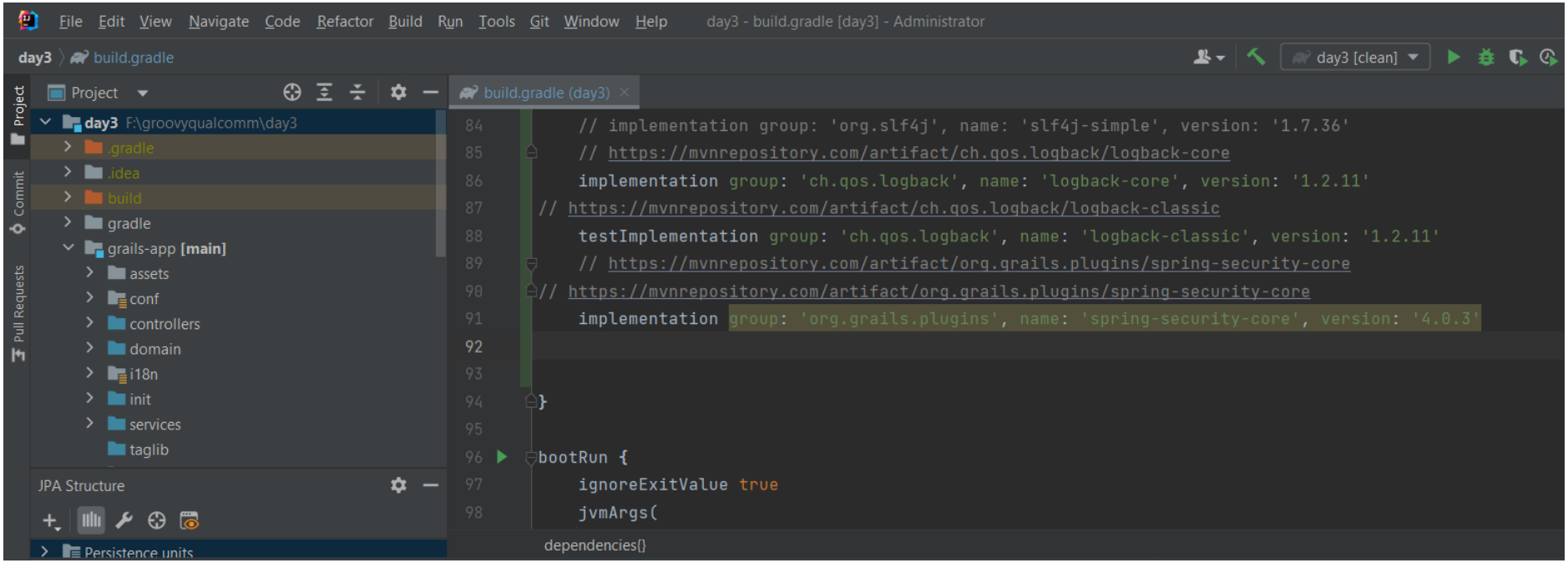
Grails Framework



Grails Framework

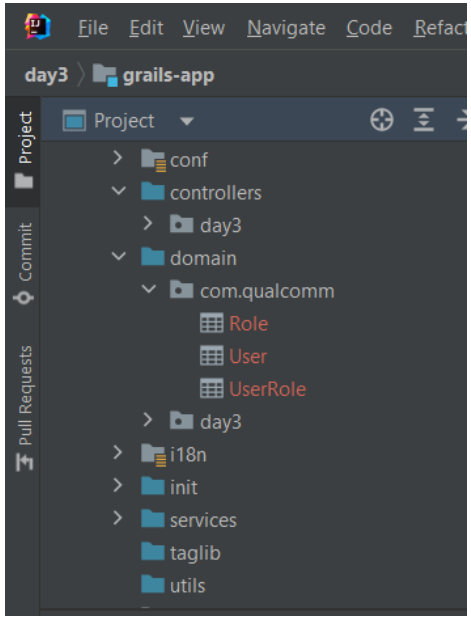


Grails Security



Grails Security

```
grails> s2-quickstart com.qualcomm User Role
| Creating User class 'User' and Role class 'Role' in package 'com.qualcomm'
| Rendered template PersonWithoutInjection.groovy.template to destination grails-app\domain\com\qualcomm\User.groovy
| Rendered template PersonPasswordEncoderListener.groovy.template to destination src\main\groovy\com\qualcomm\UserPasswo
rdEncoderListener.groovy
| Rendered template Authority.groovy.template to destination grails-app\domain\com\qualcomm\Role.groovy
| Rendered template PersonAuthority.groovy.template to destination grails-app\domain\com\qualcomm\UserRole.groovy
|
************************************************************
* Created security-related domain classes. Your            *
* grails-app/conf/application.groovy has been updated with *
* the class names of the configured domain classes;        *
* please verify that the values are correct.               *
************************************************************

grails>
```



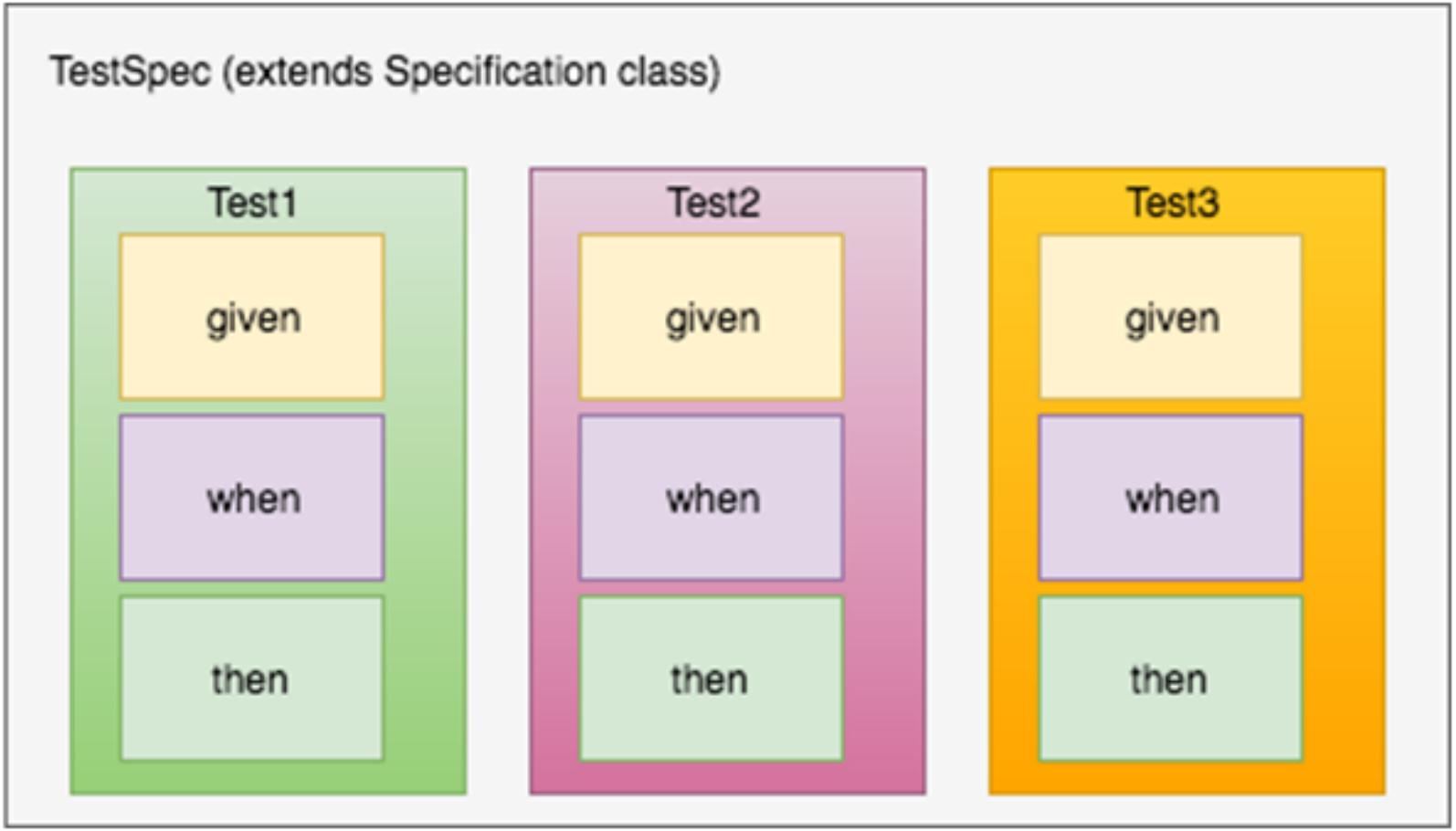
Anatomy of a Spock Specification

- Spock tests can be written for any language that runs on JVM.
- This means that even if your application code is in Java or Scala or Kotlin etc., you can still choose to write your unit/integration tests in Spock (over others like JUnit, JBehave, etc.).
- Spock test classes are called “Specifications” (Similar to Specs in BDD world) and you can extend the “Specification” class of the Spock framework. (Specification is the base class of the Spock framework).

Anatomy of a Spock Specification

- Each Spec file can contain multiple tests (similar to a JUnit test class file) and each test can optionally contain the BDD blocks i.e. given, when and then.
- This is similar to other BDD frameworks, where each of these sections can be correlated to Arrange, Act and Assert sections in the JUnit tests respectively.

Anatomy of a Spock Specification



Questions

Module Summary

- In this module we discussed
  - Overview of Groovy
  - Groovy Installation
  - Groovy vs Java
  - Data Types
  - Groovy Collections